Systèmes Multi-agents

Service REST

Claude Moulin

Université de Technologie de Compiègne

IA04

Sommaire

Sommaire

Interopérabilité

- Le Web n'est pas défini par des implémentations particulières, mais par des interfaces et des protocoles standards.
- Les utilisations augmentent et le facteur d'échelle est un problème à prendre en compte pour des systèmes distribués au niveau d'Internet.
- Il est important de baser l'Interopérabilité et donc les services Web sur des standards établis.
- Plusieurs standards peuvent être utilisés pour résoudre les mêmes problèmes, chacun avec des forces et des faiblesses et donc il faut suivre des recommandations ou des styles architecturaux pour développer des services Web interopérables.

Origine de REST

- REST : Representational State Transfer
- Thèse de Roy T. Fielding (2000) : Architectural styles and the design of network-based software architectures.
- Modèle idéalisé des interactions dans une application Web exclusive.
- Un ensemble de contraintes architecturales pour à la fois minimiser les temps d'attente et le nombre de communications et de maximaliser l'indépendance et l'extensibilité des implémentations de composants.

Eléments Architecturaux

- REST distingue trois classes d'éléments :
  - éléments de données (ressources),
  - éléments de connexion (connecteurs)
  - éléments de processus (composants)

Sommaire

Données

- Le point clé de REST est l'état des données que les composants communiquent.
- Ils transfèrent des représentations de l'état courant d'une donnée.

Ressource

- Le concept clé de REST est la ressource. Toute entité qui peut être nommée peut être une ressource.
- Une ressource peut avoir plusieurs représentations :
  - un ensemble de valeurs dans une ligne d'une table de base de données.
  - une sérialisation objet en XML, JSON.
- Seule la sémantique de la ressource est invariante (signification des champs).
- Ceci permet de référencer une donnée (instance de concept) plutôt qu'une simple matérialisation.

Identifiant

- REST utilise des identifiants pour distinguer les ressources.
- Dans un environnement Web ces identifiants doivent être uniques.
- On choisit les "Uniform Resource Identifier" (URI).
- Ce sont des identifiants logiques et non pas physiques.
- `http://<server>:<port>/images/123456`
- `http://company.com/customers/123456/orders/12`

Représentation

- Tout composant REST exécute des actions sur des représentations de ressources.
- Ces représentations capturent l'état courant ou voulu d'une ressource et sont transférées entre deux composants REST.
- Une représentation REST comprend :
  - un contenu (une séquence de bytes)
  - des meta-données décrivant le contenu
  - des méta-données décrivant les méta-données (cache)

Format de Données

- Le format d'une représentation est appelé le media type.
- Certains formats sont destinés à être traités par des ordinateurs d'autres par des humains.
- Le type de format peut affecter les performances d'un système.
- Commencer à montrer le résultat à un utilisateur avant que la ressource ne soit complètement reçue est préférable.

Format

- Dans les services Web récents les représentations chaînes utilisent très fréquemment le formalisme XML (Extensible Markup Language) ou le formalisme JSON (JavaScript Object Notation).
- Des représentations communes accessibles à des humains sont aux formats HTML, JPEG ou PDF.

Sommaire

Connecteurs

- Ils gèrent les communications pour un composant.
- Ils présentent une interface abstraite générale pour la communication conduisant à :
  - séparer les rôles,
  - cacher les implémentations sous-jacentes,
  - simplifier et rendre interchangeables les éléments.

Interaction

- Chaque interaction REST est sans état (pas de session).
- Chaque requête contient toute l'information nécessaire pour qu'un connecteur puisse comprendre une requête indépendemment des requêtes précédentes.

Avantages

- Un connecteur n'a pas besoin de retenir l'état de l'application ce qui conduit à des composants plus simples et évolutifs.
- Plusieurs requêtes peuvent être menées en parallèle.
- Des requêtes peuvent être comprises seules conduisant à une orchestration des services plus dynamique.
- Possibilité de cache.

Types de Connecteurs

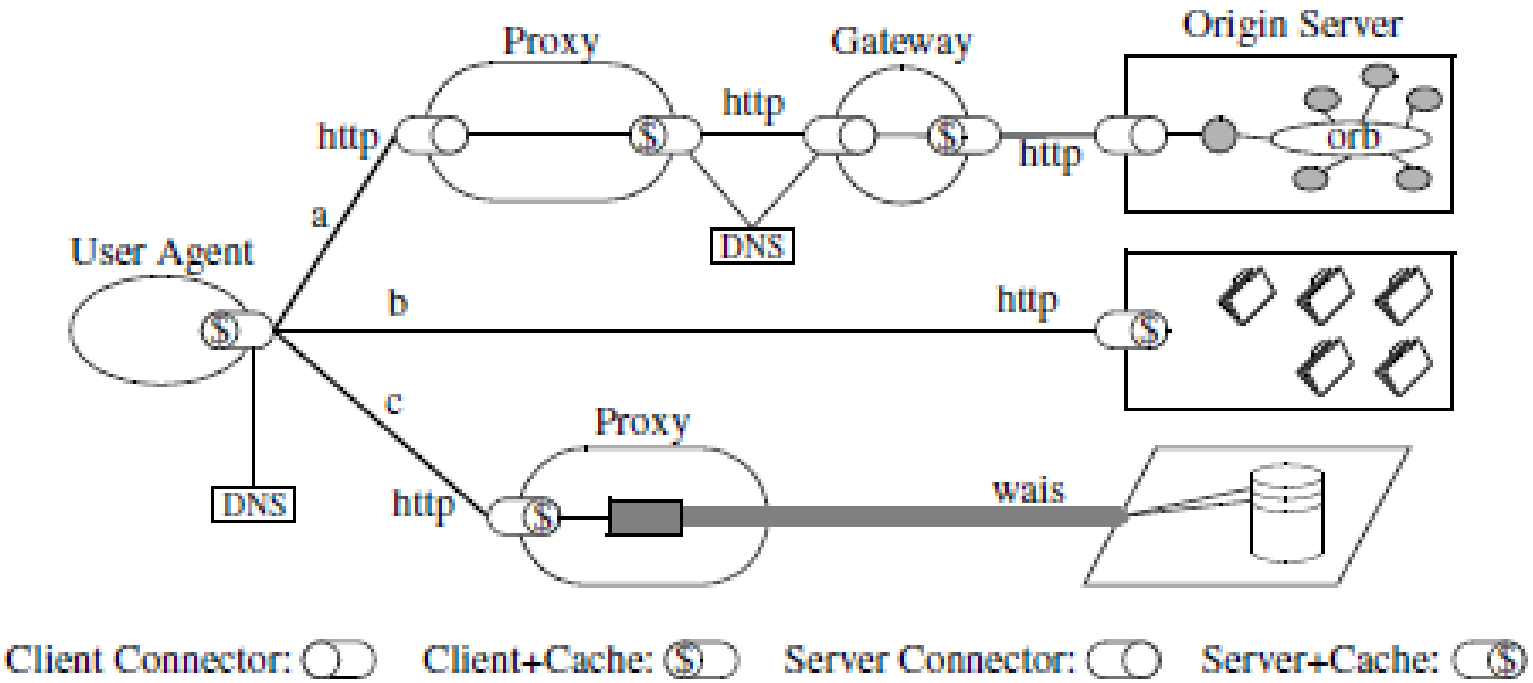
- client : envoie des reqêtes, reçoit des réponses,
- serveur : attend des requêtes, retourne des réponses,
- cache : peut être positionné à un connecteur client ou serveur pour mémoriser des réponses,
  peut aussi être partagé entre plusieurs clients,
- convertisseur : transforme des identifiants de ressources en adresses réseau.
- tunnel : relaie des requêtes. Tout composant peut servir de tunnel.

Sommaire

Types de composant

- Un composant peut contenir différents types de connecteur.
- Un composant est identifié par son rôle dans une application.
- Ex : Un agent utilise un connecteur client pour initier une requête et devient le destinataire final de la réponse.
- Ex : Un serveur Origine utilise un connecteur serveur pour recevoir une requête et décider des représentations de ses ressources.
  Chaque serveur fournit une hiérarchie de ressource comme interface générique de ses services.
- Un composant intermédiaire (proxy, gateway) agit à la fois comme client et comme serveur pour transférer des requêtes et des réponses avec de possibles traductions.

Exemple (Thèse Fielding)



- WAIS = Wide Area of Information Server

Sommaire

Objectifs

- REST définit une architecture en plaçant des restrictions sur l'interaction entre composants au lieu de définir une typologie particulière de composants.
- REST ignore les détails d'implémentation et les syntaxes de protocoles pour satisfaire :
  - l'extensibilité des interactions entre composants,
  - le déploiement indépendant des composants,
  - la réduction des attentes grâce à des composants intermédiaires.

Avantages

- Les composants d'une architecture REST peuvent être dynamiquement réarrangés, des intermédiaires peuvent être placés dans le flot des représentations et agir comme un pipe ou un filtre.
- La nature "stateless" des interactions REST permet à chaque requête d'être indépendante des autres et permet donc aux développeurs de ne pas se soucier de la topologie des composants, tâche impossible dans un système distribué à l'échelle d'Internet.
- REST n'est pas lié à un protocole particulier : il fournit une intégration souple de différents protocoles en permettant à des intermédiaires de transformer des représentations.
- HTTP est le protocole le plus largement utilisé sur Internet.

Orchestration

- L'orchestration de services Web concerne les processus de coordination des échanges d'information à travers l'interaction avec des services web.
  Elle est simplifiée puisque les protocoles sont déjà standardisés.
- Il suffit de connaître les structures de données et les URIs des services impliqués.
- Exemple : on peut utiliser des processeurs XSL pour combiner des services Web REST fournissant des représentations XML de ressources.

Sommaire

Termes REST

- Fonctionnalités de type CRUD (Create, Retrieve, Update, Delete)
- GET : est activée pour retrouver une représentation de la ressource spécifiée par son id.
- PUT : est activée pour ajouter une nouvelle ressource. Elle retourne l'id ou l'URI de la ressource créée.
- POST : est activée pour modifier la resource spécifiée par son id.
- DELETE : est activée pour supprimer la ressource spécifiée par son id.
- d'autres méthodes comme HEADER et OPTIONS OPTIONS : peut être activée pour requérir une description du service.

Termes REST

- Create = PUT ssi est envoyé le contenu complet de la ressource.
- Create = POST si est envoyée une commande au serveur pour créer une partie d'une ressource.
- Retrieve = GET + id.
- Retrieve = GET + paramètres (on retourne une liste d'id de ressources satisfaisant les paramètres).
- Update = PUT + ID ssi est envoyé le contenu complet de la ressource spécifiée (écrasée).
- Update = POST + ID si est envoyée une information modifiant une partie de la ressource spécifiée.
- Delete = DELETE + ID.

Principes REST

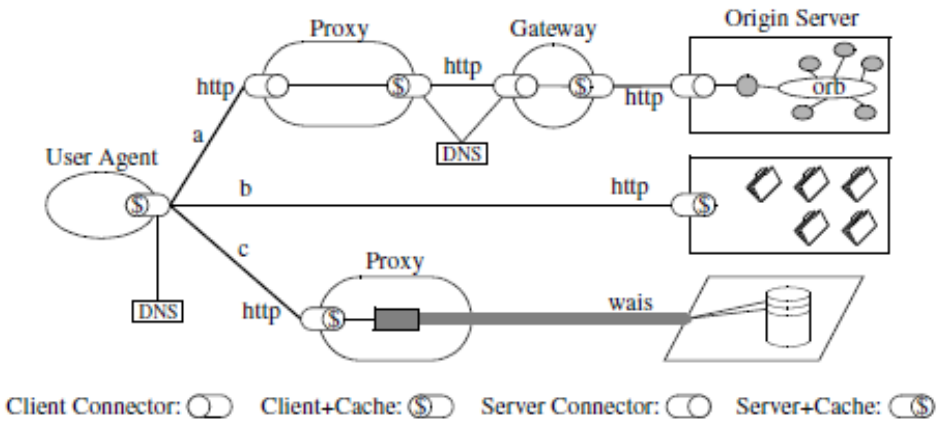
- HTTP GET, HEAD ne doivent rien modifier.
- Idempotence : la définition de HTTP impose que toutes les méthodes sauf POST soient idempotentes.
- PUT, DELETE, GET, HEAD doivent pouvoir être répétées de suite et donner toujours le même résultat.

Sommaire

Sommaire

Use cases

- Les ressources d'un serveur sont des objets : couples (image + description (métadonnées)). L'interface n'expose pas comment ces objets sont enregistrés.



auteur = R. Fielding
cours = IA04
commentaire = Schema REST

Use cases

- Un client peut demander la liste des objets (références), un objet (métadonnées ou image).
- Un client peut créer un objet en uploadant une image.
- Un client peut modifier les métadonnées d'un objet (attachées à une image).
- Des objets (images + métadonnées) peuvent être supprimées.

Ressources - Représentation

- Images : binaire
- Métadonnées : JSON

```
{"id":"json1","filename":"coursia04-json1.png",
"author":"C. M.","cours":"IA04",
"comment":"Grammaire JSON 1"}
```

- Collection d'images : JSON

```
{"values":["json1","contractnet",
"propose","jasonobject"]}
```

Adressage

- Collection d'objets : /images/index/
- Un objet (image, métadonnées) : /images/ID
- Création : /images/

Adressage

- Requête d'objets : /images/ID
- On n'adresse pas séparément les métadonnées et la représentation binaire d'une ressource.
- Le client utilise une négociation de contenu pour déterminer s'il souhaite :
  - Métadonnées seules : la propriété "Accept" avec la valeur "text/json" ajoutée dans le header HTTP.
  - L'image seule : la propriété "Accept" avec la valeur "image/png" ajoutée dans le header HTTP ;

Méthode : PUT

- PUT est utilisée pour uploader une nouvelle image sur le server.
- La requête PUT retourne le code réponse 201 (Created) et l'identifiant de la ressource créée.
- PUT est activée sur /images/.

Méthode : POST

- POST est activée pour modifier la ressource spécifiée.
- La requête POST retourne le code réponse 200 (OK) et la représentation mise à jour des métadonnées de la ressource modifiée.
- POST est activée sur /images/{id}.

Méthode : DELETE

- DELETE est activée pour supprimer une ressource sur le server.
- La requête DELETE retourne le code réponse 202 (Accepted).
- DELETE est activée sur /images/{id}.

Méthode : GET

- GET est activée sur /images/index/.
- GET est activée pour retrouver la liste des ressources.

Méthode : GET

- GET est activée sur /images/{id}.
- GET est activée pour retrouver une représentation de la ressource spécifiée.
- La requête retourne le code 400 en cas d'erreur.
- si Accept = image/png la requête retourne l'image.
- si Accept = text/json la requête retourne les métadonnées.

Exemple : GET

- Requête :

```
GET /images/propose HTTP/1.1
Host: http://localhost:8182
Accept: text/json
Apache-HttpClient/4.3.2 (java 1.5)
```

Exemple : GET

- Réponse :

```
Server: Restlet-Framework/2.3.1
HTTP/1.1 200 OK
Date: Sat, 02 May 2015 11:08:44 GMT
Content-Type: text/plain; charset=UTF-8
Content-Length: 58
```

- Représentation

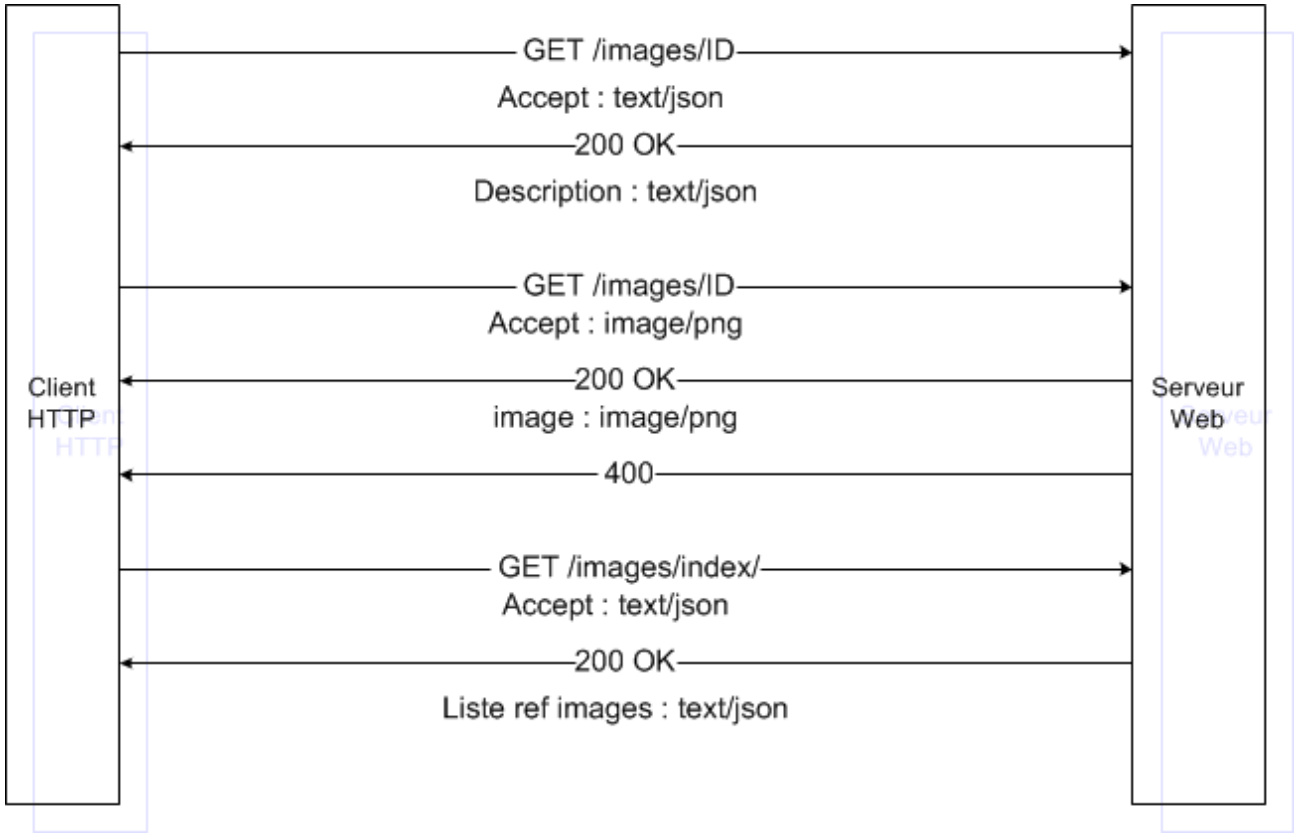
```
{"values":["json1","contractnet","propose","jasonobject"]}
```

Résumé de l'adressage

- Récapitulation

`/images/index/`
GET : liste des id des images
`/images/{ID}`
GET : ressource spécifiée par ID
Accept = image/png : image
Accept = text/json : métadonnes
POST : modifie les métadonnées de ressource ID
DELETE : supprime la ressource ID
`/images/`
PUT : ajoute une image

Exemple


Client HTTP
Serveur Web
GET /images/ID
Accept : text/json
200 OK
Description : text/json
GET /images/ID
Accept : image/png
200 OK
image : image/png
400
GET /images/index/
Accept : text/json
200 OK
Liste ref images : text/json

Sommaire

Classes nécessaires

- Application lançant le serveur Rest
- Contrôleur Web qui gère les réponses aux requêtes sur les différentes adresses.
- Bean de customisation

Lancement du serveur

```
@SpringBootApplication
public class Application {
  public static void main(String[] args) {
   SpringApplication.run(Application.class, args);
  }
}
```

Customisation

Il s'agit d'une classe annotée par `@Component`.
Elle implémente `WebServerFactoryCustomizer`.
Elle permet de spécialiser le serveur Rest (numéro de port).

Customisation - Bean

```
@Component
public class ServerPortCustomizer implements
 WebServerFactoryCustomizer
                        <ConfigurableWebServerFactory> {
 @Override
 public void customize
                (ConfigurableWebServerFactory factory) {
  factory.setPort(8182);
 }
}
```

Contrôleur

Il s'agit d'une classe annotée par `@RestController`, indiquant qu'elle peut être utilisée par Spring MVC pour répondre aux requêtes web.
`@RequestMapping` permet d'indiquer les méthodes qui répondent aux requêtes suivant une adresse particulière. La méthode retourne une chaîne (directe ou après sérialisation JSON d'un objet).
`@RestController` combine `@Controller` et `@ResponseBody`, deux annotations qui indiquent les réponses aux requêtes Web sont des données plutôt que des view.

Contrôleur : structure

```
@RestController
public class SpringImageResourceController extends
            SpringImageBase {
 @RequestMapping(method = RequestMethod.GET,
    value = Constants.IMAGES_INDEX)
 public ResourceList retrieveResourceNames() {
  return new ResourceList(docs.keySet());
 }
 // ResourceList est une classe encapsulant une
 // liste de chaînes. Un objet ResourceList est
 // automatiquement transformée en JSON

 // Méthode pour d'autres adressages
 ...
}
```

SpringImageBase

```
protected HashMap<String, ImageDescription> docs;
// docs est une map permettant de gérer "la base de
// données images"
// ImageDescription est une classe encapsulant les
// champs servant à la description des images
```

Contrôleur : structure

```
@RestController
public class SpringImageResourceController extends
            SpringImageBase {
 @RequestMapping(method = RequestMethod.GET,
   value = "images/{id}",
   headers = "Accept=application/json")
 public ImageDescription
  retrieveResourceDescription(@PathVariable(
   value = "id") String imId) {
  if (!docs.containsKey(imId)) {
   return new ImageDescription();
  }
  return docs.get(imId);
 }
}
```