Software Reuse and Component-Based SE

ITSE422

Lecture # 8: REST Services

REST

- REpresentational State Transfer
- 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" by Bill Burke.

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 resourses.
- 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.

Understanding REST

- REST is not protocol specific. It is usually associated with HTTP but its principles are more general.
- SOAP and WS-* use HTTP strictly as a transport protocol.
- But 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.

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

Principle: Uniform Interface (2)

HTTP

PUT - store the message body

- insert or update
- idempotent
- not safe

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)

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.

Principle: Uniform Interface (5)

Does HTTP have too few operations?

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

JMS and MOM have, essentially, two operations: SEND and RECEIVE

SQL and JMS have been very useful.

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.

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 Statelessly

- 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.

Principle: HATEOAS

- **Hypermedia as the Engine of Application State**
- 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:

```
<order id = “111”>
    <customer>http://…/customers/3214
    <order-entries>
        <order-entry>
            <qty>5
            <product>http://…/products/111
```

Principle: HATEOS

- From Wikipedia:
- A REST client enters a REST application through a simple fixedURL. All future actions the client may take are discovered within resource representations returned from the server. The media types used for these representations, and the link relations they may contain, are standardized. The client transitions through application states by selecting from the links within a representation or by manipulating the representation in other ways afforded by its media type. In this way, RESTful interaction is driven by hypermedia, rather than out-of-band information.[1]

Principle: HATEOS

- Knock Knock Joke Example
- Netbeans RESTKnocker

Bing Maps Using REST

- Visit the URL (using my key):

http://dev.virtualearth.net/REST/v1/Locations/New%20York?output=xml&key=AqMWEeRufD1Ch2uhYsyD10OPbLGMs_GATcB8Xd8trcvybpNuDRcMo6U6uVCqOara

- How does your browser react?
- Why?
- Change XML to JSON.
- How does your browser react?
- Why?

WeatherBug API

- Here is a nice description of the weather bug REST style API:

- http://developer.weatherbug.com/docs/read/WeatherBug_API_JSON

- WeatherBug also provides a SOAP based service.

- Questions?