REST & OpenAPI

Antonio Brogi

Department of Computer Science
University of Pisa

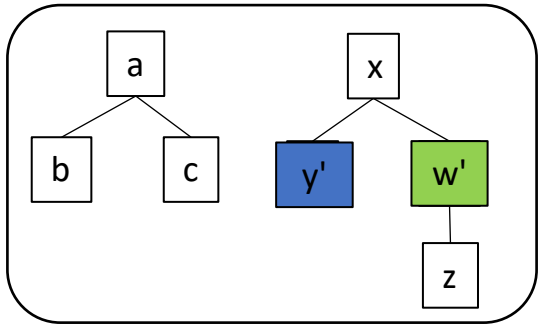
REpresentational State Transfer (REST)

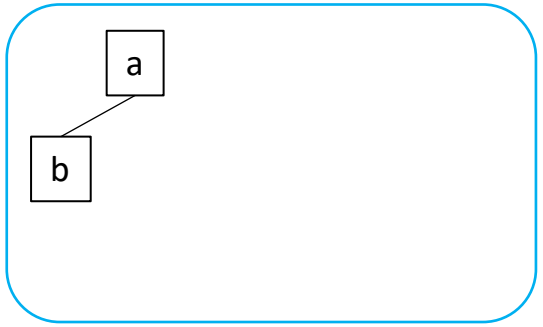
Originally introduced as an *architectural style*, developed as an abstract model of the Web architecture to guide the redesign and definition of HTTP and URIs

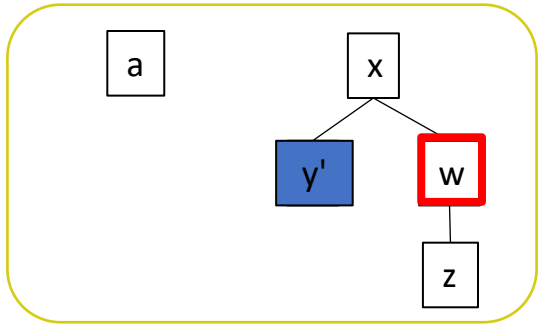
"each action resulting in a transition to the next state of the application by transferring a representation of that state to the user"

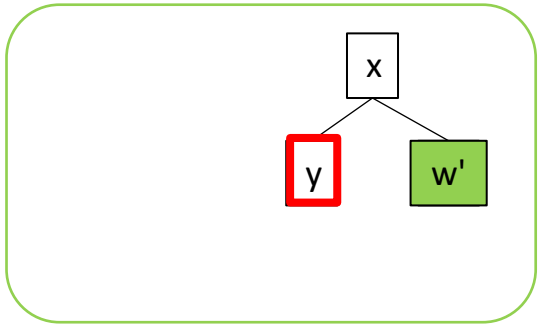
R. T. Fielding. Architectural styles and the design of network-based software architectures. PhD Thesis, University of California, Irvine. 2000.

State transfer









REST principles

1. Resource identification through URIs
   - Service exposes set of resources identified by URIs

2. Uniform interface
   - Clients invoke HTTP methods to create/read/update/delete resources:
     - POST and PUT to create and update state of resource
     - DELETE to delete a resource
     - GET to retrieve current state of a resource

3. Self-descriptive messages
   - Requests contain enough context information to process message
   - Resources decoupled from their representation so that content can be accessed in a variety of formats (e.g., HTML, XML, JSON, plain text, PDF, JPEG, etc.)

4. Stateful interactions through hyperlinks
   - Every interaction with a resource is stateless
   - Server 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

barbera.com

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
```

→

←

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

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

```
DELETE /polls/112233
```

→

←

```
200 OK
```

```
GET /polls/112233
```

←

→

```
404 Not Found
```

SOAP vs. REST

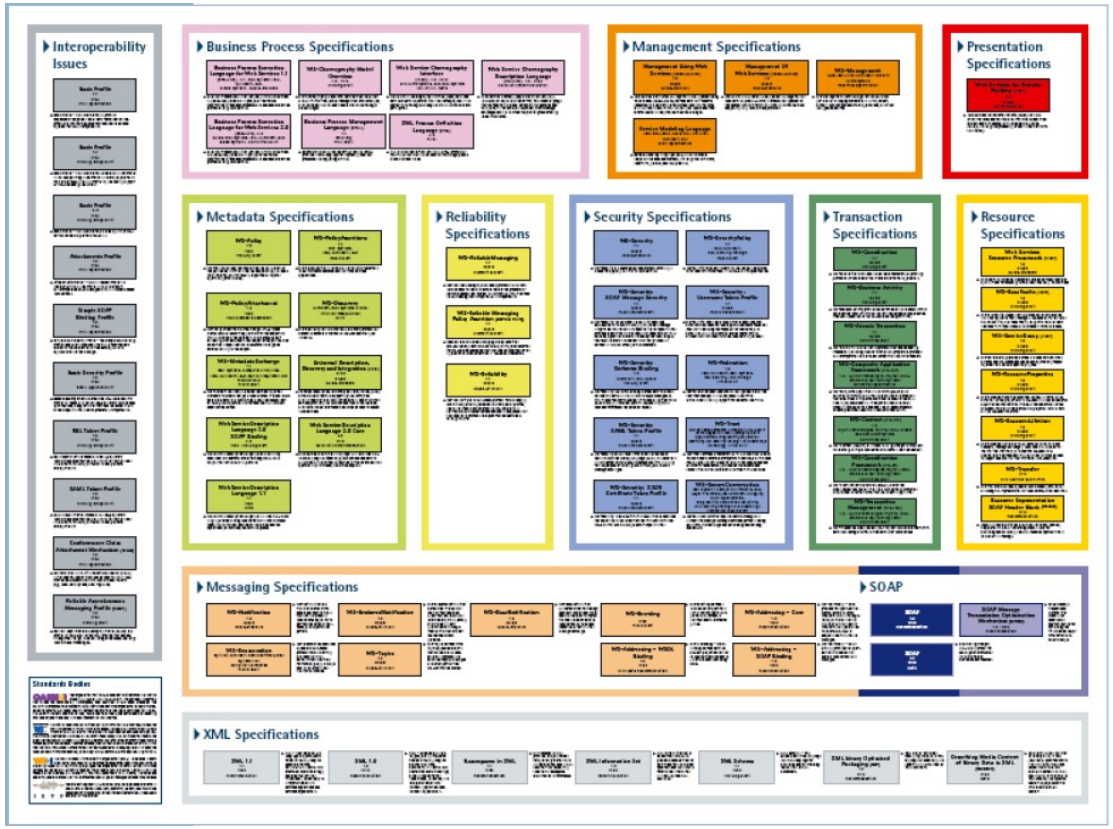
Example: Querying a phonebook application for the details of a given user (id)

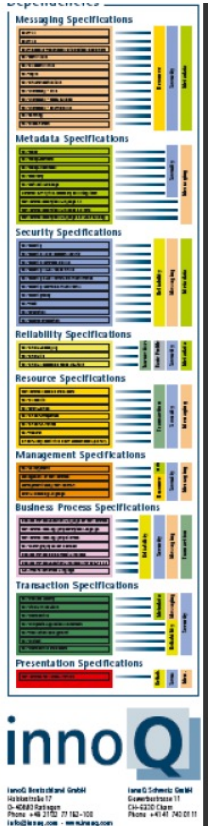
```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:body pb="http://www.acme.com/phonebook">
<pb:GetUserDetails>
<pb:UserID>12345</pb:UserID>
</pb:GetUserDetails>
</soap:Body>
</soap:Envelope>
```

vs.

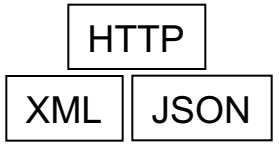
```
http://www.acme.com/phonebook/UserDetails/12345
```

WS-* vs. REST services





vs.



From Swagger to OpenAPI

https://www.youtube.com/watch?v=oxqZ9J6t420

OpenAPI

OpenAPI Initiative (Linux Foundation Collaborative Project) aims at creating a standardized, vendor neutral description format of REST APIs

- f.k.a. Swagger
- simple (JSON-based) description language to specify HTTP API endpoints, how they are used, and the structure of data that comes in and out

OpenAPI members

Industry adoption (2016)

OpenAPI

/* Simple example: One endpoint /api/users_id supporting GET to retrieve list of user Ids */

```
swagger: "2.0"
info:
  title: BipBip Data Service
  description: returns info about BipBip data
  license:
    name: APLv2
    url: https://www.apache.org/licenses/LICENSE-2.0.html
  version: 0.1.0
  basePath: /api
  paths:
    /user_ids:
      get:
        operationId: getUserIds
        description: Returns a list of ids
        produces:
        - application/json
        responses:
          '200':
            description: List of Ids
            schema:
            type: array
            items:
              type: integer
```

OpenAPI

E.g., Connexion framework for Flask automagically handles HTTP requests based on OpenAPI Specification of your API