
Service Oriented Architectures

Styles, Technologies and Applications

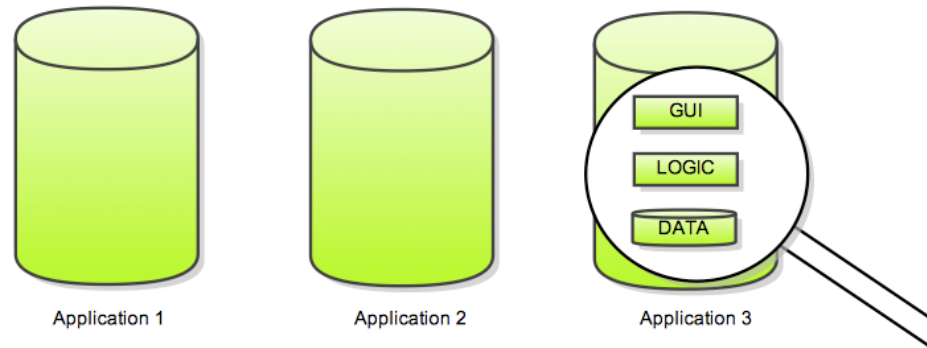
Cyril Rohr (IRISA - Equipe Myriads)
Feb 11, 2010

Agenda

1. What problem are we trying to address ?
2. Evolution towards Distributed Applications
3. Software Services - definition
4. **Web** Services ?
5. Architectural Styles - RPC vs Service-Oriented vs Resource-Oriented (REST)
6. Real-World examples

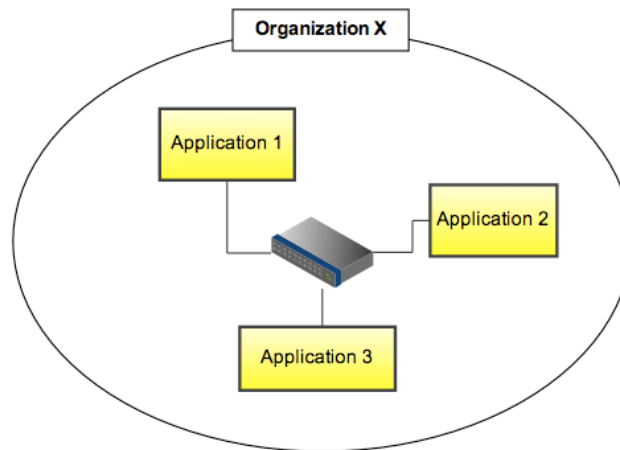
I. Problem

Complete, closed, monolithic applications



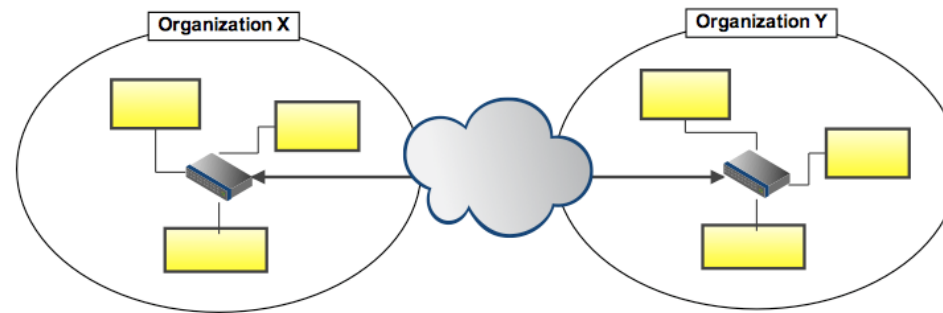
I. Objectives

Integration across application boundaries



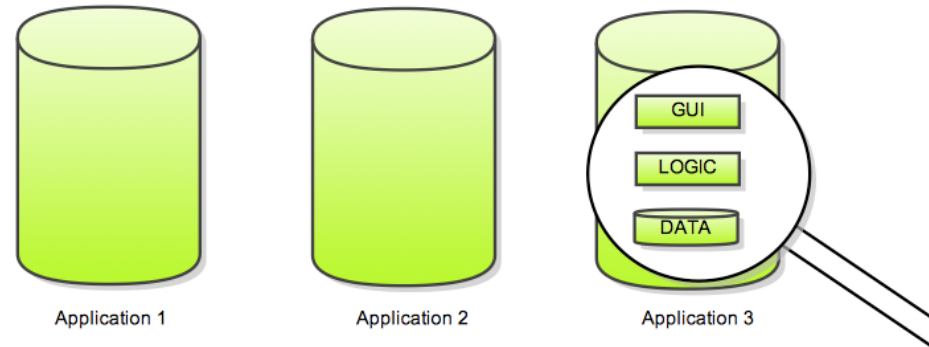
I. Objectives

Integration across organizational boundaries



2. Evolution towards Distributed Applications

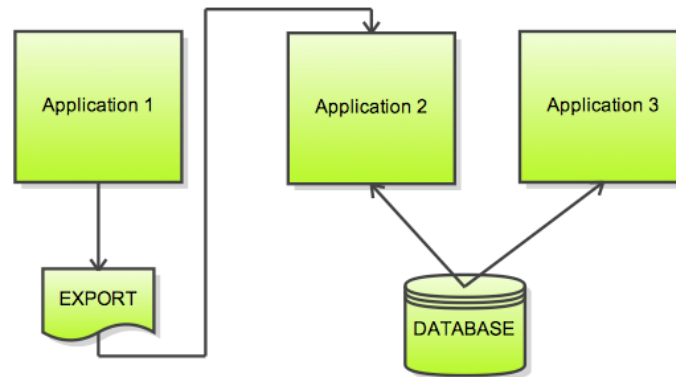
■ Monolithic Applications



- duplication, high maintenance cost, no data sharing

2. Evolution towards Distributed Applications

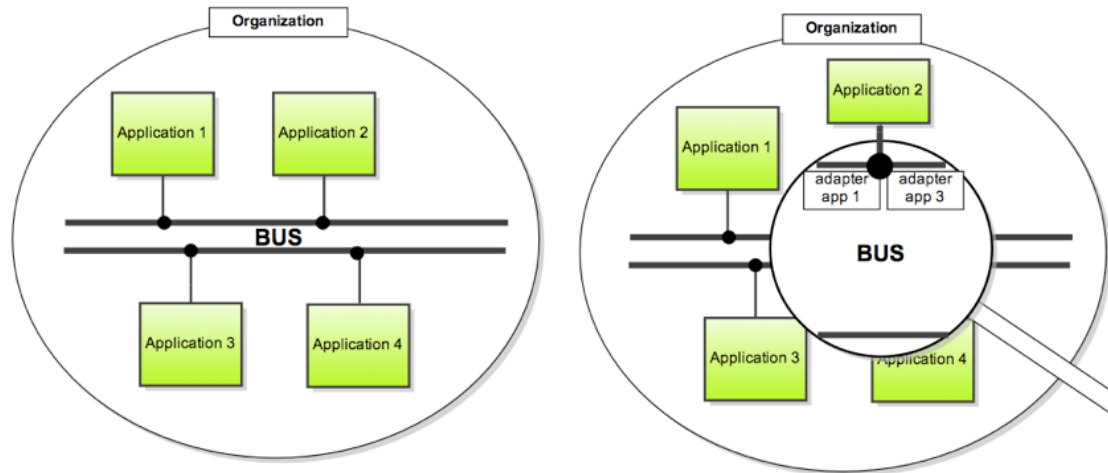
■ Applications sharing data



■ fragile, duplication

2. Evolution towards Distributed Applications

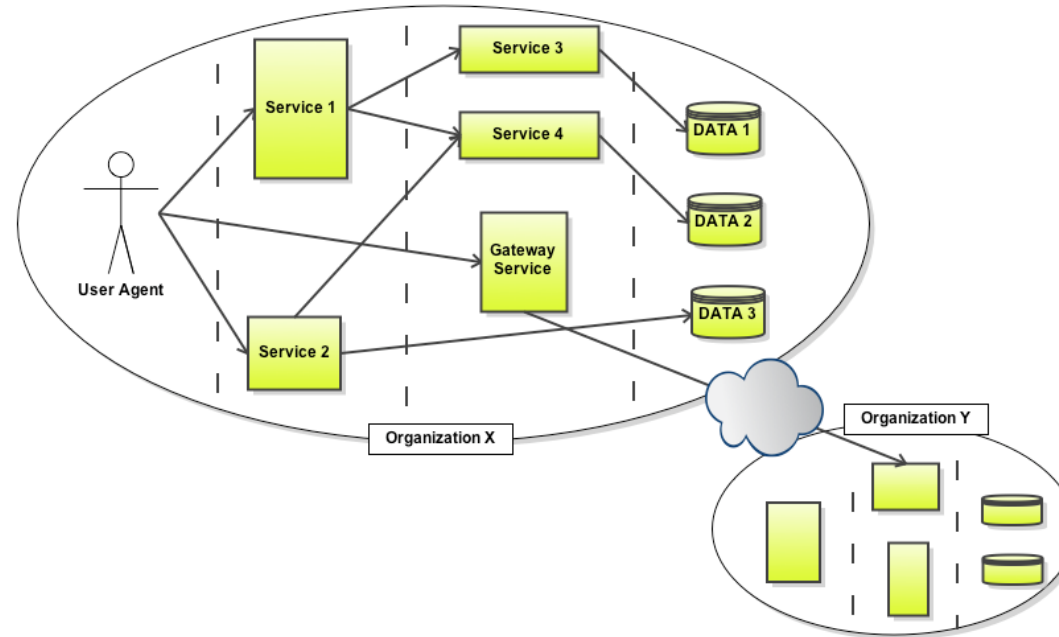
■ Applications sharing objects over the network



■ complex, vendor lock in, tight coupling

2. Evolution towards Distributed Applications

■ Service Oriented Architectures



- language independent, reusability, composability, loose-coupling

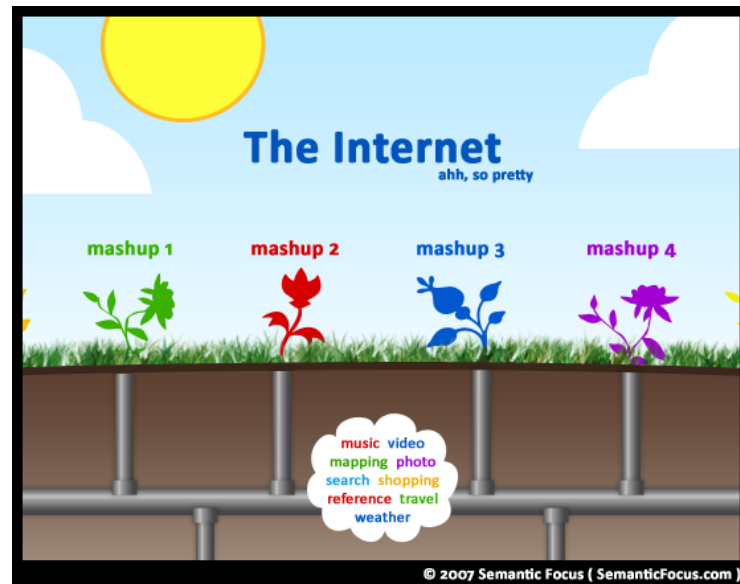
3. Software Services - definition

- Software services are units of functionality that exist at a service endpoint (**address**) that can be **remotely** accessed by clients.
- Clients can use the service by communicating with this endpoint without having direct access to the actual code files that implement the service.
- Loosely coupled
- Avoid fine-grained interactions patterns.

3. Software Services - benefits

- Operations across organizations
- Language independent
- Reusability
- Standardization, leading to better interoperability

4. Web Services ?



Term is often a catch-all for “services accessible on the web”

4. Web Services ?

W3C

a “web service” is “a **software system** designed to support **interoperable machine-to-machine interaction** over a **network**. It has an **interface** described in a **machine-processable format** (specifically Web Services Description Language WSDL). Other systems interact with the web service in a manner prescribed by its description using **SOAP messages**, typically **conveyed using HTTP** with an XML serialization in conjunction with other web-related standards.”

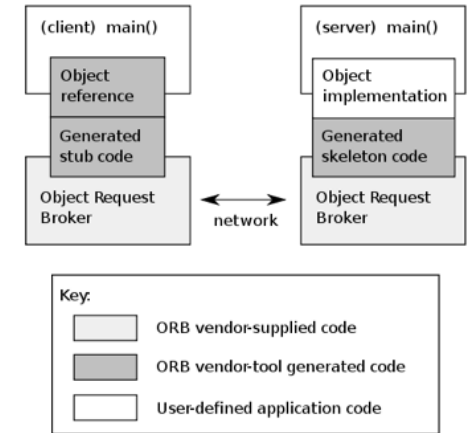
Web Services (WS) are just one of the technologies available to create distributed architectures. A software service may, but need not be exposed as a Web Service.

5. Architectural Styles for Distributed Applications

- RPC (Remote Procedure Call)
- Service/Message Oriented
- Resource Oriented

5A. RPC

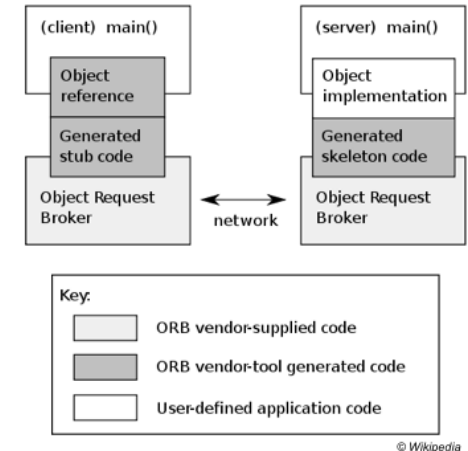
- CORBA, DCOM, RMI, XML-RPC...
- Language independency
- Apparent simplicity



© Wikipedia

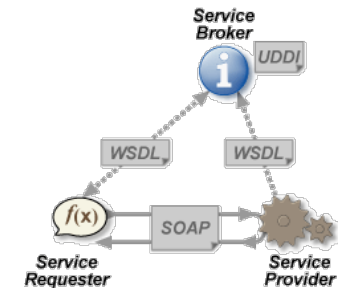
5A. RPC - drawbacks

- Tight coupling (generated client stub and server skeleton)
- No unique standard, leading to poor interoperability
- Poor scalability due to serialization/deserialization, but also:
- Hides the fact that objects are distributed, thus developer cannot make decisions according to objects locality.



5B. Message Oriented Architecture

- Most often implemented using the WS-* stack
- Web Services is a specific set of technologies for exposing Software Services:
 - the WSDL language for service description
 - the SOAP language as the message format
 - the HTTP protocol as the transport layer
 - the UDDI interface for service discovery



5B. Message Oriented Architecture - WS-*

Java method:

```
1 | public void myMethod(int x, float y);
```

WSDL

```
01 | <message name="myMethodRequest">
02 |     <part name="x" type="xsd:int"/>
03 |     <part name="y" type="xsd:float"/>
04 | </message>
05 | <message name="empty"/>
06 |
07 | <portType name="PT">
08 |     <operation name="myMethod">
09 |         <input message="myMethodRequest"/>
10 |         <output message="empty"/>
11 |     </operation>
12 | </portType>
13 |
14 | <binding .../>
```

SOAP

```
01 | POST /service
02 |
03 | <?xml version="1.0"?>
04 | <soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
```

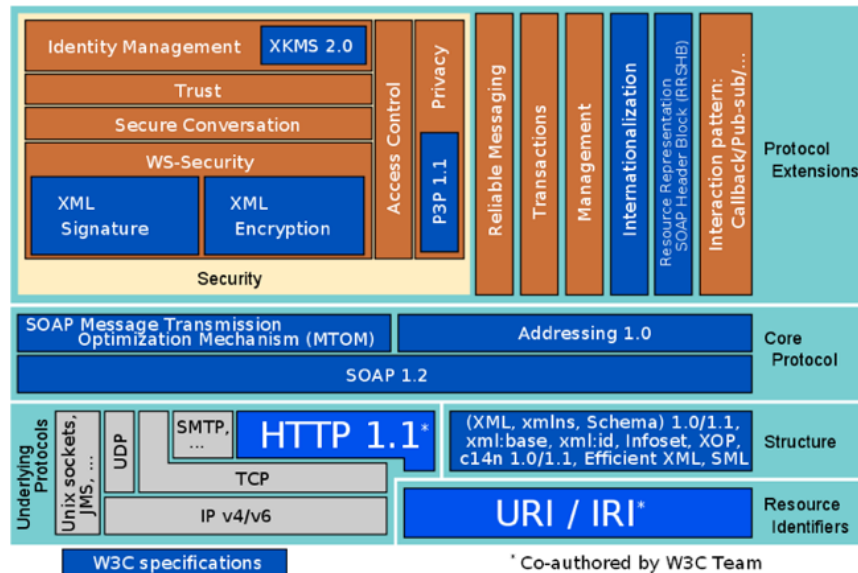
```
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
05  <soap:Header>
06    ...
07  </soap:Header>
08  <soap:Body>
09    <myMethod>
10      <x xsi:type="xsd:int">5</x>
11      <y xsi:type="xsd:float">5.0</y>
12    </myMethod>
13  </soap:Body>
14 </soap:Envelope>
```

5B. Message Oriented Architecture - benefits

- Kind of standardized, leading to better interoperability
- Easier to change the interface without breaking old clients

5B. Message Oriented Architecture - drawbacks

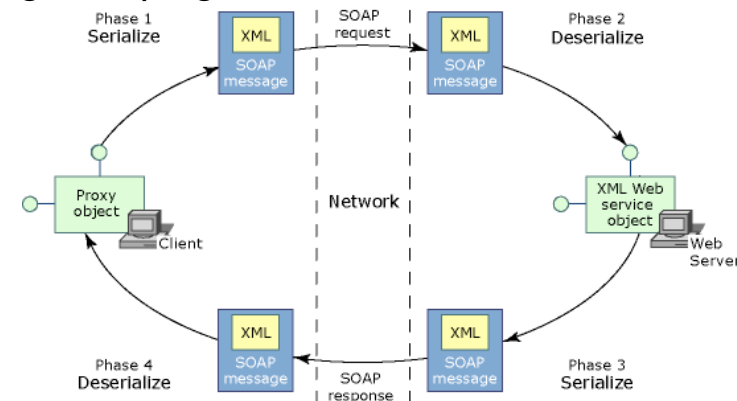
Complex, ever-changing specification



WS-Security, WS-Policy, WS-SecurityPolicy, WS-PolicyAssertions, WS-PolicyAttachment, WS-Trust, WS-Privacy, WS-Routing, WS-Referral, WS-Coordination, WS-Transaction, WS-SecureConversation, WS-Federation, WS-Authorization, WS-Attachments, WS-Transfer, WS-ResourceTransfer, WS-ReliableMessaging, WS-Addressing, ...

5B. Message Oriented Architecture - drawbacks

- Specification written and pushed by big vendors (IBM, Microsoft, HP, Intel, SAP, Sun, etc.)
- Poor interoperability between vendor implementations, leading to vendor lock-in
- Most often, developers use early binding => stubs, proxy objects => tight-coupling
- Poor scalability: requires processing power to decode SOAP messages, difficult to load-balance or route requests based on the service endpoint (URI)
- UDDI is a failure
- Does not use HTTP as an application protocol, only as a transport protocol.



However, some advanced specs may be of interest (Security, QoS) if your environment requires it.

5C. RESTful Services - concepts

- Representational State Transfer, in [Architectural Styles and the Design of Network-based Software Architectures](#)
Roy T. Fielding (2000)

*The REST Web is the subset of the WWW (based on **HTTP**) in which agents provide **uniform interface** semantics – essentially create (POST), retrieve (GET), update (PUT) and delete (DELETE) – rather than **arbitrary or application-specific** interfaces, and manipulate **resources** only by the exchange of **representations**. Furthermore, the REST interactions are “**stateless**” in the sense that the meaning of a message does not depend on the state of the conversation.*

5C. RESTful Services - concepts

Resources

- Functionalities offered by the service are exposed under the form of resources (e.g. jobs, users, payments, orders, friends...)

A resource can be anything that has identity (RFC 2396)

- Real objects (user, site) or abstract concepts (status, payment)
- A resource may have **multiple representations**. E.g. a calendar may be represented using

`image/png, text/html, application/pdf, application/xml, text/calendar`

URI

- Uniquely identifies each resource using a standardized syntax (`http://server.com/users/crohr`, `https://bank.com/accounts`, `ftp://ftp-server/document`, `mailto:cyril.rohr@irisa.fr`, `xmpp://crohr@jabber.grid5000.fr...`).
- Allow the use of **hypermedia** links to navigate from resource to resource.

5C. RESTful Services - concepts

Unified interface

All resources share the same interface for transferring state between the client and the resource:

- a restricted set of well-defined operations (HTTP verbs)

GET	fetch a resource
POST	create a new resource
PUT	update a resource
DELETE	delete a resource
...	

- a restricted set of standard media types

```
text/html, image/png, application/xml...
```

...but you can define your own.

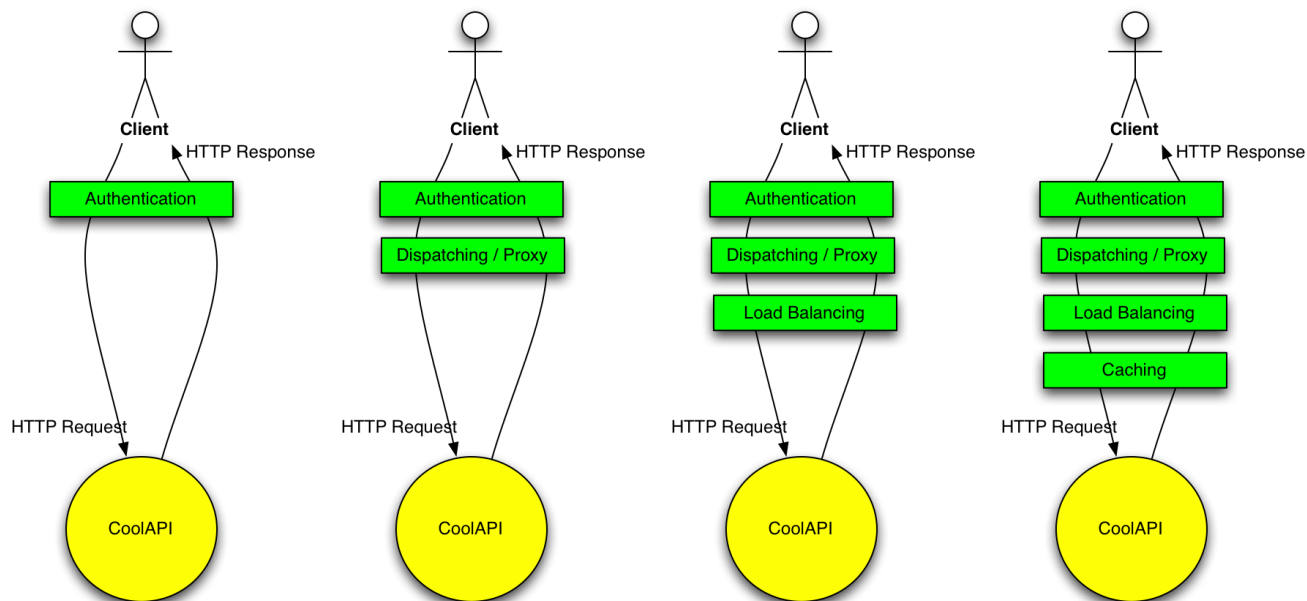
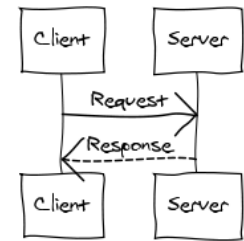
- a restricted set of status codes

```
200 (OK), 201 (Created), 202 (Accepted),  
303 (See Other), 304 (Not Modified),  
400 (Bad Request), 404 (Not Found), 409 (Conflict),  
500 (Internal Server Error)  
...
```

5C. RESTful Services - concepts

HTTP

- Client-Server protocol
- Stateless (each operation is self-sufficient)
- Cacheable
- Layered (intermediaries can be inserted between client and server, such as proxy, firewall, load-balancer, etc.)



5C. RESTful Services - benefits

- **Scalability** thanks to: cacheable requests (GET » POST,PUT,DELETE), and statelessness (requests are not bound to a specific server).
- **Interoperability**: the only thing that changes is the name of resources. Available HTTP verbs, status codes, etc. are always identical from one service to another => “given a URI, anyone already knows how to access it”.
- **Addressability**: each resource is uniquely identified.

```
https://api.grid5000.fr/sid/grid5000/sites/rennes
```

- **Versioning** is easy: insert a version number in the URI (or use a custom HTTP Header), and dispatch accordingly to the correct server.
- **Control** meta-data (HTTP headers, both in Request and Response)

```
Accept: application/json [format]
Cache-Control: max-age=120 [cache]
Accept-Encoding: gzip, deflate [compression]
Accept-Language: us,en;q=0.5 [language]
Accept-Charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7 [charset]
...You can define your own: X-API-Version, X-API-User...
```

- Removes unnecessary complexity: headers, methods, status codes, media types are already standardized. The protocol is well documented, and in use since the Web exists. It's not another protocol on top of another protocol on top of another protocol on top of...

5C. RESTful Services - benefits

- **Security**: uses the inherent HTTP security model, certain methods to certain URIs can easily be restricted by firewall/proxy configuration.
- **Ubiquity**: every programming language can speak HTTP
- **Layered** architecture

5C. RESTful Services - constraints

- Requires to **carefully choose** the HTTP verb among those available

```
GET      safe      (doesn't change the state of the resource on the server)
         idempotent (same operation can be applied multiple times
                     and will always return the same result)

         cacheable
PUT      idempotent
DELETE  idempotent
POST
...
```

- **Stateless**: each communication must include all the material required for the server to understand the context.
- Do NOT disguise RPC calls as HTTP requests (Flickr API):

```
GET /service?action=getUser&id=crohr&...

POST /service
{
  "action": "getUser",
  "id": "crohr",
  ...
}
```

5C. RESTful Services - HowTo

- A bit of specification:
 - *Choose your resources*
 - *Choose the supported HTTP verbs for each resource*
 - *Choose the supported media types*
 - *Choose the status codes that will be returned*
 - *Think about the link relationships between your resources*
- Use frameworks that encourage REST style: Sinatra (ruby), Restlet (java), Django & web.py (python)...
- Think about non-functional aspects
 - *Authentication: use Basic-Auth, Digest or SSL Client Certificates. Avoid sessions and tokens stored in cookies.*
 - *Caching: max-age, public/private, ETags, Last-Modified*
 - *Request dispatching & Load balancing, based on URI/IP/Geolocation/Traffic congestion...*

5C. RESTful Services - Client Side Code

■ Ruby

```
1 | require 'restclient'
2 | response = RestClient.get "https://api.grid5000.fr/sid/grid5000", :user
   => "crohr", :password => "whatever"
```

■ cURL

```
1 | $ curl -X GET -kni https://api.grid5000.fr/sid/grid5000
```

■ Javascript (jQuery)

```
$.ajax({
  url: "http://some/url",
  dataType: "json",
  success: function(data) {
    // do something
  },
  error: function() {},
  ...
})
{:class="brush: ruby"}
```

■ Example of a response

```
01 | HTTP/1.1 200 OK
02 | Date: Fri, 29 Jan 2010 12:12:14 GMT
03 | ETag: "ca5998adb91af61b9f9c57f2a26f71edebea5ffc"
04 | Allow: GET
05 | Cache-Control: public, must-revalidate
06 | Last-Modified: Mon, 25 Jan 2010 15:31:30 GMT
```

```
07 Content-Length: 998
08 Status: 200
09 Content-Type: application/vnd.fr.grid5000.api.grid+json;level=1
10 Age: 15023
11 X-Cache: HIT from api-proxy.rennes.grid5000.fr
12
13 {
14     "uid": "grid5000",
15     "type": "grid",
16     "version": "1cd3a84ff0c25d269cb24ac294e29b0fe8c111c5",
17     "links": [
18         {
19             "href": "/sid/grid5000/versions
20 /1cd3a84ff0c25d269cb24ac294e29b0fe8c111c5",
21             "title": "version",
22             "rel": "member",
23             "type": "application/vnd.fr.grid5000.api.Version+json;level=1"
24         },
25         {
26             "href": "/sid/grid5000/versions",
27             "title": "versions",
28             "rel": "collection",
29             "type": "application/vnd.fr.grid5000.api.Collection+json;level=1"
30         },
31         {
32             "href": "/sid/grid5000",
33             "rel": "self",
34             "type": "application/vnd.fr.grid5000.api.Grid+json;level=1"
35         },
36         {
37             "href": "/sid/grid5000/environments",
38             "title": "environments",
39             "rel": "collection",
```



```
39         "type": "application/vnd.fr.grid5000.api.Collection+json;level=1"
40     },
41     {
42         "href": "/sid/grid5000/sites",
43         "title": "sites",
44         "rel": "collection",
45         "type": "application/vnd.fr.grid5000.api.Collection+json;level=1"
46     }
47 ]
48 }
```

5C. RESTful Services - Server Side Code

```
01 require 'rubygems'
02 require 'sinatra/base' # gem install sinatra
03 require 'json'         # gem install json
04
05 class API < Sinatra::Base
06
07   configure do
08     # some configuration here
09   end
10
11   get      '/URI' { do_something }
12   post     '/URI' { do_something }
13   put      '/URI' { do_something }
14   delete   '/URI' { do_something }
15
16 end
17 API.run! :host => "localhost", :port => 4567
```

5C. RESTful Services - Example: library of scientific publications

Specification

■ root

METHOD	URI, MEDIA-TYPES, CODES	DESCRIPTION
GET	/ json	API entry-point

■ authors

GET	/authors?affiliations=x,y,z json 200,304,406,500	fetch the list of authors, optionally filtered by affiliations
GET	/authors/:author_id json 200,304,404,406,500	fetch a specific author
POST	/authors json 201,400,406,415,500	create a new author
DELETE	/authors/:author_id 204,500	delete the author
PUT	/authors/:author_id json 200,400,406,415,500	update an author

■ papers

GET	/papers?tags=x,y,z json 200,304,406,500	fetch the list of papers, optionally filtered by tags
GET	/papers/:paper_id json,text,html,pdf,latex 200,304,404,406,500	fetch a specific author
POST	/papers json 201,400,406,415,500	create a new paper

DELETE	/papers/:paper_id	delete the paper
	204,500	
PUT	/papers/:paper_id	update a paper. deal with conflicts.
	json	
	200,400,406,409,415,500	

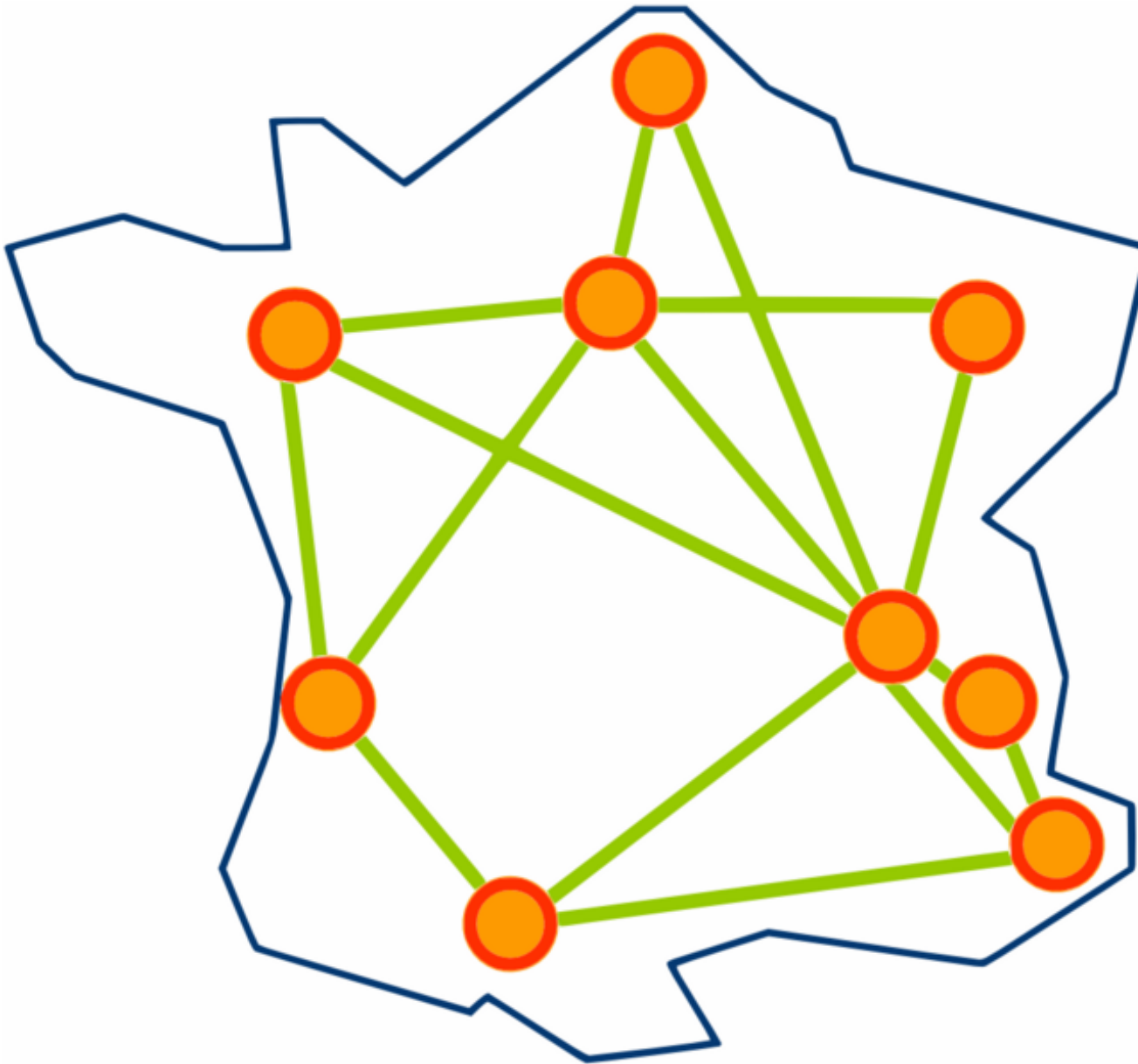
- convenience collections

GET	/authors/:author_id/papers	fetch the papers of a specific author
	json	
GET	/papers/:paper_id/authors	fetch the authors of a specific paper
	json	

See <http://github.com/crohr/rest-presentation/tree/master/app> for the implementation.

6. RESTful Services - Real-World examples - Grid5000

<http://www.grid5000.fr>



6. RESTful Services - Real-World examples - In the wild

RESTful

- Amazon S3 API: <http://docs.amazonwebservices.com/AmazonS3/latest/API/>
- Netflix API: http://developer.netflix.com/docs/REST_API_Reference
- Sun Cloud API: <http://kenai.com/projects/suncloudapis/pages/Home>
- Google Data API: <http://code.google.com/apis/gdata/docs/2.0/basics.html>

Not RESTful

- Digg API: <http://digg.com/api/docs/overview>
- Flickr API: <http://www.flickr.com/services/api/>

References

- RESTful Web Services, by Leonard Richardson and Sam Ruby, ISBN#978-0-596-52926-0
- <http://tech.groups.yahoo.com/group/rest-discuss>
- <http://code.google.com/p/implementing-rest>
- HTTP/1.1 Method Definitions - <http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>
- HTTP/1.1 Status Code Definitions - <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>
- HTTP/1.1 Header Field Definitions - <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html>
- <http://www.subbu.org/>
- [Web Services architecture](#)
- Sinatra Ruby framework: <http://www.sinatrarb.com>

Questions

- cyril.rohr@irisa.fr
- <http://crohr.me>