Università della Svizzera italiana

Faculty of Informatics

REST

The Architectural Style of the Web

Prof. Cesare Pautasso
http://www.pautasso.info
cesare.pautasso@usi.ch
@pautasso

REpresentational State Transfer

REST defines the architectural style of the Web

Four design principles explain the success and the scalability of the HTTP protocol

1. **Resource Identification** through URI
2. **Uniform Interface** for all resources
3. **Multiple representations** of the same resource
4. **Hyperlinks** indicate resource relationships and valid state transitions for dynamic protocol description and discovery

URI tell a story

How do we find something on the Internet?

- Where is the host serving the information?
- How do we communicate with the host?
- What is the protocol?
- Which port?
- Do we need a user/password to login?
- In which folder is it located?
- Which "file" should we download?
- What is the format?

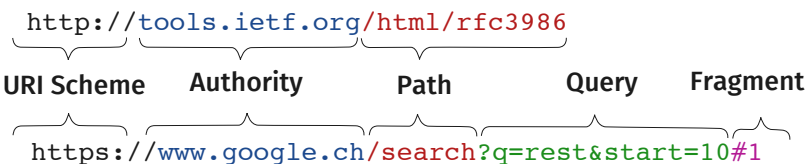
URI example

```
ftp://user:pass@ftp.usi.ch/root/public/videos/introduction.avi
```

Start your remote file transfer client, connect to `ftp.usi.ch` using the `ftp` protocol on port 21, login using your local account and password, go to my public folder `root/public` and then change to the videos sub-folder and download a copy of the `introduction.avi` file.

Uniform Resource Identifier

Internet Standard for resource naming and identification (originally from 1994, revised until 2005)



- URIs cannot have arbitrary length (4Kb)
- #Fragments are not seen by the server

URI Design Tips

- Keep URIs short
- Prefer Nouns to Verbs
- Once published, do not change URIs
- Avoid leaking implementation details (.php, .aspx) into URIs

Parametric URIs

```
http://map.com/search/Lugano/Parking
```

Prefer positional encoding

```
http://map.com/search?where=Lugano&what=Parking
```

Key-value encoding (useful for optional parameters)

Uniform Interface

HTTP Method		Safe	Idempotent
POST	Create a sub resource (Perform an action)	?	?
GET	Retrieve the current state of the resource	YES	YES
PUT	Create or update the state of a resource	NO	YES
DELETE	Clear a resource (invalidate its URI)	NO	YES

- **Safe** = no side effects: the resource state on the server remains unchanged if the same request is repeated
- **Idempotent** = regardless of how many times a given method is invoked, the end result is the same
- **Retry on Failure**: If safe/idempotent requests fail, simply repeat them (simplified exception handling).

POST vs. GET

- GET is a **read-only** operation.
  It can be repeated without affecting the state of the resource (idempotent) and can be cached.
- POST is a **read-write** operation and may change the state of the resource and provoke side effects on the server.

GET will return the current state of the resource. The result may change every time

POST vs. PUT

What is the right way of creating resources and to initialize their state?

Resources are created by many concurrent clients

```
PUT /resource/{id}
```

```
201 Created
```

- Problem: How to ensure resource `{id}` is unique?
- Solution 1: let the client choose a unique id (GUID)

```
POST /resource
```

```
301 Moved Permanently
Location: /resource/{id}
```

- Solution 2: let the server compute the unique id
- Problem: Duplicate resource instances may be created if requests are repeated due to unreliable communication

Representations

Resources may have multiple representations

- Website in English, Italian, German, French
- A picture in PNG, JPG, GIF format
- Some content in HTML, JSON or XML format

Resource representations are controlled with meta-data (**HTTP Headers**)

- Negotiation of understood content types (`Accept, Content-Type`)
- Caching of representations that did not change (`If-Modified-Since`)
- Compression to save bandwidth (`Accept-Encoding, Content-Encoding`)

Hyperlinks

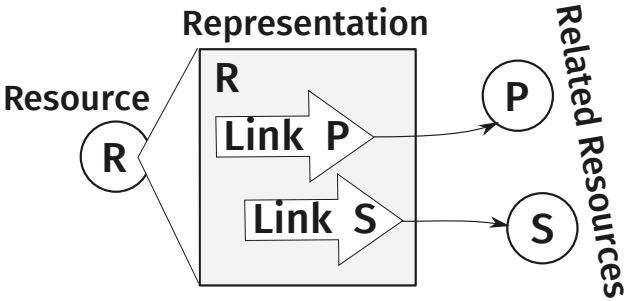
- Hyperlinks are the edges of the Web graph, linking a pair of nodes (URI)
- Hyperlinks connect and help to discover related resources
- In REST, hyperlinks are used to correctly traverse the state of a Web application. They point to the next states from the current one.

Where to store hyperlinks?

1. In the source resource representation
2. In the destination resource representation
3. Independently of the two resources

Hypermedia

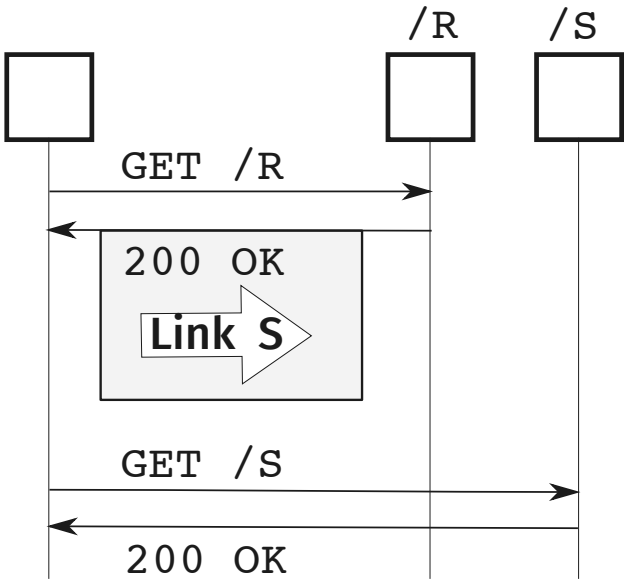
Problem: How to discover the URIs of a potentially infinite and dynamically changing set of resources?



Solution: Resource Representations contain links to other resources

Discovery by Referral

- Clients can use a service to dynamically lookup and discover other services
- Any resource can refer clients to any other resource (decentralized)
- Links can be embedded and found in any hypermedia representation format



REST API Design Process

1. Identify resources to be exposed as services (e.g., photoalbum images, book catalog, purchase order, open bugs, blog entries, polls and votes)
2. Model relationships (e.g., containment, reference, state transitions) between resources with hyperlinks that can be followed to get more details (or perform state transitions)
3. Define URIs to address the resources
4. Understand what it means to do a GET, POST, PUT, DELETE for each resource (and whether it is allowed or not)
5. Design, document and standardize resource representations (media types)

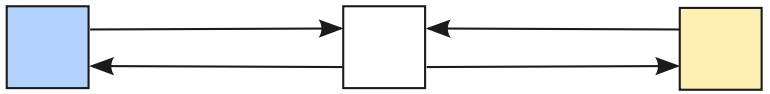
	GET	PUT	POST	DELETE
/loan	✓	✓	✓	✓
/balance	✓	✗	✗	✗
/user	✓	✓	✓	✗
/book	✓	✓	✓	✓
/order	✓	?	✓	✗

Simple Doodle API Example

1. Resources: Polls and Votes
2. Relationships: Containment
3. URIs embed IDs of "child" instance resources
4. POST on the container creates new child resources
5. PUT/DELETE for updating and removing child resources

	GET	PUT	POST	DELETE
/poll	✓	✗	✓	✗
/poll/{id}	✓	✓	✗	✓
/poll/{id}/vote	✓	✗	✓	✗
/poll/{id}/vote/{id}	✓	✓	✗	✓

/poll
/poll/42
/poll/42/vote



Creating a poll

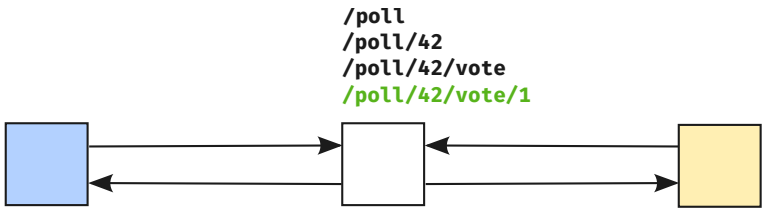
```
POST /poll
<options>A,B,C</options>
```

```
201 Created
Location: /poll/42
```

Reading a poll

```
GET /poll/42
```

```
200 OK
<options>A,B,C</options>
<votes href="/vote"/>
```



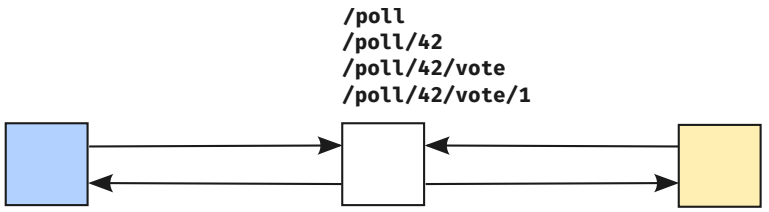
Cast a vote

```
POST /poll/42/vote
<name>C. Pautasso</name>
<choice>B</choice>
```

```
201 Created
Location: /poll/42/vote/1
```

```
GET /poll/42
```

```
200 OK
<options>A,B,C</options>
<votes href="/vote">
 <vote id="1">
  <name>C. Pautasso</name>
  <choice>B</choice>
 </vote>
</votes>
```

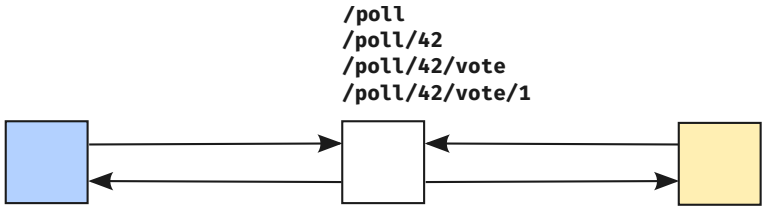
Update a vote

```
PUT /poll/42/vote/1
<name>C. Pautasso</name>
<choice>C</choice>
```

```
200 OK
```

```
GET /poll/42
```

```
200 OK
<options>A,B,C</options>
<votes href="/vote">
 <vote id="1">
  <name>C. Pautasso</name>
  <choice>C</choice>
 </vote>
</votes>
```



Remove a poll

```
DELETE /poll/42
```

```
200 OK
```

Poll is deleted

```
GET /poll/42
```

```
404 Not Found
```


References

- Tim Berners-Lee, Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web, Collins, Nov.2000
- IETF, HTTP/1.1 Standard, RFC2616, June 1999 http://www.ietf.org/rfc/rfc2616.txt (http://www.ietf.org/rfc/rfc2616.txt)
- Roy Fielding, Architectural Styles and the Design of Network-based Software Architectures (http://roy.gbiv.com/pubs/dissertation/fielding_dissertation.pdf) , University of California, Irvine, 2000
- Jacob Nielsen, URI are UI (http://www.nngroup.com/articles/url-as-ui/) , 1999
- Leonard Richardson, Sam Ruby, RESTful Web Services, O'Reilly, May 2007, ISBN 0-596-52926-0
- Jim Webber, Savas Parastatidis, Ian Robinson, REST in Practice: Hypermedia and Systems Architecture, O'Reilly, 2010
- Thomas Erl, Benjamin Carlyle, Cesare Pautasso, Raj Balasubramanian, SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST, Prentice Hall, 2012