almerys
innover pour la vie

REST

Cours Web Services ISIMA 3F3

Olivier COUPELON -
coupelon@isima.fr

Technologies de communication inter-applications

- API : on embarque directement l'application à appeler (dépendance directe, JAR)
- EJB : on appelle l'application avec un protocole de niveau transport (TCP/IP) ou directe (si même JVM)
- REST : échange niveau HTTP, on exploite à 100% le protocole
- SOAP : au dessus d'HTTP, on encapsule les messages

REST

Representational **S**tate **T**ransfer

Roy Fielding
http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

→ Exploitation de la sémantique du Web, de la mécanique HTTP

Principe

On utilise les méthodes HTTP sur les ressources exposées afin d'effectuer toutes les opération voulues. On est donc toujours sans état.

Technologies utilisées

- HTTP
- URI
- XML / JSON (représentation des ressources)
- Types MIME

XML vs JSON

- XML est standard
- Manipulation aisée
- Verbeux

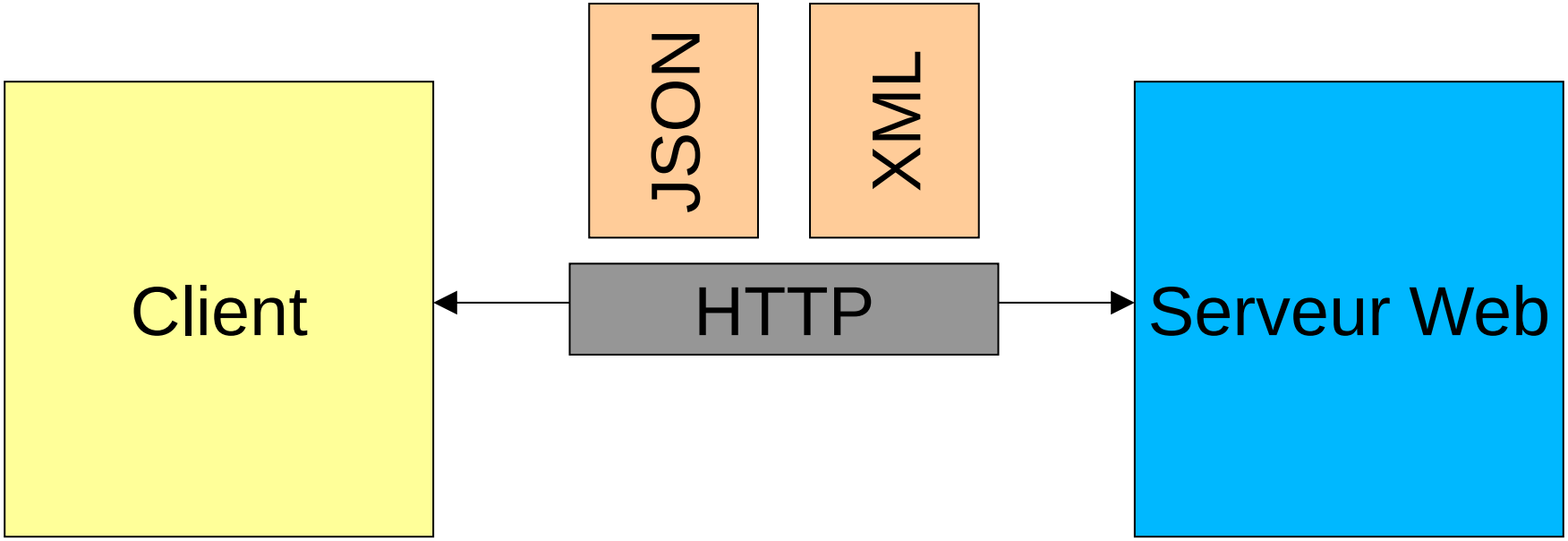
- JSON est crée pour les navigateurs web
- Interprété nativement, donc performant
- Supporté par la plupart des langages moyennant une API

Utilisation de REST

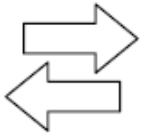
Paul Downey, BT

almerys

Communication REST



CRUD

CRUD	REST	
CREATE	POST 	Create a sub resource
READ	GET 	Retrieve the current state of the resource
UPDATE	PUT 	Initialize or update the state of a resource at the given URI
DELETE	DELETE 	Clear a resource, after the URI is no longer valid

Exemple : Création d'un sondage

POST /sondage

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

201 Created

```
Location:
/sondage/672609683
```

GET /sondage/672609683

200 OK

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

Exemple : Vote au sondage

POST /sondage/672609683

```
<name>O. Coupelon</name>
<choice>B</choice>
```

201 Created

```
Location:
/sondage/672609683/vote/1
```

GET /sondage/672609683

200 OK

```
<options>A,B,C</options>
<votes>
        <vote id='1'>
                <name>O.
Coupelon</name>
                <choice>B</choice>
        </vote>
</votes>
```

Exemple : Modification d'un sondage

PUT /sondage/672609683/vote/1

```
<name>O. Coupelon</name>
<choice>C</choice>
```

200 OK

GET /sondage/672609683

200 OK

```
<options>A,B,C</options>
<votes>
        <vote id='1'>
                <name>O. Coupelon</name>
                <choice>C</choice>
        </vote>
</votes>
```

almerys

Exemple : Suppression d'un sondage

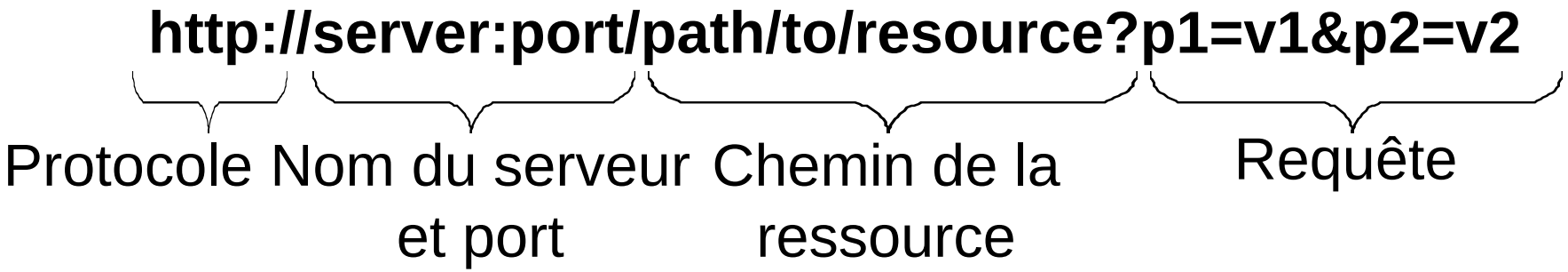
DELETE /sondage/672609683

200 OK

GET
/sondage/672609683

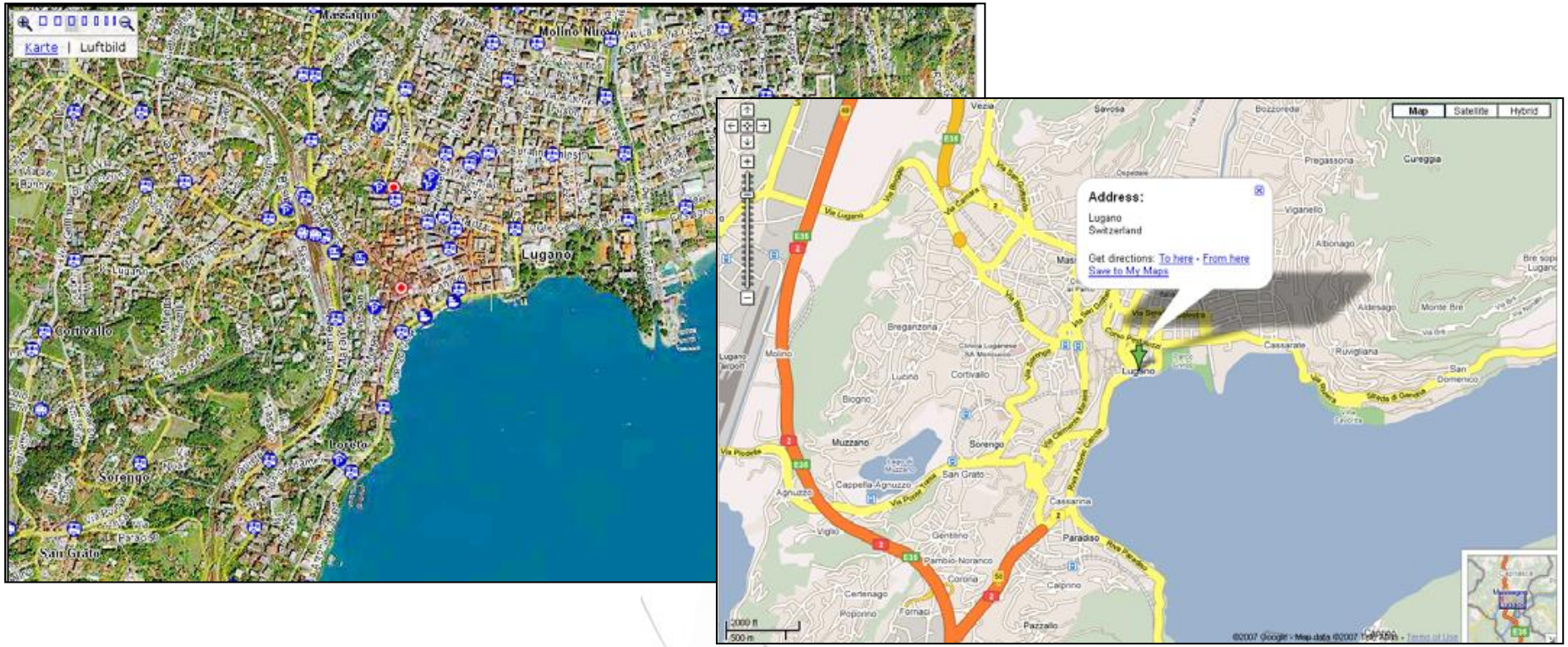
404 Not Found

URL


http://server:port/path/to/resource?p1=v1&p2=v2
Protocole
Nom du serveur et port
Chemin de la ressource
Requête

URL propre

http://map.search.ch/lugano



http://maps.google.com/maps?f=q&hl=en&q=lugano,+switzerland&layer=&ie=UTF8&z=12&om=1&iwloc=addr

URL propre – Bonnes pratiques

- Utiliser le chemin de la ressource pour spécifier les paramètres
- Une fois définie, une URL ne doit plus changer

- REST utilise des URL opaques : HATEOAS

  Hypermedia as the Engine of Application State

  – Concept unique, relativement aux autre protocoles
  – Un client n'a besoin d'aucune information pour dialoguer avec le service.
  – Concept au cœur de la thèse de Roy Fielding

Création d'une ressource REST

1. Identification des ressources
2. Définition d'url propres
3. Pour chaque url, prévoir les actions pour GET, POST, PUT & DELETE
4. Documenter chaque ressource
5. Définir les liens entre ressources
6. Déployer et tester chaque ressources sur un serveur web

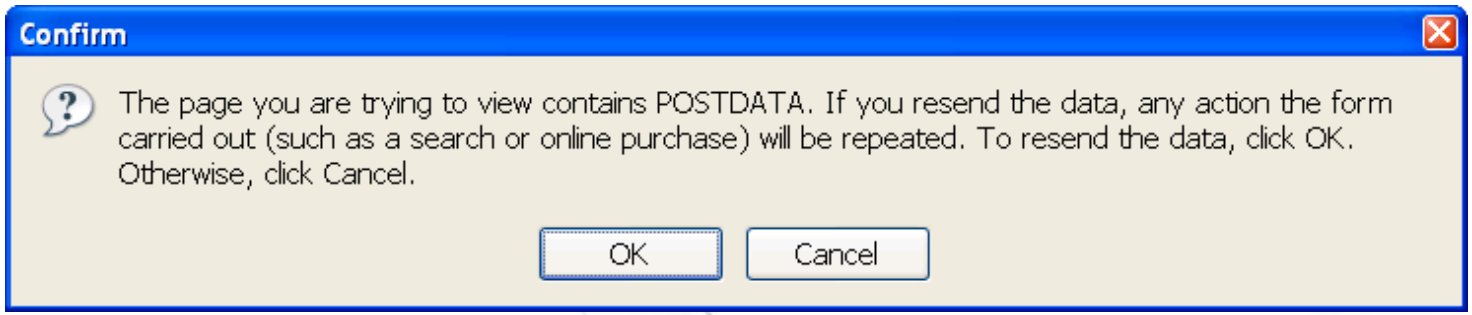
Comment gérer le versionning ?

Bonnes pratiques - GET

- Operation de lecture
- Idempotente
- Possibilité de cache

Bonnes pratiques - POST

- Operation de création
- Effet de bord possible sur le serveur



Bonnes pratiques – POST vs PUT

- Identifier une ressource créée

- Préferer

  POST /sondage/672609683
  201 CREATED
  `Location: /sondage/672609683/vote/1`

- À

  PUT /sondage/672609683/vote/1
  200 OK

Négociation de contenu

On peut ajouter en entête dans la requête le type de contenu qu'on souhaite recevoir, par ordre de référence :

GET /resource
Accept: text/html, application/xml, application/json

200 OK
Content-Type: application/json

Retours d’information

Toujours respecter les codes retour HTTP.

Doivent être interprétés correctement par le **client**, et émis correctement par le **serveur**.

WADL

Web **A**pplication **D**escription **L**anguage

Interface décrivant les opérations disponibles d'un service REST, ses entrées/sorties…

Créé automatiquement par Jersey

```
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://research.sun.com/wadl/2006/10 wadl.xsd"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:ex="http://www.example.org/types"
     xmlns="http://research.sun.com/wadl/2006/10">

    <resources base="http://www.example.org/services/">
        <resource path="getStockQuote">
            <method name="GET">
                <request>
                    <param name="symbol" style="query" type="xsd:string"/>
                </request>
                <response>
                    <representation mediaType="application/xml"
                       element="ex:quoteResponse"/>
                    <fault status="400" mediaType="application/xml"
                       element="ex:error"/>
                </response>
            </method>
        </resource>
    </resources>
</application>
```

RSDL (RESTful Service Description Language)

- Description des points d'entrée des services (URI) uniquement

```
<link rel="get" href="/api/clusters">
    <request>
        <http_method>GET</http_method>
        <headers>
            <header required="false">
                <name>Filter</name>
                <value>true|false</value>
            </header>
        </headers>
        <url>
            <parameters_set>
                <parameter context="query" type="xs:string" required="false">
                    <name>search</name>
                    <value>search query</value>
                </parameter>
                <parameter context="matrix" type="xs:boolean" required="false">
                    <name>case_sensitive</name>
                    <value>true|false</value>
                </parameter>
                <parameter context="matrix" type="xs:int" required="false">
                    <name>max</name>
                    <value>max results</value>
                </parameter>
            </parameters_set>
        </url>
        <body/>
    </request>
    <response>
        <type>Clusters</type>
    </response>
</link>
```

Gestion des états dans REST

HTTP est Stateless

Ajout d'information dans les ressources retournées représentant les **transitions valides**

Technique HTTP classiques de sessions

Sécurité

Utilisation de la sécurité HTTP

TLS

Authentification

Asynchronisme

- HTTP est un protocole synchrone
- On peut simuler une file simplement :

- POST /queue
  202 Accepted
  `Location: /queue/message/1`

- GET /queue/message/1

Use case REST

- Surtout utilisé pour exposé des services non authentifié sur le web.
- Utilisé pour ses performances
- Utilisé pour des services orienté utilisateur final
- Utilisé pour sa simplicité

JAX-RS

Spécification technique (JSR 311) côté serveur

JAX-RS 1.1 : Java EE 6

JAX-RS 2.0 : http://jax-rs-spec.java.net/ ➔ Java EE 7

Implémentations de JAX-RS

- Jersey (Implémentation de référence, 2.0 dans Java EE 7)
- RESTEasy (JBoss)
- Restlet
- Apache CXF

Annotations

JAX-RS utilise des annotations Java pour exposer les services

Masque tous les traitements HTTP

@Path : Chemin d'accès à la ressource

@GET, @PUT, @POST, @DELETE : Type de requête

@Consumes : Type de donnée en entrée

@Produces : Type de donnée en sortie

@PathParam, @QueryParam, @HeaderParam, @CookieParam, @MatrixParam, @FormParam

Client REST

Le client REST le plus commun est le navigateur Web.

Pour faire communiquer des applications REST ou effectuer des tests unitaires, des implémentations clientes existent.

Exemple de Jersey Client

- Création du client (opération lourde) :

```
Client client = Client.create();
```

- Accès à une ressource :

```
WebResource webResource =
client.resource("http://example.com/base");
```

- Opérations sur les ressources :

```
String s =
webResource.accept("text/plain").get(String.class);

ClientResponse response =
webResource.type("text/plain").put(ClientResponse.class,
"foo:bar");
```