# Pratical RESTful API design

## Exploration over technologies enabling a Model-Driven approach in RESTful API development

Giovanni Ciatto
giovanni.ciatto@studio.unibo.it

University of Bologna
Computer Science and Engineering

May 18, 2016

# Table of Contents

# Outline

# Overview I

## REST = **RE**presentational **S**tate **T**ransfert

- REST *is* basically a concept, a set of principles (best practices) referred as "constraints" and described through natural language
- REST *is not* a tight specification, and there exists more than a way to produce RESTful web services
  - Yeah, this is about web-services, so cool!
- It's not a dichotomy, nor simply a matter of RESTful and RESTless web-services:
  - there are *more RESTful* ones ...
  - ... and *more RESTless* ones
- RESTful (distributed) systems tend to be scalable, robust and easy to develop, understand and maintain
  - Yet not every system should be RESTful
  - Silver-bullets are like unicorns: they do not exist.

# Overview II

## REST constraints

RESTful systems *should* satisfy the following 6 constraints:

- Client-server
- Stateless interaction
- Chaceability
- Uniform Interface
- Layered System
- Code on Demand (Optional)

# Outline

# Client-server I

There are two kind of entities: *servers* and *clients* communicating via a *connector*.

## Servers

Reactive entities providing one or more services to multiple clients

## Clients

Triggering (proactive) entities making requests that trigger reactions from servers.

## Connectors

Mechanism that allows communication between clients and servers (e.g. HTTP protocol, Message Oriented Middleware, RPC, etc.)

# Client-server II

Separation of concerns is the key concept here.

## Separation of concerns

Once both clients and servers concerns have been fixed and some sort of common interface have been defined, the two kind of components evolve independently.

## Client-server constraint in web-based systems

Web-based systems quite often satisfy this constraint.

# Stateless interaction I

This is probably the most important constraint as well as the hardest one: each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Application state is therefore kept entirely on the client.

## Application state

Data that could vary by client, and per request.

## Exceptions are tolerated

Immutability is a utopia. For real world problems, you should just try to minimize mutability. E.g. request rate monitoring requires mutability. However, make every effort to ensure that application state doesn't span multiple requests of your services.

# Stateless interaction II

## Cookies

Cookies do not necessarily violate this constraint, since the are part of each HTTP interaction.

## Sessions identifiers

Assigning any sort of temporary session identifier to the some user *and* storing session data using some data structure within server's memory is inherently a violation of the constraint. Since this is the common usage of cookies, they are discouraged.

## Example

A stateless communication litmus test is to turn off session cookies, and determine if the API, web service, or web application still works as designed.

# Stateless interaction III

## Authentication

This constraint makes authentication critical. Other approaches have been developed which are more secure and RESTful.

# Cacheability

## Cacheable responses

- Clients can cache responses
- Responses must therefore, implicitly or explicitly, define themselves as cacheable, or not
  - to prevent clients reusing stale or inappropriate data in response to further requests
- Well-managed caching partially or completely eliminates some client-server interactions
  - further improving scalability and performance

# Uniform Interface I

- The uniform interface constraint defines the interface between clients and servers.
- This is probably secret of REST's simplicity and strength.
- RESTful systems expose a standard, unambiguous, clear and human-readable API because of this constraint.
  - Such a principle allows for model-driven approaches.

## Uniform Interface Requirements

- Resource-based
- Manipulation of Resources Through Representations
- Self-descriptive Messages
- HATEOAS (dafuq ?!)

# Uniform Interface II

## Resource-Based

- RESTful systems handle *resources*: servers host resources and clients want to CRUD them
  - ⇒ **C**reate or **R**ead or **U**pdate or **D**elete
- Resources are not accessed (CRUDed) directly but through their representation(s)
- Resources have a *hierarchical* nature
- Resources are identified and referenced by the mean of URIs
  - ⇒ **U**niform **R**esource **I**dentifier

# Uniform Interface III

## HTTP Verbs

- CRUD in Web-based systems means exploiting HTTP methods, which are often called "verbs" because of their usage within APIs:
  - POST is used for resource Creation
  - GET mean is to Read resources
  - PUT aim is to Update resources
  - DELETE is used to Delete resources
- HTTP Status codes and their general purpose semantics are part of the uniform interface too:
  - e.g. any successful request will result in a `200: Ok` or `204: No Content` status code (depending on weather the response has body or not)
  - e.g. trying to GET or DELETE any non-existent resource will result in a `404: Not Found` status code
  - e.g. POSTing an already-existing user will result in a `409: Conflict` status code

# Uniform Interface IV

- Example of resource creation:
  - we want to edit some user's username from `gciatto92` to `gciatto`
  - suppose no authentication is needed

## Example of non-RESTful approach

- `GET http://example.com/users?user=gciatto92&-operation=changeName&newName=gciatto`

× RPC style: the request contains the to-be-called operation

× No semantics for `GET` verb

× Which resource am I editing?

# Uniform Interface V

## Example of RESTful approach

- `PUT http://example.com/users/gciatto92?newName=gciatto`

✓ I'm editing the resource `gciatto92`, which is a user, composing the `users` resource, which represents the collection of registered users

✓ `PUT` verb means Update and that's what I am doing

✓ URI queries are simply a mean for payload transport

# Further Reading I


Fielding, Roy Thomas
*Architectural Styles and the Design of Network-based Software Architectures*.
Doctoral dissertation, University of California, Irvine, 200.