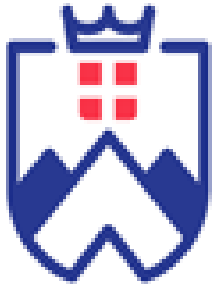


UNIVERSITÉ
SAVOIE
MONT BLANC

Architectures Orientées Services

David TELISSON

Standardiser les noms et les verbes

- L'histoire de l'informatique démontre qu'il est impossible de mettre tout le monde d'accord sur un vocabulaire commun
  - Les informations financières ne sont pas structurées de la même manière que celles d'un flux RSS destinées aux actualités
  - Il n'existe pas un seul schéma universel valide pour tout représenter
- On ne peut pas standardiser de manière universelle :
  - on va construire des annuaires (*registries/repositories)*
  - on va utiliser des transformations (XSLT par exemple)
  - c'est un processus coûteux !

Le bon exemple : les SGBD

- Que l'on conçoive une base pour la finance, les voyages, les bibliothèques
- L'interface entre votre application et les données est basée sur 4 opérations (CRUD)
- INSERT/SELECT/UPDATE/DELETE
- Vous pouvez construire des abstractions au-dessus, mais ces 4 opérations sont génériques et fixées

Revenons au Web

- Et bien, on utilise les mêmes verbes pour
  - consulter un itinéraire
  - envoyer des cartes de vœux
  - commander des livres
  - etc.
- Avec quels verbes ?
  - GET
  - POST

Services Web basés sur le Web

- Que se passe-t-il si on applique les recettes du Web aux problèmes posés par les services Web ?
- On pourrait utiliser le protocole HTTP tel qu'il est, et pas comme une couche transport pour un autre protocole (par ex. SOAP)
- On pourrait utiliser des opérations standards et universelles, comme GET et POST

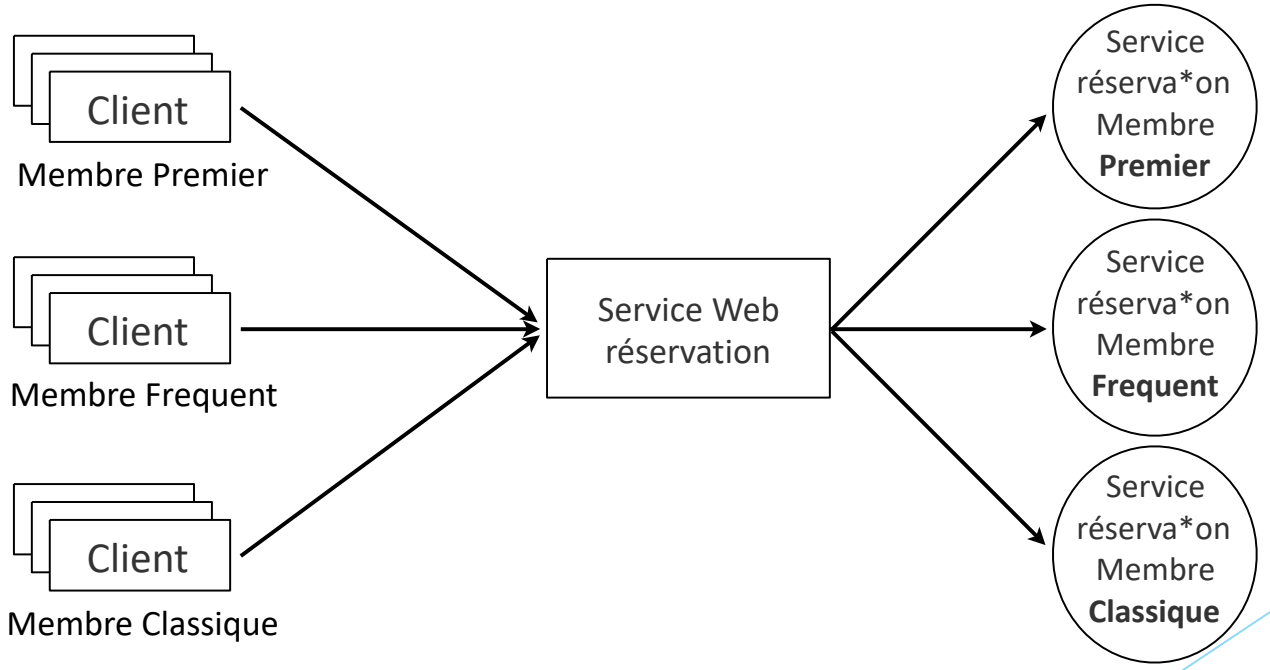
HyperText Transfer Protocol

- A l'origine, protocole permettant de publier et de retrouver des pages
- Construit au-dessus de TCP port 80 par défaut (ou 443 en SSL)
- Protocole requête/réponse sans état
- Le serveur est appelé *origin server*
- Le client est appelé *user-agent*
- Le serveur offre des *resources* qui peuvent être accédées grâce aux URIs
- Format des URLs
  - http://serveur:port/chemin/vers/la/ressource?p1=v1&p2=v2
  - Protocole + Serveur + Chemin + Requete

Requêtes et chemins

On suppose un service de réservation d'une compagnie aérienne

Approche 1



Requêtes et chemins

Inconvénients de cette approche

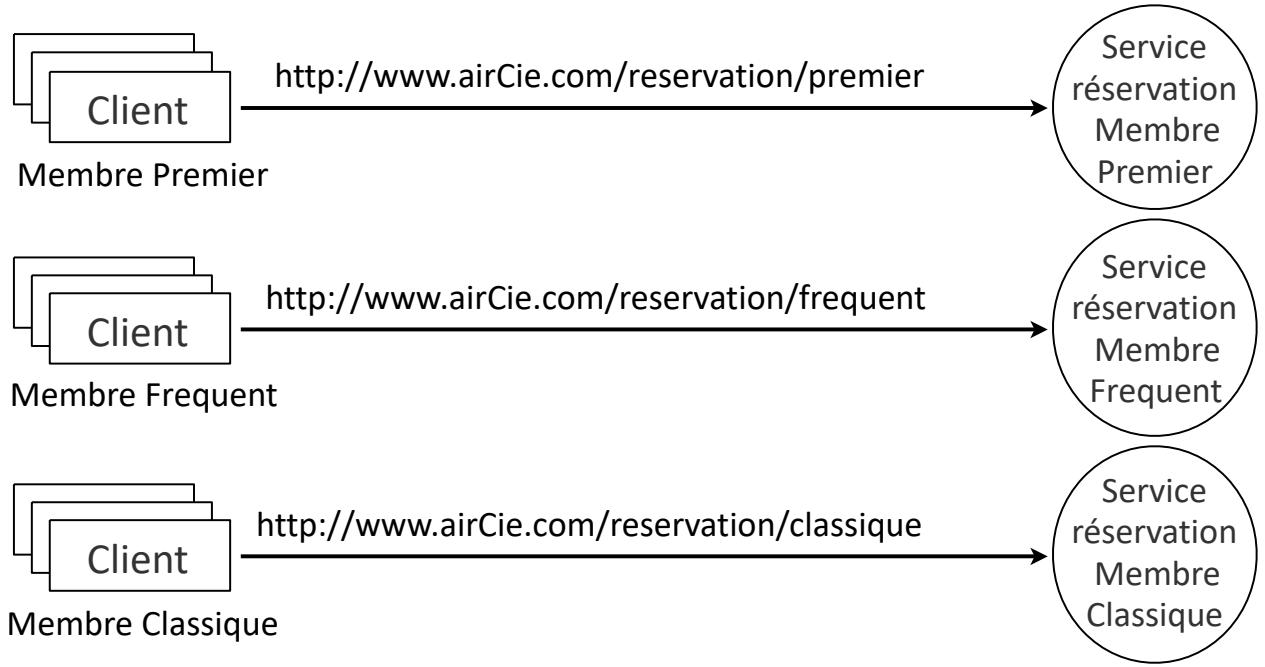
- les règles changent d'un service à un autre. Le client doit apprendre les règles, et le service doit être écrit pour comprendre les règles
- le service est un point névralgique
- elle viole l'axiome 0 de T. Berners-Lee
- les URLs sont compliqués

  www.airCie.com/idClient=45GT76&action=reserv&status=frequent

  www.airCie.com/idClient=45XF36&action=reserv&status=premier

Requêtes et chemins

Approche 2



Requêtes et chemins

- Les URLs peuvent être découverts par les moteurs de recherche
- Simple à comprendre (comme le Web)
- Pas besoin de règles. Le client sait ce qu'il veut, et sait comment y accéder
- Équilibrage facile: un serveur rapide pour les membres premiers
- Cohérent avec l'axiome 0

HTTP: format requête/réponse

- Basé sur du texte
- Requête
  - Méthode + chemin + version HTTP
  - Hôte
  - Contenu (optionnel)
- Réponse
  - Version HTTP + code d'état
  - Entêtes
  - Contenu (optionnel)

HTTP: exemple requête/réponse

Requête

```
GET /index.html HTTP/1.1
Host: www.example.com
```

Réponse

```
HTTP/1.1 200 OK
Date: Mon, 10 November 2008 08:38:34 GMT
Server: Apache/1.3.27 (Unix)  (Red-Hat/Linux)
Last-Modified: Wed, 02 Jan 2008 23:11:55 GMT
Accept-Ranges: bytes
Content-Length: 10
Connection: close
Content-Type: text/html; charset=UTF-8

<html><body>Hello world!</body></html>
```

HTTP: méthodes

- pour récupérer la représentation d'une ressource
  - GET
- pour créer une nouvelle ressource
  - POST
- pour modifier une ressource existante
  - PUT
- pour supprimer une ressource existante
  - DELETE

HTTP: codes d’état

- 1xx : méta-données
- 2xx : tout va bien
- 3xx : redirection
- 4xx : le client a fait quelque chose d’incorrect
- 5xx : le serveur a fait quelque chose d’incorrect

REST

- *REpresentational State Transfer*
  - Roy Fielding, thèse soutenue en 2000
  - http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- C’est un style d’architecture
- Ce n’est pas un standard
- C’est une émanation de la façon dont fonctionne le WEB
- C’est L’alternative à SOAP !!

REST et les standards

- REST utilise les standards existants
  - HTTP
  - URIs/URLs
  - HTML
  - XML, JSON
- HTTP est utilisé comme protocole applicatif et non protocole de transport !

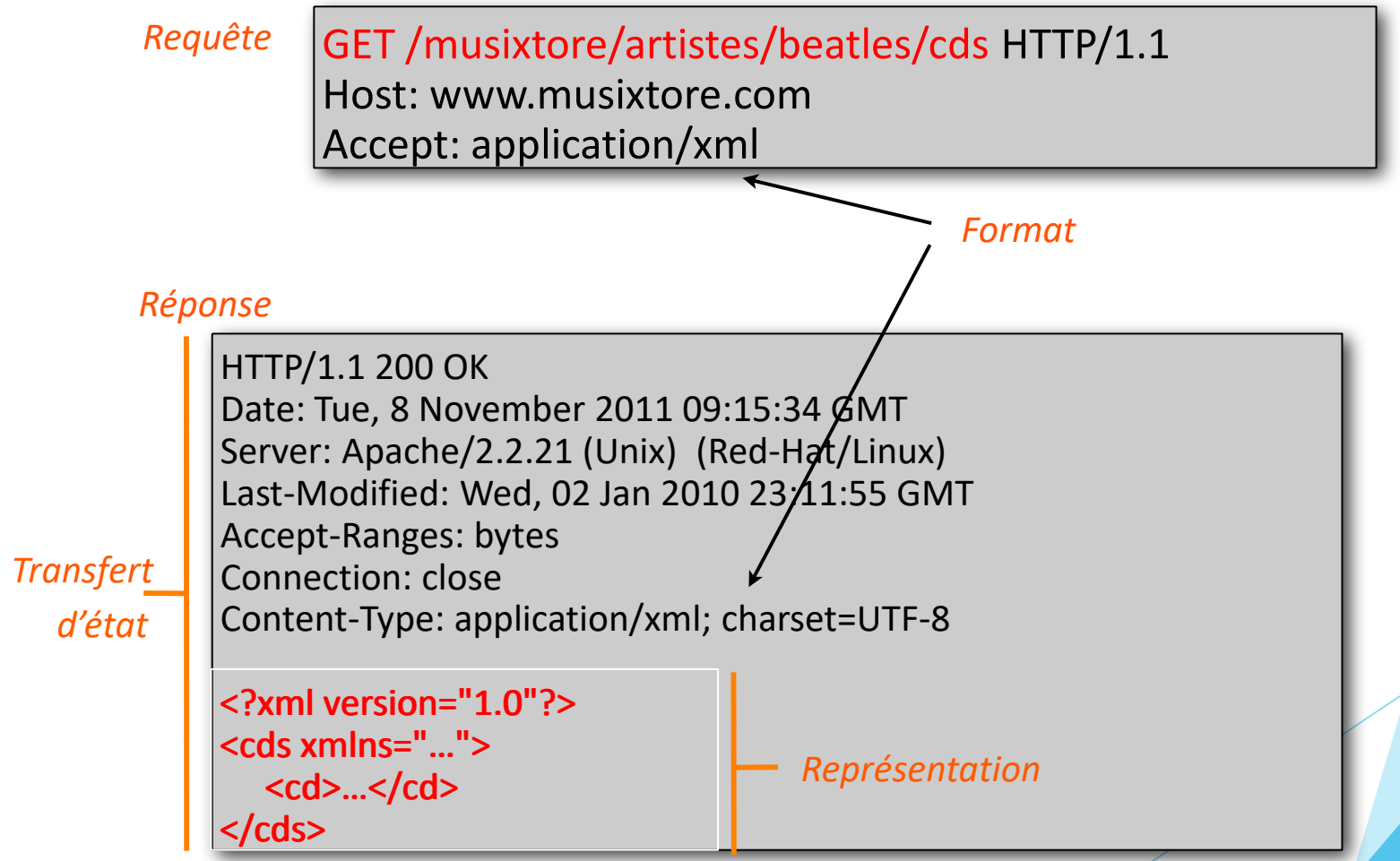
Les principes généraux de REST

- Adressabilité
  - donner un ID à tout !
- Uniformité de l'interface
  - utiliser les standards
- Communication stateless
  - le serveur ne mémorise pas l'état d'une conversation avec un client
- Transfert d'état en utilisant les possibilités hypermédia
  - on relie les choses entre elles

Adressabilité

- Les ressources sont
  - identifiées par des URIs
  - manipulées grâce à leurs représentations, et non directement
- Les URIs permettent d'exposer
  - les données
  - ET l'état des données

Fonctionnement



Les opérations

Méthodes HTTP comparables à CRUD pour des ressources accessibles via des URIs (analogie avec les BDD)

- la ressource est créée avec POST
- elle est lue avec GET
- et mise à jour avec PUT
- pour finalement être supprimée en utilisant DELETE

Principe

- On crée une URL pour chaque ressource
- Les ressources doivent être des noms, pas des verbes
- Exemple
  - http://www.airCie.com/getReservation?idClient=00345
    - getReservation est un verbe
  - http://www.airCie.com/reservation/00345
    - reservation est un nom

Exemple de requête

```
POST /creation_cd HTTP/1.1

<?xml version="1.0" encoding="utf-8"?>
<cd xmlns=“…”>
 <auteur>
   <nom>John Lennon</nom>
  …
 </auteur>
…
</cd>
```

Exemple de réponse

```
HTTP/1.1 201 Created
Location: http://www…/cd_1

<?xml version="1.0" encoding="utf-8"?>
<CD-URI xmlns="…">http://www…/cd_1</CD-URI>
```

La requete pour afficher le résultat

```
GET /cds/cd_1 HTTP/1.1
```

Le résultat

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

<?xml version="1.0" encoding="utf-8"?>
<cd>
   <auteur>
     <nom>John Lennon</nom> …
   </auteur>
…
</cd>
```

Avantages de REST

- Passage à l'échelle
  - HTTP définit une interface standard pour tout le monde
- Tolérance aux fautes
  - un serveur Web est remplaçable par un autre
  - toute l'information nécessaire pour le traitement d'une requête est dans la requête
- Secure
  - HTTPs est relativement mature
  - SSL/TLS pour les échanges point-à-point
- Couplage TRES faible
  - Ajouter une opération métier ne modifie pas les autres opérations métier

SOAP ou REST ?

- Emacs ou vi ;-)
- contre SOAP
  - on réinvente un peu le Web avec HTTP POST
  - c'est relativement complexe
  - on perd les avantages du Web
- contre REST
  - impossible de tout modéliser avec GET, POST, PUT et DELETE

Comment choisir

- Les Web services sont très ambitieux
  - SOAP: extensibilité
  - indépendant du protocole (peut être porté sur autre chose que HTTP)
  - description des services analysable par une machine

- REST est simple, mais reste au niveau technologique
  - rien sur fiabilité, intégrité, sécurité,… car on utilise HTTP
  - modélisation du système comme une ressource

Quand utiliser SOAP ?

- Si vous estimez nécessaire d'avoir un contrat entre le service et le client à travers une interface (WSDL)
- Si vous avez des besoins non fonctionnels (adressage fiabilité, sécurité, coordination,…) et que vous ne voulez pas les coder vous-même
- Si vous avez besoin d'autres protocoles que HTTP, en particulier sur des protocoles permettant des communications asynchrones (SMTP, JMS)

Quand utiliser REST ?

- Si vos services sont entièrement « stateless »
- Si vos services renvoient des données qui peuvent être mises dans un cache
- Si le fournisseur et le client du service ont une compréhension commune du contexte
- Si la bande passante est importante pour vous
  - REST est performant même avec des configurations limitées
  - REST ne nécessite pas d'infrastructure particulière