



What is REST?

Glen Campbell
Engineering Manager, Yahoo! Tech
glen@broadpool.com



“There are only two hard things in computer science: cache invalidation and naming things.”

– Phil Karlton



What is REST?

- It stands for “Representational State Transfer”
- Coined by Roy Fielding in his Ph.D dissertation
- One of several architectural styles for implementing services in a distributed hypermedia system
- REST *typically* (but not mandated) involves serving XML over HTTP (but so do many other web services).



Specifically, REST is...

- Client-Server – specifically aimed at serving information to a client
- Stateless – state is not maintained by the system
- Cacheable – services can be cached, and can signal if not cacheable
- Uniform Interface – a universal name for data



REST is made up of...

- Data - resources, representations
- Connectors - client, server, cache, resolver, tunnel
- Components - origin server, gateway, proxy, user agent



What REST is NOT

- NOT a protocol
- NOT a language or data transfer specification
- NOT a development methodology
- NOT procedural
- NOT XML/HTTP (so is SOAP)
- NOT XML-RPC/HTTP (remember, it's not procedural)



More ...

- There's no such thing as "REST format..."
- There's no such thing as "the REST standard..."
- There's no such thing as "REST-compliant" (i.e., there's no body that validates REST compliance)



Defining Features of REST

- Information represented as URIs
- Stateless connections
- Information is transferred between clients and servers using connectors such as HTTP
- Data representation (resources) is separate from actions (verbs)



“There is no problem in computer science that can’t be solved by adding an additional level of indirection.”

– Robert Cousins



What is a resource?

- A resource is an abstraction with a name: a Universal Resource Locator: `{protocol}://{host}/{resource}?{querystrings}`
- No guarantee that a URL maps to anything specific in the real world
<http://example.com/account/cgross> - simply a name for a resource; no guarantee that maps to anything physical in the real world.
- In the Real World, a resource is a *document* (see “10 principles of SOA”)
- A document may have embedded hyperlinks that reference other, related documents. You can think of this as a *web* of documents. Since the URL is global in scope, this makes it a *world-wide web*.



URL Semantics

- There are none! The URL is simply a unique identifier for a resource somewhere on the world-wide web. Any semantics are provided by the application.
- <http://example.com/this/is/a/rather/long/url/dont/you/think?> is a perfectly valid URL, though it does not make sense.
- <http://example.com/user/glen> is much more meaningful to people who view the URL.



Some REST(?) Examples



Yahoo! Shopping API

- Information represented by a URL
[http://api.shopping.yahoo.com/ShoppingService/v1/catalogSpecs?
&catalogid=1991675140&appid=YahooDemo](http://api.shopping.yahoo.com/ShoppingService/v1/catalogSpecs?&catalogid=1991675140&appid=YahooDemo)
- Data returned as XML (JSON, Serialized PHP)
- Actions defined by HTTP verbs (GET, POST, DELETE)
- Not quite REST-ful:
 - Query strings—this means that the results are not cacheable (according to the HTTP standard)



Amazon S3 REST API

- Information represented by URIs:
`http://{yourapp}.s3.amazonaws.com/{bucket}/{object}`
- Data returned as...anything! (REST does NOT mean XML)
- Actions defined by HTTP verbs: GET, PUT, POST, DELETE
- Status returned using HTTP codes: 2xx, 3xx, 4xx, 5xx



Flickr Public API

- Not really REST:
<http://api.flickr.com/services/rest/?method=flickr.test.echo&name=value>
- Uses URIs to represent functions (procedures), not information
 - flickr.photos.delete
 - flickr.photos.search
- Non-cacheable
- Any time you see “?method=” or “?action=”, it’s probably not REST



Example: Siteframe

- <http://siteframe.org/api/V1/User/glen?key=c90ef573fffb43c17d4>
- URL format:
`{protocol}://{host}/api/V{version}/{class}/{identifier}`
- Response: an XML document for the specified class/identifier pair.
- Methods supported: GET, PUT, POST, DELETE
- Status: 200 OK, 404 NOT FOUND, 500 ERROR, etc.

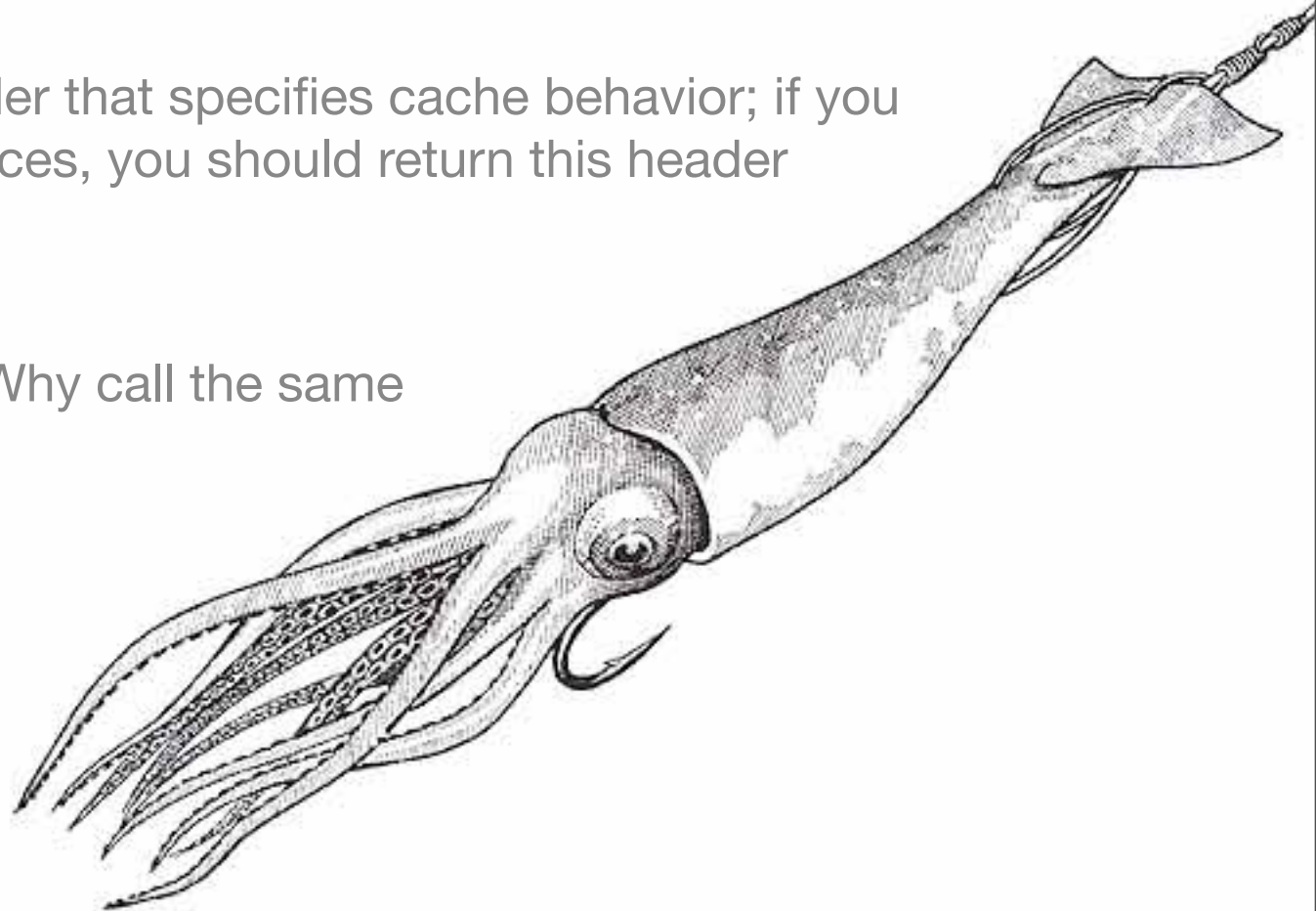


Benefits of REST

- Cacheable—since it follows the HTTP protocol, responses can be cached using any HTTP-compliant cache mechanism
- No additional software or licenses required
- No knowledge of the internals of a URL is needed to determine if it changes data or not.
- Bookmark-able: resource URLs are not dependent upon cookies or browser states
- Easily debugged with a browser

Caching Web Services

- ICP: Internet Caching Protocol
- Squid—<http://www.squid-cache.org>: the *de facto* standard for web caching
- Cache-Control: HTTP header that specifies cache behavior; if you are writing REST web services, you should return this header expressly.
- Why cache webservices? Why call the same service multiple times?





Where is REST used?

- Between servers and clients: for example, to retrieve data from Amazon S3 and serve it on a different website
- Between servers and other servers (the second server is effectively a client)
- Between websites and browsers: for example, to perform an update using AJAX and display the result without a page refresh.



What's REST bad at?

- Stateful transactions: for complex web applications requiring intermediate states to be maintained, either REST should be avoided or the application must be redesigned to be stateless.
- Bidirectional applications (e.g., chat). Since REST is client-server (because of HTTP), it's not suitable for any bidirectional protocol.
- Authentication: HTTP, in general, is not very good at authentication. Any really secure REST application will probably have a separate authentication layer.



Ok, make it simple for me

- Information (the YHOO stock quote, today's weather, my résumé) is a resource and has a name
- Names are URLs (Uniform Resource Locator)
- The names don't change over time (e.g., "today's weather" always has the same name)
- I specify what to do with the information using HTTP: GET (retrieve), PUT (create), POST (update), DELETE



“There are only two things you can do with a mouthful of too-hot coffee. Both of them are wrong.”
– Dave Wyland



ROA vs. SOA

- Is there a difference between a Resource-Oriented Architecture (REST) and a Service-Oriented Architecture?
- Some distinctions:
 - SOA – services are provided by the server and accessed via messages
ROA – information is provided by the server and services are provided by the client
 - SOA – services (applications) require state, can be complex
ROA – not good at multi-tier applications; state maintained by client



Wyland's Law: Anything that can automatically be done *for* you, can automatically be done *to* you.



Resources

- Roy Fielding's Dissertation (Fielding, 2000)
http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
- How I Explained REST to my Wife (Ryan Tomayko)
<http://tomayko.com/articles/2004/12/12/rest-to-my-wife>
- 10 Principles of SOA (Stefan Tilkov)
<http://www.infoq.com/articles/tilkov-10-soa-principles>
- Representational State Transfer (Wikipedia)
http://en.wikipedia.org/wiki/Representational_State_Transfer
- Web Services Theory and Practice (Tim Bray)
<http://www.tbray.org/ongoing/When/200x/2004/04/26/WSTandP>