REST - REpresentational State Transfer

Prof. Emanuele Panizzi

REpresentational State Transfer

- Architectural style for distributed hypermedia systems.
- Proposed by Roy Fielding, 2000[1]
- Transfer representation of resources from one component (e.g., the server) to another (e.g., the client).

[1] https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Resource

- A resource is any information that can be named: a document, an image, a service, a non-virtual object (e.g., a person), a collection of other resources.
- A resource is a set of elements or values that can vary over time.
- Two resources may map to the same values at a given time. E.g., “version v2.1” of a program and “latest version” of the same program.

Representation

- Resource representation: the current or intended state of a resource, i.e., the value of the resource at any particular time.
- REST components (e.g., clients or servers) perform actions on a resource by using a representation
- Representation consists of data and metadata. The data format is known as “media type”.

Resource Identifiers

- Used to identify, i.e. address, a resource.
- A Uniform Resource Identifier (URI) is a unique sequence of characters that identifies a logical or physical resource
- E.g.: *`http://example.com/users`*

URI best practices

1. use nouns to represent resources
2. use singular nouns for a single resource, e.g.:
   `http://example.com/users/admin`
3. use plural nouns for a collection of resources, e.g.:
   `http://example.com/users/`

URI notation

1. Forward slash (/): used to express hierarchy. Suggest: use trailing slash only if the resource is not a leaf.
2. Prefer hyphens (-) to underscores (_)
3. Use lowercase letters
4. Do not use file extensions (media type is communicated in headers)
5. Use query component to filter, e.g.:
   http://example.com/managed-devices/?region=USA

Operations: not in the URI

URIs are used to uniquely identify the resources and not any action upon them. Different actions can be executed on a resource through supported methods (interface between components).

Examples of methods:

```
GET http://example.com/managed-devices/{id}

PUT http://example.com/managed-devices/{id}

DELETE http://example.com/managed-devices/{id}
```

RESTful system constraints

1. client-server
2. stateless
3. cacheable
4. uniform-interface
5. layered system

Client-Server

- Enforces separation of concerns (client: UI –- server: data storage)
- Improves portability
- Improves scalability

Stateless

- Each client request must contain all the information necessary to understand it
- and cannot take advantage of any stored context on the server
- Session state is kept entirely on the client (resource state is kept on the server)

Cacheable

- The client can later reuse a cacheable resource representation (data)
- Time period is specified in the response.

Uniform interface

- uniform interface between components: standardization vs. efficiency
- implementations are decoupled from the services they provide
- four interface constraints:
  - identification of resources;
  - manipulation of resources through representations;
  - self-descriptive messages;
  - hypermedia as the engine of application state (the client needs only the initial URI)

Layered System

- There may be different components involved in a communication, in a layered architecture e.g.:
  - origin server
  - gateway
  - proxy
  - user agent
- Intermediary components act as both a client and a server. They forward requests and responses, with possible translation.
- Each component cannot “see” beyond the immediate layer with which they interact.

References

- https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
- https://restfulapi.net
- https://datatracker.ietf.org/doc/html/rfc3986
- https://en.wikipedia.org/wiki/Uniform_Resource_Identifier