Development of Web Applications

Principles and Practice

Vincent Simonet, 2013-2014
Université Pierre et Marie Curie, Master Informatique, Spécialité STL

2 Communication

Vincent Simonet, 2013-2014

Université Pierre et Marie Curie, Master Informatique, Spécialité STL

Today’s agenda

- HTTP (HyperText Transfer Protocol),
- RPC (Remote Procedure Call),
- REST (Representational State Transfer).

HTTP

Hypertext Transfer Protocol

History

HTTP is the main protocol of the World Wide Web. It was invented between **March 1989** and **December 1990** by Berners-Lee, together with HTML.

1991: HTTP 0.9

1996: HTTP 1.0 (RFC 1945)

1997: HTTP 1.1 (RFC 2068)

1999: HTTP 1.1 improved (RFC 2616)

Main characteristics

- Application layer protocol,
- Asymmetric (the *client* submits a *request* to the *server* which answers with a *response*),
- Stateless,
- Handle caches and proxies.

Request

Structure:

- Command: `<method>␣<URI>␣HTTP/<version>`
- Headers: `<name>:␣<value>`
- Empty line
- Body (when applicable)

Example:

```
GET /foo?id=1&name=bar HTTP/1.1
User-Agent: Mozilla/5.0
Accept: text/html

EOF
```

URL Universal Resource Locator

```
scheme://domain[:port]/path[?query_string]
                              [#fragment_id]
```

- **Scheme:** underliying protocol (*e.g.* http),
- **Host:** name or IP address of the server,
- **Port:** port number of the server,
- **Path:** path of the resources on the server,
- **Query-string:** dynamic parameters associated with the request,
- **Fragment identifier:** a part or a position within the overall resource or document.

Request Methods

Most used:

- **GET:** Retrieves an URI.
- **HEAD:** Same as GET, but without response body.
- **POST:** Requests that the server accept the enclosed entity enclosed as a new subordinate of the resource identified by the URI.
- **OPTIONS:** Returns the HTTP methods that the server supports for the specified URI.

Also used by REST:

- **PUT:** Requests that the enclosed entity be stored under the supplied URI.
- **DELETE:** Deletes the specified resource.
- **PATCH:** Applies partial modifications to a resource.

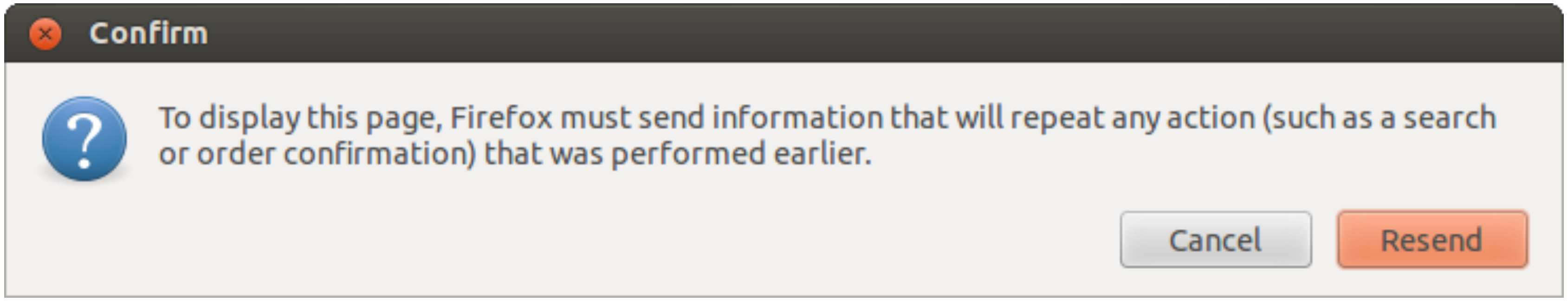
Others: **CONNECT** and **TRACE**.

Safe Request Methods

Safe request methods should not change the state of the server. They should be idempotent.

Safe methods	**Unsafe methods**
GET	POST
HEAD	PUT
OPTIONS	DELETE
TRACE	TRACE
	CONNECT
	PATCH

Unsafe Methods



To display this page, Firefox must send information that will repeat any action (such as a search or order confirmation) that was performed earlier.

Request Headers (1/2)

Expected format:

```
Accept: text/plain
Accept-Language: en-US
Accept-Encoding: gzip, deflate
```

Cache:

```
Cache-Control: no-cache
If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

Proxy:

```
Host: en.wikipedia.org:80
Max-Forwards: 10
Proxy-Authorization: Basic QWxhZGRpbjpvcGVuIH==
Via: 1.0 fred, 1.1 example.com (Apache/1.1)
```

Request Headers (2/2)

Content:

```
Content-Length: 348
Content-Type: text/html; charset=utf-8
Content-MD5: Q2hlY2sgSW50ZWdyaXR5IQ==
```

Meta:

```
Date: Tue, 15 Nov 1994 08:12:31 GMT
User-Agent: Mozilla/5.0
Referer: http://en.wikipedia.org/wiki/Main_Page
```

Session:

```
Cookie: Version=1; Skin=new;
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
Connection: keep-alive
DNT: 1 (Do Not Track Enabled)
```
[non standard]

Response

Structure:

- Command:
  `HTTP/<version>␣<status>␣<reason>`
- Headers: `<name>:␣<value>`
- Empty line
- Body (when applicable)

Example:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

<html>
  ...
```

Response Status Codes (1/2)

1xx: Informational *(Request received, continuing process.)*

2xx: Success *(The action requested by the client was received, understood, accepted and processed successfully.)*

- **200:** OK

3xx: Redirection *(The client must take additional action to complete the request.)*

- **301:** Moved Permanently *(use GET)*
- **303:** See Other *(use GET)*
- **307:** Temporary Redirect *(use same method)*
- **308:** Permanent Redirect *(use same method)*

Response Status Codes (2/2)

4xx: Client Error *(The client seems to have erred.)*

- **400:** Bad Request
- **403:** Forbidden
- **404:** Not Found

5xx: Server Error (The server failed to fulfill an apparently valid request.)

- **500:** Internal server error.
- **501:** Not implemented.
- **503:** Service Unavailable.

(The above list includes only the most common codes for each category.)

Response Headers (1/2)

Content:

```
Content-Type: text/html; charset=utf-8
Content-Encoding: gzip
Content-Language: da
Content-Length: 348
Content-Location: /index.html
```

Cache:

```
Cache-Control: no-cache
ETag: "737060cd8c284d8af7ad3082f209582d"
Expires: Thu, 01 Dec 1994 16:00:00 GMT
Last-Modified: Tue, 15 Nov 1994 12:45:26 +0000
```

Response Headers (2/2)

Proxy:

```
Age: 12
Via: 1.0 fred, 1.1 example.com (Apache/1.1)
```

Meta:

```
Allow: GET, HEAD
Date: Tue, 15 Nov 1994 08:12:31 GMT
Location: http://www.w3.org/pub/WWW/People.html
Retry-After: 120
Server: Apache/2.4.1 (Unix)
```

Session:

```
Set-Cookie: UserID=X; Max-Age=3600; Version=1
Connection: keep-alive
DNT: 1 (Do Not Track Enabled)
```
[non standard]

RPC

Remote Procedure Call

Definition

A remote procedure call (RPC) is an **inter-process communication** that allows a computer program to cause a subroutine or procedure to execute in **another address space** (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction.

Principle

General principle: a server provides a service which

- can be called from a client,
- may take parameters,
- may return a value.

This usually comes with:

- A service (or service provider) directory,
- A typed interface for every service, specific or not to a programming language,
- An integration to one or several programming languages (remote call ~ function call)

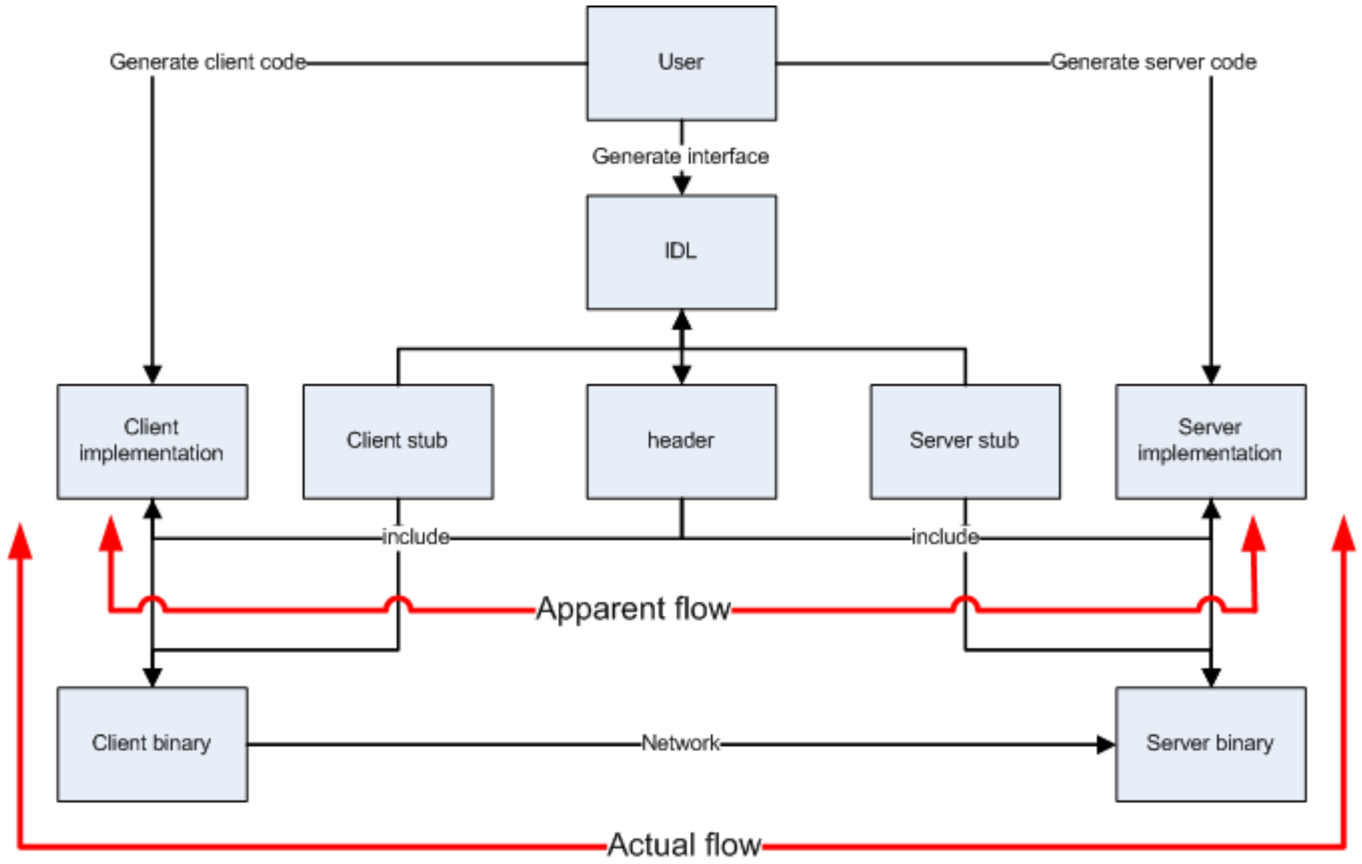
Purposes:

- Distributed programming,
- Inter-process communication,
- Web services.

RPC: Sequence of Events

1. The client calls the client stub. The call is a local procedure call, with parameters pushed on to the stack in the normal way.
2. The client stub packs the parameters into a message and makes a system call to send the message. Packing the parameters is called marshalling.
3. The client's local operating system sends the message from the client machine to the server machine.
4. The local operating system on the server machine passes the incoming packets to the server stub.
5. The server stub unpacks the parameters from the message. Unpacking the parameters is called unmarshalling.
6. Finally, the server stub calls the server procedure. The reply traces the same steps in the reverse direction.

RPC: Code generation



Difficulties

In practice, a Remote Procedure Call is very similar to a normal function call, but:

- Information transfer is asynchronous,
- Error may happen in the information transfer,
- Data is copied,
- Communication can be costly,
- Security issues.

Main Choices

- Service description:
    - Description languages (IDL, XML, etc.),
    - Source code annotations (*e.g.* in Java),
    - No service description (reflexivity).
- Serialization:
    - Binary vs text,
    - How to manage pointers/functions?
    - Security (authentication, confidentiality, etc.)
    - Versioning.
- Transport:
    - Synchronous (*e.g.* HTTP) or asynchronous (*e.g.* SMTP)

History

1976: Description of the RPC principle in RFC 707

1981: Courier, by Xerox (first commercial application)

1986: RPC/XDR by Sun (RFC 1057) for NFS

1997: Java RMI

1998: XML-RPC (Dave Winer, Microsoft)

1998: SOAP

2001: WSDL & UDDI

2005: JSON-RPC

XML-RPC

- The client sends an HTTP request to the server. The request body is an XML document specifying a single call to a method (method name + parameters).
- The server replies with a response whose body contains an XML document.
- Structured values can be encoded in the XML documents.

XML-RPC: Request example

```
<?xml version="1.0"?>
<methodCall>
  <methodName>example.getResult</methodName>
  <params>
    <param>
      <value><int>40</int></value>
    </param>
    <param>
      <value>
        <array>
          <data>
            <value><i4>1404</i4></value>
            <value><string>...</string></value>
            <value><i4>1</i4></value>
          </data>
        </array>
      </value>
    </param>
  </params>
</methodCall>
```

XML-RPC: Response example

```xml
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>Result value</string></value>
    </param>
  </params>
</methodResponse>
```

JSON-RPC

- The same, but replacing XML by JSON.
- Became popular as JSON is easy to handle in JavaScript.

Example of JSON-RPC

Request:

```
{"jsonrpc": "2.0",
 "method": "subtract",
 "params": {"subtrahend": 23, "minuend": 42},
 "id": 3}
```

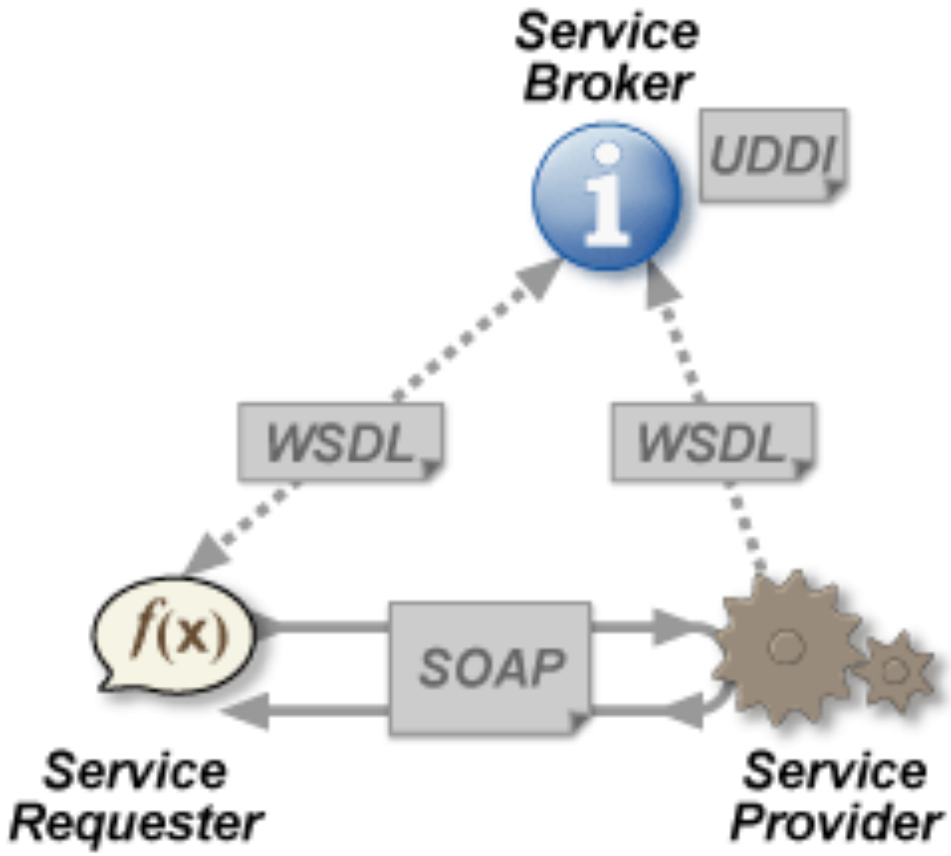
Response:

```
{"jsonrpc": "2.0",
 "result": 19,
 "id": 3}
```

Error response:

```
{"jsonrpc": "2.0",
 "error": {"code": -32601, "message": "..."},
 "id": 10}
```

SOAP / WSDL / UDDI



SOAP Simple Object Access Protocol

SOAP provides a basic messaging framework upon which web services can be built.

It has three major characteristics:

- Extensibility (e.g. adding security features),
- Neutrality (can be used over any transport protocol),
- Independance (allows for any programming model).

SOAP Message Format

A SOAP message is an XML document containing the following elements:

- **Envelope:** Identifies the XML document as a SOAP message (required).
- **Header:** Contains header information (optional).
- **Body:** Contains call and response information (required).
- **Fault:** Provides information about errors that occurred while processing the message (optional).

SOAP Transport Methods

HTTP

HTTPS

SMTP (not really used)

Example of SOAP request (over HTTP)

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="...">
  <soap:Body xmlns:m="...">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Example of SOAP response (over HTTP)

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="...">
  <soap:Body xmlns:m="...">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>
</soap:Envelope>
```

Comparison to JSON/XML

- SOAP is capable of representing general graph structures, not just tree structures, of objects.
- SOAP messages can be sent to multiple recipients.
- SOAP has the ability to encrypt parts of the message so that some recipients but not others can see those parts. (This ability is standardised in XML too but not JSON).
- SOAP has guaranteed message delivery - if a connection is severed, it will try to re-send the message.
- Naturally this all comes at a cost of increased complexity.

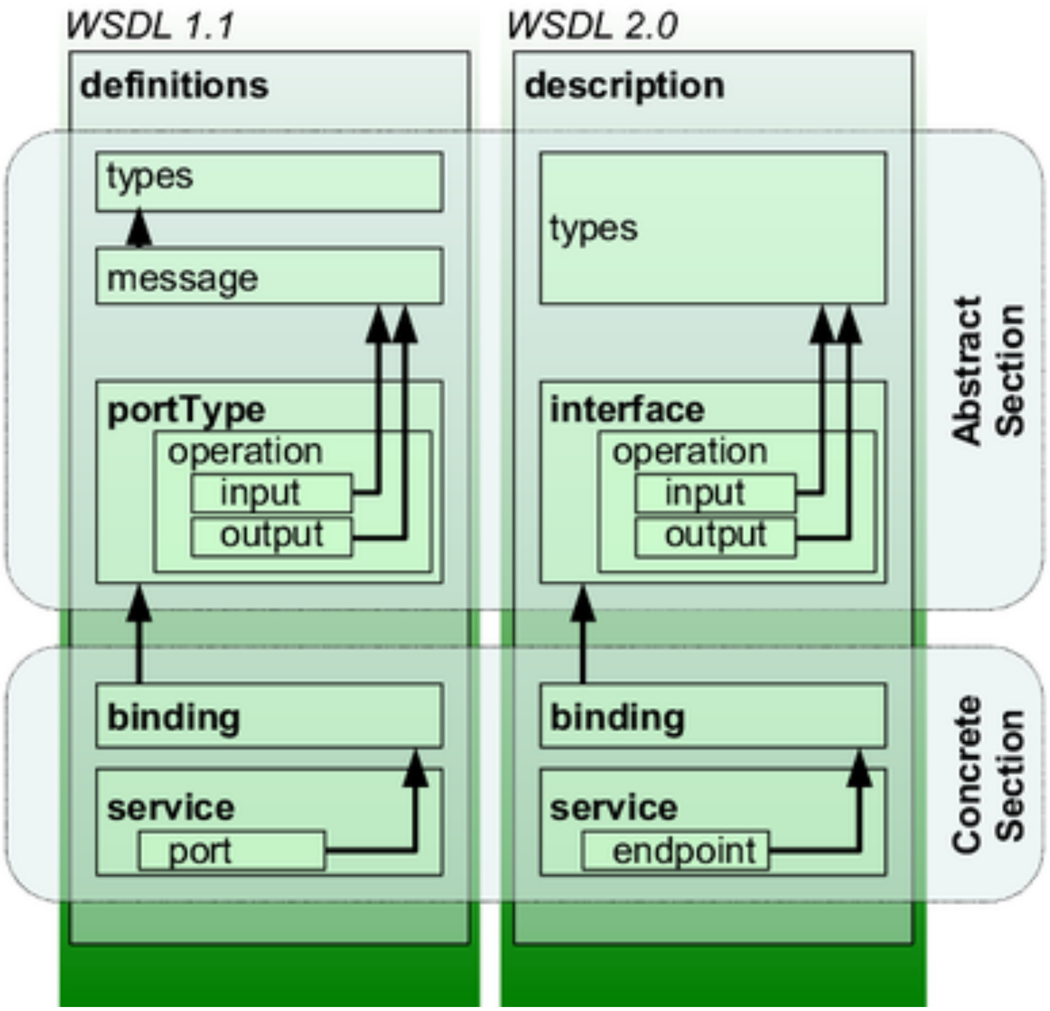
WSDL Web Services Description Language

WSDL is an XML-based interface description language that is used for describing the functionality offered by a web service.

A WSDL description of a web service (WSDL file) provides a machine-readable description of how the service can be called, what parameters it expects, and what data structures it returns.

WSDL is often used in combination with SOAP.

WSDL objects



WSDL objects

- **Service:** Contains a set of system functions that are exposed to the protocols.
- **Endpoint:** Defines the address or connection point to the service. Typically a simple URL.
- **Binding:** Specifies the interface and defines the SOAP binding style (RPC/Document) and transport (SOAP Protocol). It also defines the operations.
- **Interface:** Defines a service, the operations that can be performed, and the messages that are used to perform the operation.
- **Operation:** Defines the SOAP actions and the way the message is encoded, for example, "literal". Like a method or function in a traditional programming language.
- **Type:** Describes the data. The XML Schema language (XSD) is used.

WSDL example

```
<message name="GetStockPriceRequest">
  <part name="StockName" type="xs:string"/>
</message>

<message name="GetStockPriceResponse">
  <part name="Price" type="xs:double"/>
</message>

<portType name="GetStockPrices">
  <operation name="GetStockPrice">
    <input message="GetStockPriceRequest"/>
    <output message="GetStockPriceResponse"/>
  </operation>
</portType>
```

WSDL example: SOAP binding

```
<binding type="glossaryTerms" name="b1">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation>
   <soap:operation
     soapAction="http://example.com/GetStockPrice"/>
   <input><soap:body use="literal"/></input>
   <output><soap:body use="literal"/></output>
  </operation>
</binding>
```

UDDI Universal Description Discovery & Integration

UDDI is a platform-independent, XML-based registry by which businesses worldwide can list themselves on the Internet, and a mechanism to register and locate web service applications.

Most commonly found inside companies, where they are used to dynamically bind client systems to implementations

UDDI Structure

- **White pages:** businesses directory (name, contact, etc.).
- **Yellow pages:** classification of services and businesses based on standard taxonomies.
- **Green pages:** how to access the services.

REST

Representational State Transfer

What is REST?

A style of architecture for client/server protocol.

In REST, requests and responses are built around the transfer of representations of resources. A **resource** can be essentially any coherent and meaningful concept that may be addressed. A **representation** of a resource is typically a document that captures the current or intended state of a resource.

Introduced and defined in 2000 by Roy Fielding in his doctoral dissertation.

REST constraints (1/2)

- **Client/Server.** A uniform interface separates clients from servers, which have different concerns. E.g. clients are not concerned by storage while servers are not concerned with the user interface.
- **Stateless.** No client context being stored on the server between requests. Each request from any client contains all of the information necessary to service the request, and session state is held in the client.
- **Cacheable.** Clients can cache responses. Responses must therefore, implicitly or explicitly, define themselves as cacheable, or not.

REST constraints (2/2)

- **Layered system.** A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way.
- **Code on demand (optional).** Servers can temporarily extend or customize the functionality of a client by the transfer of executable code.
- **Uniform interface.** The uniform interface between clients and servers, discussed on next slide, simplifies and decouples the architecture, which enables each part to evolve independently.

A service fulfilling these constraints is said to be **RESTful**.

REST Uniform Interface

- **Identification of resources.** Individual resources are identified in requests, for example using URIs.
- **Manipulation of resources through representations**, which are conceptually separate from the resources.
- **Self-descriptive messages.** Each message includes enough information to describe how to process the message.
- **Hypermedia as the engine of application state.** Clients make state transitions only through actions that are dynamically identified within hypermedia by the server.

RESTful Web API

A RESTful Web API (or RESTful Web Service) is an API implemented using REST principles and HTTP. It is a collection of resources with four defined aspects:

- the base URI for the web API,
- the Internet media type of the data supported by the web API,
- the set of operations supported by the web API using HTTP methods,
- The API must be hypertext driven.

Example 1: Google Maps API

http://maps.googleapis.com/maps/api/geocode/json?address=4+Place+Jussieu+Paris&sensor=false

```
{
  "results" : [
    {
      "address_components" : [...],
      "formatted_address" : "4 Place Jussieu,
Université Pierre et Marie Curie, Université
Jussieu, 75005 Paris, France",
      "geometry" : {
        "location" : {
          "lat" : 48.8464111,
          "lng" : 2.3548468
        },
        ...
```

Example 2: Google Drive API

```
GET https://.../drive/v2/files/fileId
POST https://.../upload/drive/v2/files
PUT https://.../upload/drive/v2/files/fileId
DELETE https://.../drive/v2/files/fileId
POST https://.../drive/v2/files/fileId/copy

(... == www.googleapis.com)
```

RESTful API in practice

Four operations (CRUD), implemented by four HTTP methods:

- Create (POST),
- Read (GET),
- Update (PUT),
- Delete (DELETE).

Failure/Success: HTTP status code.

Resources are identified by URI, and represented in XML, JSON, YAML, ...

RSDL RESTful Service Description Language

This is similar to WSDL, for RESTful Web APIs.

Services *vs* Resources

Interaction Model:

- SOAP: exchanges
  - The server keeps data about the session,
  - Messages contain only what they express.
- REST: independent operations
  - Stateless server,
  - Messages must carry context.

Usages:

- SOAP: Transactional services,
- REST: Data or document exchanges.

Versioning

A practical issue which has to be handled in both approaches.

In a client/server application, it is usually not posssible to deploy evolutions on all servers and all clients at the same time.

Two main approaches:

- Ensuring backward compatibility,
- Including version numbers in requests.