web API design choices

Daniel Jackson
what exactly is ReST?

nowadays, it’s cool to be ReSTful
› but what does that mean?

where ReST came from
› Roy Fielding thesis at UC Irvine
› software architecture as a lens on the design of the web
› chapter on “representation state transfer”

but like many web notions
› not precisely defined
› extended and evolved
naughty and nice URIs

naughty
› http://store.com/shop.asp?cmd=nextPage
	nice
› http://store.com/items/123/images/4

request structure tells you a lot
› about how the app was designed
resources vs. operations on state

traditional approach
› app has an internal state $S$, often implicit
› use of API involves calling operations
› $S.addReview(subject, review)$

resource-based approach
› app stores collection of resources $R$ with names $N$
› use of API involves resolving names & updating collection
› $POST /subjects/12/reviews/$
SOAP example

POST /reviews HTTP/1.1
Host: www.tipster.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body xmlns:m="http://www.tipster.com/review">
  <m:GetReviews>
    <m:SubjectName>Peets Coffee</m:SubjectName>
    <m:ReviewAuthor>Chloe Closure</m:ReviewAuthor>
  </m:GetReviews>
</soap:Body>
</soap:Envelope>

request body
› XML document with rich structure
› conforms to interface defined by given XML namespace
navigation expressions

URIs describe sets of resources
> /subjects
> /subjects/4
> /subjects/4/reviews
> /subjects/4/reviews/3
disciplined use of HTTP methods

old approach
› request is GET, with query string giving op and args
› or request is POST, with form data for args
› eg, GET /reviews.php?action=update&reviewid=23&rating=5

ReSTful approach
› request is resource name + HTTP method
› eg, PUT /subject/3/review/2

standard interpretations
› GET: return resource(s) cacheable
› POST: add new resource
› PUT: replace resource idempotent
› DELETE: replace resource idempotent
**examples of ReSTful requests**

get reviews of subject 3
› GET /subjects/3/reviews

create new review of subject 3
› POST /subjects/3/reviews

update review 2 of subject 3
› PUT /subjects/3/reviews/2

delete review 2 of subject 3
› DELETE /subjects/3/reviews/2

obtaining forms: not so clear
› GET /subjects/3/reviews/new
› GET /subjects/3/reviews/2/edit

**note:** sadly, can’t specify PUT as action in plain HTML form, so often use POST instead
self-describing data

when a response comes back
› what do you do with the data?

traditional approach
› API doc tells you what to do with it

web approach
› resource carries its own description
› media type (eg, as MIME)
› human-readable fields (in XML or JSON)
a DNS resource record
Google directions response message

```json
{
  "status": "OK",
  "routes": [ {
    "summary": "I-40 W",
    "legs": [ {
      "steps": [ {
        "travel_mode": "DRIVING",
        "start_location": { 
          "lat": 41.8507300,
          "lng": -87.6512600
        },
        "end_location": { 
          "lat": 41.8525800,
          "lng": -87.6514100
        },
        "polyline": { 
          "points": "a~l~Fjk~uOwHJy@P"
        },
        "duration": { 
          "value": 19,
          "text": "1 min"
        }  
      }]
    }
  }]
}
```
hypertext vs published interfaces

when you get a response
› what do you do next?

traditional approach
› given type of response and state of app...
› API doc tells you which ops you can call
› eg, DNS packet seen before

hypertext approach
› response includes operations to use next
› as machine-readable as possible

example: atom feed format
› now used as a general data format (eg by Google Data)
example: atom feed

```xml
<?xml version="1.0" encoding="utf-8"?>

<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Example Feed</title>
  <subtitle>A subtitle.</subtitle>
  <link href="http://example.org/feed/" rel="self"/>
  <link href="http://example.org/"/>
  <id>urn:uuid:60a76c80-d399-11d9-b91C-0003939e0af6</id>
  <updated>2003-12-13T18:30:02Z</updated>
  <author>
    <name>John Doe</name>
    <email>johndoe@example.com</email>
  </author>
  <entry>
    <title>Atom-Powered Robots Run Amok</title>
    <link href="http://example.org/2003/12/13/atom03"/>
    <link rel="alternate" type="text/html" href="http://example.org/2003/12/13/atom03.html"/>
    <link rel="edit" href="http://example.org/2003/12/13/atom03/edit"/>
    <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
    <updated>2003-12-13T18:30:02Z</updated>
    <summary>Some text.</summary>
  </entry>
</feed>
```

› note use of link relations, how server returns self id too
discussion

hypertext vs. published API
› decoupling of client from server
› but makes links less permanent!

fixed methods vs. operations
› only good for data repositories?
› not good for search (hence Yahoo’s YQL, eg)

stateless vs. stateful interactions
› iterators awkward? article/page/5?
› hypertext to rescue: can use nextpage link relation
takeaways

you should
› design your URI scheme carefully
› make your interactions stateless
› exploit hypertext

you’ll probably want to
› use ReST rather than SOAP
› use JSON rather than XML

don’t
› abuse HTTP methods (eg, side-effecting GET)
› expose implementation structure in URLs
› use ad hoc query strings for operations