REST

REpresentational State Transfert

Toward Resource-Oriented Architecture (ROA)

Plan

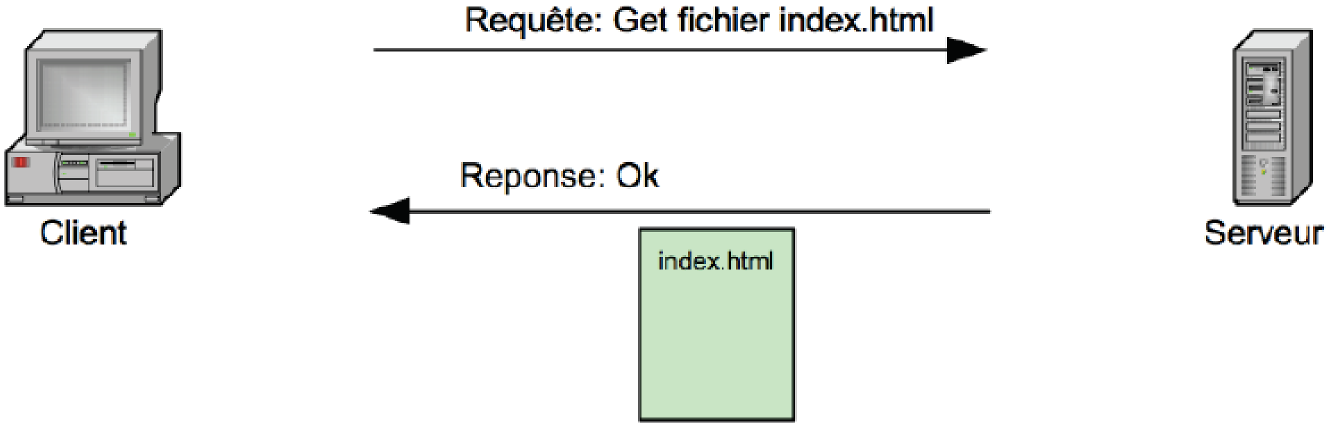
1. Reminders on HTTP
2. Maturity Model of L. Richardson
3. REST Web services

Part 1

Reminders on HTTP

HyperText Transfert Protocol

- HTTP enables to access files that are located on the Internet. It is used for the *World Wide Web*
- HTTP is above TCP and operates according to a request/response principle
  - The client sends a request containing information about the requested document.
  - The server sends the document if available, or an error message.



HTTP is a synchronous protocol initially **connectionless**, and each couple request/response is then independent.

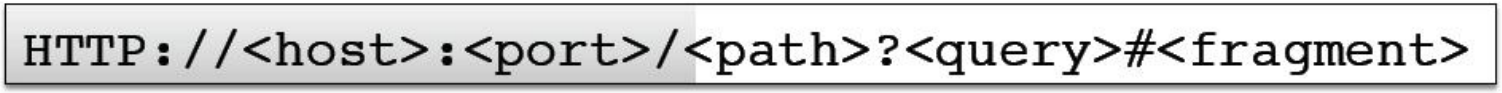
HTTP – Brief history

- From HTTP 1.0 to HTTP 1.1
- Protocol at CERN in the early 1990s to provide a simple *Web transfer protocol.*
- Two protocol versions that exist:
  - HTTP 1.0 is defined in 1996 by RFC 1945
  - HTTP 1.1 is defined in 1999 by RFC 2616
- The version 1.1 brings the following enhancements :
  - Five new methods
  - Persistant connections

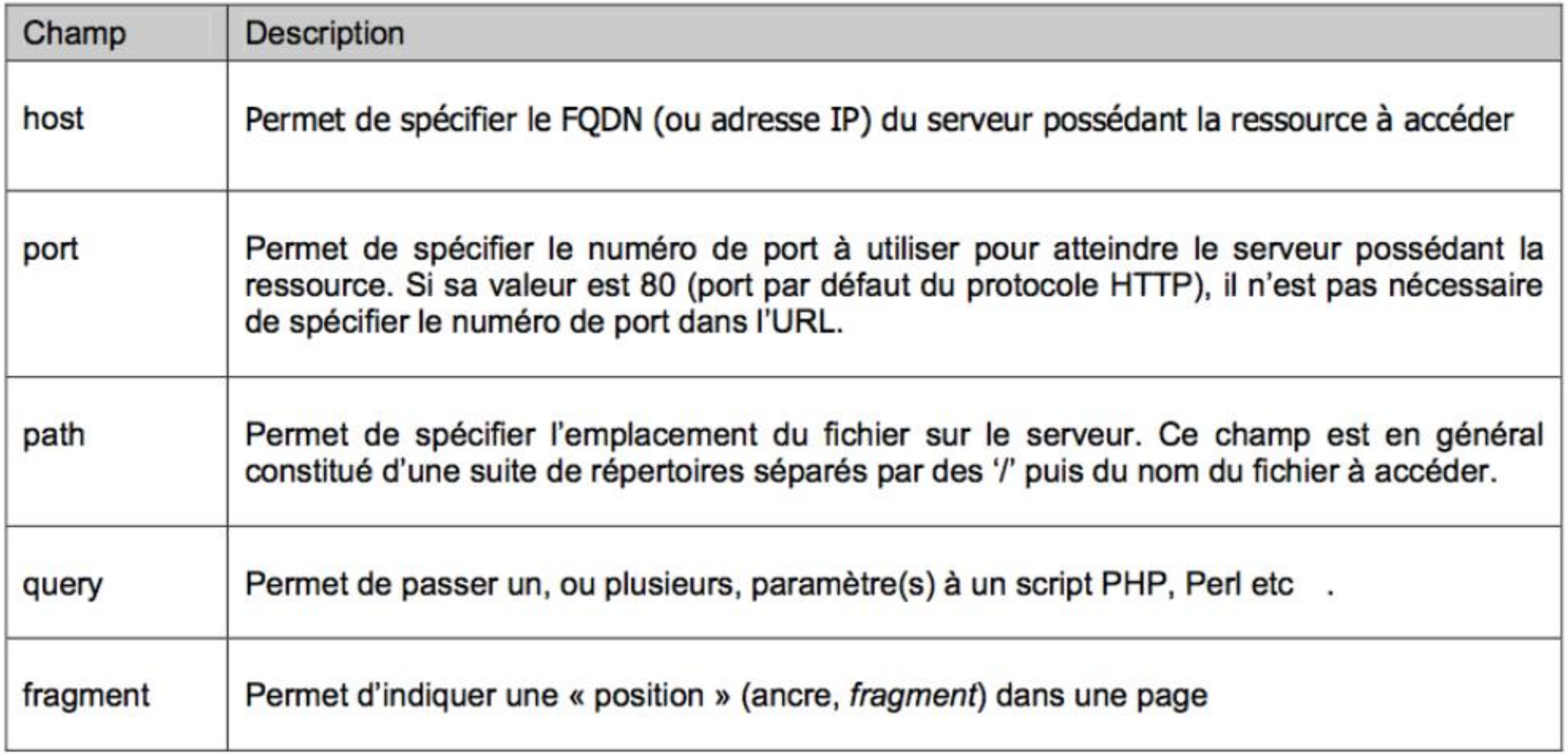
« Adresses » HTTP

- *Uniform Resource Identifier URI*
  - String of characters structured to **uniquely identify a resource in a space of a defined name.**
  - This resource may be designated either by a URN or by a URL.
  - URN and URL are subsets of URI.
- *Uniform Resource Name or URN*
  - Enables to **identify a resource by his name even when this latter is no longer available**.
- *Uniform Resource Locator or URL*
  - Allows to **locate a resource.**
  - In the case of HTTP, URL locates an HTML page, a text file, a CGI script, an image ...

URI Format



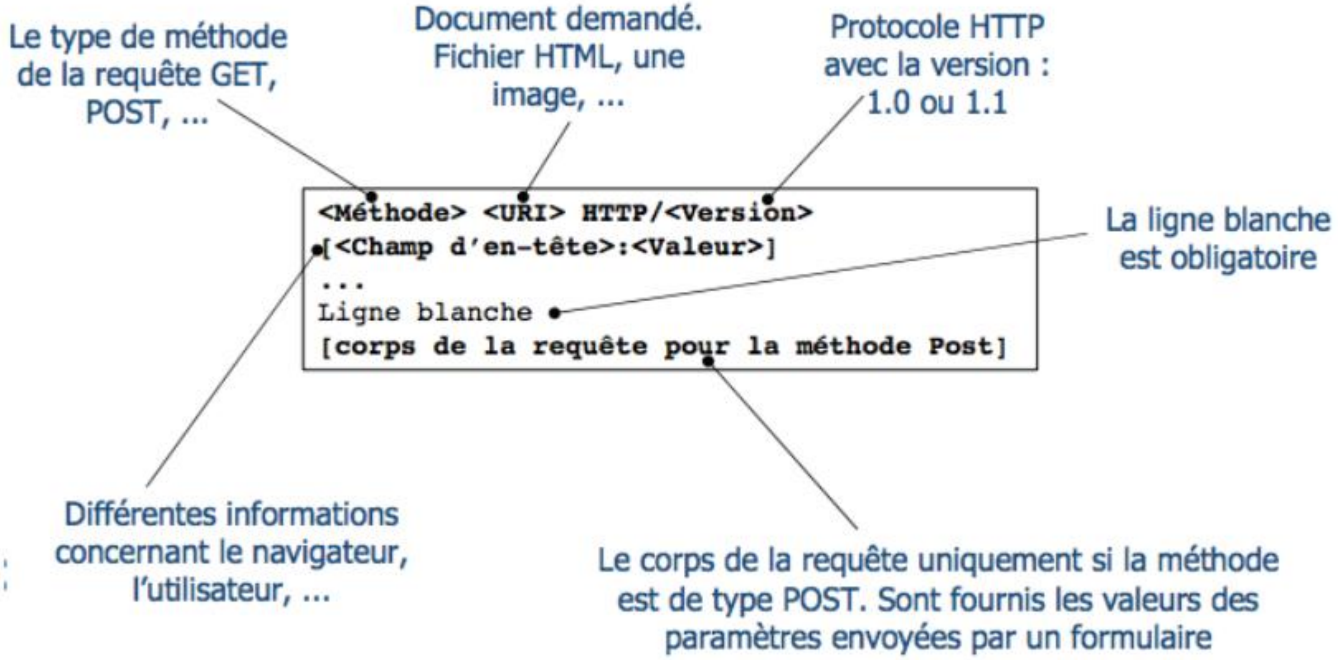
```
HTTP://<host>:<port>/<path>?<query>#<fragment>
```



Champ	Description
host	Permet de spécifier le FQDN (ou adresse IP) du serveur possédant la ressource à accéder
port	Permet de spécifier le numéro de port à utiliser pour atteindre le serveur possédant la ressource. Si sa valeur est 80 (port par défaut du protocole HTTP), il n'est pas nécessaire de spécifier le numéro de port dans l'URL.
path	Permet de spécifier l'emplacement du fichier sur le serveur. Ce champ est en général constitué d'une suite de répertoires séparés par des '/' puis du nom du fichier à accéder.
query	Permet de passer un, ou plusieurs, paramètre(s) à un script PHP, Perl etc .
fragment	Permet d'indiquer une « position » (ancre, *fragment*) dans une page

HTTP Protocol : Request (1/2)

- The request that is transmitted by the client to the server comprises:
  - A **request line** (request-line) containing the method used, the URI of the requested service, and the version used of HTTP
  - One or more **header lines**, each having a name and a value.



Le type de méthode de la requête GET, POST, ...

Document demandé. Fichier HTML, une image, ...

Protocole HTTP avec la version : 1.0 ou 1.1

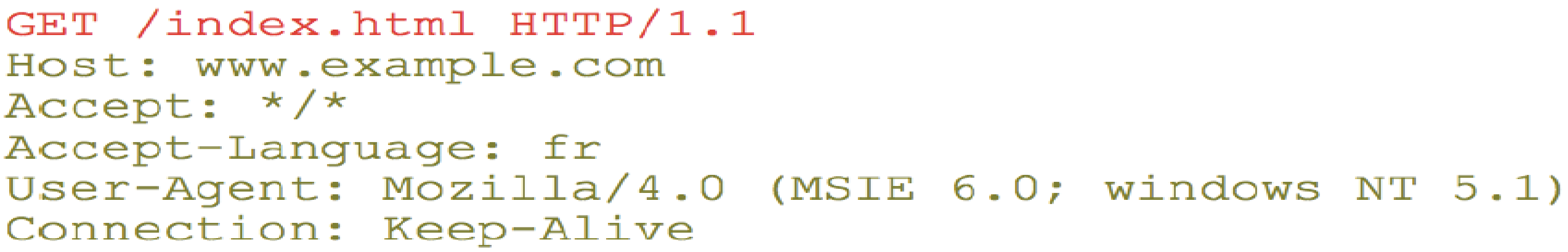
```
<Méthode> <URI> HTTP/<Version>
[<Champ d'en-tête>:<Valeur>]
...
Ligne blanche
[corps de la requête pour la méthode Post]
```

La ligne blanche est obligatoire

Différentes informations concernant le navigateur, l'utilisateur, ...

Le corps de la requête uniquement si la méthode est de type POST. Sont fournis les valeurs des paramètres envoyées par un formulaire

HTTP Protocol : Request (2/2)



```
GET /index.html HTTP/1.1
Host: www.example.com
Accept: */*
Accept-Language: fr
User-Agent: Mozilla/4.0 (MSIE 6.0; windows NT 5.1)
Connection: Keep-Alive
```

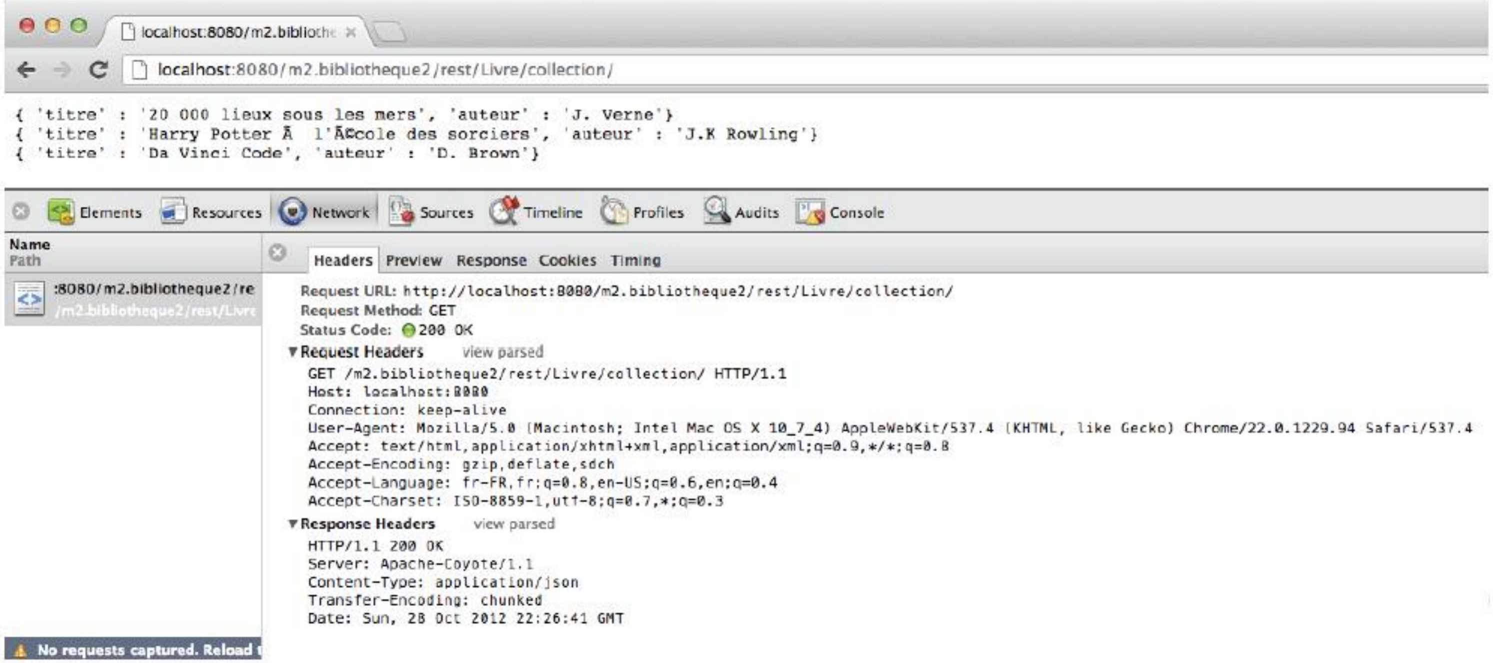
- **Example**
- The client requests the document at the adress
  - http://www.example.com/index.html
- He accepts, in return, all types of the document
- Prefers the documents in French language
- Uses a browser that supports Mozilla 4.0 on a Windows NT 5.1 system (Windows XP)
- Indicates to the TCP server that it should keep the connection opened after the request (because it has other requests to be transmitted).

HTTP Protocol: methods types (1/2)

- When a client connects to a server and sends a query, this query may take several types, called **methods**
- Requests of type GET
  - To retrieve information
    - Document, chart
  - Integrates URI formatting of the data (query string)
    - www.toto.com/hello?key1=titi&key2=tata&…
- Requests of type POST
  - To post secret information (not visible in the header), graphic data, ... Transmitted in the body of the request
  - Transmitted in the body of the request

Principal methods – GET (1/2)

- The method **GET is an information request on a resource**
  - The information provided as a response takes the following form:
    - A set of headers
    - And a content
  - The client never sends a representation with the request (request body is empty)



localhost:8080/m2.biblioth
localhost:8080/m2.bibliotheque2/rest/Livre/collection/

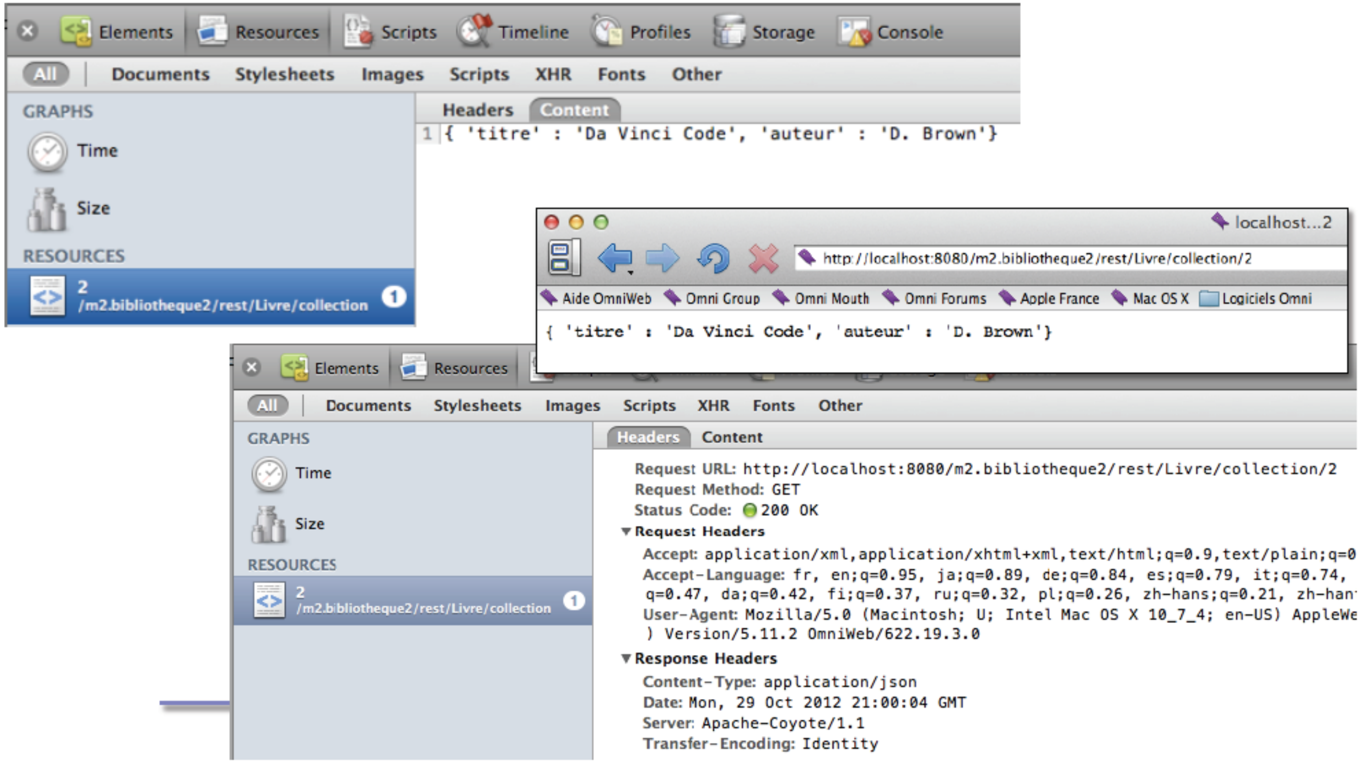
```
{ 'titre' : '20 000 lieux sous les mers', 'auteur' : 'J. Verne'}
{ 'titre' : 'Harry Potter Ã  l'Ã©cole des sorciers', 'auteur' : 'J.K Rowling'}
{ 'titre' : 'Da Vinci Code', 'auteur' : 'D. Brown'}
```

Elements
Resources
Network
Sources
Timeline
Profiles
Audits
Console
Name
Path
:8080/m2.bibliotheque2/re
/m2.bibliotheque2/rest/Livre
Headers
Preview
Response
Cookies
Timing
No requests captured. Reload t

```
Request URL: http://localhost:8080/m2.bibliotheque2/rest/Livre/collection/
Request Method: GET
Status Code: 200 OK
▼Request Headers      view parsed
  GET /m2.bibliotheque2/rest/Livre/collection/ HTTP/1.1
  Host: localhost:8080
  Connection: keep-alive
  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_4) AppleWebKit/537.4 (KHTML, like Gecko) Chrome/22.0.1229.94 Safari/537.4
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
  Accept-Encoding: gzip,deflate,sdch
  Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
  Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
▼Response Headers      view parsed
  HTTP/1.1 200 OK
  Server: Apache-Coyote/1.1
  Content-Type: application/json
  Transfer-Encoding: chunked
  Date: Sun, 28 Oct 2012 22:26:41 GMT
```

Principal methods– GET (2/2)

- The method **GET is an information request on a resource**
  - The information provided as a response takes the following form:
    - A set of headers
    - And a content



```
1 { 'titre' : 'Da Vinci Code', 'auteur' : 'D. Brown'}
```

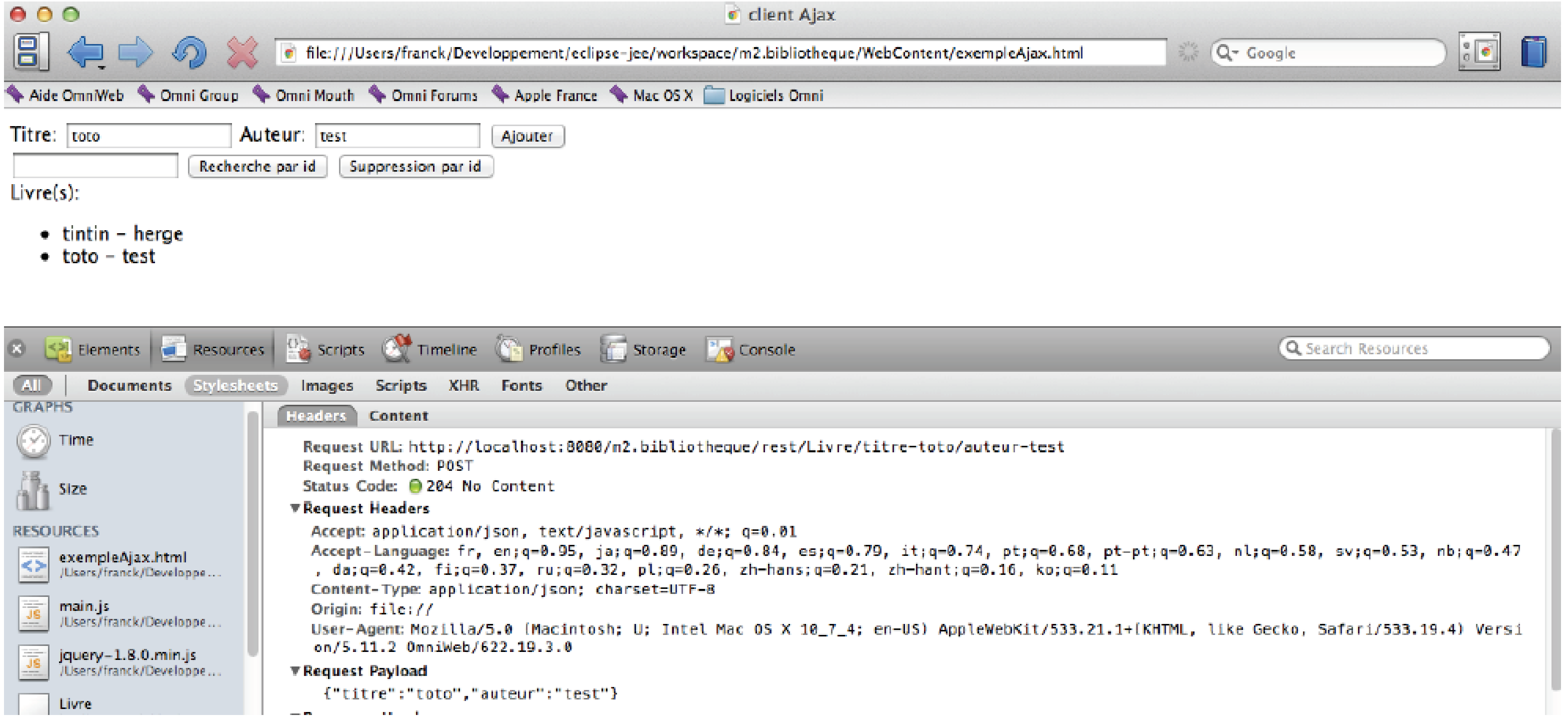
http://localhost:8080/m2.bibliotheque2/rest/Livre/collection/2

```
{ 'titre' : 'Da Vinci Code', 'auteur' : 'D. Brown'}
```

```
Request URL: http://localhost:8080/m2.bibliotheque2/rest/Livre/collection/2
Request Method: GET
Status Code: ● 200 OK
▼ Request Headers
  Accept: application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0
  Accept-Language: fr, en;q=0.95, ja;q=0.89, de;q=0.84, es;q=0.79, it;q=0.74,
  q=0.47, da;q=0.42, fi;q=0.37, ru;q=0.32, pl;q=0.26, zh-hans;q=0.21, zh-han
  User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_7_4; en-US) AppleWe
  ) Version/5.11.2 OmniWeb/622.19.3.0
▼ Response Headers
  Content-Type: application/json
  Date: Mon, 29 Oct 2012 21:00:04 GMT
  Server: Apache-Coyote/1.1
  Transfer-Encoding: Identity
```

Principal methods– POST

- The method **POST allows to create / add a new resource**
  - All parameters, to be moved to the services, can be in the header
  - No data is expected in response (but this still be possible)



Titre: toto Auteur: test

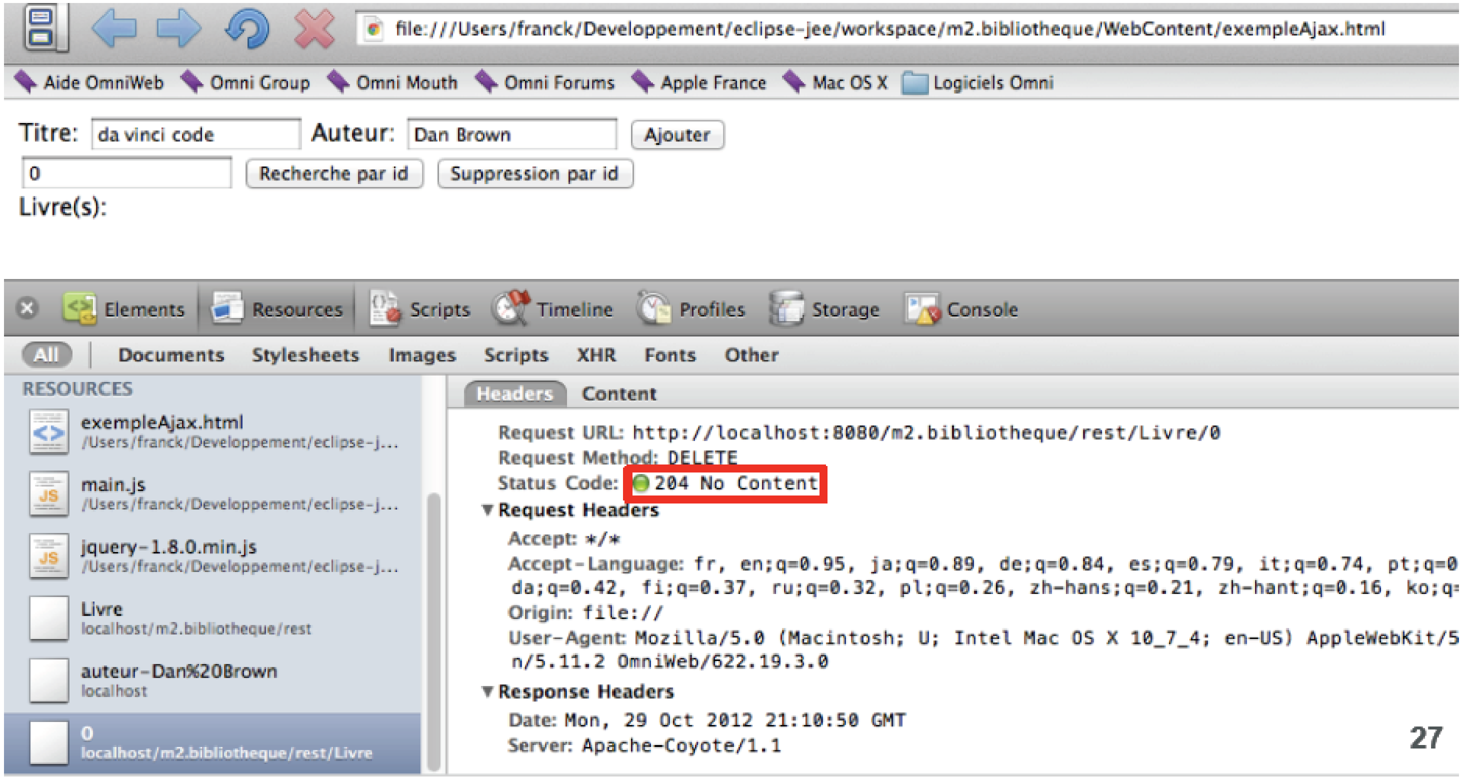
Livre(s):

- tintin - herge
- toto - test

```
Request URL: http://localhost:8080/m2.bibliotheque/rest/Livre/titre-toto/auteur-test
Request Method: POST
Status Code: 204 No Content
▼Request Headers
  Accept: application/json, text/javascript, */*; q=0.01
  Accept-Language: fr, en;q=0.95, ja;q=0.89, de;q=0.84, es;q=0.79, it;q=0.74, pt;q=0.68, pt-pt;q=0.63, nl;q=0.58, sv;q=0.53, nb;q=0.47
  , da;q=0.42, fi;q=0.37, ru;q=0.32, pl;q=0.26, zh-hans;q=0.21, zh-hant;q=0.16, ko;q=0.11
  Content-Type: application/json; charset=UTF-8
  Origin: file://
  User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_7_4; en-US) AppleWebKit/533.21.1+(KHTML, like Gecko, Safari/533.19.4) Versi
  on/5.11.2 OmniWeb/622.19.3.0
▼Request Payload
   {"titre":"toto","auteur":"test"}
```

Principal methods– DELETE

- The method **DELETE enables to delete a resource**
  - All parameters, to be moved to the services, can be in the header
  - No data is expected in response (but this still be possible)



Titre: da vinci code Auteur: Dan Brown

0

Livre(s):

exempleAjax.html
/Users/franck/Developpement/eclipse-j...

main.js
/Users/franck/Developpement/eclipse-j...

jquery-1.8.0.min.js
/Users/franck/Developpement/eclipse-j...

Livre
localhost/m2.bibliotheque/rest

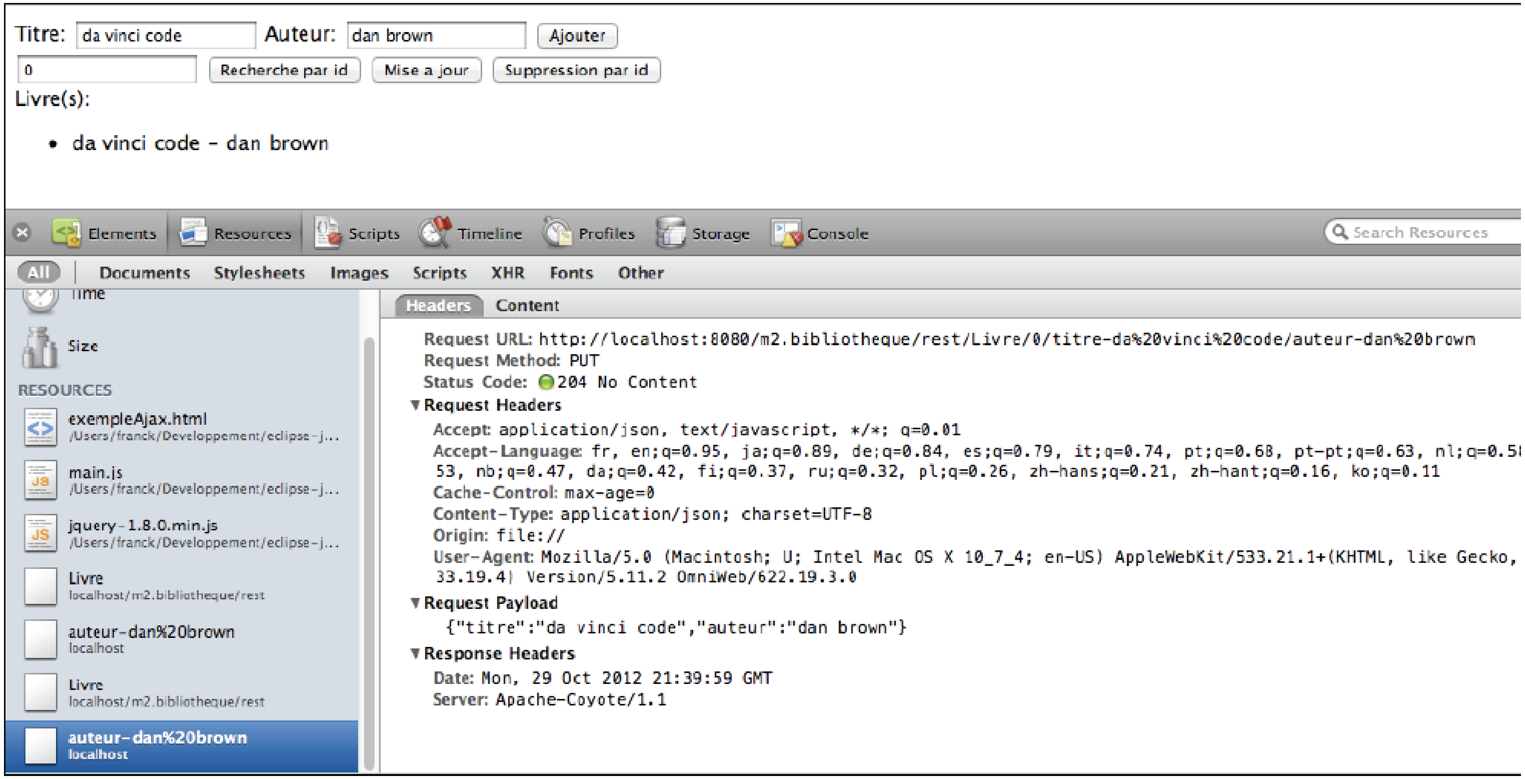
auteur-Dan%20Brown
localhost

0
localhost/m2.bibliotheque/rest/Livre

```
Request URL: http://localhost:8080/m2.bibliotheque/rest/Livre/0
Request Method: DELETE
Status Code: 204 No Content
▼Request Headers
  Accept: */*
  Accept-Language: fr, en;q=0.95, ja;q=0.89, de;q=0.84, es;q=0.79, it;q=0.74, pt;q=0
  da;q=0.42, fi;q=0.37, ru;q=0.32, pl;q=0.26, zh-hans;q=0.21, zh-hant;q=0.16, ko;q=
  Origin: file://
  User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_7_4; en-US) AppleWebKit/5
  n/5.11.2 OmniWeb/622.19.3.0
▼Response Headers
  Date: Mon, 29 Oct 2012 21:10:50 GMT
  Server: Apache-Coyote/1.1
```

Principal methods– PUT

- The method **PUT allows to update a resource**
  - Includes adding a sub-resource
  - All parameters, to be moved to the services, can be in the header
  - No data is expected in response (but this still be possible)



Titre: da vinci code Auteur: dan brown

0

Livre(s):

- da vinci code - dan brown

```
Request URL: http://localhost:8080/m2.bibliotheque/rest/Livre/0/titre-da%20vinci%20code/auteur-dan%20brown
Request Method: PUT
Status Code: ●204 No Content
▼Request Headers
  Accept: application/json, text/javascript, */*; q=0.01
  Accept-Language: fr, en;q=0.95, ja;q=0.89, de;q=0.84, es;q=0.79, it;q=0.74, pt;q=0.68, pt-pt;q=0.63, nl;q=0.5
  53, nb;q=0.47, da;q=0.42, fi;q=0.37, ru;q=0.32, pl;q=0.26, zh-hans;q=0.21, zh-hant;q=0.16, ko;q=0.11
  Cache-Control: max-age=0
  Content-Type: application/json; charset=UTF-8
  Origin: file://
  User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_7_4; en-US) AppleWebKit/533.21.1+(KHTML, like Gecko,
  33.19.4) Version/5.11.2 OmniWeb/622.19.3.0
▼Request Payload
   {"titre":"da vinci code","auteur":"dan brown"}
▼Response Headers
  Date: Mon, 29 Oct 2012 21:39:59 GMT
  Server: Apache-Coyote/1.1
```

Principal methods– safety et idempotence

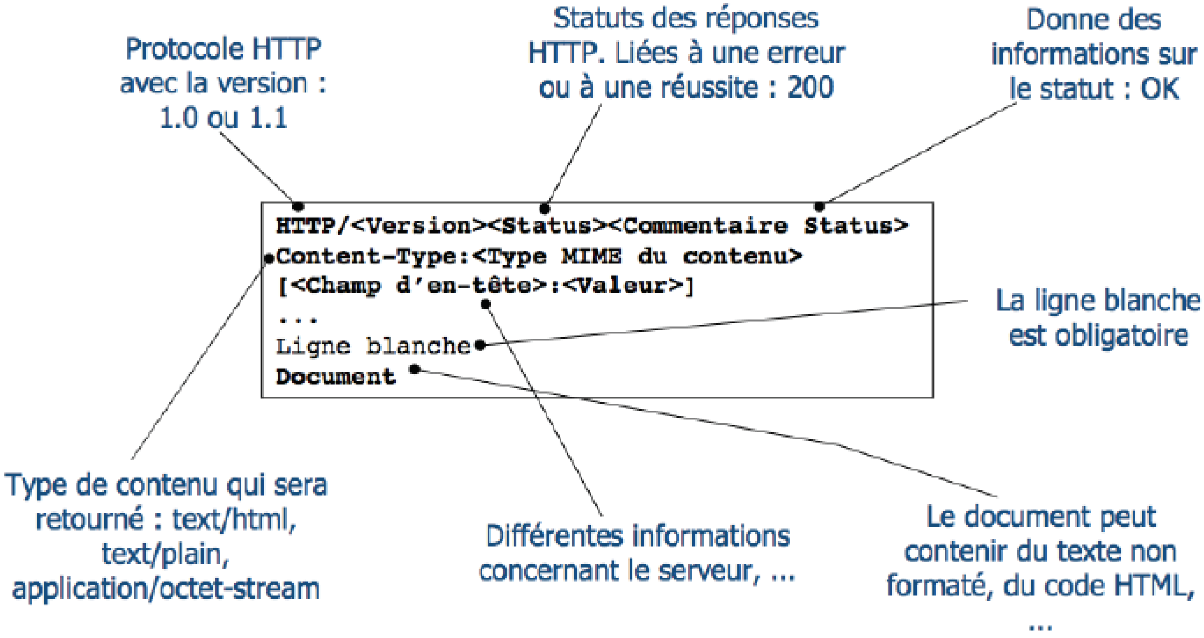
- Two important characteristics
  - **The safety**
  - **The idempotence**
- A **safe** method should never change the resource state**.**
  - Case of methods GET and HEAD
  - POST, DELETE and PUT are not safe
- A method is idempotent if it can be repeated any number of times, the set of resources always remains in the same state after applying the method
  - In other words, the result of an idempotent operation remains the same in a given context with given parameters

Principal methods– safety et idempotence

- The method GET is safe and idempotent
  - A client that makes a request of type GET on a resource does not require any change of this ressource
  - The server can evidently change however the client can not be responsible (e.g., log the request or increment a counter)
  - Repeating GET any number of times has the same effect
- Methods PUT and DELETE are idempotent
  - Make several requests PUT (or DELETE) on a resource must have the same effect as to make only one request.
  - Known Issue :
    - PUT that change the resource state by an increment of 5 on a value
    - Such specification is not possible to be idempotent
- The method POST is neither safe nor idempotent
  - It serves from «tool box» in various frameworks (custom messages, etc.)

HTTP Protocol : response (1/3)

- The transmitted response by the server to the client comprises :
  - A **status line** containing the used version of HTTP and a status code
  - One or more **header lines** including a name and a value
  - The document body returned (e.g., HTML or binary data).
    - A response does not necessarily contain a body (e.g, if it is a response to a HEAD request, only the status line and the headers are returned).



```
HTTP/<Version><Status><Commentaire Status>
Content-Type:<Type MIME du contenu>
[<Champ d'en-tête>:<Valeur>]
...
Ligne blanche
Document
```

Protocole HTTP avec la version : 1.0 ou 1.1

Statuts des réponses HTTP. Liées à une erreur ou à une réussite : 200

Donne des informations sur le statut : OK

La ligne blanche est obligatoire

Type de contenu qui sera retourné : text/html, text/plain, application/octet-stream

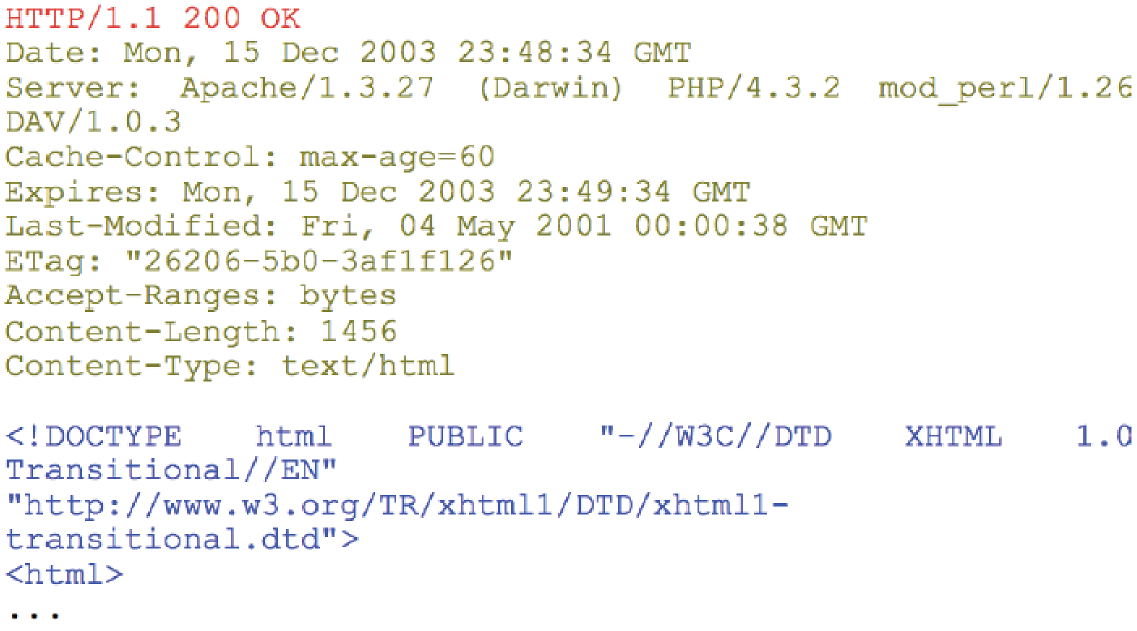
Différentes informations concernant le serveur, ...

Le document peut contenir du texte non formaté, du code HTML, ...

HTTP Protocol : response(2/3)

Example

- The code 200 indicates that the requested document has been found.
- To facilitate the management the client cache, the server transmits
  - The current date,
  - The date of the last modification of the document
  - The expiration date (after which the document can be requested again).

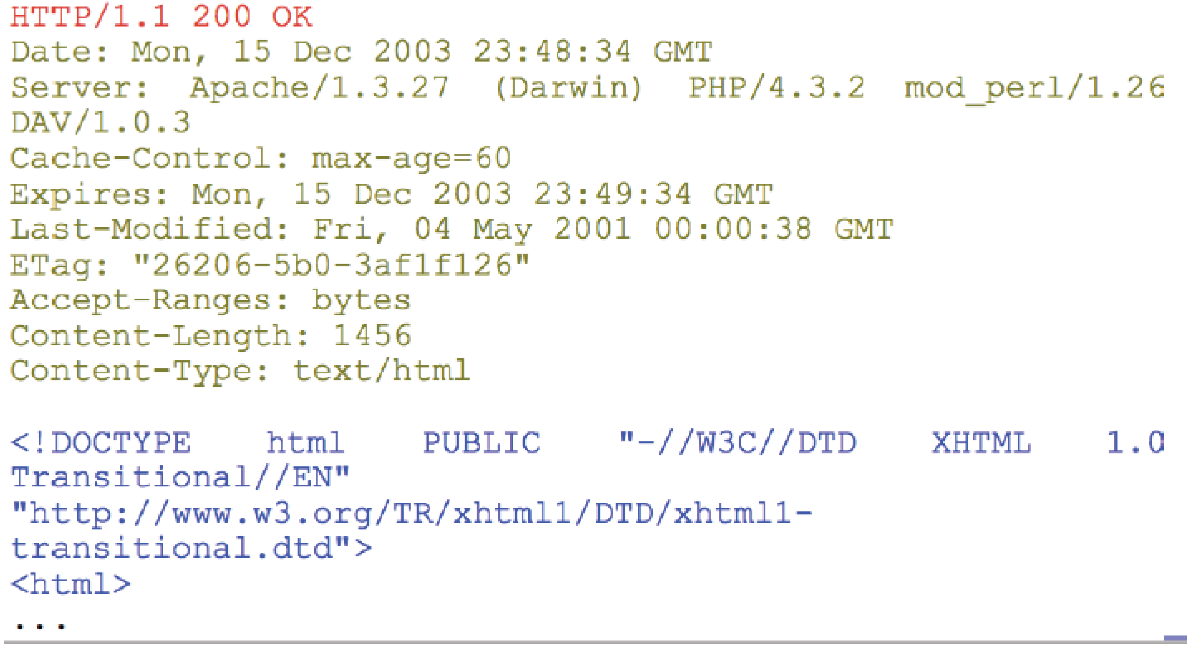


```
HTTP/1.1 200 OK
Date: Mon, 15 Dec 2003 23:48:34 GMT
Server: Apache/1.3.27 (Darwin) PHP/4.3.2 mod_perl/1.26
DAV/1.0.3
Cache-Control: max-age=60
Expires: Mon, 15 Dec 2003 23:49:34 GMT
Last-Modified: Fri, 04 May 2001 00:00:38 GMT
ETag: "26206-5b0-3af1f126"
Accept-Ranges: bytes
Content-Length: 1456
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html>
...
```

HTTP Protocol : response(3/3)

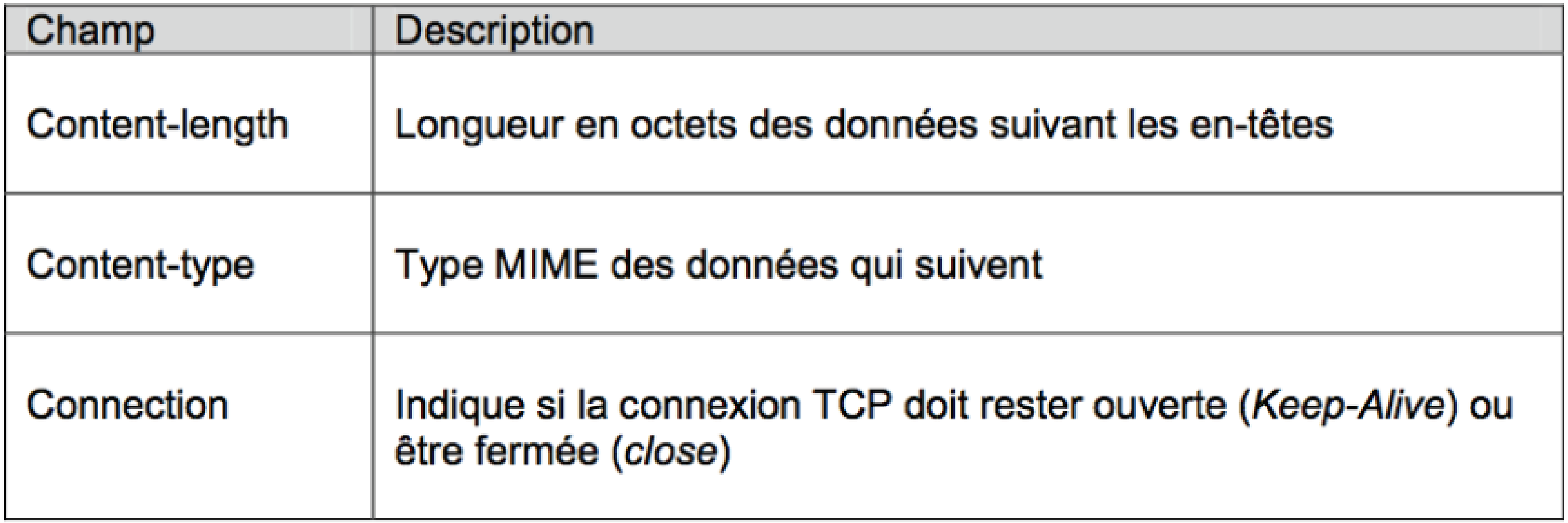
Example

- The header *Content-Type indicates that the returned document is of type HTML*
- The header *Content-Length indicates that the document body have a length of* 1456 bytes.
- The header *Server indicates the used software server.*
  - Sending such information is not recommended from a security point of view.



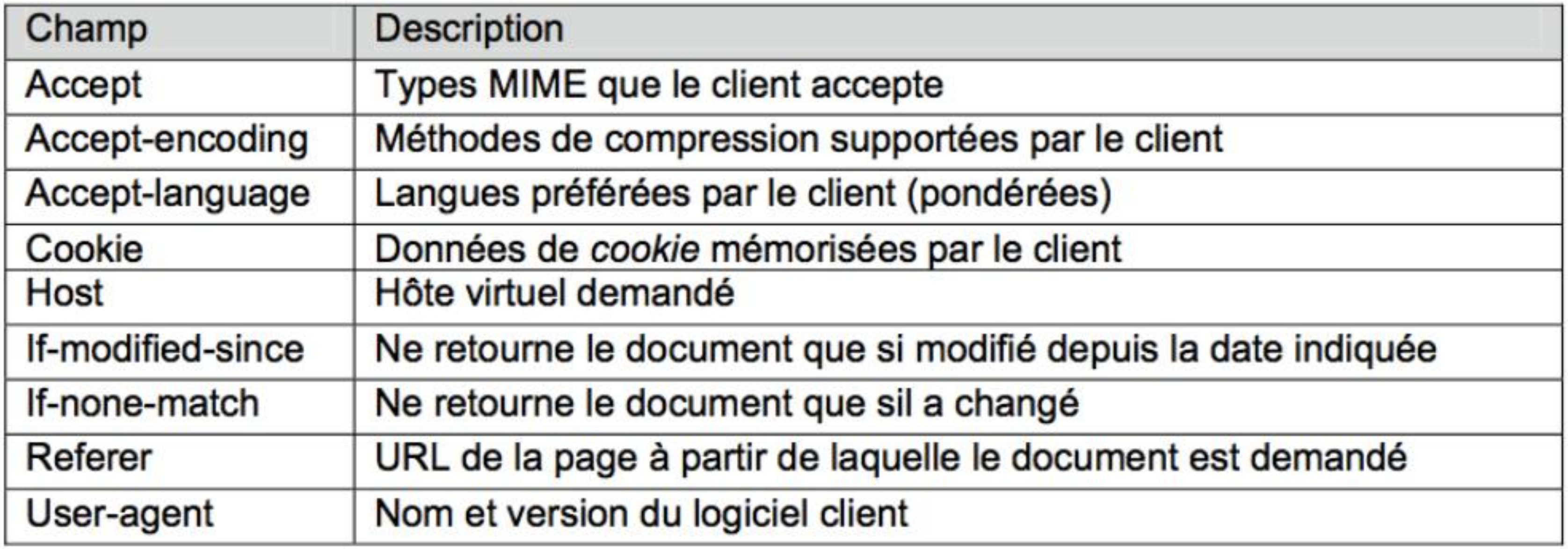
```
HTTP/1.1 200 OK
Date: Mon, 15 Dec 2003 23:48:34 GMT
Server:  Apache/1.3.27  (Darwin)  PHP/4.3.2  mod_perl/1.26
DAV/1.0.3
Cache-Control: max-age=60
Expires: Mon, 15 Dec 2003 23:49:34 GMT
Last-Modified: Fri, 04 May 2001 00:00:38 GMT
ETag: "26206-5b0-3af1f126"
Accept-Ranges: bytes
Content-Length: 1456
Content-Type: text/html

<!DOCTYPE    html    PUBLIC    "-//W3C//DTD    XHTML    1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html>
...
```

Generic headers of HTTP messages



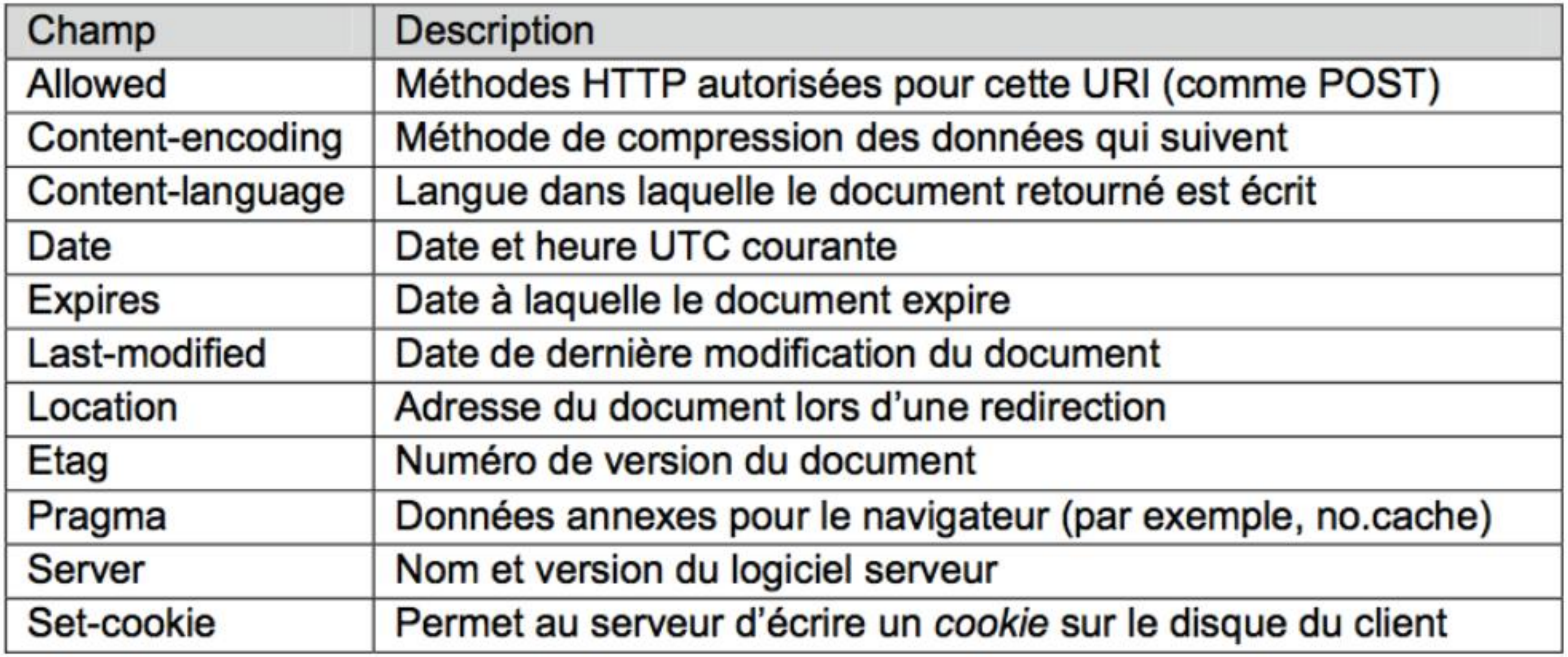
Champ	Description
Content-length	Longueur en octets des données suivant les en-têtes
Content-type	Type MIME des données qui suivent
Connection	Indique si la connexion TCP doit rester ouverte (*Keep-Alive*) ou être fermée (*close*)

HTTP requests Headers



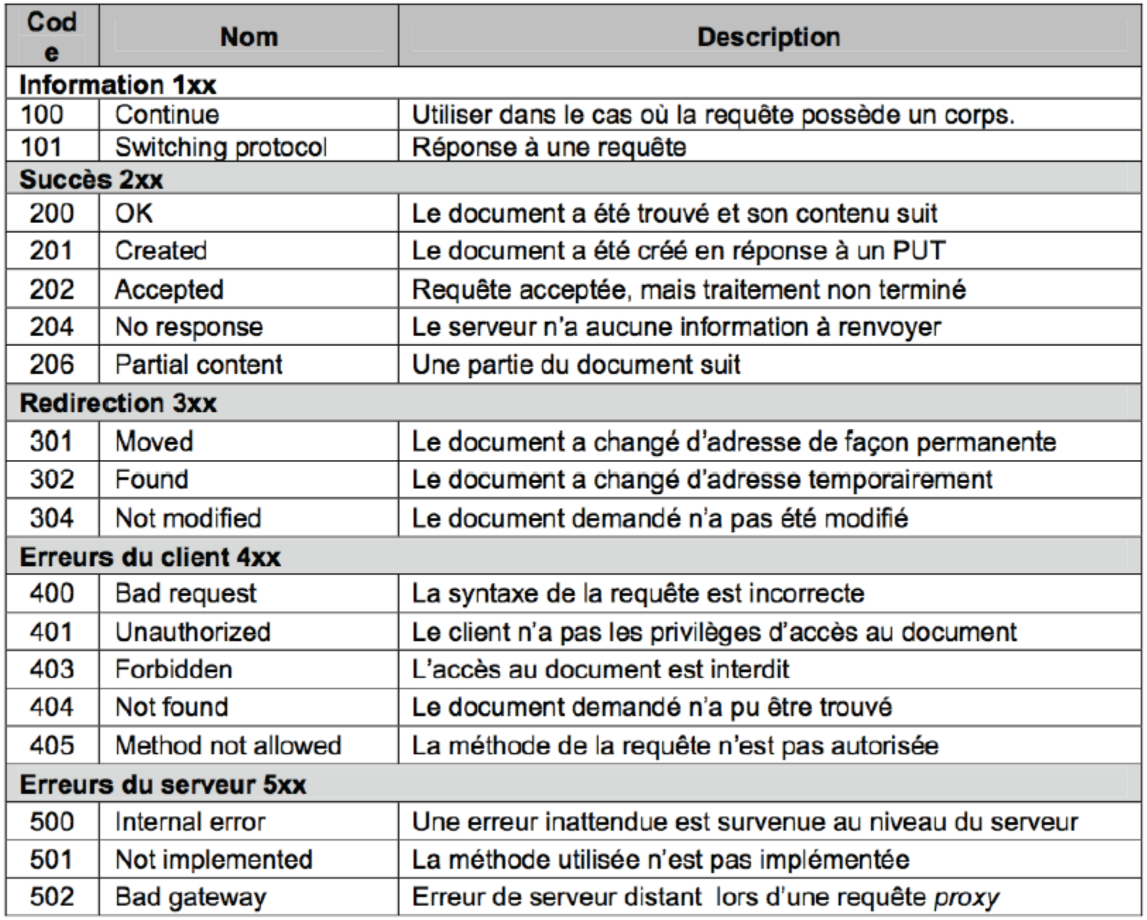
Champ	Description
Accept	Types MIME que le client accepte
Accept-encoding	Méthodes de compression supportées par le client
Accept-language	Langues préférées par le client (pondérées)
Cookie	Données de *cookie* mémorisées par le client
Host	Hôte virtuel demandé
If-modified-since	Ne retourne le document que si modifié depuis la date indiquée
If-none-match	Ne retourne le document que sil a changé
Referer	URL de la page à partir de laquelle le document est demandé
User-agent	Nom et version du logiciel client

HTTP responses Headers



Champ	Description
Allowed	Méthodes HTTP autorisées pour cette URI (comme POST)
Content-encoding	Méthode de compression des données qui suivent
Content-language	Langue dans laquelle le document retourné est écrit
Date	Date et heure UTC courante
Expires	Date à laquelle le document expire
Last-modified	Date de dernière modification du document
Location	Adresse du document lors d'une redirection
Etag	Numéro de version du document
Pragma	Données annexes pour le navigateur (par exemple, no.cache)
Server	Nom et version du logiciel serveur
Set-cookie	Permet au serveur d'écrire un *cookie* sur le disque du client

Some return codes of HTTP responses

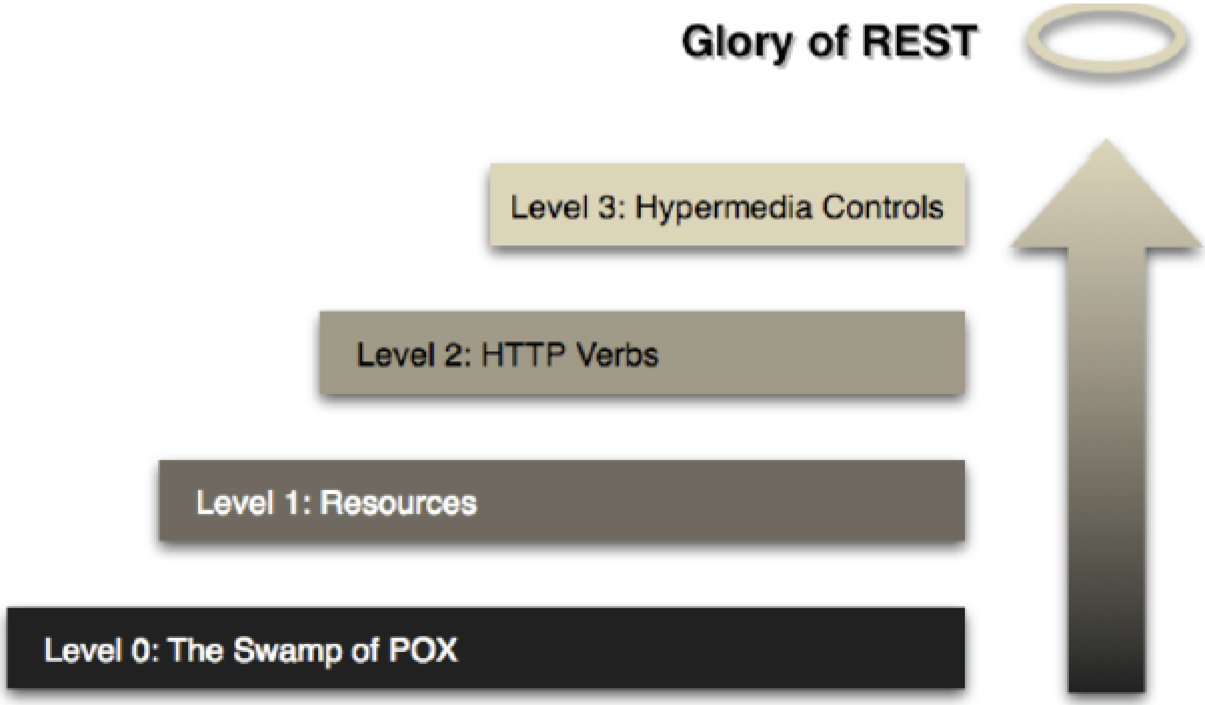


Code	Nom	Description
Information 1xx		
100	Continue	Utiliser dans le cas où la requête possède un corps.
101	Switching protocol	Réponse à une requête
Succès 2xx		
200	OK	Le document a été trouvé et son contenu suit
201	Created	Le document a été créé en réponse à un PUT
202	Accepted	Requête acceptée, mais traitement non terminé
204	No response	Le serveur n'a aucune information à renvoyer
206	Partial content	Une partie du document suit
Redirection 3xx		
301	Moved	Le document a changé d'adresse de façon permanente
302	Found	Le document a changé d'adresse temporairement
304	Not modified	Le document demandé n'a pas été modifié
Erreurs du client 4xx		
400	Bad request	La syntaxe de la requête est incorrecte
401	Unauthorized	Le client n'a pas les privilèges d'accès au document
403	Forbidden	L'accès au document est interdit
404	Not found	Le document demandé n'a pu être trouvé
405	Method not allowed	La méthode de la requête n'est pas autorisée
Erreurs du serveur 5xx		
500	Internal error	Une erreur inattendue est survenue au niveau du serveur
501	Not implemented	La méthode utilisée n'est pas implémentée
502	Bad gateway	Erreur de serveur distant lors d'une requête *proxy*

Part 2

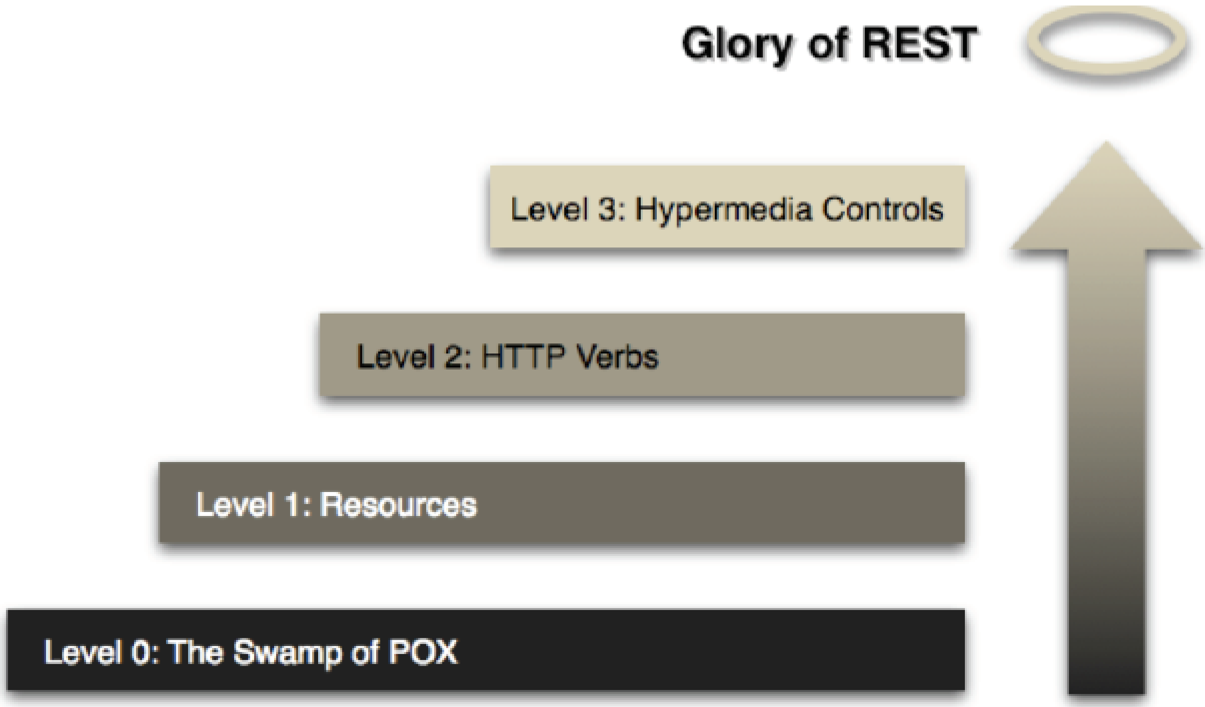
Maturity Model of L. Richardson

The Maturity Model of Richardson



- Original presentation of Leonard Richardson(QCON conference 2009)
  - http://www.crummy.com/writing/speaking/2008-QCon/act3.html
- Decryption by Martin Fowler in March 2010
  - http://martinfowler.com/articles/richardsonMaturityModel.html

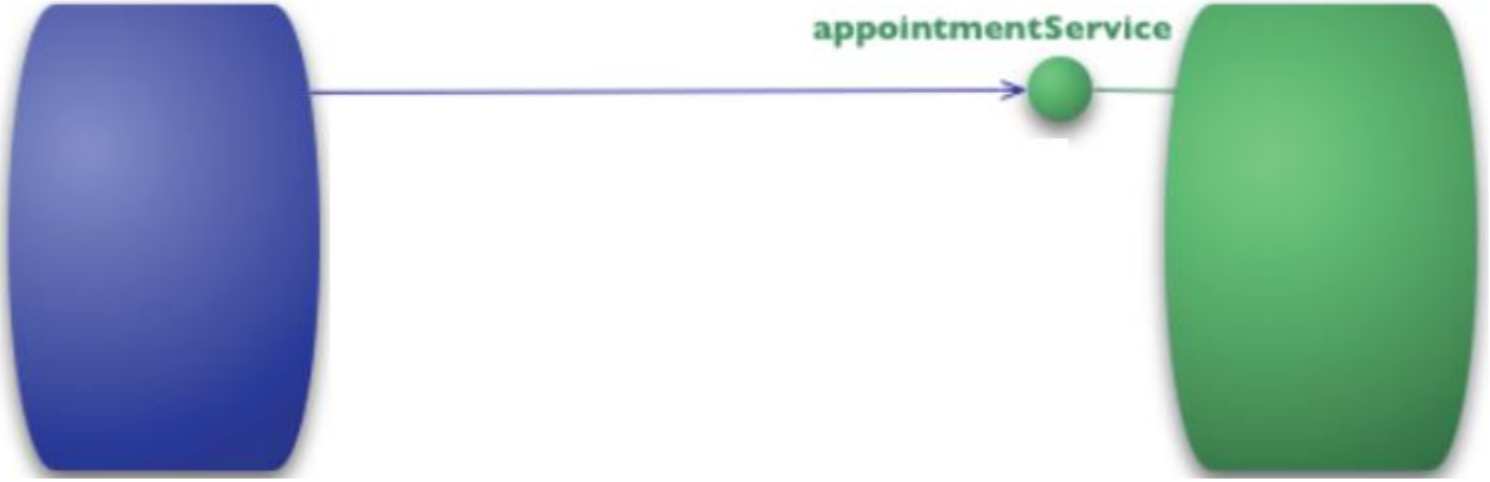
The Maturity Model of Richardson



- Level 0: The RPC over HTTP in POX
- Level 1: The use of differentiated resources
- Level 2: The use of verbs and HTTP return codes
- Level 3: The use of hypermedia controls

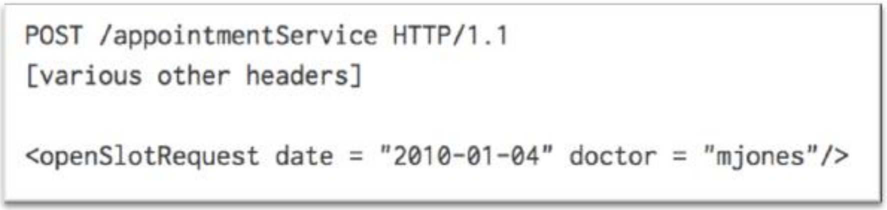
MMR – Level 0 – «tunneling» Mecanism

- HTTP as the transport system for remotely interact with a ”service”
  - Model RPC/RPI (Remote Procedure Call / Invocation)
  - All requests are sent to the same URI (or endpoint)
  - Example: Making an appointment with the doctor
    - A unique URI **appointmentService**

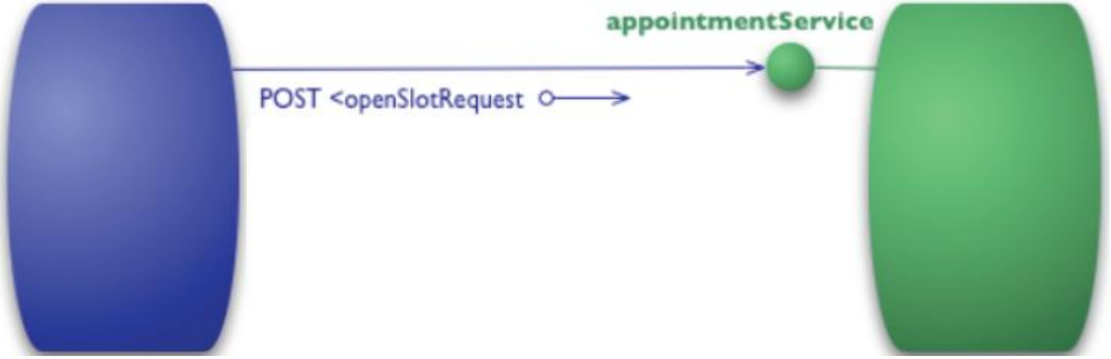


MMR – Level 0 – «tunneling» Mecanism

- Example: Making an appointment with the doctor
  - A unique URI **appointmentService**
  - The client component must first ask the server component for available time (open slots) at a given date

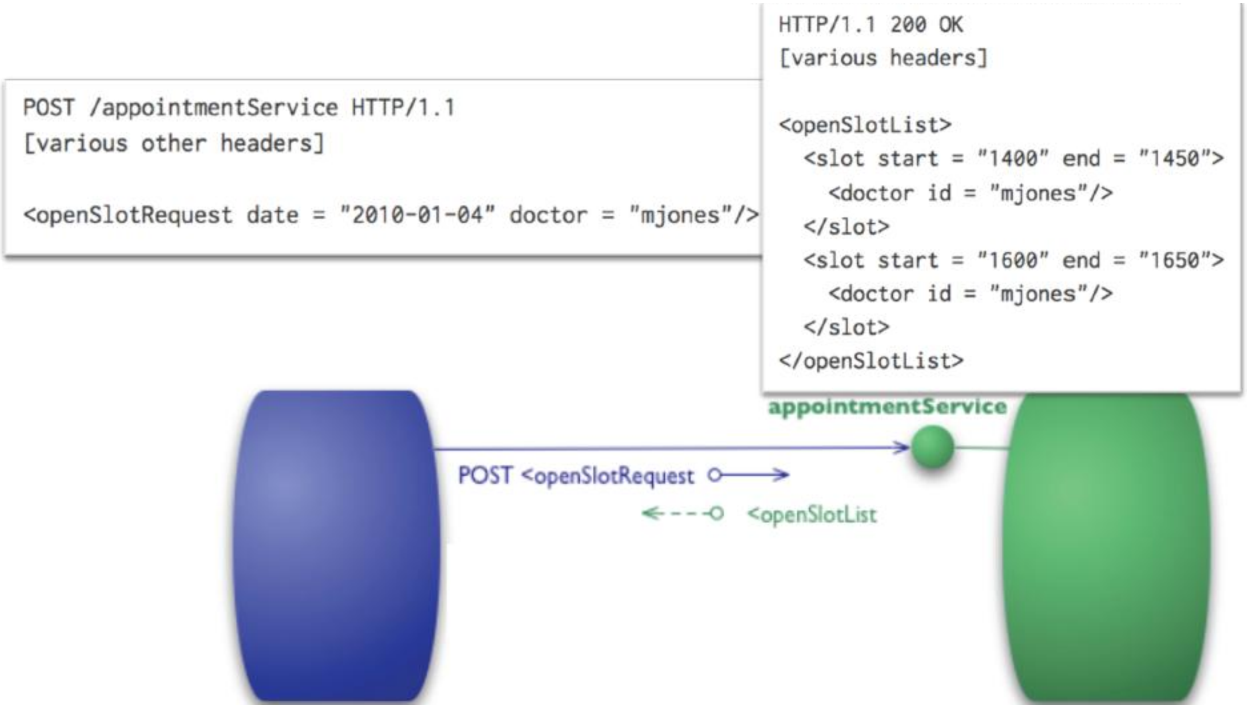


```
POST /appointmentService HTTP/1.1
[various other headers]

<openSlotRequest date = "2010-01-04" doctor = "mjones"/>
```



MMR – Level 0 – «tunneling» Mecanism

- Example: Making an appointment with the doctor
  - A unique URI **appointmentService**
  - The client component must first ask the server component for available time (open slots) at a given date

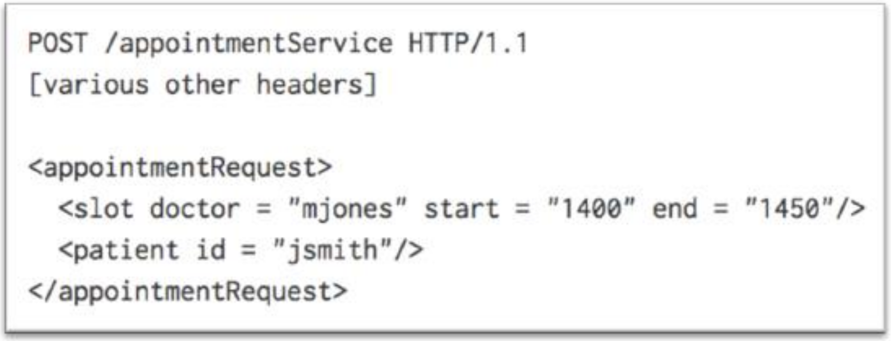


```
POST /appointmentService HTTP/1.1
[various other headers]

<openSlotRequest date = "2010-01-04" doctor = "mjones"/>
```

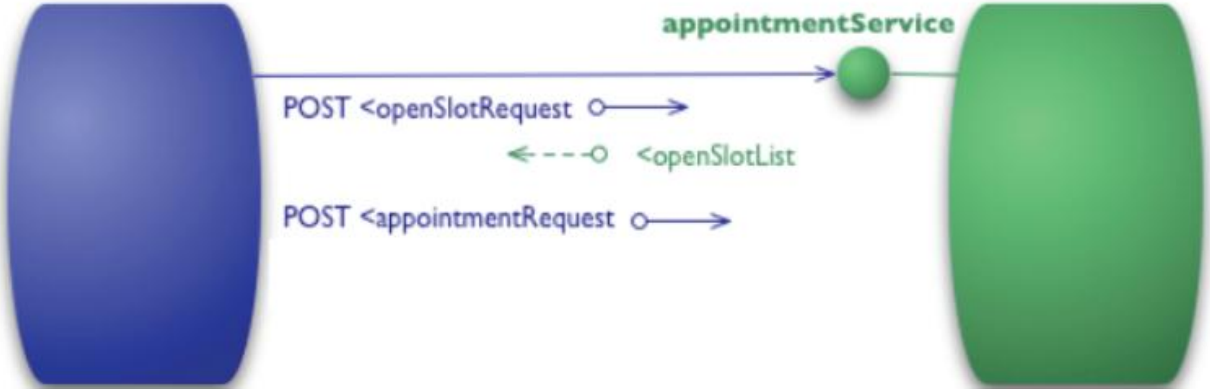
```
HTTP/1.1 200 OK
[various headers]

<openSlotList>
  <slot start = "1400" end = "1450">
    <doctor id = "mjones"/>
  </slot>
  <slot start = "1600" end = "1650">
    <doctor id = "mjones"/>
  </slot>
</openSlotList>
```

MMR – Level 0 – «tunneling» Mecanism

- Example: Making an appointment with the doctor
  - A unique URI **appointmentService**
  - Then take one appointment among the possible choices

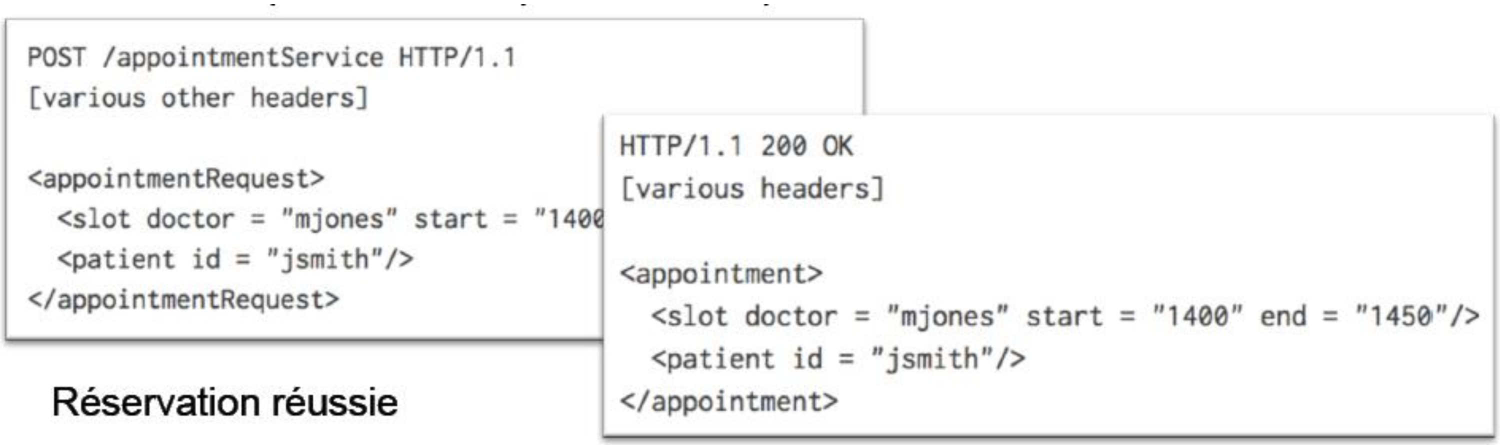


```
POST /appointmentService HTTP/1.1
[various other headers]

<appointmentRequest>
  <slot doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointmentRequest>
```



MMR – Level 0 – «tunneling» Mecanism

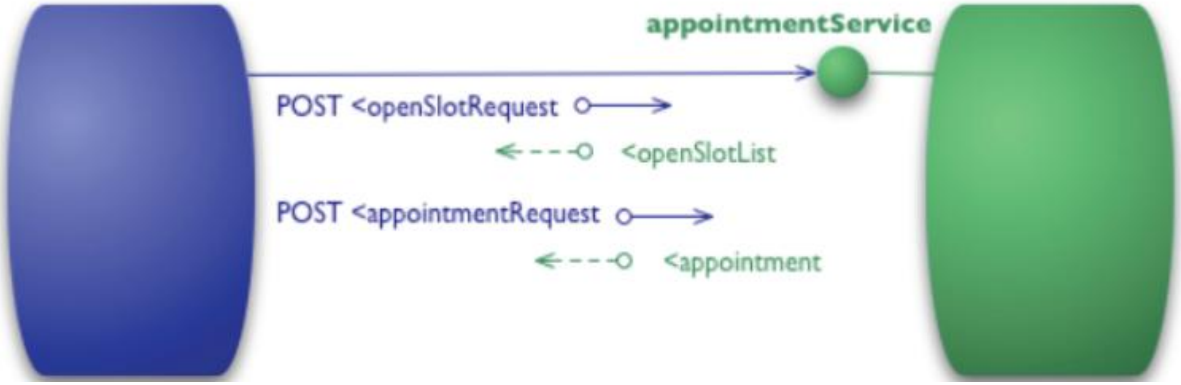
- Example: Making an appointment with the doctor
  - A unique URI **appointmentService**
  - Then take one appointment among the possible choices



```
POST /appointmentService HTTP/1.1
[various other headers]

<appointmentRequest>
  <slot doctor = "mjones" start = "1400
  <patient id = "jsmith"/>
</appointmentRequest>
```

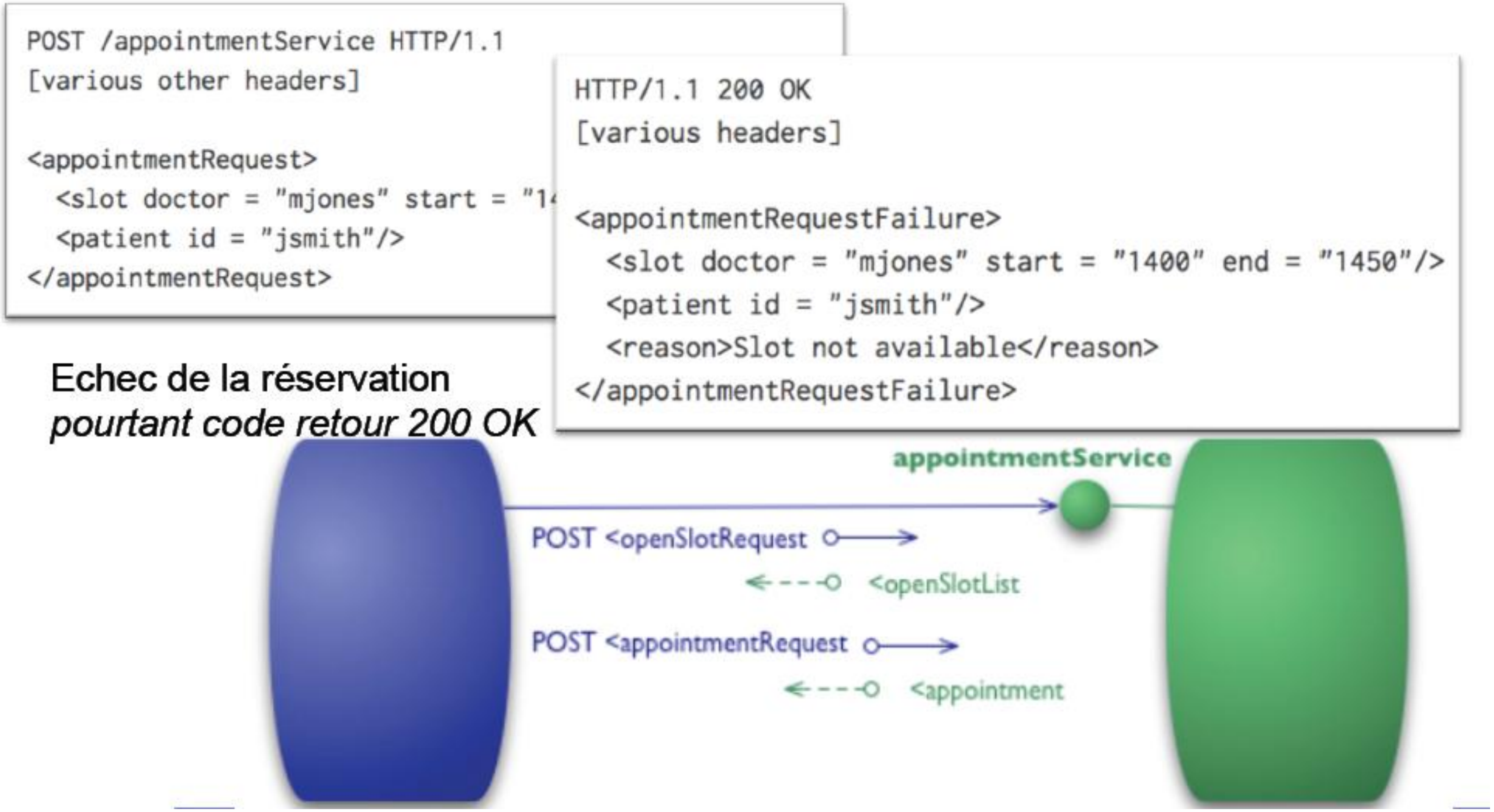
```
HTTP/1.1 200 OK
[various headers]

<appointment>
  <slot doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointment>
```

Réservation réussie



MMR – Level 0 – «tunneling» Mecanism

- Example: Making an appointment with the doctor
  - A unique URI **appointmentService**
  - Then take one appointment among the possible choices



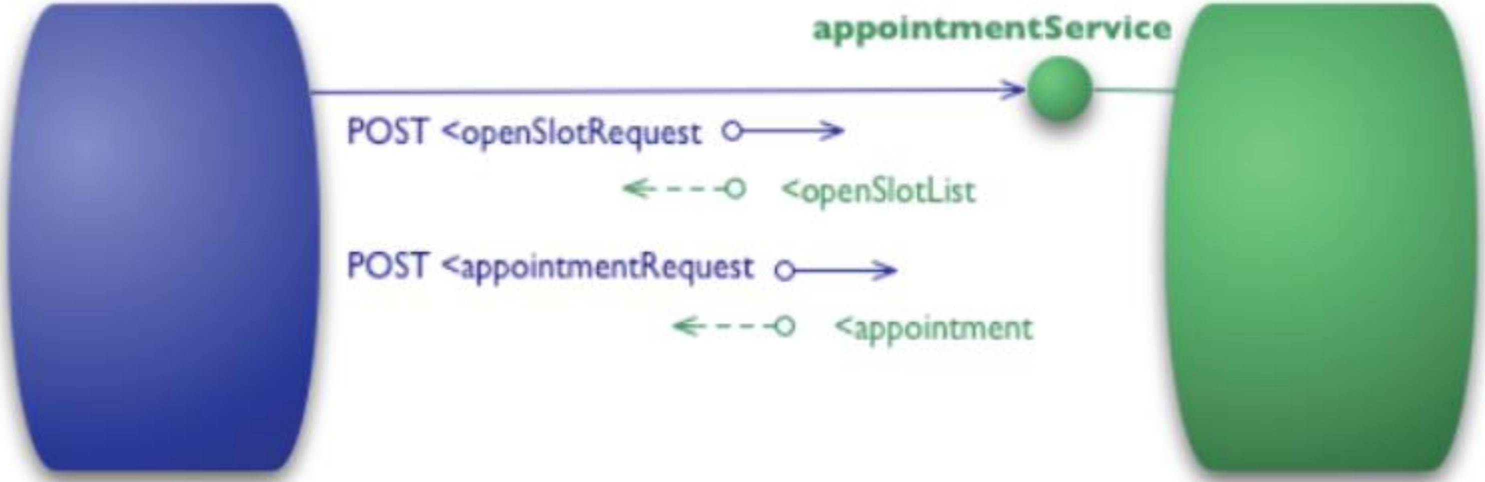
```
POST /appointmentService HTTP/1.1
[various other headers]

<appointmentRequest>
  <slot doctor = "mjones" start = "14
  <patient id = "jsmith"/>
</appointmentRequest>
```

```
HTTP/1.1 200 OK
[various headers]

<appointmentRequestFailure>
  <slot doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
  <reason>Slot not available</reason>
</appointmentRequestFailure>
```

Echec de la réservation
pourtant code retour 200 OK

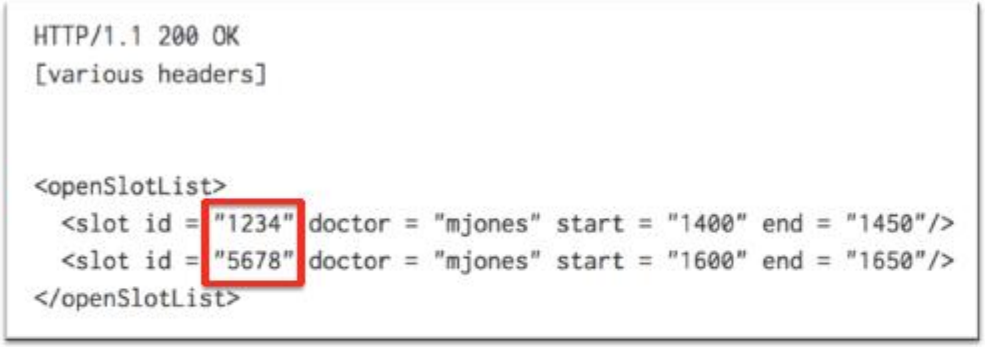
MMR – Level 0 – «tunneling» Mecanism

- HTTP as the transport system for remotely interact with a ”service”
  - Model RPC/RPI (Remote Procedure Call / Invocation)
- A unique URI
- A unique HTTP verb is used (POST) which does not distinguish the type of action to execute on the server side
  - ➔Furthermore neither safe nor idempotent ➔ no possible optimization
- We call functions 🡐RPC Approach !!!!
  - signatures and return content are in the body of messages
  - Here XML, but any other format is possible
- Idem SOAP or RPC-XML – the only difference XML + specific grammar



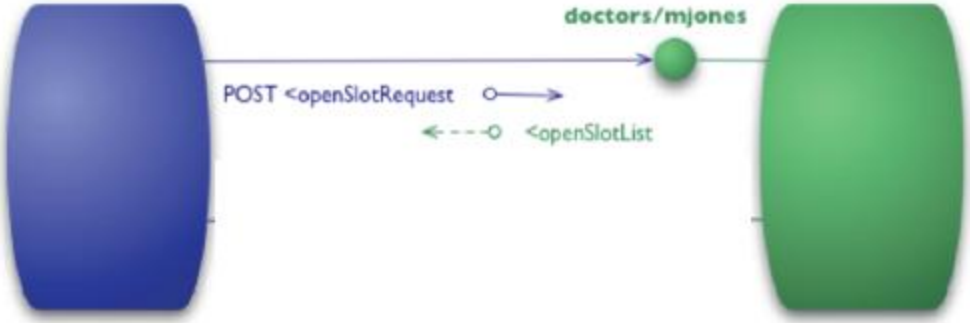
MMR – Level 1 –
The use of resources

- Distinction of several URIs but still a single verb



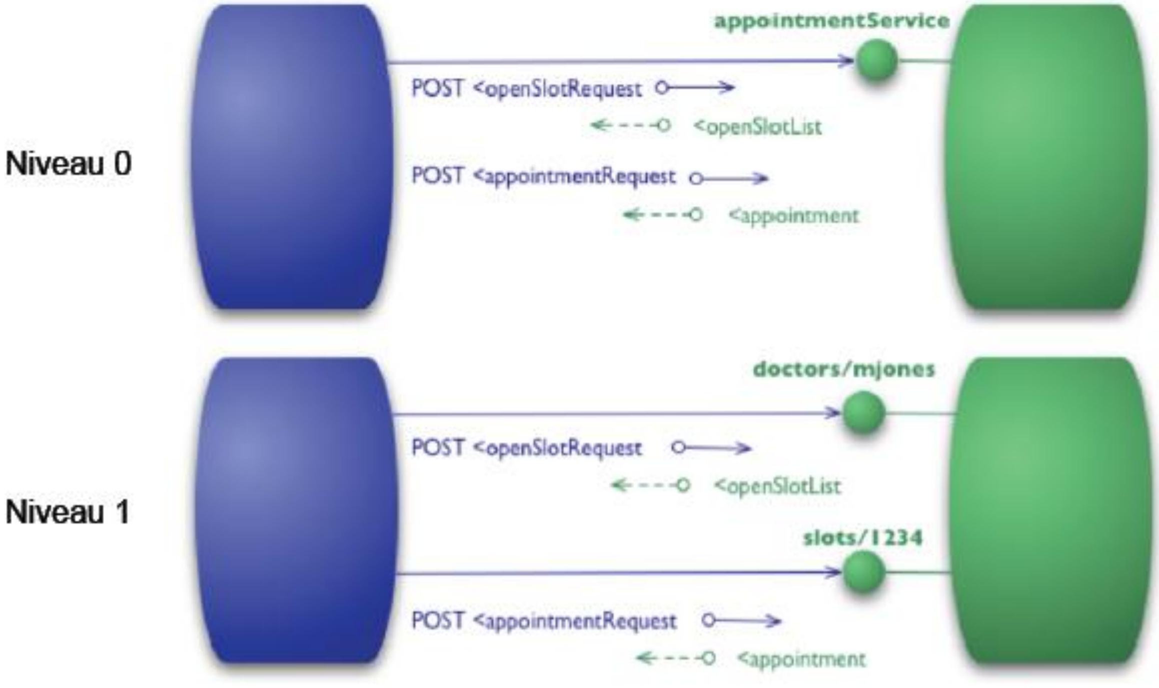
```
HTTP/1.1 200 OK
[various headers]


<openSlotList>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650"/>
</openSlotList>
```

URI partielle pour répondre



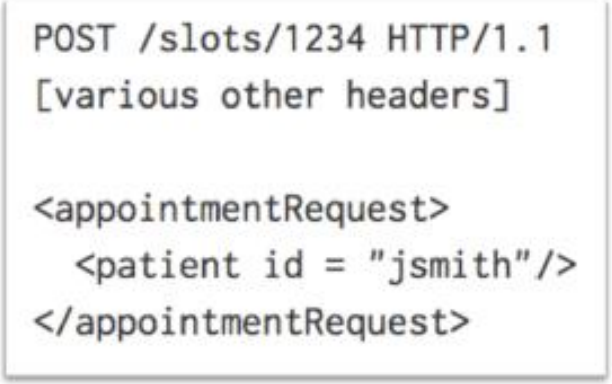
MMR – Level 1 –
The use of resources

- Distinction of several URIs but still a single verb

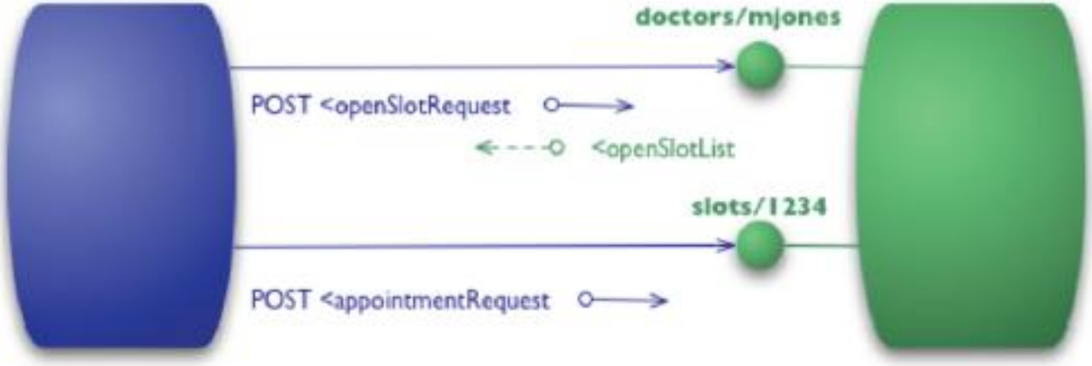


MMR – Level 1 –
The use of resources

- Distinction of several URIs but still a single verb

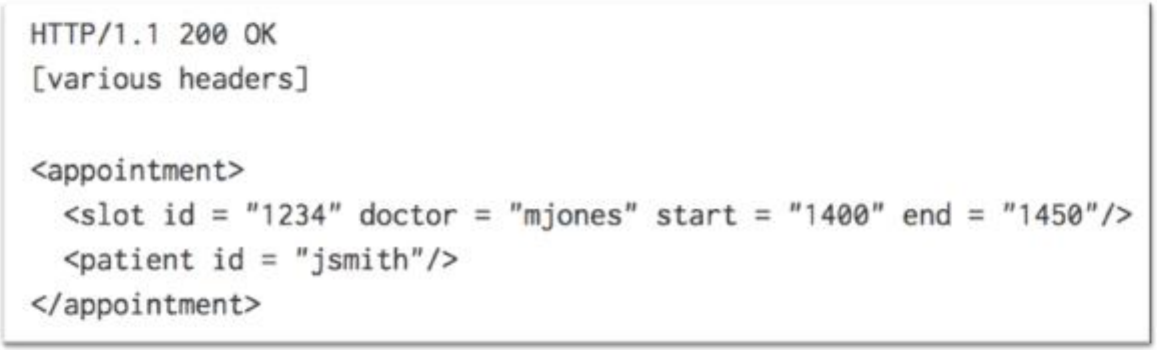


```
POST /slots/1234 HTTP/1.1
[various other headers]

<appointmentRequest>
  <patient id = "jsmith"/>
</appointmentRequest>
```

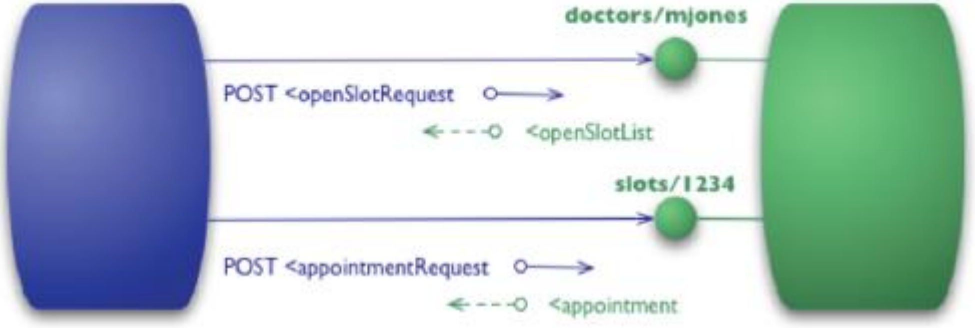


MMR – Level 1 – The use of resources

- Distinction of several URIs but still a single verb

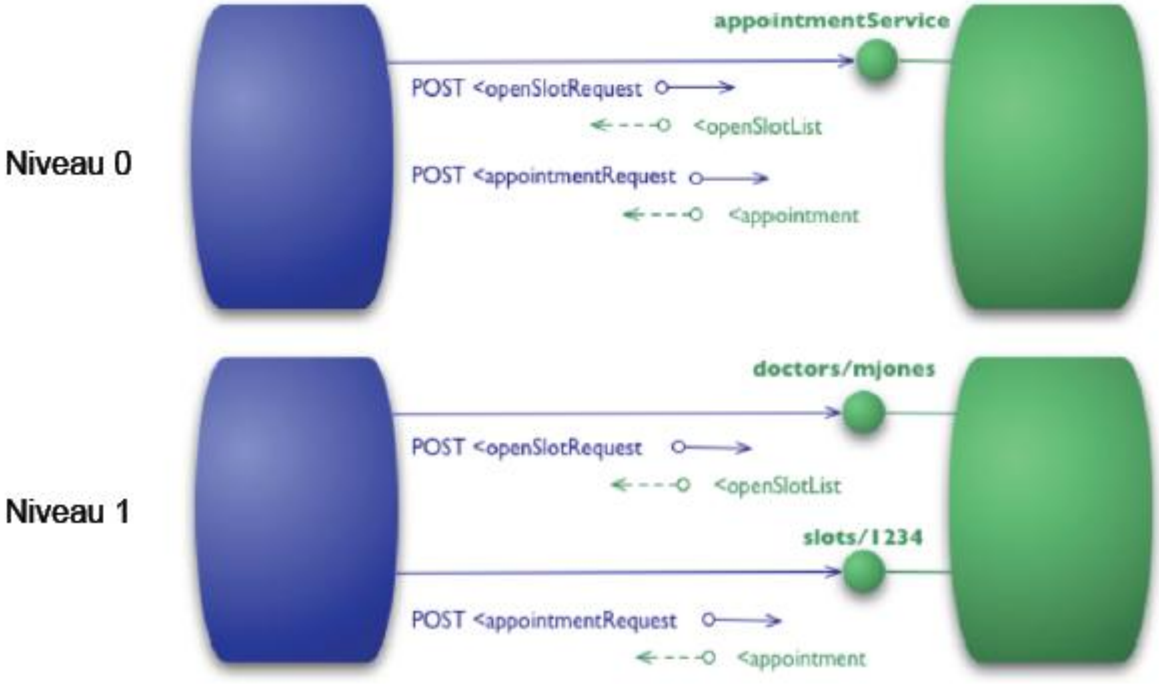


```
HTTP/1.1 200 OK
[various headers]

<appointment>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointment>
```



MMR – Level 1 – The use of resources

- Distinction of several URIs but still a single verb



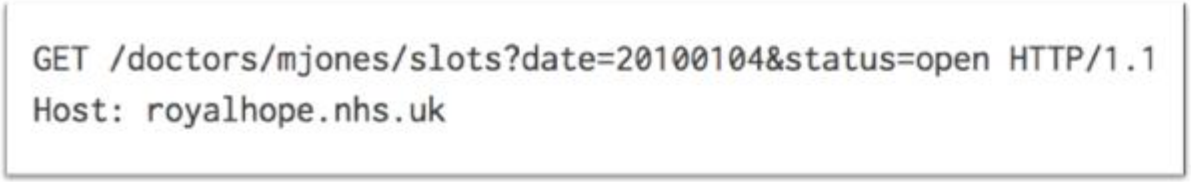
MMR – Level 1 – The use of resources

- Distinction of several URIs but still a single verb
- It is no longer what we want from the client station ➔ Start of discoverability !!
  - The server has a responsibility to indicate the resource for the future of the exchange (http://royalhope.nhs.uk/slots/1234/ par exemple)
- Introduction of the notion of identity of an object
  - We no longer call simply a function
  - We call a method on an identified resource (i.e. an object)
- Important benefits
  - Differentiation of URIs by application domain provides **semantics** to the system, which is one of the great strengths of the REST architectural style.

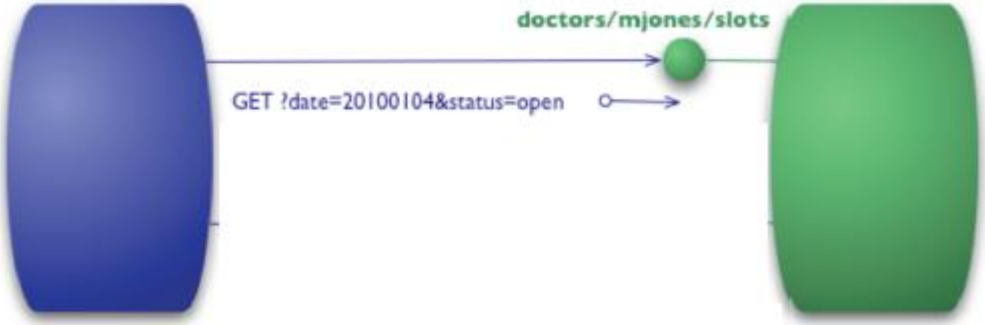
  ➔ beginnings of identifying resources

MMR – Level 2 – Verbs and HTTP return codes

- Using all HTTP verbs in compliance with their specifications
  - For our example GET and POST
  - GET is safe and idempotent for the first request

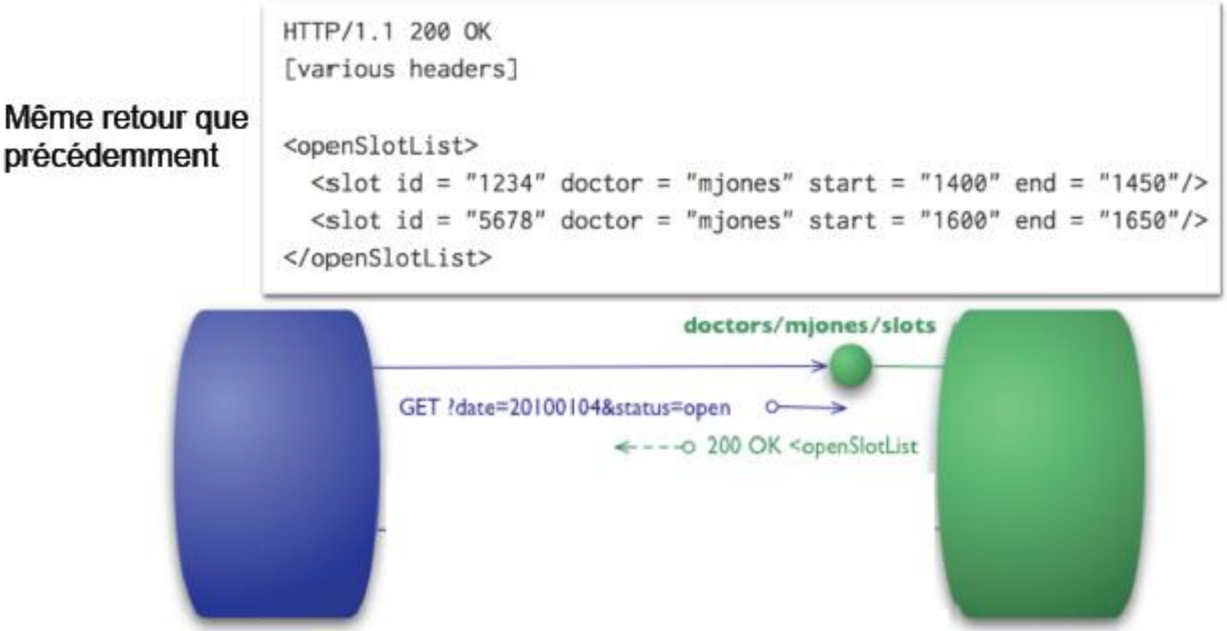


```
GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
Host: royalhope.nhs.uk
```



MMR – Level 2 – Verbs and HTTP return codes

- Using all HTTP verbs in compliance with their specifications
  - For our example GET and POST
  - GET is safe and idempotent for the first request

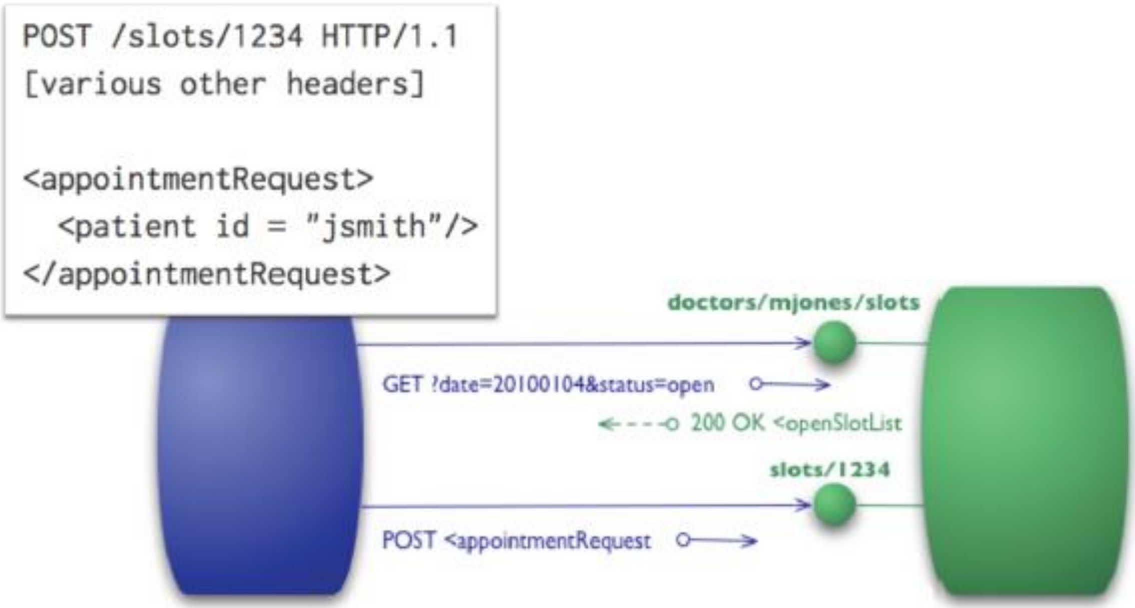


Même retour que précédemment

```
HTTP/1.1 200 OK
[various headers]

<openSlotList>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650"/>
</openSlotList>
```

MMR – Level 2 – Verbs and HTTP return codes

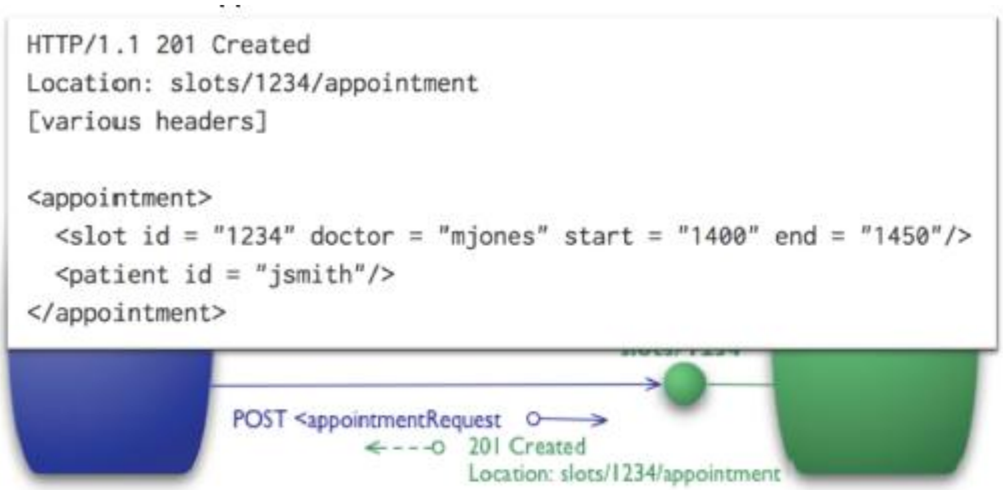
- Using all HTTP verbs in compliance with their specifications
  - For our example GET and POST
  - POST (as well as PUT) allows the state change



```
POST /slots/1234 HTTP/1.1
[various other headers]

<appointmentRequest>
  <patient id = "jsmith"/>
</appointmentRequest>
```

MMR – Level 2 – Verbs and HTTP return codes

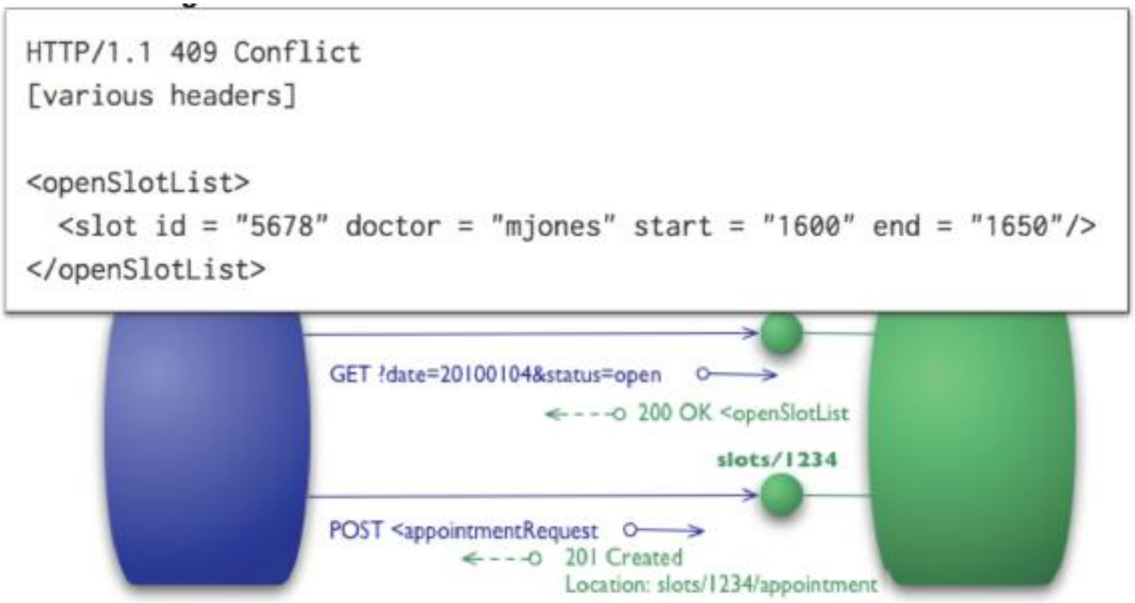
- Using all HTTP verbs in compliance with their specifications
  - For our example GET and POST
  - POST (as well as PUT) allows the state change
    - In the header: response code 201 + the URI slot accessed later to access the modification ➔ beginning of discoverability again
    - In the body: representation of the resource to prevent access for consultation on slots/1234/appointment



```
HTTP/1.1 201 Created
Location: slots/1234/appointment
[various headers]

<appointment>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointment>
```

MMR – Level 2 – Verbs and HTTP return codes

- Using all HTTP verbs in compliance with their specifications
  - For our example GET and POST
  - POST (as well as PUT) allows the state change
    - Return code change on error



```
HTTP/1.1 409 Conflict
[various headers]

<openSlotList>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650"/>
</openSlotList>
```

MMR – Level 2 – Verbs and HTTP return codes

- Using all HTTP verbs in compliance with their specifications
  - PUT and DELETE are little used in practice
- Using return codes of HTTP verbs
- important benefits
  - the semantic use of verbs and HTTP return codes enriches the protocol level between the client and the server
  - Supported by tools (browsers, firewalls, routers, etc.) as standard, so possible optimizations
  - Using the POST verb allows to clearly signify **the creation of a resource** with type appointment using the URI slots/1234/. As this is the creation of an appointment (POST verb), the part /1234 of the URI is not ambiguous for the server: This is of course of the identifier of appointment.
  - the use of HTTP return codes allows for a **clear semantics** to the client without reading the message body
    - 201 Created : the creation has succeeded
    - 404 Not Found : the customer concludes that the resource has moved / deleted

MMR – Level 3 – hypermedia controls

HATEOAS

Hypertext As The Engine Of Application State

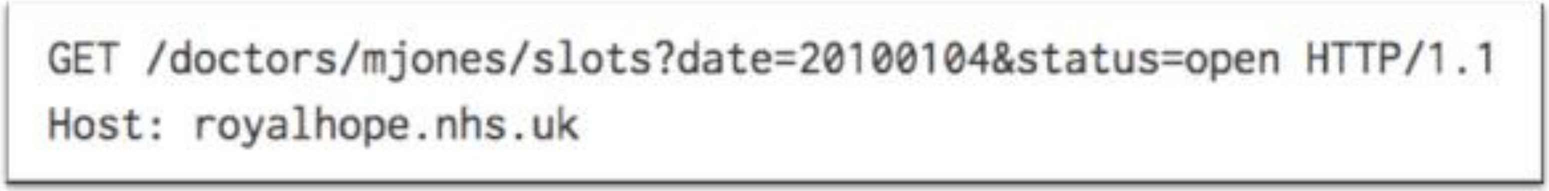
- At level 2, the client must know in advance
  - all URIs correspond to different features of the server
  - possible actions on these URIs (HTTP methods)

  the client must be aware of the possible request during its application path: he must know in advance the **possible application states** of the system.

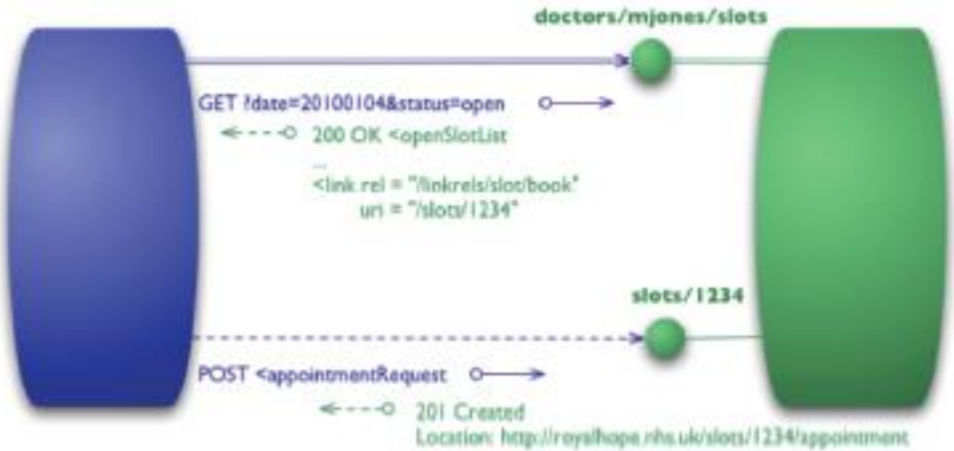
- At level 3, the client discovers step by step **what he is not allowed to do at** the application level, **thanks to hypermedia.**
  - Hypermedia links take us from an application state to another without having to know them in advance.

MMR – Level 3 – hypermedia controls

- HATEOAS – *Hypertext As The Engine Of Application State*
- Same initial request as level 2



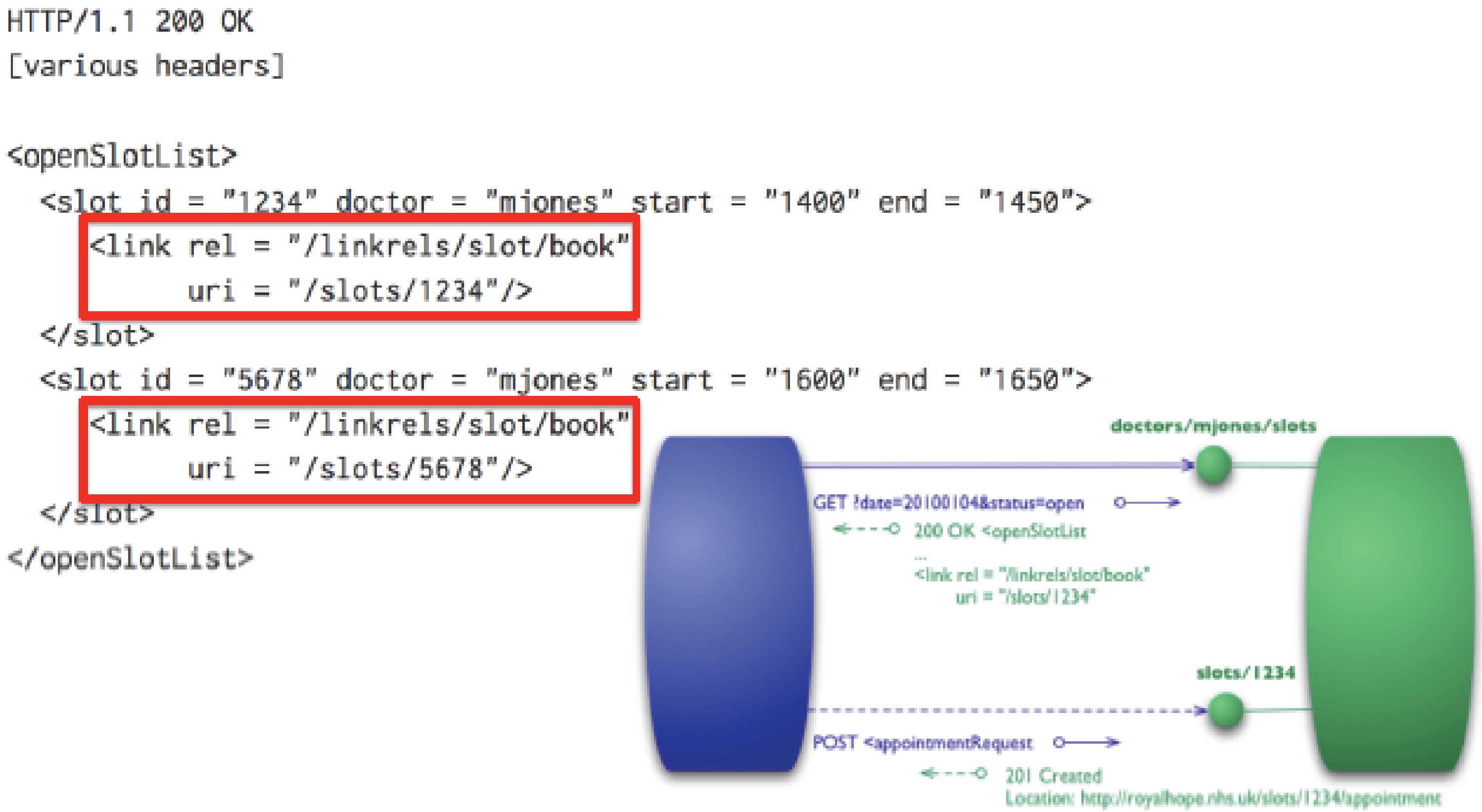
```
GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
Host: royalhope.nhs.uk
```



doctors/mjones/slots
GET ?date=20100104&status=open
200 OK <openSlotList
...
<link rel = "/linkrels/slot/book"
uri = "/slots/1234"
slots/1234
POST <appointmentRequest
201 Created
Location: http://royalhope.nhs.uk/slots/1234/appointment

MMR – Level 3 – hypermedia controls

- But different return containing "hyperlinks" to find out where to take the respective appointments

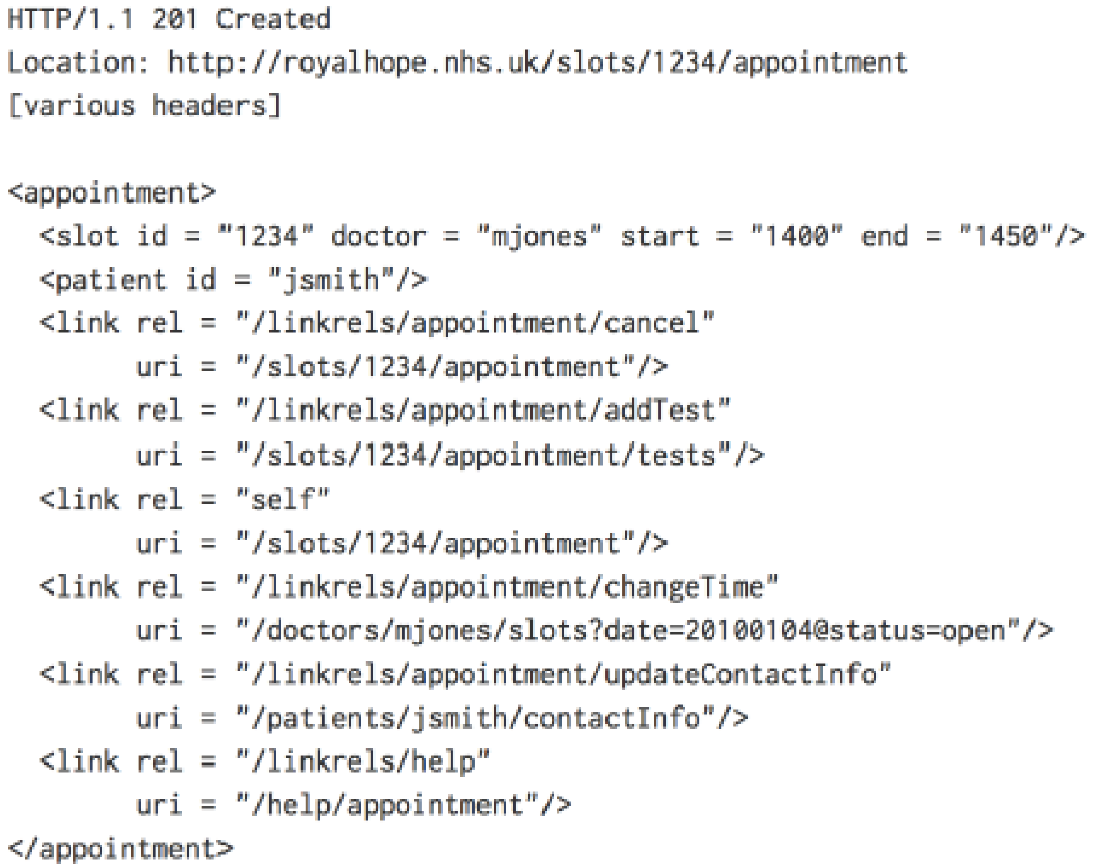


```
HTTP/1.1 200 OK
[various headers]

<openSlotList>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450">
     <link rel = "/linkrels/slot/book"
           uri = "/slots/1234"/>
  </slot>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650">
     <link rel = "/linkrels/slot/book"
           uri = "/slots/5678"/>
  </slot>
</openSlotList>
```

```
GET ?date=20100104&status=open
200 OK <openSlotList
...
<link rel = "/linkrels/slot/book"
     uri = "/slots/1234"

POST <appointmentRequest
201 Created
Location: http://royalhope.nhs.uk/slots/1234/appointment
```

MMR – Level 3 – hypermedia controls

- But different return containing "hyperlinks"



```
HTTP/1.1 201 Created
Location: http://royalhope.nhs.uk/slots/1234/appointment
[various headers]

<appointment>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
  <link rel = "/linkrels/appointment/cancel"
        uri = "/slots/1234/appointment"/>
  <link rel = "/linkrels/appointment/addTest"
        uri = "/slots/1234/appointment/tests"/>
  <link rel = "self"
        uri = "/slots/1234/appointment"/>
  <link rel = "/linkrels/appointment/changeTime"
        uri = "/doctors/mjones/slots?date=20100104@status=open"/>
  <link rel = "/linkrels/appointment/updateContactInfo"
        uri = "/patients/jsmith/contactInfo"/>
  <link rel = "/linkrels/help"
        uri = "/help/appointment"/>
</appointment>
```

MMR – Level 3 – hypermedia controls (HC)

- The CHs allow the server to change its URIs without "breaking" customers
  - weak coupling
  - Links are no longer known as "hard" by the client but provided by the server
- The links tell the client application developer opportunities ahead (but not all information)
  - The controls "latest" and "cancel" point to the same URI (respectively GET and DELETE verbs but it is not specified by the link)
- No standard yet but ATOM recommandations (RCF 4287) to define a link <link>
  - The *uri* attribute gives the address of the resource
  - The *rel* attribute describes the type of relationship

The Maturity Model of Richardson

- Level 0
  - A URI, a verb
- Level 1
  - Several URIs, a single verb
  - Uses a "divide and conquer" approach to break a single point in several
- Level 2
  - Several URIs, Several verbs
  - Introduces a standard set of verbs to use similarly in similar situations
- Level 3
  - Several URIs, Several verbs
  - Links between pages
  - Introduces discoverability and self-documentation exchange protocols

Part 3

REST and ressource-oriented approach

A word on the REST Web services

- Exploited for Data-Oriented Architectures (DOA)
- REST is not a standard, there is no W3C specification defining a specification
- REST is an architectural style based on a mode of understanding the web
- REST is based on web standards:
  - HTTP protocol
  - URIs
  - File Formats
  - Secure SSL

Ressource-oriented approach or REST

- REST is an acronym for **REpresentational State Transfert**
- The principles were defined in the thesis of Roy FIELDING in 2000
  - One of the main authors of the HTTP specification
  - Founding member of the Apache Foundation
  - Apache web server developer
- **REST is an architectural style** inspired from web architecture
- So it's
  - A way to build an application
- And it isn't
  - A format, a protocol, a standard

What is REST?

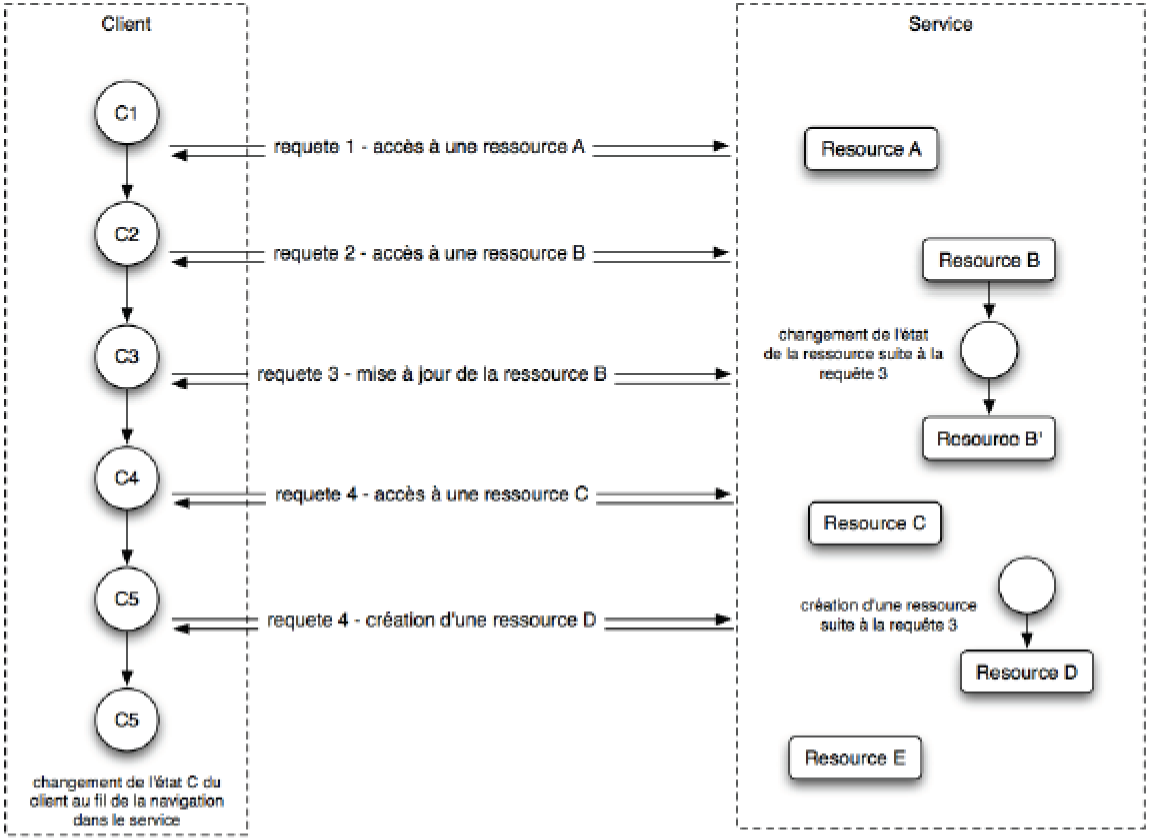
- REST Web Services are used to develop resource-oriented architectures
- Different denominations available in the literature
  - Data Oriented Architecture (DOA)
  - Resource Oriented Architecture (ROA)
- Applications that meet the resource-oriented architectures are named respectively ***RESTful***
- In the rest of the course we indiscriminately use the name ***REST*** and ***RESTful***

Application state type client – server (1/2)

- There exist 2 types of states
  - **The state of the resource is information relating to the own resource,** so the service (eg BD)
  - **The state of the application** is the information for each customer so related to each "user".

➔ It is this state which we want to be independent

Application state type client – server (2/2)


Client
C1
C2
C3
C4
C5
C5
changement de l'état C du client au fil de la navigation dans le service
requete 1 - accès à une ressource A
requete 2 - accès à une ressource B
requete 3 - mise à jour de la ressource B
requete 4 - accès à une ressource C
requete 4 - création d'une ressource D
Service
Resource A
Resource B
changement de l'état de la ressource suite à la requête 3
Resource B'
Resource C
création d'une ressource suite à la requête 3
Resource D
Resource E

REST Characteristics

- REST Web services are stateless
  - The server never has to know the client's condition and vice versa
  - The client maintains the state of the implementation of its view
  - The server does the same maintaining the state of its resources

    ➔ These states are never shared!!!

  - Any change of state occurs as a result of an exchange of messages between client and server ➔ transfer of representations
  - Since server and client do not know their respective states
    - Each request sent to the server must contain all the information necessary for its treatment
    - The server does not store customer information

    ➔ Minimizing system resources, no session neither state

REST Characteristics

- REST Web services provide a uniform interface based on HTTP methods
  - GET, POST, PUT et DELETE
- REST-oriented architectures are built from resources that are uniquely identified by URIs

Recommendation:
Treatments must be installed client side,
not server side

The resource-oriented architecture based on 3 concepts …

- **Resource (Identifier)**
  - Identified by a URL
  - Example : http://localhost:8080/libraryrestwebservice/books
- **Method (Verb)**
  - Allows you to manipulate the username or resource
  - Method HTTP : GET, POST, PUT and DELETE
- **Representation gives a view of a resource**
  - We often talk about the view of the state of a resource
  - This is the information transferred between the client and the server
  - Example : XML, JSON

… and 4 properties

- The representation must be addressable
- Services must be stateless
- Services / Resources must be connected
- The services respect an interface standard (Uniform Interface, or UI)

Resources and URI (1/3)

- A resource is something that is identifiable in a system
  - Personne, Agenda, Document, set, map

- http://cours-rest.fr/api?method=findStudent&userid=nLegoff&sessionid=06102015
- http://cours-rest.fr/students/nicolas-legoff

Resources et URI (1/3)

- A resource is something that is identifiable in a system
  - Personne, Agenda, Document, Ensemble, Carte

Bad URI

- http://cours-rest.fr/api?method=findStudent&userid=nLegoff&sessionid=06102015

Good URI

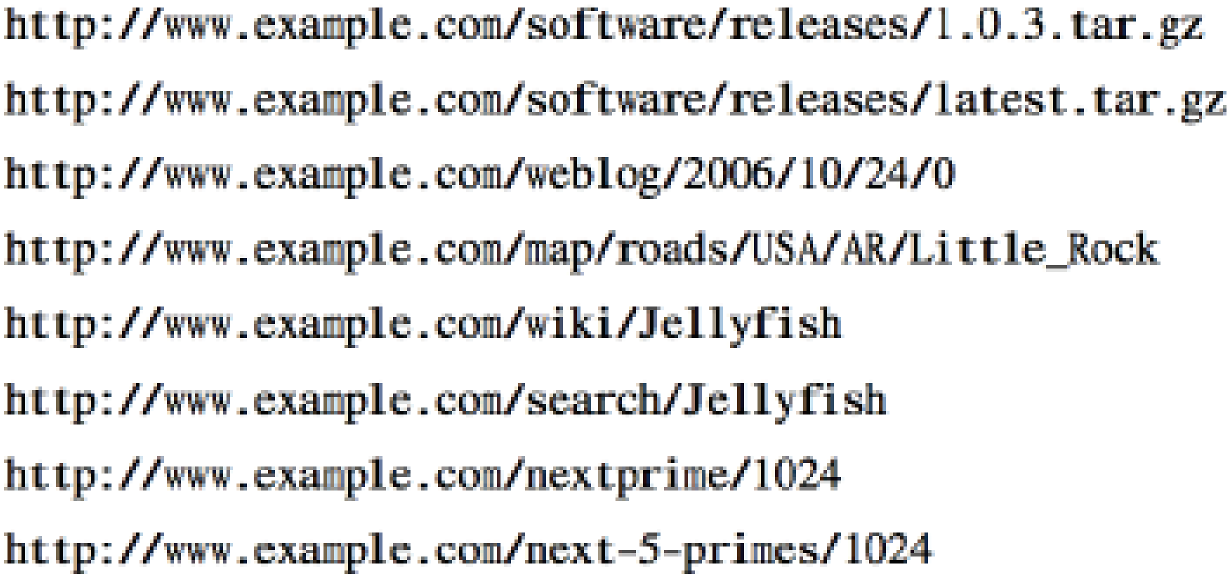
- http://cours-rest.fr/students/nicolas-legoff

From an architectural point of view

- First solution = implementation choice
  - Here the method call to a remote service
  - HTTP used simply as message transport only
- Second solution
  - Less impression to invoke a remote operation
  - URL reflecting a concept: a student
  - AND no action

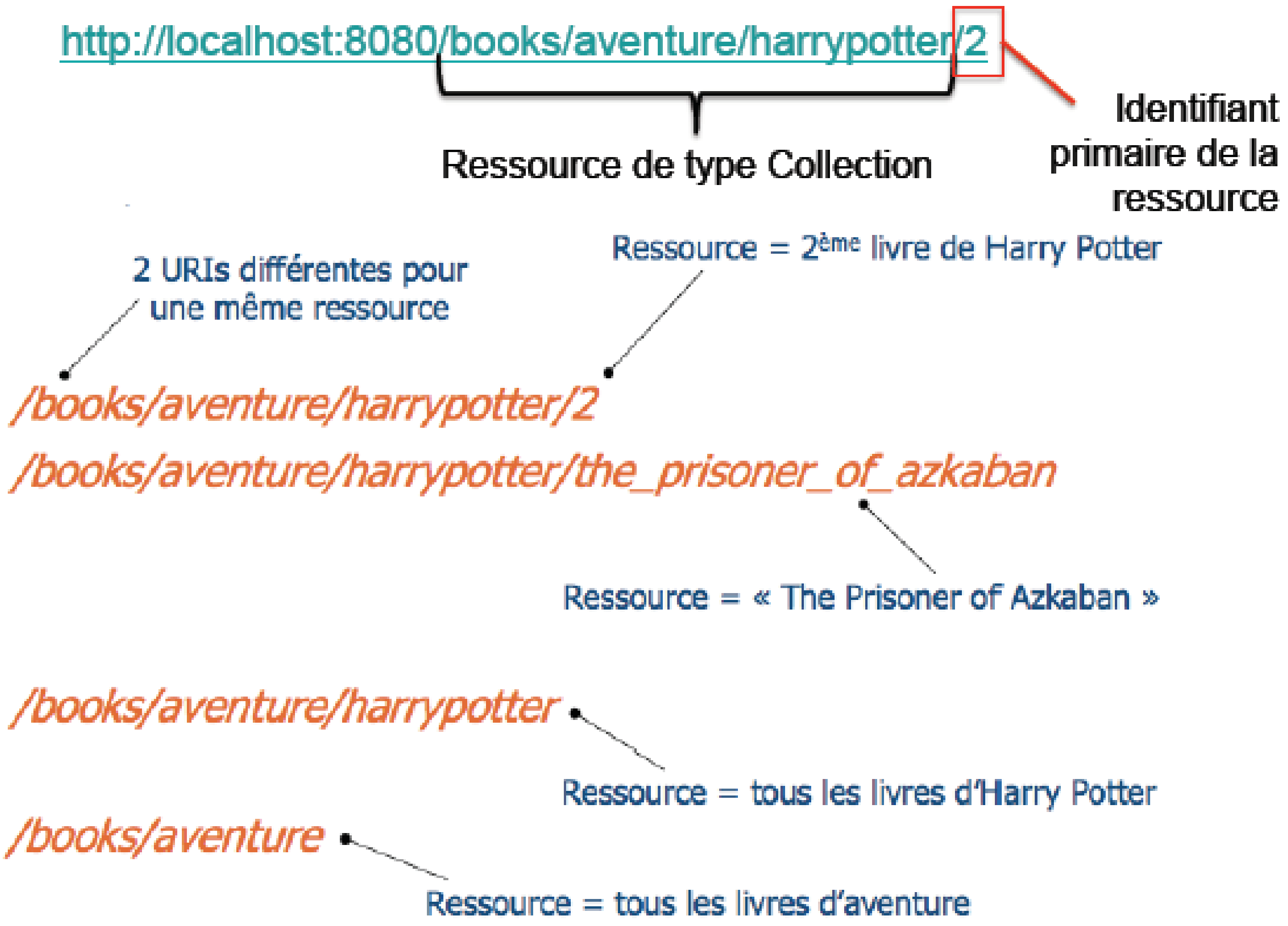
Resources et URI (2/3)

- A resource is something that is identifiable in a system
  - Personne, Agenda, Document, Ensemble, Carte
- A URL uniquely identify a resource on the system
  - Attention, resource can have multiple URIs
  - The representation of the resource is not related to the URI, it can change over time and the customer
  - A URI should be **descriptive** (***Fielding*** thesis, **W3C recommendations**)
  - A URI must have **a structure**



```
http://www.example.com/software/releases/1.0.3.tar.gz
http://www.example.com/software/releases/latest.tar.gz
http://www.example.com/weblog/2006/10/24/0
http://www.example.com/map/roads/USA/AR/Little_Rock
http://www.example.com/wiki/Jellyfish
http://www.example.com/search/Jellyfish
http://www.example.com/nextprime/1024
http://www.example.com/next-5-primes/1024
```

Resources et URI (3/3)



http://localhost:8080/books/aventure/harrypotter/2

Ressource de type Collection

Identifiant primaire de la ressource

2 URIs différentes pour une même ressource

Ressource = 2ème livre de Harry Potter

/books/aventure/harrypotter/2

/books/aventure/harrypotter/the_prisoner_of_azkaban

Ressource = « The Prisoner of Azkaban »

/books/aventure/harrypotter

Ressource = tous les livres d'Harry Potter

/books/aventure

Ressource = tous les livres d'aventure

Methods CRUD

- A resource may undergo four basic operations referred to as **CRUD**
  - ***Create –***
  - ***Retrieve –***
  - ***Update –***
  - ***Delete –***
- REST uses HTTP to directly express these four basic operations via HTTP methods
  - ***Create by the method POST***
  - ***Retrieve by the method GET***
  - ***Update by the method PUT***
  - ***Delete by the method DELETE***

➔ Few verbs to be standard, interoperable

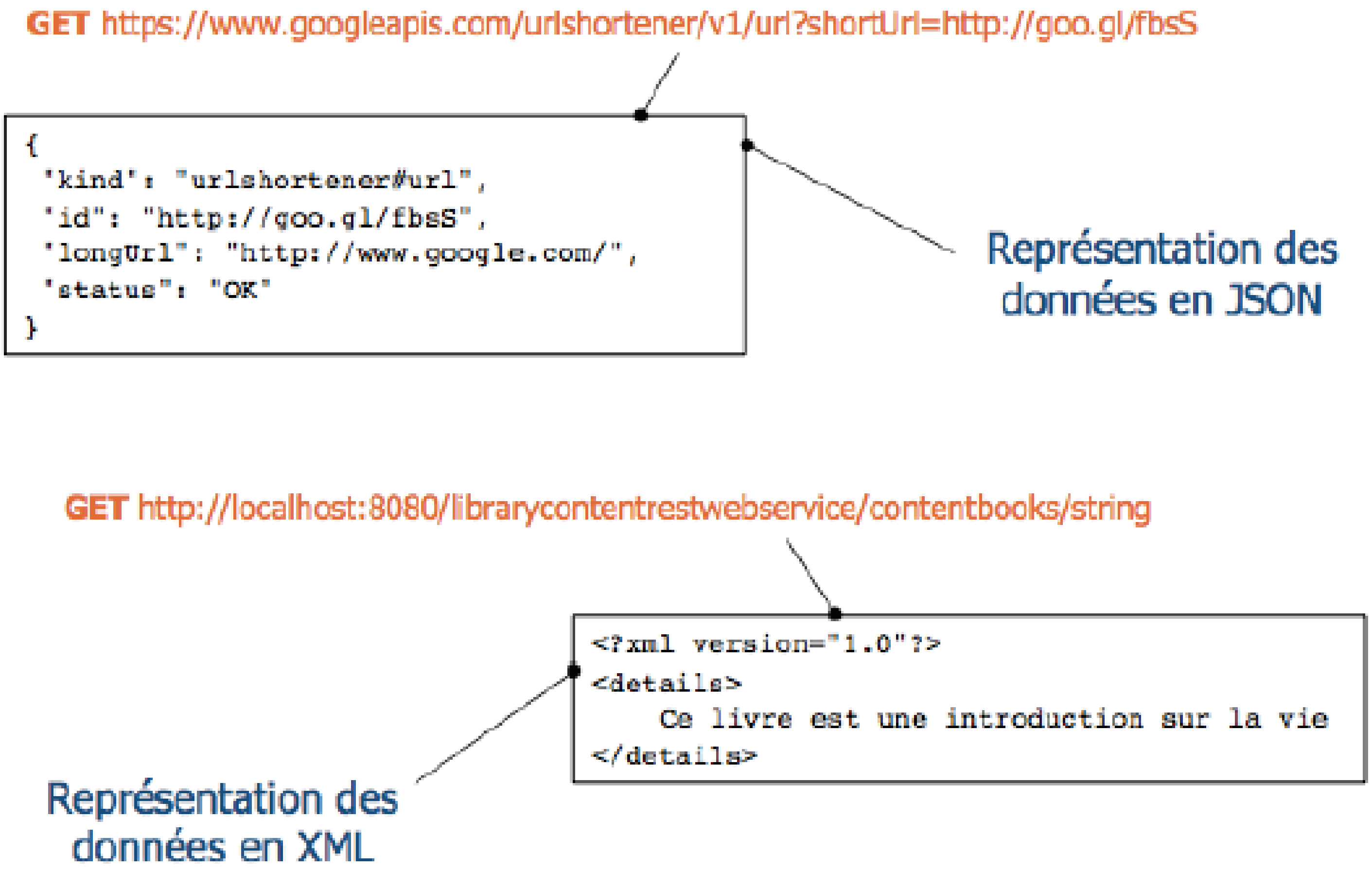
- Additional opportunities can be expressed through other HTTP methods (HEAD, OPTIONS)

The representation (1/2)

- Provide the data according to a representation for
  - The client (GET)
  - The server (PUT andPOST)
- The data returned in different formats
  - XML
  - JSON
  - (X)HTML
  - CSV
  - …
- The input format (POST) and the output format (GET) of a Web service of a resource can be different

The representation(2/2)

- Examples : JSON and XML formats



GET https://www.googleapis.com/urlshortener/v1/url?shortUrl=http://goo.gl/fbsS

```
{
 'kind': "urlshortener#url",
 'id": "http://goo.gl/fbsS",
 'longUrl": "http://www.google.com/",
 'status": "OK"
}
```

Représentation des données en JSON

GET http://localhost:8080/librarycontentrestwebservice/contentbooks/string

```
<?xml version="1.0"?>
<details>
    Ce livre est une introduction sur la vie
</details>
```

Représentation des données en XML

The resource-oriented architecture based on 4 properties

- The representation must be addressable
- Services must be stateless
- Services / Resources must be connected
- The services respect an interface standard (Uniform Interface, or UI)

Property 1 - A representation should be addressable

- A web service is addressable since it exposes some of its data in a visible resources
  - cf. annotation @Path java classes visible in Jersey
- A URL should never represented more than one resource (otherwise no more of universality)

Example : A resource accessible in English and French

- A frequently used solution **URI ➔a representation**
  - www.mylibrary/2012/books/en 🡐 a representation in English
  - www.mylibrary/2012/books/fr 🡐 a representation in French
- Other solution
  - www.mylibrary/2012/books/ 🡐 a unique URI
  - Both performances still exist (2 GET methods annotated with different @Produces)
  - The customer choose with the **Accept-Language of the header** of the query

The two solutions are RESTful. It only deals with URI, representation and all happening in the header of the request

Property 2 - Stateless service (1/7)

Any HTTP request must run in a completely isolated way

- When a client makes an HTTP request,
  - All necessary information for the execution of the request by the server are sent to the server
  - The server never reuses information from previous queries
- In practice, information is transfered via the addresses (URIs)

Property 2 - Stateless service (2/7)

Be stateless

- A Web application must scale up
  - Server clusters with load balancing management, proxies, of input points form topologies enable applications to move between servers ← **in order to reduce the response time to the client**
  - This means you can transfer independent and comprehensive queries, ie queries freestanding → **the state should not be specific to the server**
  - A self-supporting request must not therefore store / use any information on the server peculiar to itself
- A REST Web service included in the header and body of the HTTP request all that is needed to operate the called service
  - Settings, context, necessary data to server

- Being stateless simplifies the design and implementation of server-side services because the lack of state removes the need to synchronize data from the session with an external application.

Property 2 - Stateless service (3/7)

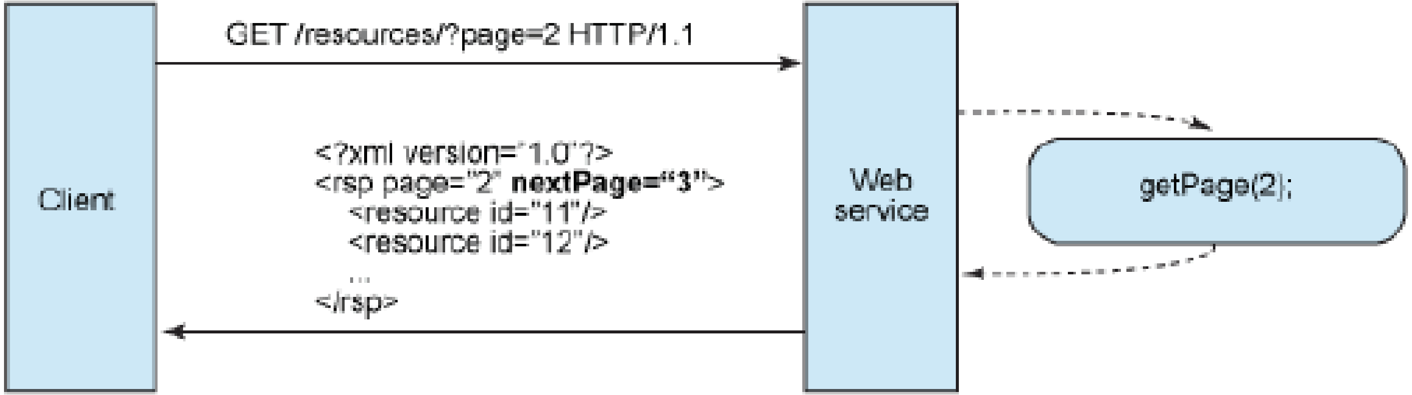
Example of a solution with state

- An application requests a "next page" in a set of several result pages
- With the concept of state
  - It is assumed that the service keeps trace of the last page visited
  - For this, the increment service and maintains a variable **previousPage** to pass then to the next page.
- Such a variable is problematic to update among several Java servers (EJC or servlet/Java Server Pages for eg. With j**ava.io.NotSerializableException** during session replication.
- In addition, the synchronization sessions adds additional costs impacting server performance.
- Finally, what about the idempotence??

Property 2 - Stateless service (4/7)

Exempleof a Solution a stateless

- In the case of a web service REST, the server is only responsible for generating responses
- The client manages the state of the application itself
- In our example, it is the customer that indicates the page he wants and not the server that has the knowledge !!!
- In addition, the web service indicating the next page in the response to the client and allows the customer to manage this value



```
GET /resources/?page=2 HTTP/1.1
```

```
<?xml version='1.0'?>
<rsp page="2" nextPage="3">
    <resource id="11"/>
    <resource id="12"/>
    ...
</rsp>
```

Property 2 - Stateless service (5/7)

Good practices

- **Server Side**
  - Generates responses that include links to other resources to permit client applications to navigate to these resources
  - Generates responses which indicate whether they are "cacheables" or not to improve performance by reducing the number of requests per resource duplication or complete elimination request)
    - **Cache-Control and Last-Modified (date value) HTTP header.**
- **Client Side**
  - Use the **Cache-Control header value of** the response to determine if the answer can be copied locally or not
  - The customer also read the **Last-Modified header value** of the response and returns the value **if If-Modified-Since header value has changed (*called GET conditional)***
    - Code 304 (Not Modified) indicates that the current resource has not changed
    - The client can use his local copy of the resource so that the resource is updated
  - Sending Freestanding queries.

Property 2 – Stateless service (6/7)

- A stateless service influence only one type of states
- To remember, there are 2 types of states in a REST service
  - **The resource status is the information related to the resource**
  - **The state of the application is the information related to each customer**
- The state of the application may appear when you does not expect it
  - Various websites create unique keys for each registered user
  - This key is sent with each request (limiting the number of request per day / right of access)

Property 2 – Stateless service (7/7)

The importance of being stateless

Moves to scale by

- Putting resource in cache
- By separating the requests to be processed across multiple servers
  - If a server can not handle everything, since services are independent, without states, we distribute them among different servers (balancing randomly load, round-robin, etc.)
  - If both servers are not enough, we add a third, etc.
  - If a server fails, the others substitute it (fault tolerance)
- No replication of the application state

Property 3 – The resources should be connected

- The server may guide the client from a ressource to another by sending links to other resources in responses according to requests
  - Hypermedia as the engine of application state (Fielding’s PHD thesis)
- It is the case of « human web » where links to other pages are present in almost all web pages
- In constrast « programmable web» is hardly connected

Property 4 – The UI (Uniform Interface) is respected

- All the interactions between the client and the server passes through the UI
  - GET
  - HEAD
  - PUT
  - DELETE
  - POST
    - OPTIONS
- If you ever want another operation, then overload the POST operation
- But it is probably rrather than a new resource to add
- Importance of safety and idempotence
  - Warning not to POST
- Importance of using the same interface than everyone with the same operation semantic !!!