HYPERMEDIA APIS

Jonathan R. Engelsma, Ph.D.

TOPICS

- Definition of Hypermedia APIs
- The "Richardson Maturity Model" for describing RESTful APIs.
- The State of WEB APIs Today
- Hypermedia Controls
- HATEOAS: Hypermedia As The Engine Of Application State

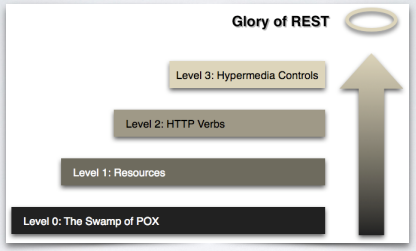
Def of Hypermedia API: an API in which application state transitions are driven by the client selection of server-provided choices that are present in representations received by the client.

REST's key insight: the human web is extremely powerful, and this power should be fully given to programs.

THE IMPLICATIONS

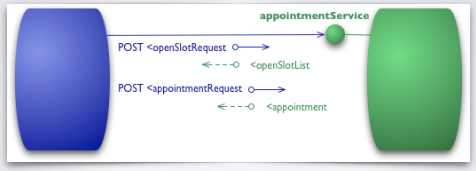
- Out-of-band information (e.g. human API Docs) should not be driving interaction, hypermedia controls should be.
- Client's should not assume a fixed resource structure defined by out-of-band info, as that is the data-oriented equivalent of RPC's functional coupling between client/server.

RICHARDSON MATURITY MODEL



Source: http://martinfowler.com/articles/richardsonMaturityModel.html

LEVEL 00: SWAMP OF POX



Source: http://martinfowler.com/articles/richardsonMaturityModel.html

LEVEL 00: SWAMP OF POX

Step 1: Request an appointment on a given day with a given doctor

```
POST /appointmentServ
[various other heade

<openSlotRequest dat
```

```
HTTP/1.1 200 OK
[various headers]

<openSlotList>
  <slot start = "1400" end = "1450">
    <doctor id = "mjones"/>
  </slot>
  <slot start = "1600" end = "1650">
    <doctor id = "mjones"/>
  </slot>
</openSlotList>
```

Step 2: Response - available time slots

LEVEL 00: SWAMP OF POX

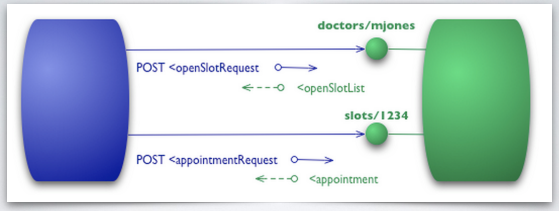
Step 3: Request an appointment

```
POST /appointmentService HTTP/1.1
[various other headers]
```

```
HTTP/1.1 200 OK
[various headers]

<appointment>
  <slot doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointment>
```

Step 4: Response - Appointment confirmed.

LEVEL 01: ADD RESOURCES



Source: http://martinfowler.com/articles/richardsonMaturityModel.html

LEVEL 01: ADD RESOURCES

Step 1: Request appointment on a given day with a given doctor:

```
POST /doctors/mjones HTTP/1.1
[various other headers]

<openSlotRequest date = "2010-01-04"/>
```

```
HTTP/1.1 200 OK
[various headers]

<openSlotList>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650"/>
</openSlotList>
```

Step 2: Response - available time slots

LEVEL 01: ADD RESOURCES

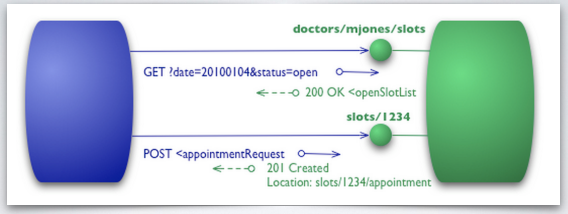
Step 3: Request an appointment

```
POST /slots/1234 HTTP/1.1
[various other headers]

<appointmentRequest>
```

```
HTTP/1.1 200 OK
[various headers]

<appointment>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointment>
```

Step 4: Response - Appointment confirmed.

LEVEL 02: ADD HTTP VERBS



Source: http://martinfowler.com/articles/richardsonMaturityModel.html

LEVEL 02: ADD HTTP VERBS

Step 1: Request appointment on a given day with a given doctor:

```
GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
Host: royalhope.nhs.uk
```

```
HTTP/1.1 200 OK
[various headers]

<openSlotList>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650"/>
</openSlotList>
```

Step 2: Response - available time slots

LEVEL 02: ADD HTTP VERBS

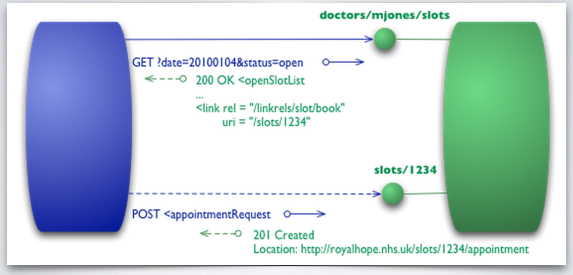
Step 3: Request an appointment

```
POST /slots/1234 HTTP/1.1
[various other headers]

<appointmentRequest>
```

```
HTTP/1.1 201 Created
Location: slots/1234/appointment
[various headers]

<appointment>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointment>
```

Step 4: Response - Appointment confirmed.

LEVEL 03: ADD HYPERMEDIA CONTROLS



Source: http://martinfowler.com/articles/richardsonMaturityModel.html

LEVEL 03: ADD HYPERMEDIA CONTROLS

Step 1: Request appointment on a given day with a given doctor:

```
GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
Host:
```

```
HTTP/1.1 200 OK
[various headers]

<openSlotList>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450">
    <link rel = "/linkrels/slot/book"
          uri = "/slots/1234"/>
  </slot>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650">
    <link rel = "/linkrels/slot/book"
          uri = "/slots/5678"/>
  </slot>
</openSlotList>
```

Step 2: Response - available time slots

LEVEL 03: ADD HYPERMEDIA CONTROLS

Step 3: Request an appointment

```
POST /slots/1234 H
[various other hea

<appointmentReques
  <patient id = "js
</appointmentReque
```

```
HTTP/1.1 201 Created
Location: http://royalhope.nhs.uk/slots/1234/appointment
[various headers]

<appointment>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
  <link rel = "/linkrels/appointment/cancel"
        uri = "/slots/1234/appointment"/>
  <link rel = "/linkrels/appointment/addTest"
        uri = "/slots/1234/appointment/tests"/>
  <link rel = "self"
        uri = "/slots/1234/appointment"/>
  <link rel = "/linkrels/appointment/changeTime"
        uri = "/doctors/mjones/slots?date=20100104&status=open"/>
  <link rel = "/linkrels/appointment/updateContactInfo"
        uri = "/patients/jsmith/contactInfo"/>
  <link rel = "/linkrels/help"
        uri = "/help/appointment"/>
</appointment>
```

Step 4: Response - Appointment confirmed.

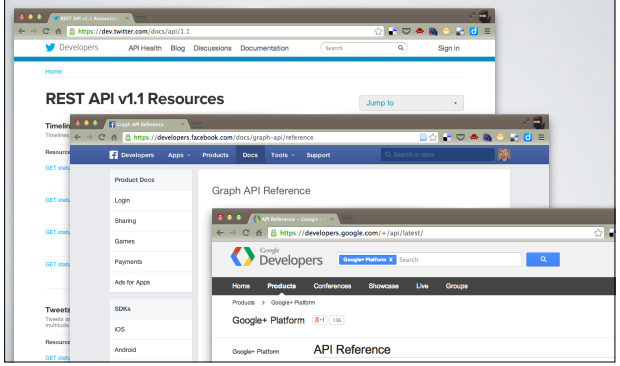
RELATIONSHIP TO DESIGN TECHNIQUES

- Level 1: handle complexity with divide and conquer. Break large server endpoint into multiple resources.
- Level 2: Introduce common set of verbs to remove unnecessary variation.
- Level 3: Introduces discoverability, makes API more self documenting.

HYPERMEDIA API BENEFITS

- Avoid breaking clients when API changes. (especially URL changes).
- Client depends on API to provide the control logic/workflow.
- Helps developers (and maybe clients as well) explore/ discover and make better use of an API.
- Help standardize API representations and workflow across the web!

STATE OF WEB APIS TODAY



- ~~Client Server~~
- ~~Stateless Interaction~~
- ~~Caching~~
- ~~Uniform Interface~~
  - ~~Identification of resources~~
  - ~~manipulation through representations~~
  - self-descriptive messages
  - HATEOAS
- ~~Layered System~~

THE MISSING LINK

- We need a hypermedia format with hypermedia controls that:
  - tell us how to construct an HTTP request: e.g. URL, method, and headers.
  - make promises about the HTTP response: e.g. status code, headers, likely data.
  - suggest how the client should integrate the response into its workflow.

EXAMPLE: HTML

```
<a href="dashboard.php"> Display Dashboard </a>
```

- Meaning: if user clicks on text "Display Dashboard"
  - form the URL, e.g. http://example.com/dashboard.php
  - send HTTP **GET** request to this URL.
  - replace current browser page with a rendering of whatever content type you get back. (probably text/html)

EXAMPLE: HTML

```
<img src="thumbnail.png"/>
```

- Meaning: when rendering an HTML document and an <img> tag is encountered:
  - form a URL using the src attribute, e.g. http://example.com/thumbnail.png
  - send HTTP **GET** request to this URL.
  - you should get back image data - embed a rendering of it within the current rendered web page.

EXAMPLE: HTML

```
<form action="save.php" method="post">
 <input type="text" name="email" value=""/>
 <input type="submit" value="Post"/>
</form>
```

- Meaning: when submit button is pressed:
  - form the URL (http://example.com/save.php)
  - submit an HTTP **POST** with the value of the email text input field included in the body.
  - you might get back a redirect status code, follow it and render, replacing current page with new content.

EXAMPLE: HTML

```
<link type="text/css" rel="stylesheet"
href="css/fluid_grid.css" />
```

- Meaning: when link is encountered in the HTML
  - create URI from href (http://example.com/css/fluid_grid.css)
  - fetch via an HTTP **GET**.
  - resulting document should be CSS which contains rules that will guide the styling in the page render.

HATEOAS

- Hypermedia as the engine of application state
  - REST client enters API through a fixed URL.
  - All future actions client may take are discovered within resource representations from the server.
  - Media types of these representations and the links they contain are "standardized".
  - Interaction is driven by hypermedia rather than out-of-band information.

CAPTURING SEMANTICS

- To describe protocol and application level semantics, we need to have:
  - semantic descriptors: allow us to machine process the attributes within representations.
  - link semantics (relationships): let us define the meaning of links.

NO STANDARD EXISTS!

- Some possible hypermedia representation options:
  - Using HTML as hypermedia format with Microformats or Microdata
  - Use a more general purpose hypermedia format: HAL, Siren
  - Use a "standard collections pattern" format: AtomPub, Collection+JSON
  - Create a totally new custom hypermedia format.

PROFILES

- Formal (e.g. machine processable) descriptions of link relationships and semantic descriptors (see RFC 6906)
- Possible approaches to authoring profiles:
  - XMDP, ALPS, JSON-LD

REPRESENTATION FORMATS

- Popular formats include JSON, XML
  - Neither define hypermedia controls!
  - Possible Solutions:
    - JSON: use JSON-LD (http://json-ld.org/)
    - XML: use Xlink? (http://www.w3.org/TR/xlink/)

CREATING A HYPERMEDIA API

1. List all the info clients will want to get from the API (e.g. identify semantic descriptors)
2. Draw a state diagram of API (state transitions are links).
3. Use existing link relations and semantic descriptors where possible. (see also alps.io)

CREATING A HYPERMEDIA API

4. Choose your representation media type (JSON, HAL, Siren, etc.)
5. Write a profile (JSON-LD, ALPS, free format).
6. Implement!
7. Publish initial URL!

READING ASSIGNMENT

- http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven - Roy Fielding Rant on Hypermedia.
- http://martinfowler.com/articles/richardsonMaturityModel.html - Fowler on RMM
- http://blog.programmableweb.com/2014/02/27/hypermedia-apis-the-benefits-of-hateoas/
- http://signalvnoise.com/posts/3373-getting-hyper-about-hypermedia-apis - DHH opposition to hypermedia APIs
- On Using JSON-LD to Create Evolvable RESTful Services. Lanthaler et al. (see Blackboard)

RECOMMENDED READ

- RESTful Web APIs by Richardson and Amundsen.