REST

Un Survol des principaux concepts

(basé essentiellement sur le support SUN par C.McDonald)

REST?

- **REST (REpresentational State Transfer)** est un style d’architecture pour les systèmes hypermédia distribués

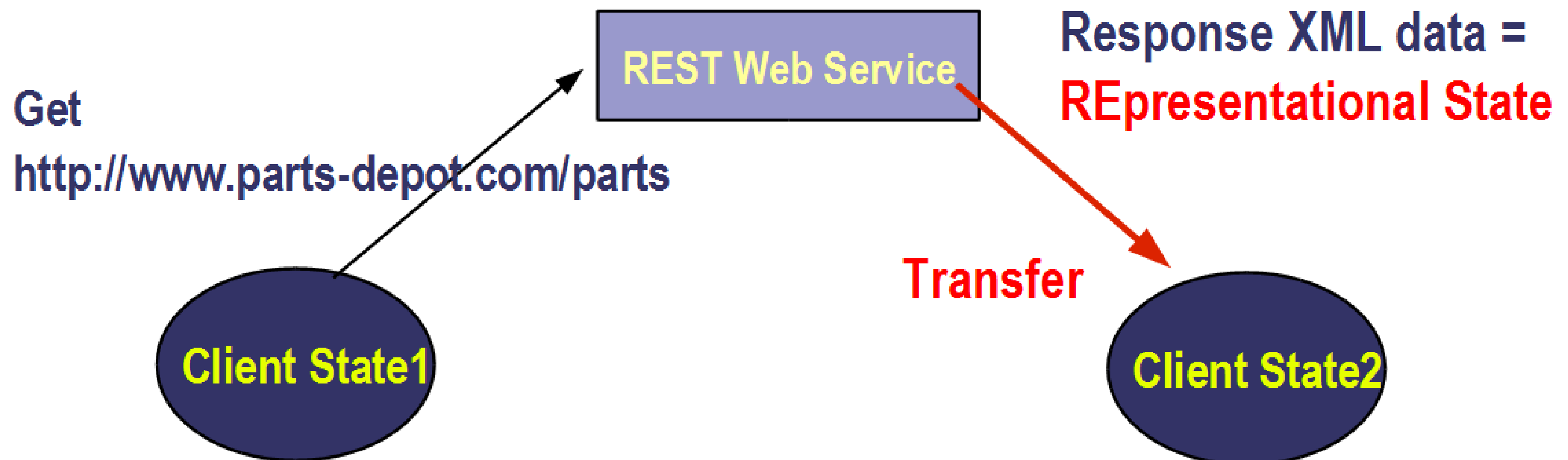
- Créé par Roy Fielding en 2000 (thèse de doctorat)

- REST n’est pas un protocole (tel que HTTP) ou un format

- Style d'architecture particulièrement bien adapté au WWW mais n'est pas dépendant du Web.
  - Peut s'appliquer à d'autres protocoles d'application que HTTP.

REpresentational State Transfer (REST)

- L’URL c’est la Ressource
- GET pour afficher la page à partir du serveur
  - Transfert de l’état de la ressource sur le navigateur du client
- Les ressources sont accessibles à travers des liens hyperlink



Concepts clés

- Ressources (noms)
  - Identifiées par une URI, Exemple: http://www.parts-depot.com/parts
- Méthodes (verbes) afin de manipuler les ressources
  - Create, Read, Update, Delete
- Représentation est la manière de voir/échanger l'état de la ressource
  - Transfert de données et d'état entre le serveur et le client
  - XML, HTML, JSON...

Exemple

Request

```
GET /music/artists/beatles/recordings HTTP/1.1
Host: media.example.com
Accept: application/xml
```

Méthode

Ressource

Response

```
HTTP/1.1 200 OK
Date: Tue, 08 May 2007 16:41:58 GMT
Server: Apache/1.3.6
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0"?>
<recordings xmlns="…">
  <recording>…</recording>
  …
</recordings>
```

Transfert d'état

Représentation

REST en 5 étapes

- Donner un ID pour chaque ressource
- Utiliser les méthodes standard d’HTTP
- Lier les ressources entre elles
- Choix entre multiples représentations
- Communication sans état (Stateless)

Etape 1: Donner un ID pour chaque ressource

- http://example.com/customers/1234
  - Le client num 1234 de la collection de clients
- http://example.com/products/4554
- http://example.com/customers/1234/orders/12
  - La commande num 12 du client num 1234

Etape 2: Utiliser les méthodes standards d'HTTP

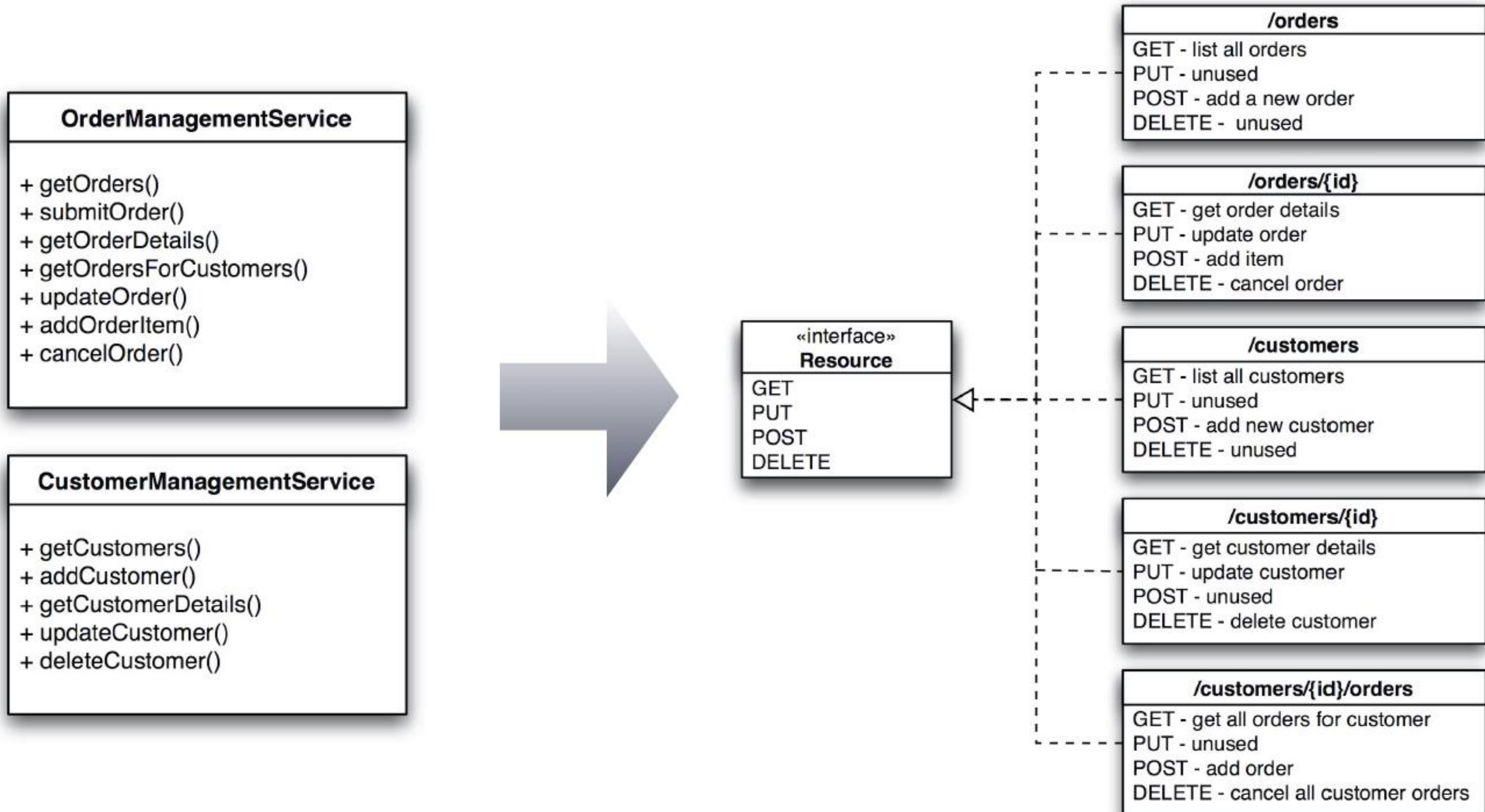
- GET: Lecture d'une information (ressource)
  - éventuellement déjà présente dans le cache
  - Sans effet de bord
    - Exp. **GET /toto/customers/1234**
- POST: Créer une nouvelle information (ressource) sans l'ID
  - Créer la ressource et la rajouter à une collection
  - Exp. POST **/toto/customers**
    - Ajoute le client spécifié dans le POSTDATA à la collection des clients.
    - L'opération retourne l'URI de la nouvelle ressource créée

Etape 2: Utiliser les méthodes standards d'HTTP

- PUT: Mise à jour /création d'une ressource avec un ID connu
  - Exp. PUT /toto/customers/1234
    - Remplace le client num 1234 avec une nouvelle version

- DELETE: Effacer une ressource
  - Exp. /toto/customers/1234
    - Efface le client num 1234 du système

Ce qui change dans la conception



Etape 3: Lier les ressources entre elles

- Permet au client de faire évoluer l'application d'un état à un autre en suivant des liens et en remplissant des formulaires

```
<order self='http://example.com/orders/1234'>
        <amount>23</amount>
        <product ref='http://example.com/products/4554' />
        <customer ref='http://example.com/customers/1234' />
</order>
```

Etape 4: Choix entre multiples représentations

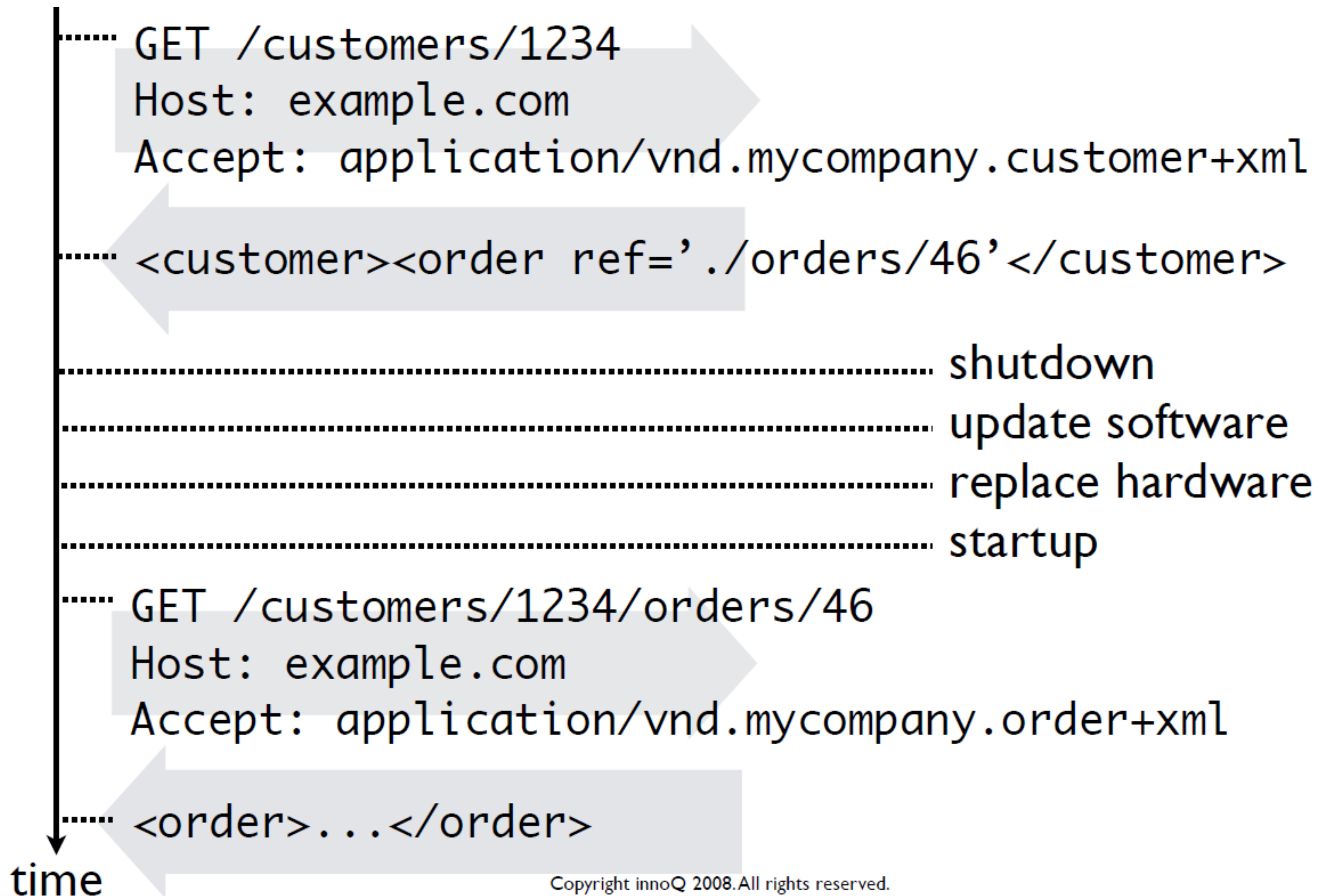
- Plusieurs formats possibles selon les besoins
  - XML, JSON, (x)HTML

```
// 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> ";
}
```

Etape 5: Communication sans état (Stateless)

- HTTP est Stateless (sans état)
- Tout ce qui est nécessaire pour traiter une demande est dans l'objet Request
- Le client est responsable de l'état de l'application
- Le serveur est responsable de l'état de la ressource
- Exp. Agence de voyage en ligne
  - Créer un voyage, définir l'itinéraire, le soumettre, etc.
  - Le tout est géré coté client et non pas sur la session du serveur

Etape 5: Communication sans état (Stateless)



Copyright innoQ 2008. All rights reserved.

REST: Principaux avantages

- Coté Serveur
  - Plus de passage à l’échelle (scalable)
  - Reprise après panne
  - Utilisation optimisée du cache
  - Couplage réduit
  - Fonctionne avec les infrastructures actuelles
  - Interface uniforme

- Coté Client
  - Lien bookmarkable (favoris)
  - Besoin d’un simple navigateur
  - Plusieurs langages supportés
  - Plusieurs choix de formats de données

REST
Un exemple
- Utilisation d’annotations JAX-RS

Etape 1: Donner un ID pour chaque ressource

- Une ressource => Classe POJO (Plain Old Java Object)
  - Pas d'interface requise!
- L'ID est défini par l'annotation **@Path**
  - Relative au contexte de déploiement
  - Peut être utilisée pour annoter la classe ou directement la méthode censée retourner la ressource

http://host/ctx/orders/12

```
@Path("orders/{id}")
public class OrderResource {
```

http://host/ctx/orders/12/customer

```
  @Path("customer")
  CustomerResource getCustomer(...) {...}
}
```

Etape 1: Donner un ID pour chaque ressource

- Comment mapper les URIs aux Classes:

```
@Path("/items")          Collection contenant les items du catalogue
public class Items {
    @Get
    public ItemsConverter get() {
        ... return new ItemsConverter(itemList);
    }
    @Path("{id}/")                          retourne 'item' selon l'id
    public ItemResource getItem(@PathParam("id")int id) {
        ... return itemResource;
    }
}
```

Etape 1: Donner un ID pour chaque ressource

- Deux manières de créer des sous-ressources

```
public class ItemResource {
    @Path("/items/{id}/")
    @GET
    public ItemConveter get(@PathParam("id") Long id) {
 .....
    }
```

```
@Path("/items/")
 public class ItemsResource {
 @Path("{id}/")
 public ItemResource getItemResource(@PathParam("id") Long id) { ...
   return resource;
  }
```

Etape 2: Utiliser les méthodes standards d'HTTP

- Annoter les classes de ressources avec les méthodes standards selon le besoin
  - @GET, @PUT, @POST, @DELETE

```java
// For the browser
@GET
@Produces(MediaType.TEXT_XML)
public Todo getTodoHTML() {
  Todo todo = TodoDao.instance.getModel().get(id);
  if(todo==null)
    throw new RuntimeException("Get: Todo with " + id +  " not found");
  return todo;
}

@PUT
@Consumes(MediaType.APPLICATION_XML)
public Response putTodo(JAXBElement<Todo> todo) {
  Todo c = todo.getValue();
  return putAndGetResponse(c);
}

@DELETE
public void deleteTodo() {
  Todo c = TodoDao.instance.getModel().remove(id);
  if(c==null)
    throw new RuntimeException("Delete: Todo with " + id +  " not found");
}

private Response putAndGetResponse(Todo todo) {
  Response res;
  if(TodoDao.instance.getModel().containsKey(todo.getId())) {
    res = Response.noContent().build();
  } else {
    res = Response.created(uriInfo.getAbsolutePath()).build();
  }
  TodoDao.instance.getModel().put(todo.getId(), todo);
  return res;
}
```

Le @Path n'est pas donné dans l'exemple, il est défini avant la signature de la classe

Etape 2: Utiliser les méthodes standards d'HTTP

- Possibilité d'extraire les informations à partir des paramètres de la requête avec **@QueryParam**

```
@Path("/items/")
@ConsumeMime("application/xml")
public class ItemsResource {

  @GET
  ItemsConverter get(@QueryParam("start")
    int start) {
   ...
  }

  @Path("{id}/")
  ItemResource getItemResource(@PathParam("id") Long id){
  ...
  }

}
```

http://host/catalog/items/?start=0

http://host/catalog/items/123

Etape 3: Lier les ressources entre elles

- **UriInfo** donne l'information sur le contexte de déploiement, l'URI, et le chemin jusqu'à la ressource
- **UriBuilder** offre des facilités pour créer les URIs des ressources

```
@Context UriInfo i;
OrderResource r = ...
UriBuilder b = i.getBaseUriBuilder();
URI u = b.path(OrderResource.class).build(r.id);

List<URI> ancestors = i.getAncestorResourceURIs();
URI parent = ancestors.get(ancestors.size()-1);
```

Etape 4: Choix entre multiples représentations

- Annoter les méthodes ou bien les classes avec
  - **@ProduceMime, @ConsumeMime**

```
@GET
@ProduceMime({"application/xml","application/json"})
Order getOrder(@PathParam("order_id") String id) {
   ...
}
@GET
@ProduceMime("text/plain")
String getOrder(@PathParam("order_id") String id) {
   ...
}
```

Etape 4: Choix entre multiples représentations

Request

```
GET /music/artists/beatles/recordings HTTP/1.1
Host: media.example.com
Accept: application/xml
```

Accept HTTP header

Format

Response

```
HTTP/1.1 200 OK
Date: Tue, 08 May 2007 16:41:58 GMT
Server: Apache/1.3.6
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0"?>
<recordings xmlns="…">
  <recording>…</recording>
  …
</recordings>
```

Content-type HTTP header

Representation

Etape 4: Choix entre multiples représentations

- JAX-RS peu automatiquement faire du Marshalling/ UnMarshaling entre les messages HTTP et les types Java. Support de:
  - **text/xml, application/xml, application/json - JAXB class**
  - ***/*** – byte[], InputStream, File, DataSource
  - **text/*** – String
  - **application/x-www-form-urlencoded** -MultivaluedMap<String, String>

- Il suffit d'annoter votre pojo avec @XmlRootElement

```java
package de.vogella.jersey.jaxb.model;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
// JAX-RS supports an automatic mapping from JAXB annotated class to XML and JSON
// Isn't that cool?
public class Todo {
  private String summary;
  private String description;
  public String getSummary() {
    return summary;
  }
  public void setSummary(String summary) {
    this.summary = summary;
  }
  public String getDescription() {
    return description;
  }
  public void setDescription(String description) {
    this.description = description;
  }


}
```

Etape 4: Choix entre multiples représentations

- Exemple complet

La classe de service annontée par JAX-RS

```
package de.vogella.jersey.jaxb;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import de.vogella.jersey.jaxb.model.Todo;

@Path("/todo")
public class TodoResource {
// This method is called if XMLis request
@GET @Produces({ MediaType.APPLICATION_XML,
MediaType.APPLICATION_JSON })
public Todo getXML() {
Todo todo = new Todo();
todo.setSummary("This is my first todo");
todo.setDescription("This is my first todo");
return todo; }
// This can be used to test the integration with the browser
@GET @Produces({ MediaType.TEXT_XML })
public Todo getHTML() {
Todo todo = new Todo();
todo.setSummary("This is my first todo");
todo.setDescription("This is my first todo");
return todo; }
}
```

La classe utilisée par JAXB pour créer la représentation JSON ou XML

```
package de.vogella.jersey.jaxb.model;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
// JAX-RS supports an automatic mapping from JAXB annotated class to XML and JSON
// Isn't that cool?
public class Todo {
  private String summary;
  private String description;
  public String getSummary() {
    return summary;
  }
  public void setSummary(String summary) {
    this.summary = summary;
  }
  public String getDescription() {
    return description;
  }
  public void setDescription(String description) {
    this.description = description;
  }


}
```

Etape 5: Communication sans état (Stateless)

- Une nouvelle instance est créée pour chaque requête
  - Réduit les problèmes de concurences
- Les sessions HTTP ne sont pas supportées
- Le développeur doit gérer l'état de l'application à travers les représentations

Et le WSDL dans tout ça?

- Pas nécessaire mais un format existe, WADL!!
  - WADL (Web Application Description Language)
  - https://wadl.dev.java.net/

```
<resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
    <resource path="newsSearch">
        <method name="GET" id="search">
            <request>
                <param name="appid" type="xsd:string" style="query" required="true"/>
                <param name="query" type="xsd:string" style="query" required="true"/>
                <param name="type" style="query" default="all">
                    <option value="all"/>
                    <option value="any"/>
                    <option value="phrase"/>
                </param>
                <param name="results" style="query" type="xsd:int" default="10"/>
                <param name="start" style="query" type="xsd:int" default="1"/>
                <param name="sort" style="query" default="rank">
                    <option value="rank"/>
                    <option value="date"/>
                </param>
                <param name="language" style="query" type="xsd:string"/>
            </request>
            <response>
                <representation mediaType="application/xml" element="yn:ResultSet"/>
                <fault status="400" mediaType="application/xml" element="ya:Error"/>
            </response>
        </method>
    </resource>
</resources>
```

Conclusion

- Style architectural de plus en plus utilisé
- Défini comme étant le vrai WEB contrairement aux WS SOAP
- Jeunesse encore au niveau des standards de sécurité, transactions, etc.
- Eviter de généraliser!! Le tout Rest ou le tout SOAP!!