RESTful services

Antonio Brogi

Department of Computer Science
University of Pisa

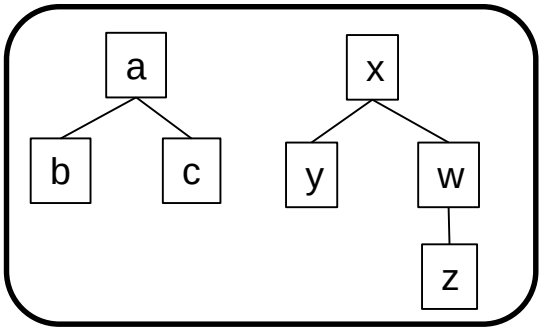
REpresentational State Transfer (REST)

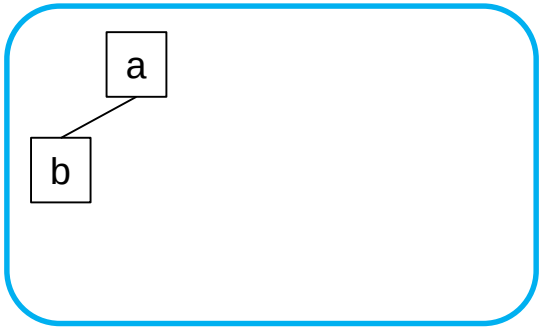
Originally introduced as an *architectural style*, developed as an abstract model of the Web architecture to guide the redesign and definition of HTTP and URIs

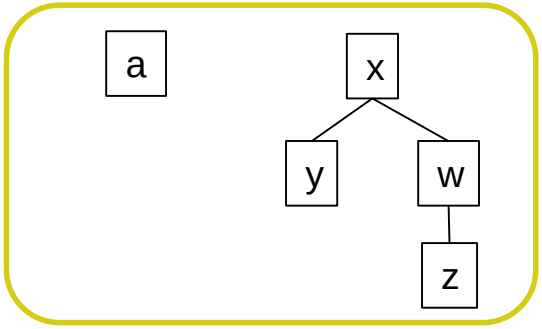
Q: Why the name "*REpresentational State* Transfer"?

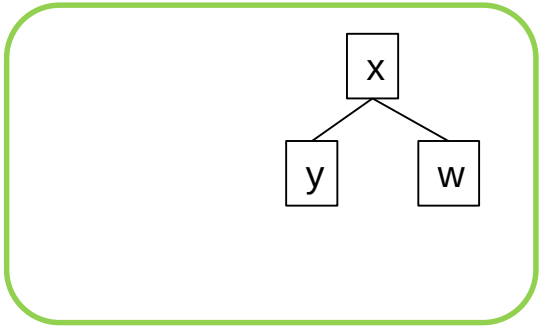
A: "[...] to evoke an image of how a well-designed Web application behaves: a network of Web pages forms a virtual state machine, allowing a user to progress through the application by selecting a link or submitting a short data-entry form, with each action resulting in a transition to the next state of the application by transferring a representation of that state to the user"

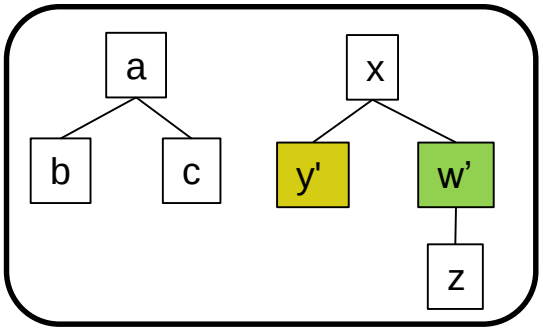
R.T. Fielding. Architectural styles and the design of network-based software architectures. *PhD Thesis, University of California, Irvine, 2000.*

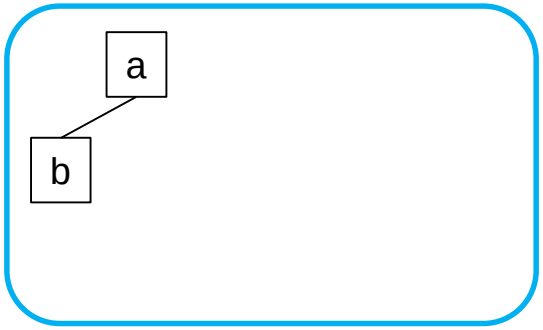

a
x
b
c
y
w
z

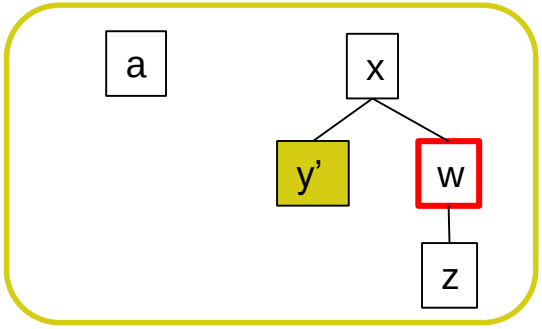

a
b

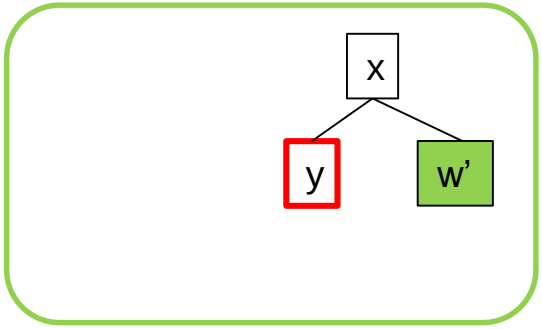

a
x
y
w
z


x
y
w


a
x
b
c
y'
w’
z


a
b


a
x
y’
w
z


x
y
w’

RESTful services

- RESTful services:
  - **Services are viewed as *resources* that can be uniquely address by their URIs**
  - **Clients invoke HTTP methods to create/read/update/delete resources**
- REST "resource-centric" (SOAP "message-centric")
- Requests and responses to transfer representations of resources

REST motivations

- Simplicity

  rely on a few principles and a small set of well-defined operations

- Scalability

  stateless protocol and distributed state

- Layeredness

  allow (any number of) intermediaries (proxies, gateways, firewalls)

REST principles

REST principles

1. Resource identification through URIs
   - Service exposes a set of resources which identify the targets of the interaction with its clients
   - Resources identified by URIs, which define global addressing space for resource & service discovery
2. Uniform interface
   - Resources manipulated using a fixed set of operations:
     - PUT and POST to create and update state of resource
     - DELETE to delete a resource
     - GET to retrieve current state of a resource

```
PUT is idempotent
POST is not idempotent
                [RFC 2616]
```

3. Self-descriptive messages
   - Requests contain enough context information to process the message
   - Resources decoupled from their representation so that their content can be accessed in a variety of formats (e.g., HTML, XML, JSON, plain text, PDF, JPEG, etc.)
   - Metadata about the resource can be used to control caching or to negotiate representation format
4. Stateful interactions through hyperlinks
   - Every interaction with a resource is stateless
   - Serever contains no client state, any session state is held on the client
   - Stateful interactions rely on the concept of explicit state transfer

Example

Customer wants to update his last food order

GET /customers/fred

```
            200 OK
            <customer>
               <name>Fred Flinstone</name>
               <address> 45 Cave Stone Road, Bedrock</address>
               <orders>http://barbera.com/customers/fred/orders</orders>
            </customer>
```

GET /customers/fred/orders

```
            200 OK
            <orders>
               <customer>http://barbera.com/customers/fred</customer>
               <order id="1">
                  <orderURL>http://barbera.com/orders/1122</orderURL>
                  <status>open</status>
               </order>
               ...
            </orders>
```

GET /orders/1122

```
            200 OK
            <order>
               <customer>http://barbera.com/customers/fred</customer>
               <item quantity="1">brontoburger</item>
            </order>
```

PUT /orders/1122

```
<order>
   <customer>http://barbera.com/customers/fred</customer>
   <item quantity="50">brontoburger</item>
</order>
```

```
            200 OK
```

Example

Using a simple Doodle service to organize next Friday night

POST /polls
<title>Friday night</title>
<options>
<option>bowling</option>
<option>pool</option>
<option>poker</option>
</options>
...

201 Created
Content-Location: /polls/112233

GET /polls/112233

200 OK
<poll>
<title>Friday night</title>
...
<votes href="/vote">
</poll>

POST /polls/112233/vote
<name>Barnie>/name>
<choice>pool</choice>

GET /polls/112233

201 Created
Content-Location: /polls/112233/vote/1

200 OK
<title>Friday night</title>
...
<votes>
<vote id="1">
<name>Barnie</name>
<choice>pool>/choice>
</vote>
</votes>

DELETE /polls/112233

200 OK

GET /polls/112233

404 Not Found

Some other (real) examples

flickr http://api.flickr.com/services/search?...&text=pisa

twitter http://search.twitter.com/search.json?q=merkel

YouTube

facebook

Google maps

...

Strengths of REST

- Simplicity
  - low learning curve
    - REST leverages well-known standards (HTTP, XML, URI)
    - necessary infrastructure has already become pervasive
  - services can be built with minimal tooling
    - deploying a RESTful Web service very similar to building dynamic Web site
    - effort required to build a client to a RESTful service small
      - developers can begin testing service from ordinary Web browser
      - no need to develop custom client-side software
    - thanks to URIs and hyperlinks, it is possible to discover Web resources without compulsory registration to a repository
- Efficiency
  - lightweight protocol
  - lightweight message formats
- Scalability
  - stateless RESTful Web services can serve a very large number of clients

Weaknesses of REST

- Confusion on what are the best practices for building RESTfulWeb services, e.g.,
  - **Hi-REST** recommends the use of 4 verbs (GET, POST, PUT, DELETE)
  - Since proxies and firewalls may not always allow HTTP connections that use any other verb than GET and POST, **Lo-REST** uses only two verbs (GET for idempotent requests, and POST for everything else)
    - workarounds: "real" verb sent using a special HTTP header (`X-HTTP-Method-Override`) or – like with Ruby on Rails– a hidden form field
    - these workarounds may not be understood by all Web servers → additional development and testing effort required
- Encoding complex data structures into a URI can be challenging as there is no commonly accepted marshalling mechanism
- Idempotent requests having large amounts of input data cannot be encoded in a URI ("`414 - Request-URI too long`")

SOAP vs. REST

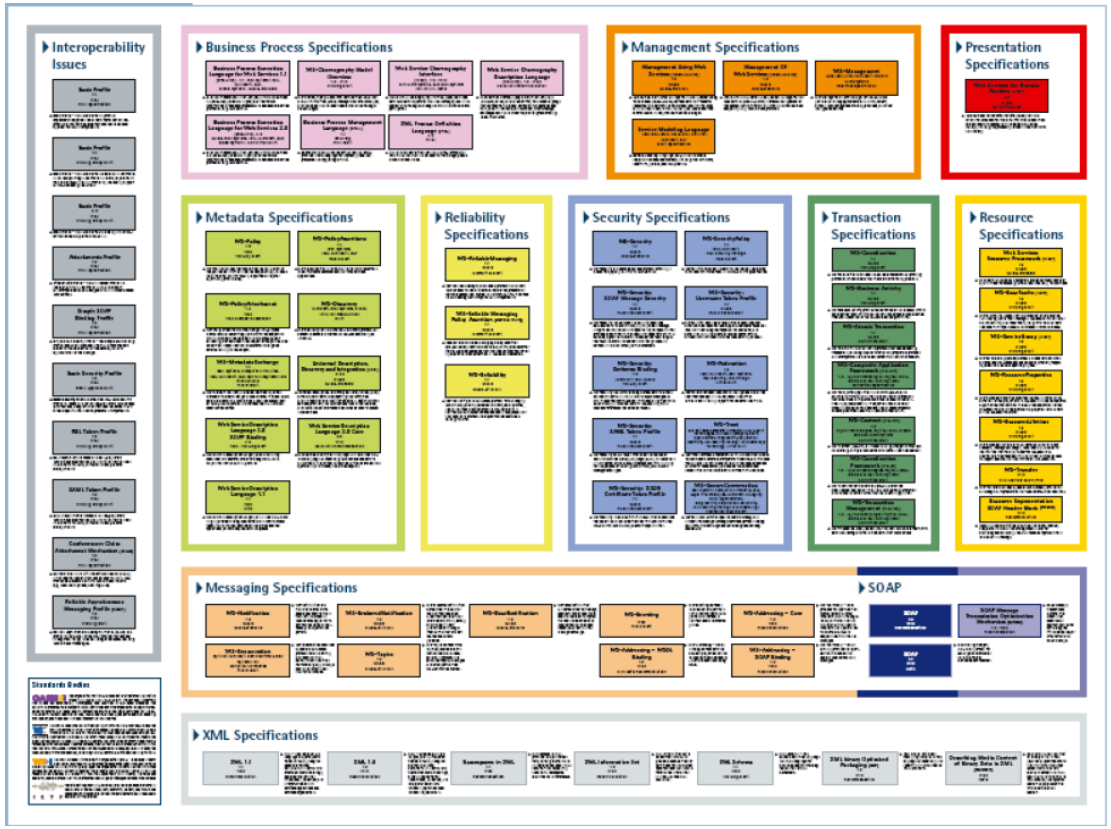
Example: Querying a phonebook application for the details of a given user (id)

```
<?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 pb="http://www.acme.com/phonebook">
<pb:GetUserDetails>
<pb:UserID>12345</pb:UserID>
</pb:GetUserDetails>
</soap:Body>
</soap:Envelope>
```

vs.

```
http://www.acme.com/phonebook/UserDetails/12345
```

WS-* vs. REST services



vs.

HTTP

XML | JSON

WS-* vs. REST services

- REST does not have the complexity of the layers of the WS-* stack
- Not easy to extend a RESTful Web service to support advanced functionalities in an interoperable manner
- Various decisions that are very easy to make for RESTful services may lead to significant development efforts and technical risks
  - e.g., the design of the exact specification of the resources and their URI addressing scheme
- If the enterprise-level features of WS-* (transactions, reliability, message-level security) are not required
  - REST can provide better flexibility and control, but requires a lot of low-level coding
  - WS-* provides better tool support and programming interface convenience

- REST convenient
  - to get (simple) services running
  - for tactical, ad hoc integration over the Web (e.g., mashups)
- WS-* preferable for professional enterprise application integration scenarios with longer lifespan and advanced QoS requirements

C. Pautasso, O. Zimmermann, F. Leymann. Restful web services vs. "big"' web services: making the right architectural decision.

https://nordicapis.com

https://nordicapis.com

https://www.youtube.com/watch?v=vhpjfAnqXlM

Mashup

[noun - Music, Slang.]: a recording that combines vocal and instrumental tracks from two or more recordings

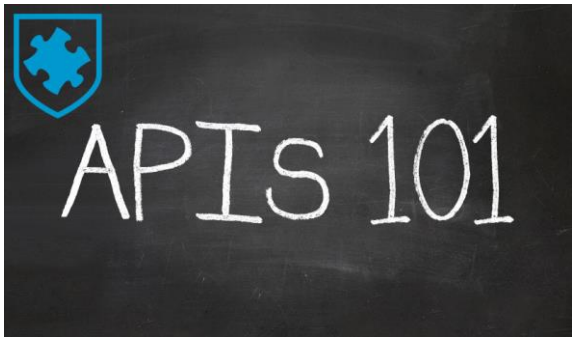
Mashup

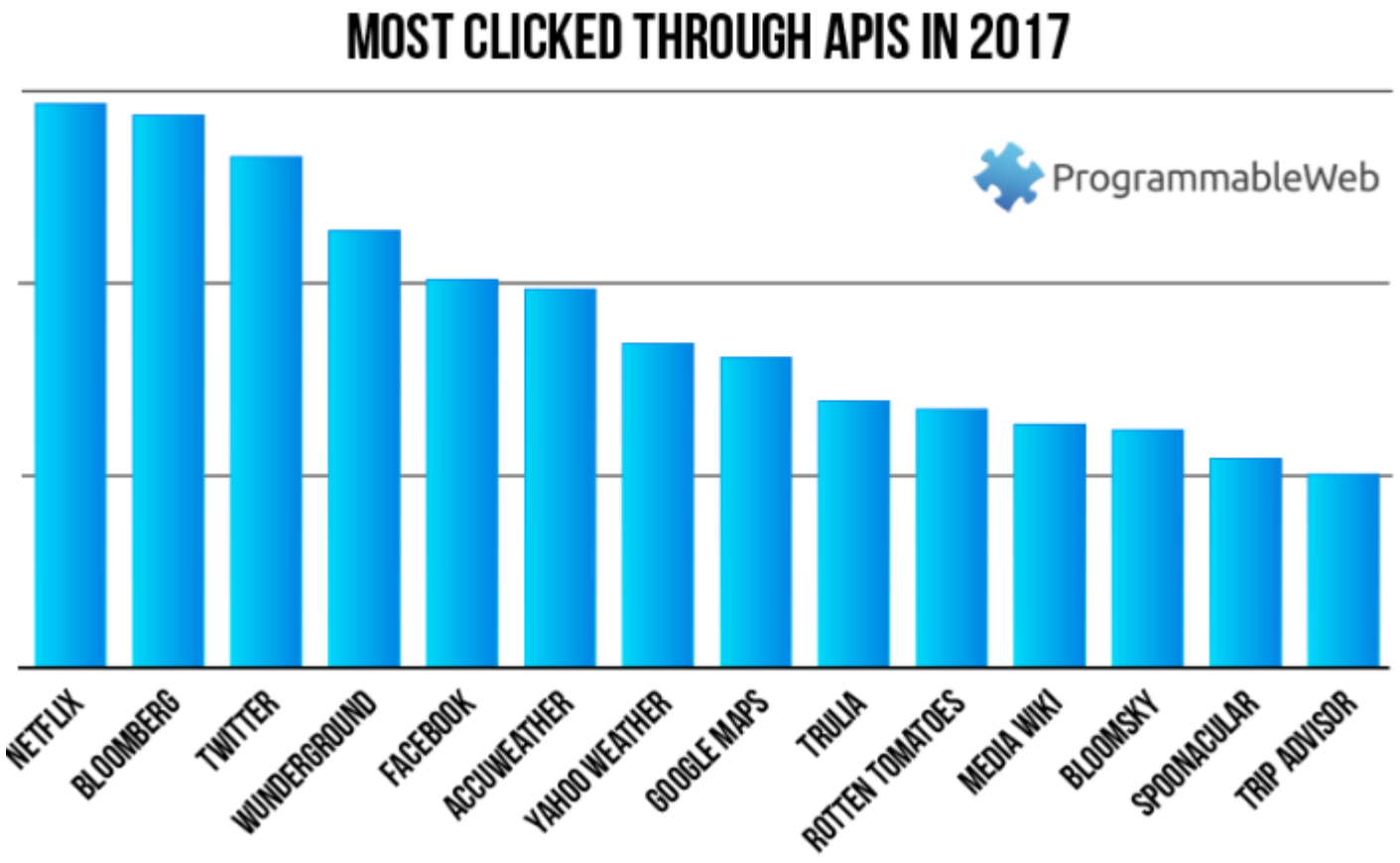
[noun - Music, Slang.]: a recording that combines vocal and instrumental tracks from two or more recordings

[noun – Computer Science]: **a light-weight Web application innovatively combining (data provided by) services available on the net**

A couple of random examples

- Given the title of a movie *m*, display map with cinemas featuring *m* and restaurants nearby
- BBC news map – places on a map (links to) BBC news reports





https://www.programmableweb.com

Kloudless

Kloudless helps aggregate multiple APIs and integrate these services into an application. The

Read More

Healthcare Messenger – KPN

The Healthcare Messenger API is provided by KPN, allowing 1-1 chat communications for

Read More

Live Objects

The Orange API connects with a platform for managing data from IoT sources. Using the API,

Read More

Avalara AvaTax API

Avalara offers a suite of REST APIs to interface with AvaTax, an enterprise tax service. According

Read More

NASA API

The NASA APIs open up NASA data and imagery for developers to create all sorts of spacey

Read More

api.video

The api.video services provides a way to host and deliver videos using their content delivery

Read More

RingCentral

The RingCentral API enables multi-faceted business communication abilities, including

Read More

Home Connect API

Home Connect offers the possibility to control your Home Connect enabled kitchen appliances

Read More

APItude by Hotelbeds

Described as "the world's bed bank," Hotelbeds is networking the world's travel trade. Through

Monzo API Docs

Monzo is an innovative, user-friendly mobile-first bank. Their web API provides an intuitive

https://nordicapis.com/best-public-api-of-2018/