Representational State Transfer

Alexandre Rodrigues

14-04-2016

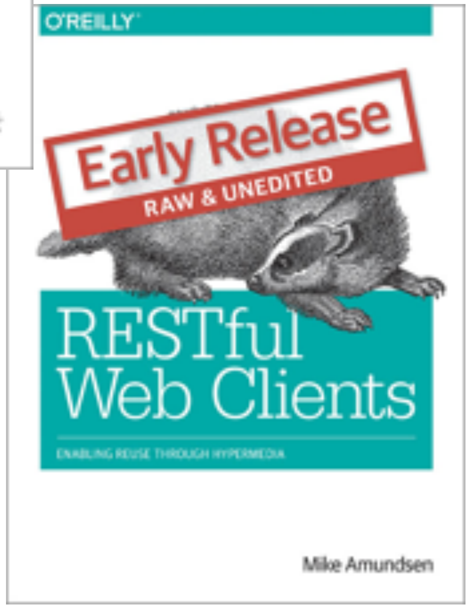
(Or) Design of evolvable client-server applications

A little bit of history

- 2000 - Roy Fielding's dissertation "Architectural Styles and the Design of Network-based Software Architectures" [1]
- 2007 - RESTful Web services book [2]
- 2008 - Roy Fielding's blog post explaining the importance of Hypermedia in REST [3]
- 2013 - RESTful Web APIs book [4]
- ? - RESTful Web Clients book [5]

Web Services for the Real World
RESTful Web Services
O'REILLY
Leonard Richardson & Sam Ruby

Services for a Changing World
RESTful Web APIs
O'REILLY
Leonard Richardson, Mike Amundsen & Sam Ruby



REST is not

- a synonym of HTTP (Hypertext Transfer Protocol)
  - HTTP is an application layer protocol that is adequate to the Web requirements
- a message format
- a protocol
- a type of interface
- a cool and better way do implement Web Services
- SOAP substitute

REST is an Architectural Style

"An architectural style is a coordinated set of constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms that style."

– Roy Fielding

Architectural Styles examples

- Pipe and Filter
- Replicated Repository
- Client-Server
- Client-Stateless-Server
- Client-Cache-Stateless-Server
- Remote Data Access
- Event-based Integration
- Brokered Distributed Objects

“REST is a named set of constraints on component interaction that, when obeyed, cause the resulting architecture to have certain properties (preferably, desirable properties).”

– Roy Fielding

Architectural properties of key interest

- Performance
    - Network performance, User-perceived performance, Network efficiency
- Scalability
- Simplicity (separation of concerns)
- Modifiability
    - **Evolvability**, Extensibility, Customisability, Configurability, Reusability
- Visibility
- Portability
- Reliability

WWW Application Domain Requirements

- Low Entry-Barrier
- Extensibility
- Distributed Hypermedia (holds extensibility property and lowers the entry-barrier when a human being is making all the decisions)
- Internet-Scale
    - Anarchic Scalability
    - Independent Deployment

How REST helps holding those requirements?

REST constraints

- Client-Server
- Stateless (self contained messages)
- Cache
- <u>Uniform Interface</u>
- Layered Systems
- Code-On-Demand (Extensibility)

Uniform Interface

"REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of the application state."

– Roy Fielding

Identification of Resources

- Every relevant resource should have an uniform resource identifier (URI)
- The resource state may change but its URI stays the same

Manipulation of resources through representations

- A resource can be anything (a resource exists if it has an URI)
- A resource is never manipulated directly, always through representations
- Clients and servers manipulate resources by sending representations back and forth using a set of standardised methods

Self-descriptive messages

- “Interaction is stateless between clients”
- “Standard methods and media types are used to indicate semantics and exchange information”
- “Responses explicitly indicate cacheability”

Hypermedia as the engine of application state

1. All application state is kept on the client side (resource state is server’s responsibility)

2. The client changes the application state through interaction with the server

3. The client knows which requests can do next only by looking at the hypermedia controls in the representations it has received so far

4. Hypermedia controls are the driving force behind changes in application state

Common API designs

Not evolvable

“Most of today’s APIs have a big problem: once deployed they can’t change.”

– Leonard Richardson and Mike Amundsen in “RESTful Web APIs”

Contracts on human readable documentation

"When those clients contain hardcoded information that could have gone into distributed hypermedia instead, you lose extensibility."

– Leonard Richardson and Mike Amundsen in "RESTful Web APIs"

Lack of focus on the clients

“Unfortunately, the server API can only enable loose coupling; it cannot prevent tight coupling. No matter how much we employ hypermedia, standard media types, and self-descriptive messages, **we cannot prevent clients from hardcoding URIs or assuming they know the content type and semantics of a response**.”

– Glenn Block, Pedro Félix *et al* in Designing Evolvable Web APIs with ASP .NET

Wrapper libraries are tackling the wrong problems

```
var api = new IssueTrackingApi(apiToken);

var issue = api.GetIssue(issueId);

issue.Description = “Issue description” ;

issue.Save();
```

Example APIs

Should we go back to using RPC or can we do better?

(but first think if you really need to)

"The hypermedia constraint allows a smart client to automatically adapt to changes on the server side. It allows a server to change underlying implementation without breaking all of its clients."

– Leonard Richardson and Mike Amundsen in "RESTful Web APIs"

Collection+JSON

```
{ "collection" :
  {
    "version" : "1.0",
    "href" : "http://example.org/friends/",

    "links" : [
      {"rel" : "feed", "href" : "http://example.org/friends/rss"}
    ],

    "items" : [
      {
        "href" : "http://example.org/friends/jdoe",
        "data" : [
          {"name" : "full-name", "value" : "J. Doe", "prompt" : "Full Name"},
          {"name" : "email", "value" : "jdoe@example.org", "prompt" : "Email"}
        ],
        "links" : [
          {"rel" : "blog", "href" : "http://examples.org/blogs/jdoe", "prompt" : "Blog"},
          {"rel" : "avatar", "href" : "http://examples.org/images/jdoe", "prompt" : "Avatar", "render" : "image"}
        ]
      },
      {
        (…)
      }
    ],

    "queries" : [
      {"rel" : "search", "href" : "http://example.org/friends/search", "prompt" : "Search",
        "data" : [
          {"name" : "search", "value" : ""}
        ]
      }
    ],

    "template" : {
      "data" : [
        {"name" : "full-name", "value" : "", "prompt" : "Full Name"},
        {"name" : "email", "value" : "", "prompt" : "Email"},
        {"name" : "blog", "value" : "", "prompt" : "Blog"},
        {"name" : "avatar", "value" : "", "prompt" : "Avatar"}

      (…)
```

References

1. https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
2. RESTful Web Services, Leonard Richardson and Sam Ruby
3. http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven
4. RESTful Web APIs, Leonard Richardson and Mike Amundsen
5. [Not released yet] RESTful Web Clients, Mike Amundsen
6. Designing evolvable Web APIs with ASP .NET, Glenn Block, Pedro Félix, *et al*

Topics for future sessions

- More on Hypermedia Formats
  - HAL
  - Collection + JSON
  - Siren
- Semantic gap and Profiles
- How to design RESTful Web APIs
- How to design RESTful Web Clients
- HTTP/2 - RFC 7540 (Standardization of Google’s SPDY)