Architectures Web Services RESTful

Alexandre Denis – Alexandre.Denis@inria.fr

Inria Bordeaux – Sud-Ouest
France

ENSEIRB PG306

REST

- REST – Representational State Transfer
  - Roy Fielding (2000)
  - *Décollage* vers 2006-2007
  - Alternative à SOAP – protocole historique des Web Services
  - Désormais l'interface prédominante dans les WS
- Approche **minimaliste**

RESTful

- Vrai REST / faux REST
- REST est une **philosophie de conception d'interface**
  - Pas réellement une norme
  - Pas un protocole
- RESTful
  - Beaucoup de protocoles web sur HTTP prétendent être du REST
  - **RESTful** = *vrai* REST
  - Parfois : architecture REST, service RESTful

Propriétés REST

- Client/serveur
- Sans état – serveur sans mémoire
- Cacheable, proxy
  - Conséquence de « sans état »
- Interface uniforme
  - Ressources identifiées de façon unitaire
  - Manipulation des ressources au travers de représentations
  - Messages auto-descriptifs
  - HATEOS – Hypermedia as the Engine of Application State
- Système en couches

Ressources REST

- Architecture centrée sur les ressources
  - Pas sur les actions
- Entité de base : une **ressource**
  - ~ objet
  - Arboressence de ressources
    - Point d'entrée de l'application : **ressource racine**
  - **Verbes** (~méthodes) appliqués aux ressources
    - Façon *base de données*: put, get, create, delete
  - Liens direct entre ressources
    - Façon *hypertexte*

HATEOS
Hypermedia as the Engine

- RPC **en HTTP** en non **au-dessus** de HTTP
- Pourquoi ré-implémenter un mécanisme RPC (SOAP) sur HTTP, alors que HTTP est déjà un mécanisme RPC avec nativement :
  - Un système de **requêtes** (GET, POST, PUT, CREATE, DELETE)
  - Un système d'**adressage** d'entités (URI)
  - Un système de retour d'**erreur** (404, etc.)
  - Un système de **typage** (Content-Type, types MIME, etc.)
- HTTP est conçu depuis l'origine comme un **vrai protocole de requêtes**
  - Même si le WWW utilise quasi-exclusivement GET

Représentations

- Les données échangées sur le fil sont une **représentation**
  - Clairement distincte de l'implémentation
  - Sérialisées en un format à la carte
    - Souvent **JSON** et XML
    - Personnalisable par le client
      - Headers HTTP `Accept` et `Content-Type`
- Représentation auto-descriptive
  - Content-Type, charset, compression
  - Tout est déjà prévu dans HTTP

Représentations - Exemples

- Représente une **ressource complète**
  - S'il y a besoin de découper, c'est que le modèle de données est à revoir
- Représentation JSON et XML, pour une même donnée
  - Applications Web/AJAX **souvent en JSON**
  - XML : habituel en WS
  - D'autres formats sont possibles

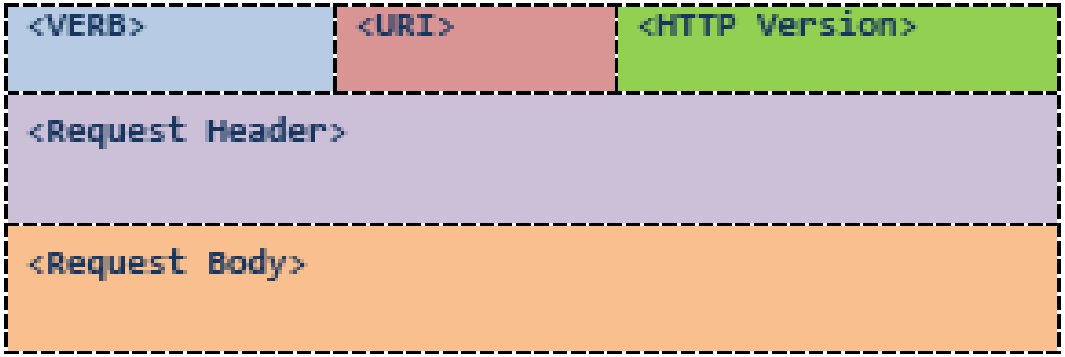
```
{
    "ID": "1",
    "Name": "M Vaqqas",
    "Email": "m.vaqqas@gmail.com",
    "Country": "India"
}
```

```
<Person>
  <ID>1</ID>
  <Name>M Vaqqas</Name>
  <Email>m.vaqqas@gmail.com</Email>
  <Country>India</Country>
</Person>
```

Exemple : Dr. Dobbs, M. Vaqqas

Messages

- Format habituel des messages HTTP
- Les entités HTTP sont mappées directement sur les entités REST :
  - Les requêtes HTTP sur les **verbes** REST
  - Les URI HTTP sur les **ressources** REST



Requête - exemple

- Exemple de requête POST

```
POST http://MyService/Person/
Host: MyService
Content-Type: text/xml; charset=utf-8
Content-Length: 123

<?xml version="1.0" encoding="utf-8"?>
<Person>
  <ID>1</ID>
  <Name>M Vaqqas</Name>
  <Email>m.vaqqas@gmail.com</Email>
  <Country>India</Country>
</Person>
```

- PUT v.s. POST – par convention :
  - PUT : crée/actualise une ressource en donnant l'**URI de la ressource**
  - POST : on donne l'URI d'une **méthode**
    - C'est le serveur qui assigne l'URI de la ressource

Bonne conduite

- A l'utilisateur d'appliquer les conventions
- Si on peut utiliser une URI de ressource, c'est que c'est une ressource !
  - À ne pas faire : `GET http://MyService/Persons?id=1`
  - Préférer : `GET http://MyService/Persons/1`
- GET ne fait pas d'effet de bord
  - À ne pas faire : `GET http://MyService/DeletePerson/1`
  - Préférer : `DELETE http://MyService/Persons/1`
- PUT est idempotent
  - Appeler la requête plusieurs fois à le même effet qu'une seule fois
- Pas d'état dans le serveur
  - À ne pas faire :
    ```
    GET http://MyService/Persons/1 HTTP/1.1
    GET http://MyService/NextPerson HTTP/1.1
    ```
  - Préférer : liste de liens, **itérateur côté client**

Description d'interfaces

- Description d'une interface REST
  - WSDL 2.0 a une projection REST
  - WADL - Web Application Description Language
    - Décrit les types
  - RSDL - RESTful Service Description Language
    - Décrit les URI
- Langages d'usage marginal
- Le plus souvent, description texte en ~~français~~ anglais

```
<method name="GET" id="ItemSearch">
 <request>
  <param name="Service" style="query"
   fixed="AWSECommerceService"/>
  <param name="Version" style="query"
fixed="2005-07-26"/>
  <param name="Operation" style="query"
fixed="ItemSearch"/>
  <param name="SubscriptionId" style="query"
   type="xsd:string" required="true"/>
  <param name="SearchIndex" style="query"
   type="aws:SearchIndexType"
required="true">
    <option value="Books"/>
    <option value="DVD"/>
    <option value="Music"/>
  </param>
  <param name="Keywords" style="query"
   type="aws:KeywordList" required="true"/>
  <param name="ResponseGroup" style="query"
   type="aws:ResponseGroupType"
repeating="true">
    <option value="Small"/>
    <option value="Medium"/>
    <option value="Large"/>
    <option value="Images"/>
  </param>
 </request>
 <response>
  <representation mediaType="text/xml"
   element="aws:ItemSearchResponse"/>
 </response>
</method>
```

Exemple : Amazon

Outils pour services RESTful

- Assemblage de protocoles et formats normalisés
  - **Pas de modèle de programmation** spécifique !
  - Pas d'outil / API / workflow standard
  - Outil dédié ou assemblage ad-hoc d'une lib HTTP et d'une lib JSON
- Approches asymétriques côté client et serveur - typiquement :
  - Serveur : framework, annotations, serveurs d'applications
  - Client : scripts, langages web, assemblage ad-hoc HTTP+parseur JSON/XML

Middleware pour service RESTful

- JAX-RS - Java API for RESTful Web Services
  - Annotations de code Java
- Frameworks Java
  - Restlet
    - Framework Java, pionnier (2007), interface spécifique et JAX-RS
  - Apache CXF
    - Fondation Apache, WS-*, RESTful spécifique ou JAX-RS, etc.
  - Oracle Jersey
    - JAX-RS, Maven, Tomcat
  - Oracle WebLogic, RedHat JBoss RESTEasy, Java JSP
- Ruby-on-rails, Python Django
- Solutions ad-hoc
  - Servlets ad-hoc, php+rewrite rule, Node.js, etc.

Clients REST

- Grand nombre d'approches !
  - Tout ce qui sait faire du HTTP peut être client REST
- Partie cliente d'un framework REST
- Bibliothèques Ruby, python
- Assemblage ad-hoc en Javascript, PHP
- En C : libcurl + Jansson, libcurl + expat
- Script shell, curl

JAX-RS

- JAX-RS - Java API for RESTful Web Services
  - **Annotations** de code Java
  - Pour préciser : type de requête, URI, type

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/hello")
public class Hello
{
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  public String sayPlainTextHello()
  {
    return "Hello Jersey";
  }
}
```

Exemple : Jersey

JAX-RS

```
@Path("/hello")
public class Hello
{
  // This method is called if TEXT_PLAIN is request
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  public String sayPlainTextHello() {
    return "Hello Jersey";
  }

  // This method is called if XML is request
  @GET
  @Produces(MediaType.TEXT_XML)
  public String sayXMLHello() {
    return "<?xml version=\"1.0\"?>" + "<hello> Hello Jersey" + "</hello>";
  }

  // This method is called if HTML is request
  @GET
  @Produces(MediaType.TEXT_HTML)
  public String sayHtmlHello() {
    return "<html> " + "<title>" + "Hello Jersey" + "</title>"
        + "<body><h1>" + "Hello Jersey" + "</body></h1>" + "</html> ";
  }
}
```

Exemple : Jersey

Référence JAX-RS

- Liste des annotations

`@Path`	indique un morceau du chemin de l'URI
`@GET`	désigne une méthode pour traiter les requêtes GET
`@PUT`	désigne une méthode pour traiter les requêtes PUT
`@POST`	désigne une méthode pour traiter les requêtes POST
`@DELETE`	désigne une méthode pour traiter les requêtes DELETE
`@HEAD`	désigne une méthode pour traiter les requêtes HEAD
`@HeaderParam`	extrait un paramètre à partir du header http
`@PathParam`	extrait un paramètre à partir du chemin
`@QueryParam`	extrait un paramètre de l'URI
`@Consumes`	type MIME des données acceptées par la méthode
`@Produces`	type MIME des données produites par la méthode
`@Provider`	indique qu'une classe fournit des méthodes auxiliaires pour JAX-RS

JAXB

- JAXB - Java Architecture for XML Binding
  - Sérialiseur/parseur XML en Java
  - Entièrement généré par annotations de code
  - Conversion XSD <-> code Java annoté
  - Permet la sérialisation désérialisation JSON (`MediaType` explicite)
    - Pas implémenté dans l'implem de référence du JDK, il faut utiliser *Jackson* ou *Moxy*

```
import javax.xml.bind.annotation.*;

@XmlRootElement(name="hello")
public class HelloObject
{
  public HelloObject(int a)
  {
    id = a;
  }
  @XmlElement(name="id")
  int id;
}
```

Travail à faire

Exemple REST

- Téléchargez l'exemple hello-jaxrs.tar.gz sur la page du cours
- Les exemples utilisent :
  - L'interface JAX-RS du framework Restlet côté serveur
    - `ExampleServer.java` : *main* générique pour application restlet
    - `ExampleApplication.java` : application restlet générique pour une ressource
    - `HelloRootResource.java` : code JAX-RS
    - `HelloObject.java` : classe représentant une resource en JAXB
  - Un assemblage ad-hoc Apache HttpComponents + JAXB côté client
    - `ClientHttp.java` : client gérant manuellement le protocole HTTP
    - `HelloObject.java` : classe JAXB commune avec le serveur
  - Des routines de test en ligne de commande à l'aide de curl
    - Dans le `Makefile`

REST avec `curl`

- Parcourez le code du serveur
  - Identifiez les différents types de requêtes
  - Identifiez les manipulations et conversions de données (texte, XML)
- Lancez des requêtes vers le serveur à l'aide de l'outil `curl`
  - Consultez le manuel de curl si besoin
    - Note : REST utilise intensivement le `Content-Type`. Précisez-le **toujours** à curl
- Testez les différents types de requêtes : GET, PUT, POST

REST avec le client Java HttpComponents

- Parcourez le code du client
  - Identifiez les parties relatives à HTTP et à XML
- Ajoutez d'autres requêtes vers les ressources du serveur

REST comme vous voulez

- Vous êtes **encouragés** à tester d'autres solutions côté client selon les langages que vous maîtrisez (Ruby, Python, PHP, Javascript, Go, Rust)
  - du moment qu'une bibliothèque http cliente est disponible et un parseur XML ou JSON
  - libcurl + jansson pour ceux qui font du C
  - JAXB en mode JSON pour ceux qui font du Java

Construire un service REST

- Nous proposons d'écrire un service à la IMDb
- Proposez une interface pour décrire un film avec : titre, réalisateur, acteurs
  - Quelles sont les différentes ressources ?
  - Comment construire l'arboressence ?
  - Quelles sont les opérations ?
    - Il faut au moins : ajout d'entrée, interrogation d'une fiche, recherche par titre, recherche par personne (réalisateur, acteur)
    - Comment gérer une requête qui retourne plusieurs réponses ?
  - Écrivez le serveur avec JAX-RS / Restlet
- Créez un script shell utilisant curl pour remplir la base à l'initialisation
- Créez un client Java effectuant des recherches

Vers un service externe

- Un grand nombre de sites « Web 2.0 » proposent une interface REST : Facebook, Google, Twitter, Instagram, IMDb, Wikipedia, etc.
- Choisissez votre site favori, consultez sa documentation, écrivez un client !
  - Note : il y a toujours une **authentification** qui peut être délicate à gérer

Retour sur l'application bancaire

- L'application bancaire du TP 2 utilise :
  - CORBA pour les serveurs internes à la banque et en interbancaire
  - REST pour l'interaction avec les clients
- Ajoutez une interface REST à la banque avec :
  - Création d'un nouveau client
  - Liste des comptes d'un client
  - Ouverture/fermeture de compte
  - Obtention du solde d'un compte
  - Ordre de virement
- Écrivez un code client REST pour lancer ces différentes opérations
  - Dans la réalité, le client REST est embarqué dans une application web...
  - On laisse de côté la problématique d'authentification
- Cette partie est **à rendre avec le projet**

À vous de jouer !

http://dept-info.labri.fr/~denis/