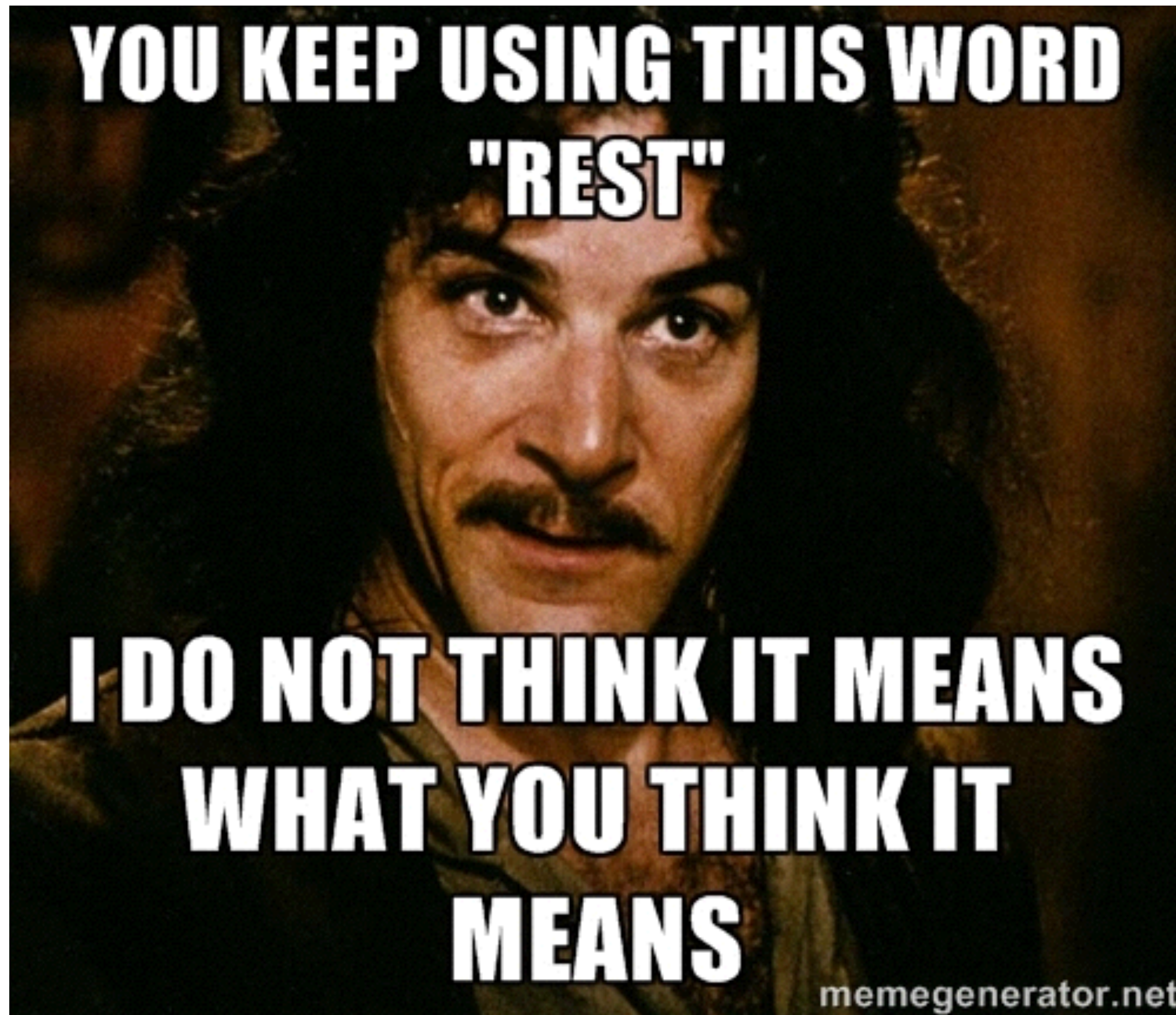# Representational State Transfer

Alexandre Rodrigues

12-08-2015

What I'll cover

- <u>What you may think REST is</u>

- What is an arquitectural style and why I need one

- Imagine a unRESTful Web

- REST constraints and it's benefits

- Contrast REST with other know styles like RPC

- Who is using REST principles in production

- Who should be using REST principles

YOU KEEP USING THIS WORD "REST"

I DO NOT THINK IT MEANS WHAT YOU THINK IT MEANS

memegenerator.net

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 cool and better way do implement Web Services

- SOAP substitute

REST is an Arquitectural 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

Let's talk about the World Wide Web and it's success

Imagine if

- You had to use one different Web browser for each Website you wanted to view

- Or you had to type enter the address each time you wanted to change the page (i.e. there were no links in the pages)

- Websites didn't tolerate more than a few users accessing a page at a time

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

- Cache

- <u>Uniform Interface</u>

- Layered Systems

- Code-On-Demand (Optional)

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

- If the resource changes the server uses hypermedia (in HTTP can be the response's Location header) to direct clients to the new URI

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

"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"

Can REST be successfully applied to Web APIs (HTTP based services)?

- It is getting easier to implement an API that follows the REST architectural style but it is still hard, specially:

  - Ensure the self-descriptive messages constraint (profiles are a possible solution for this, but we won't talk about them today)

  - Getting the hypermedia constraint right

  - Implement a client that takes advantage of the REST evolvability property, but that's not a Web browser reinvention.

- There is still much work to be done defining how should a client be implemented when there is no human making all the decisions, regarding the state transitions.

Why should I care?

"The reason to make a real REST API is to get evolvability … a "v1" is a middle finger to your API customers, indicating RPC/HTTP (not REST)"

– Roy Fielding

"When was the last time you saw a version number on a Website?"
"a REST API is just a website for users with a limited vocabulary (machine to machine interaction)"

– Roy Fielding

Can I use just a few constraints?

- Yes, but please don't call it REST

- Call it HTTP based service, HTTP service or Web API

"What needs to be done to make the REST architectural style clear on the notion that hypertext is a constraint? In other words, if the engine of application state (and hence the API) is not being driven by hypertext, then it cannot be RESTful and cannot be a REST API. Period. Is there some broken manual somewhere that needs to be fixed?"

– Roy Fielding

Respect Roy Fielding's work

- REST was defined by him

- If you had also defined something, would you like that people used that term to refer to other thing?

- He designed the HTTP 1.1 that solved many of the problems of the Web at that time

- He worked on a set of new RFCs (7230-7235) to replace the 2616 (HTTP bis working group)

Now that we know what REST is, let's see who may benefit from it

Real world example

- A news company that has many contents and information channels

- They maintain lots of clients (iOS, Android, Web, Windows Phone)

- There may be a set of representation that can be used

- Now imagine if all the news companies used the same type of representations (media types)

HATEOAS and the PayPal REST Payment API

One of the key features of the PayPal REST API is HATEOAS (Hypertext As The Engine Of Application State).

At its core, HATEAOAS provides a way to interact with the REST Payment API entirely through hyperlinks. With each call that you make to the API, we'll return an array of links that allow you to request more information about a call and to further interact with the API. You no longer need to hard code the logic necessary to use the PayPal REST API.

HATEOAS links are contextual, so you only get the information that is relative to a specific request.

Building Blocks of HATEOAS

Let's take a look at a PayPal payment using the REST API. When you send the initial payment request, you'll get back a response that contains a set of HATEOAS links like this:

```
[
  {
    "href": "https://api.sandbox.paypal.com/v1/payments/payment/PAY-6RV70583SB702805EKEYSZ6Y",
    "rel": "self",
    "method": "GET"
  },
  {
    "href": "https://www.sandbox.paypal.com/webscr?cmd=_express-checkout&token=EC-60U79048BN7719609",
    "rel": "approval_url",
```

There are three components for each link in a HATEOAS links array:

- `href` : URL of the related HATEOAS link you can use for subsequent calls.
- `rel` : Link relation that describes how this link relates to the previous call.
- `method` : The HTTP method required for the related call.

href component

The key component within HATEOAS, `href` provides the linkage between the completed call and a future call. Each `href` is a fully formed target URL, so you there is no further logic required to access a resource other than the `method` .

rel component

31

REST is better than SOAP right?

Totally wrong!

They are different tools with different purposes, clearly SOAP wasn't designed to be on the World Wide Web

SOAP merits

- Widely adopted and mature (SOAP 1.2 solves many of the problems of 1.1)

- Enables Contract-First design approach

- Clients are coupled with the interface (contract) and not with the specific implementation

- Interoperability

- Agnostic to the transport protocol (only SOAP 1.2)

- Good tool support (specially in Java or .NET)

SOAP de-merits

- Not all languages have the same tool support as Java and .NET, what decreases ease of use (e.g. JavaScript)

- Too verbose, bad for mobile devices and slow networks (e.g. transatlantic traffic)

- Caching is not an easy task

- It's not evolvable

And now the slide almost no REST advocate shows.

Maybe we were better without the REST buzz…

- We have HTTP friendly APIs, each with it's on flavour, therefore we have one client for each API, much like with SOAP

- We don't have contracts, we have human readable documentation that most of the times is insufficient

- We have poor designs like APIs that in case of error give us responses with 200 Status code but an error payload

- We have efforts on languages to document APIs (RAML, Blueprint) that try to close the gap to SOAP, but are not helping in the evolvability (no hypermedia support)

Thank you!
We are almost there, just more
a few things before finishing

Topics for future sessions

- Hypermedia Formats

  - HAL

  - Collection + JSON

- Profiles

- How to design RESTful Web APIs

- CoAP: REST for embedded systems

- HTTP/2 - RFC 7540 (Standardization of Google's SPDY)

References and further reading material

- RESTful Web APIs, Leonard Richardson and Mike Amundsen

- Arquitectural Styles and the Design of Network-based Software Architecture, Roy Fielding

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

- http://roy.gbiv.com/untangled/

- http://www.slideshare.net/royfielding/

- [Upcoming] Learning Client Hypermedia, Mike Amundsen