Introduction aux services web

Alexandre Niveau

GREYC — Université de Caen

Adapté du cours de Jean-Marc Lecarpentier

Service web

- Programme qui tourne sur un serveur web
- Fournit un service à des utilisateurs « machines »
- Comme une application web, mais utilisable par des *programmes*

Utilisation d'un service web

- Les services web sont utilisés par les sites web pour améliorer l'expérience utilisateur, ou pour faire des widgets
- Quelques exemples :
  - page d'un magasin intégrant un plan Google Maps
  - page décorée automatiquement avec des photos issues de Flickr
  - page personnelle intégrant les derniers tweets de l'auteur
  - outil de recherche intégré à la page mais fourni par Google ou Yahoo

Service Oriented Architecture

- Architecture orientée service : modèle d'application qui met en œuvre des services complètement indépendants mais interopérables
- Les services sont des briques de base qui permettent de créer des services plus complexes (qui peuvent à leur tour être utilisés comme briques de base)
- Dans cette définition, un web service est :
  - un composant métier
  - autosuffisant
  - exécuté sur le web
  - remplissant un contrat (format standard pour les requêtes et réponses, API et protocole connus)
- Recommandation W3C sur l'architecture des web services [http://www.w3.org/TR/ws-arch/]

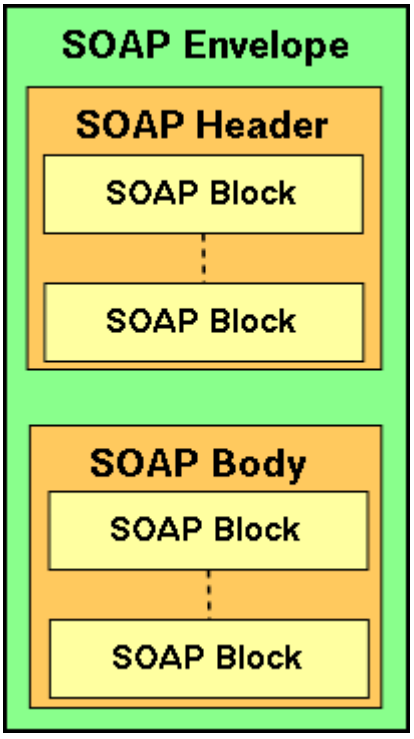
Standards pour les services web

- Dans le cadre d'une architecture orientée services, nécessité de standards pour que les services soient interopérables
- Standards du W3C :
  - XML pour le format des données
  - SOAP pour le format des requêtes
  - WSDL pour la description de l'interface du service

SOAP

- Simple Object Access Protocol
- Protocole pour l'envoi des messages
- Basé sur XML
- Au-dessus de la couche du protocole de transmission (généralement HTTP ou SMTP)
- Créé par Microsoft en 1998, puis proposé comme standard au W3C (recommandation W3C) [http://www.w3.org/TR/soap/]

Principe de SOAP



Exemples de messages SOAP (service de recherche Google, supprimé)

- requête SOAP [demo/reqSoap.xml]
- réponse SOAP [demo/repSoap.xml]

WSDL

- Après la standardisation de SOAP : volonté de développer une architecture orientée services
- Il fallait pouvoir définir les API des services de façon standard
- Septembre 2000 : WSDL, Web Services Description Language, développé notamment par IBM et Microsoft, puis standardisé par le W3C [http://www.w3.org/TR/wsdl20/]

- Quoi : modèle du service
- Comment : lie le modèle à son implémentation
- Où : point d'accès au service
- Exemple [demo/GoogleSearch.wsdl] de Google (service supprimé)

SOAP et PHP

- Classe `SoapClient` [http://fr.php.net/manual/en/class.soapclient.php]
- Peut être instanciée à partir d'un fichier WSDL
- Méthode = fonction de service interrogé
- `resultElements` est un tableau composé des résultats (les données XML ont été parsées)

Un client SOAP

```php
<?php
try{
    $options = array(
        'proxy_host' => "proxy.info.unicaen.fr", 'proxy_port'=> 3128,
        'trace' => true);
    $client = new SoapClient("GoogleSearch.wsdl", $options);
}
catch(Exception $e) { echo $e->getMessage();}

$cle = "FjHwko9QFHL/EYmp3hblb/3YsAMQR/Ys";
$res = $client-
>doGoogleSearch($cle, "ajax", 0, 10, false, "", false, "", "", "");
$tab  = $res->resultElements;

for ($i=0; $i < sizeof($tab); $i++) {
  echo $tab[$i]->title ." à l'url ".$tab[$i]->URL ."<br>";
}
?>
```

Avantages et inconvénients de SOAP

- SOAP est un protocole standard, indépendant du protocole de transport utilisé
- Développé pour être extensible autant que possible...
- ... ce qui le rend naturellement lourd à manipuler
- Couplage fort entre client et serveur : modification de l'API implique une modification chez le client
- Conséquence : perte de vitesse depuis des années au profit de REST

REST

- REST : Representational State Transfer
- Style d'architecture, pas un protocole ou un format

- Ensemble de contraintes, proposées par Roy Fielding dans sa thèse en 2000
- Un système est conforme (« RESTful ») s'il respecte les contraintes
- Objectifs : simplicité des interfaces, portabilité, scalabilité, fiabilité à long terme

Contraintes de REST

- Modèle client-serveur
- Sans état : les requêtes sont indépendantes
- Possibilité d'utiliser un cache pour les réponses
- Interface uniforme séparant le client et le serveur

Interface uniforme

- C'est un des points fondamentaux, qui permet le découplage strict entre client et serveur
- En particulier, l'architecture REST manipule des *ressources*, qui doivent
  - avoir un identifiant unique (ex. URI)
  - être manipulables par les clients via des représentations bien définies (ex. HTML, JSON…)
- Les messages :
  - doivent être autodescriptifs, i.e. contenir une explication de comment les utiliser
  - doivent contenir des hyperliens indiquant ce qu'il est possible de faire ensuite

REST en pratique

- Exemple d'architecture REST : *le web*, en tant que système de diffusion de documents (implémenté par la combinaison HTTP, HTML, URI)

  ➤ mais les *services* web ne sont pas forcément conformes
- Le concept de REST a été très à la mode dans les années 2000, comme solution à la lourdeur de SOAP/WSDL
- Souvent mal interprété : nombreux services web basés sur HTTP autoproclamés RESTful pour des raisons marketing
- Au sens large, on parle de REST quand la requête est faite via HTTP GET/POST (exemple [https://developers.google.com/custom-search/json-api/v1/using_rest])

Conception d'une API REST (1)

- Dans un sens plus strict, l'API d'un service REST a les caractéristiques suivantes (dans l'ordre des priorités)
  - utilise des URI bien choisies qui représentent des *ressources* :

```
/users/bob/messages/354
```

en exploitant l'aspect hiérarchique, chaque composant du chemin ayant une sémantique et n'utilise les paramètres que pour filtrer ou rechercher:

```
/users/bob/messages/?date=2015-02-23&sort=title
/users/?q=bob
```

- utilise les verbes HTTP pour les actions :
  - `GET /users/bob` renvoie Bob
  - `GET /users/` renvoie la liste des utilisateurs
  - `POST /users/` ajoute un nouvel utilisateur
  - `PUT /users/bob` modifie Bob
  - `DELETE /users/bob` supprime Bob

Conception d'une API REST (2)

- utilise les codes de statut HTTP pour indiquer le résultat de la requête :
  - les habituels 200 OK, 403 Forbidden, 404 Not Found, 500 Internal Server Error, etc.
  - mais aussi 201 Created, 405 Method Not Allowed, 409 Conflict, etc.
- plus rarement, utilise des liens dans les réponses pour une API auto-descriptive et découvrable :

```
<message>
  <text>bla bla bla</text>
  <link rel = "self"
        uri = "/users/bob/messages/354" />
  <link rel = "prev"
        uri = "353" />
  <link rel = "next"
        uri = "355" />
  <link rel = "/linkrels/message/new"
        uri = "../"
        method = "POST" />
</message>
```

Sémantique du lien donné par l'attribut `rel`, pour laquelle il existe des valeurs standard [http://www.iana.org/assignments/link-relations/link-relations.xhtml] (on peut spécifier une valeur non-standard avec une URI).

Interroger un service REST

- Un service REST s'interroge par une URI avec les paramètres nécessaires
- Faisable directement dans un navigateur par un humain (pour les requêtes GET)
- En Javascript, utilisation de XMLHttpRequest
- En PHP, utiliser CURL [https://php.net/manual/fr/book.curl.php]
- Résultats en XML ou JSON généralement

- En PHP, utiliser SimpleXML [http://fr2.php.net/manual/en/book.simplexml.php] et json_decode [http://fr2.php.net/manual/fr/function.json-decode.php]

Exemple en REST

- Service web de recherche de localités : geonames-search [http://www.geonames.org/export/geonames-search.html]
- Exemple de requête : http://api.geonames.org/search?q=caen&maxRows=10&username=demo [http://api.geonames.org/search?q=caen&username=unicaen1]
- Et en JSON : http://api.geonames.org/searchJSON?q=london&maxRows=10&username=demo [http://api.geonames.org/searchJSON?q=caen&maxRows=10&username=unicaen1]

Démo : client PHP très simple interrogeant Geonames [demo/phpClientREST.php]

- Autres exemples : Google Books API [https://developers.google.com/books/docs/v1/using], Twitter [https://dev.twitter.com/rest/public], Flickr [http://www.flickr.com/services/api/flickr.photos.search.html], API Wikipédia [http://en.wikipedia.org/w/api.php]…
- Exemple de classe PHP interrogeant Flickr [demo/Flickr.php] (pas exécutable sans clef)

Spécifications et normes

- RFC 5988 — Web linking [http://tools.ietf.org/html/rfc5988]

Références et guides

- Representational State Transfer [http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm] (chapitre de la thèse de Roy Fielding)

Tutoriels

- A Beginner’s Guide to HTTP and REST [http://code.tutsplus.com/tutorials/a-beginners-guide-to-http-and-rest--net-16340]
- Best Practices for Designing a Pragmatic RESTful API [http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api]
- RESTful web services: a tutorial [http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069]
- REST API quick tips [http://www.restapitutorial.com/lessons/restquicktips.html] (voir aussi la suite du tutorial)
- Learn REST: a tutorial [http://rest.elkstein.org/]
- Using the Google Maps API to add cool stuff to your applications [http://blog.smartbear.com/how-to/using-the-google-maps-api-to-add-cool-stuff-to-your-applications/]

Lectures complémentaires

- A Brief Introduction to REST [http://www.infoq.com/articles/rest-introduction] (services web et principes théoriques)
- Richardson Maturity Model [http://martinfowler.com/articles/richardsonMaturityModel.html] Explication très claire et progressive des principes de REST
- REST et HATEOAS [http://timelessrepo.com/haters-gonna-hateoas]
- Why trailing slashes on URIs are important [https://cdivilly.wordpress.com/2014/03/11/why-trailing-slashes-on-uris-are-important/]
- The S stands for Simple [http://web.archive.org/web/20150314044423/http://wanderingbarque.com/nonintersecting/2006/11/15/the-s-stands-for-simple/]

Outils

- Services web GeoNames [http://www.geonames.org/export/ws-overview.html]
- API Flickr [https://www.flickr.com/services/developer/api/]
- Open Movie Database API [http://www.omdbapi.com/]

 [http://creativecommons.org/licenses/by-nc-sa/4.0/]