Introduction to REST

Kenneth M. Anderson
University of Colorado, Boulder
CSCI 7818 — Lecture 6 — 08/27/2008

Credit Where Credit is Due

- Portions of this lecture are derived from material in “RESTful Web Services” by Leonard Richardson & Sam Ruby. As such, they are Copyright 2008 by O’Reilly

- Portions of this lecture are also derived from material in Wikipedia’s article on REST located here:

  - <http://en.wikipedia.org/wiki/Representational_State_Transfer>

Agenda

- Presentation by Haitao Cheng and Yibo Zhu on how to implement a BPEL process on Oracle BPEL Server and JDeveloper Designer
- Introduction to REST
- Ideas for future presentations
  - I will be presenting the material covered in the REST textbook and pointing you at relevant articles on-line
  - Students can present on things like
    - REST support in Ruby on Rails, Tools/Languages for creating REST, Comparison of REST/SOAP-based Web Services

REST

- Stands for Representation State Transfer
    - It is a software architectural style for distributed hypermedia systems
        - This style was created by Roy Fielding, and described in his dissertation from UC Irvine, by examining what made the Web such a successful technology
            - Due to popularity of REST, Roy Fielding has the distinction of having one of the most widely read Ph.D. dissertations
    - Aside: Roy and I went to UCI at the same time. He has had a hand in creating the HTTP and URL specifications, Apache, and finally REST
    - More info on Roy: <http://roy.gbiv.com/>

REST Design Principles (I)

1. Stateless Client/Server Protocol

    - Each message in the protocol contains all the information needed by the receiver to understand and/or process it. This constraint attempts to “keep things simple” and avoid needless complexity

2. Set of Uniquely Addressable Resources

    - “Everything is a Resource” in a RESTful system
    - Requires universal syntax for resource identification (e.g. URI)

REST Design Principles (II)

3. Set of Well-Defined Operations
    - that can be applied to all resources
    - In context of HTTP, the primary methods are
        - POST, GET, PUT, DELETE
    - these are similar (but not exactly) to the database notion of
        - CRUD (Create, Read, Update, Delete)
4. The use of Hypermedia both for Application Information and State Transitions
    - Resources are typically stored in a structured data format that supports hypermedia links, such as XHTML or XML

Web as embodiment of REST

- Stateless Protocol: HTTP
- Uniquely Addressable Resources: URIs
- Well-Defined Operations: HTTP Methods defined in HTTP 1.1 spec
- Use of Hypermedia: HTML is the primary Web format

- But it is important to note that while the Web is ONE embodiment of these principles, it is NOT the ONLY one
  - You can create RESTful systems using other protocols, methods, resources, and data formats but those new entities must conform to the previous four priniciples

REST Operations

- If we assume the standard four operations from HTTP 1.1, then

Method	Meaning	Idempotent?
GET	Retrieve a COPY of a Resource	YES
DELETE	Remove a Resource	YES
POST	Update a Resource	NO
PUT	Create a Resource	YES

- NOTE: Different people assign different meanings to POST and PUT

Idempotent?

- Definition of Idempotent: “Mathematics adjective denoting an element of a set that is unchanged in value when multiplied or otherwise operated on by itself.”
- From the HTTP/1.1 Spec: “Methods can also have the property of *idempotence* in that (aside from error or expiration issues) **the side-effects of N > 0 identical requests is the same as for a single request.**
  - The methods GET, HEAD, PUT and DELETE share this property. Also, the methods OPTIONS and TRACE SHOULD NOT have side effects, and so are inherently idempotent.”
- The concept of a method being idempotent is again one of simplicity. An identical request on an idempotent method causes at most one state change in the receiving system no matter how many times it is invoked.

Are These Operations Enough?

- Four (main) methods for ALL resources?
  - Is that enough?
  - What about collections?
    - How do I list the members of a collection?
    - How do I find resources?
- Answer: Collections and Search Results are viewed as “just another resource”
  - They have their own unique URLs and the standard methods can then be applied to them just like any other resource
    - Example: GET /users — Get resource listing all users
    - Example: GET /search/DNC — Get results that match keyword DNC

More on PUT and POST

- HTTP 1.1 spec says that POST's URI should point to a script whereas PUT's URI should point to the resource in question
    - That's nice, BUT Apache, for instance, doesn't handle the PUT method UNLESS you write a script that handles PUT requests and configure Apache to know about the script!
- Furthermore, I've seen forums where people insist that POST is only used for Create (ignoring much of the Web, IMHO) and that PUT is only used for Update
    - In practice, I've seen them used for both purposes
    - For instance, Amazon's S3 service uses PUT to create objects in buckets
        - PUT the OBJECT in the BUCKET

REST vs. RPC

- Useful to compare these approaches as RPC underlies all of the middleware systems we discussed previously along with SOAP-based web services
  - In RPC systems, the design emphasis is on **verbs**
    - What operations can I invoke on a system?
      - getUser(), addUser(), removeUser(), updateUser(), getLocation(), updateLocation(), listUsers(), listLocations(), etc.
  - In REST systems, the design emphasis is on **nouns**
    - User, Location
    - In REST, you would define XML representations for these resources and then apply the standard methods to them

Example

```
<user>
    <name>Jane</name>
    <gender>female</gender>
    <location href="http://www.example.org/locations/us/ny/new_york_city">
        New York City, NY, USA</location>
</user>
```

- This documentation is a representation used for the User resource
  - It might live at
    - http://www.example.org/users/jane/
  - If a user needs information about Jane, they GET this resource
  - If they need to modify it, they GET it, modify it, and PUT it back
  - The href to the Location resource allows savvy clients to gain access to its information with another simple GET request
- Implication: Clients cannot be “thin”; need to understand resource formats

From the Textbook

- Programmable Web
  - We are used to the human-readable Web provided by Web browsers
  - Web services should provide the Web's information to programs
- Two key questions for Web Services
  - How does it specify method information?
  - How does it specify scoping information?
- http://www.flickr.com/services/rest?method=flickr.photos.search&api_key=xxx&tag=penguins
  - Method: flickr.photos.search
  - Scope: tag=penguin

The Competing Architectures

- RESTful, Resource-Oriented Architectures
  - Method: HTTP Method
  - Scope: URI
- RPC-Style Architectures
  - Method: In the document envelope or as a “sticker on the envelope”
    - SOAP’s envelope exists inside HTTP’s envelope
  - Scope: In the body of the envelope
- REST-RPC Hybrid Architectures
  - Method: Varies; flickr: GET is used for retrieving stuff, but also for DELETING stuff
  - Scope: Varies; sometimes in body, sometimes in URI

Coming Up Next

- Chapters 2 and 3 in Textbook
- Any student presentations that I receive between now and next Wednesday!