95-702 Distributed Systems

Lecture 3: Principles of REST
REST

Representational State Transfer
REST

• Roy Fielding’s doctoral dissertation (2000)
• Fielding (along with Tim Berners-Lee) designed HTTP and URI’s.
• The question he tried to answer in his thesis was “Why is the web so viral”? What is its architecture? What are its principles?

Notes from “Restful Java with JAX-RS, Bill Burke, Orielly
REST Architectural Principles

• The web has addressable resources. Each resource has a URI.
• The web has a uniform and constrained interface. HTTP, for example, has a small number of methods. Use these to manipulate resources.
• The web is representation oriented – providing diverse formats.
• The web may be used to communicate statelessly – providing scalability
• Hypermedia is used as the engine of application state.

Notes from “Restful Java with JAX-RS, Bill Burke, Orielly”
Understanding REST

• REST is not protocol specific.
• SOAP and WS-* use HTTP strictly as a transport protocol.
• HTTP may be used as a rich application protocol.
• Browsers usually use only a small part of HTTP.
• HTTP is a synchronous request/response network protocol used for distributed, collaborative, document based systems.
• Various message formats may be used – XML, JSON,..
• Binary data may be included in the message body.

Notes from “Restful Java with JAX-RS, Bill Burke, Orielly
Principle: Addressability

- Addressability (not restricted to HTTP)
  Each HTTP request uses a URI.
  The format of a URI is well defined:

  scheme://host:port/path?queryString#fragment

  The scheme need not be HTTP. May be FTP or HTTPS.
  The host field may be a DNS name or a IP address.
  The port may be derived from the scheme. Using HTTP implies port 80.
  The path is a set of text segments delimited by the “/”.
  The queryString is a list of parameters represented as name=value pairs. Each pair is delimited by an “&”.
  The fragment is used to point to a particular place in a document.

  A space is represented with the ‘+’ characters. Other characters use % followed by two hex digits.
Principle: Uniform Interface (1)

A uniform constrained interface

- No action parameter in the URI

- HTTP
  - GET - read only operation
    - idempotent (once same as many)
    - safe (no important change to server’s state)
  - may include parameters in the URI
    http://www.example.com/products?pid=123

Notes from “Restful Java with JAX-RS, Bill Burke, Orielly
Principle: Uniform Interface (2)

HTTP
PUT - store the message body
- insert or update
- idempotent
- not safe

Notes from “Restful Java with JAX-RS, Bill Burke, Orielly
Principle: Uniform Interface (3)

HTTP
POST - Not idempotent
  - Not safe
  - Each method call may modify the resource in a unique way
  - The request may or may not contain additional information
  - The response may or may not contain additional information
  - The parameters are found within the request body (not within the URI)

Notes from “Restful Java with JAX-RS, Bill Burke, Orielly
Principle: Uniform Interface (4)

HTTP
DELETE  - remove the resource
  - idempotent
  - Not safe
  - Each method call may modify the resource in a unique way
  - The request may or may not contain additional information
  - The response may or may not contain additional information

HTTP HEAD, OPTIONS, TRACE and CONNECT are less important.

Notes from “Restful Java with JAX-RS, Bill Burke, Orielly
Principle: Uniform Interface (5)

Note that SQL has only four operations: SELECT, INSERT, UPDATE and DELETE.

JMS and MOM have, essentially, two operations: SEND and RECEIVE.
What does a uniform interface buy?

Familiarity
We do not need a general IDL that describes a variety of method signatures. We already know the methods.

Interoperability
WS-* has been a moving target
HTTP is widely supported

Scalability
Since GET is idempotent and safe, results may be cached by clients or proxy servers.
Since PUT and DELETE are both idempotent neither the client or the server need worry about handling duplicate message delivery

Notes from “Restful Java with JAX-RS, Bill Burke, Orielly
Principle: Representation Oriented(1)

• Representations of resources are exchanged.
• GET returns a representation.
• PUT and POST passes representations to the server so that underlying resources may change.
• Representations may be in many formats: XML, JSON, YAML, etc., ...
Principle: Representation Oriented (2)

• HTTP uses the CONTENT-TYPE header to specify the message format the server is sending.
• The value of the CONTENT-TYPE is a MIME typed string. Versioning information may be included.
• Examples:
  - text/plain
  - text/html
  - application/vnd+xml;version=1.1
• “vnd” implies a vendor specific MIME type
Principle: Representation Oriented(3)

- The ACCEPT header in content negotiation.
- An AJAX request might include a request for JSON.
- A Java request might include a request for XML.
- Ruby might ask for YAML.
Principle: Communicate Statelessy

• The application may have state but there is no client session data stored on the server.
• If there is any session-specific data it should be held and maintained by the client and transferred to the server with each request as needed.
• The server is easier to scale. No replication of session data concerns.

Notes from “Restful Java with JAX-RS, Bill Burke, Orielly
Principle: HATEOAS

- Hypermedia is document centric but with the additional feature of links.
- With each request returned from a server it tells you what interactions you can do next as well as where you can go to transition the state of your application.
- Example:
  
  ```xml
  <order id = "111">
    <customer>http://.../customers/3214
    <order-entries>
      <order-entry>
        <qty>5
        <product>http://.../products/111
  ```

Notes from “Restful Java with JAX-RS, Bill Burke, Orielly"