Avant-propos

Organisation de ce livre

Ce livre présente un aperçu des services web que l'on peut concevoir dans le style d'architecture REST. Plutôt que de se focaliser sur un framework particulier, nous mettrons en lumière les principes de l'architecture, les bonnes pratiques associées et comment tirer parti au mieux des protocoles pour concevoir des applications et tenir compte de la latence, des caches, de la montée en charge, etc.

> **Avertissement**
>
> Ce livre n'a pas la prétention d'être une référence sur REST, ne serait-ce que par son format, mais il donne un tour d'horizon des concepts de base et des apports de ce style d'architecture. Le lecteur averti devra nous pardonner d'avoir simplifié légèrement certains concepts – le prix de la concision.

Après une introduction générale, nous verrons au **chapitre 2**, sur un exemple minimaliste comment concevoir une application selon les principes REST, et quel en est l'impact sur la structuration des données, sur la lecture ou la mise à jour des informations.

Au **chapitre 3**, nous reviendrons sur REST et sur certains points d'architecture spécifiques, en étudiant comment tirer parti au mieux de HTTP et des standards associés. Nous verrons comment une utilisation soigneuse du protocole permet de bénéficier d'une architecture de cache, de gestion des versions, et d'une meilleure montée en charge.

Nous verrons au **chapitre 4** dans le détail quelques principes simples d'implémentation pour exploiter facilement les caches, la distribution, ou le contrôle de version. Bien sûr, nous y aborderons également les grands écueils classiques.

Au **chapitre 5**, nous explorerons une application REST existante, l'API Google Contacts, et nous analyserons comment les concepteurs de cet outil ont mis en oeuvre les concepts REST.

Nous conclurons par une check-list méthodique, avant de proposer quelques pistes et références bibliographiques.

Remerciements

Je tiens à remercier, pour leur relecture attentive et leurs conseils, Muriel Shan Sei Fan, Jean Zundel, Luc Heinrich, Sébastien Tanguy, Loïc Ségalou, et Véronique Heinrich.

Le Web pour les humains – le Web pour les machines

Ma journée démarre : j'ouvre mon navigateur web, je pars butiner quelques blogs du matin. Un billet de l'un de mes auteurs favoris suggère la lecture d'un autre billet d'un inconnu. Je file le lire, et commence à parcourir les archives du mois de ce nouveau blog. Je trouve un lien sur la page personnelle de l'auteur, je passe sur la liste de ses publications.

Pendant ce temps, une application sur ma machine se connecte sur un site de photos, télécharge la liste des albums auxquels je suis abonné, récupère la liste de commentaires récents sur une de mes photos, trouve et charge dans un de ces commentaires une image que l'on me conseille d'aller voir, puis revient sur les informations de mon compte, vérifie mon quota, et met en ligne les dernières photos que j'ai stockées sur ma machine.

Quoi de commun entre mon activité et celle de mon programme ? Superficiellement, pas grand-chose. Je suis dans mon navigateur et je passe de page en page, et l'application travaille silencieusement à synchroniser des données assez hétérogènes.

Et pourtant…

Mon programme et moi sommes finalement en train de faire exactement la même chose : parcourir le Web, naviguer d'une information à l'autre en suivant des liens, en prenant des décisions à chaque étape sur quoi aller voir ensuite. C'est-à-dire présenter une requête à un serveur sur une URL donnée, attendre sa réponse. Examiner le contenu, y trouver des liens. Suivre un ou plusieurs de ces liens, c'est à dire refaire une requête sur une autre URL, peut-être sur un autre serveur, attendre la réponse, etc.

Mon navigateur me présente des données dans différents formats : images, texte, et même d'autres sans que je m'en aperçoive, comme un peu de javascript pour me faciliter - ou compliquer - ma navigation. L'application navigue dans la même toile de liens, téléchargeant parfois du XML, parfois un peu de HTML ou de JSON, parfois des formats plus spécialisés comme ATOM ou GData.

Table des matières

1 Introduction

Ce livre traite exactement du sujet suivant : comment faire pour que les services web et les programmes qui les utilisent aient la même souplesse de navigation dans l'information que tout internaute dans son navigateur web ? Comment utiliser les mêmes principes ? On verra que les bonnes pratiques du web « humain » doivent se retrouver lorsqu'on conçoit des services web en REST.

Les services web : appel de procédure ou exploration d'espace ?

Imaginons un service web qui nous permette d'interroger ou de modifier un carnet d'adresses, sur le Web, mais aussi *via* un client spécialisé. Prenons un premier exemple en examinant deux façons d'écrire une URL pour accéder à la fiche d'une hypothétique amie, Ada Veen :

Quelle différence entre ces deux URL :

```
http://carnet-rest.com/api?method=findcard&userid=aveen&sessionid=0679725229
```

et

```
http://carnet-rest.com/cards/ada-veen
```

Du point de vue de l'implémenteur, probablement pas grand-chose. L'une comme l'autre se trouvent renvoyer la même information : une fiche dans deux services virtuels de carnet d'adresses.

Du point de vue de l'architecte, la différence est déjà plus nette : la première semble exhiber directement un choix d'implémentation, en l'occurrence un appel de méthode sur un service distant. On imagine aisément être directement en train d'interroger un objet Java, C#, Ruby, etc. On utilise une technologie web, HTTP, comme simple transport pour interroger directement un programme.

Dans le second cas, on voit bien que le concepteur du service a conçu son service un peu différemment : on a moins l'impression de parler directement à une machine, d'invoquer une action (rechercher une fiche). On est face à une URL qui semble traduire directement un concept : la fiche d'une personne nommée Ada Veen.

Dans ce dernier cas, l'implémentation sous-jacente est bien sûr moins apparente, mais la différence fondamentale est que dans le premier cas, on exprimait via cette URL une action, alors qu'ici on se focalise sur l'information, le concept.

Cette distinction est révélatrice de deux grandes approches pour concevoir un service web. L'une exprime un concept finalement assez ancien : l'appel de procédure à distance. L'autre se focalise sur des concepts à explorer.

L'une suppose que le client a une connaissance exacte de la structure des URL qu'il faudra demander au serveur – et probablement du format de réponse – en encapsulant un appel de procédure à distance dans un protocole qui se trouve être ici HTTP. L'autre donne un point d'entrée dans un espace que l'on partira explorer, en exploitant des réponses dans des formats bien connus.

Ce principe d'exploration *via* des liens a un autre impact fondamental. Il permet de mettre en place une architecture où le serveur n'a pas à connaître le client *spécifique* auquel il s'adresse. Comme le client ne fait que suivre des liens, l'un après l'autre, la notion « d'état » du client n'a même pas à être connue par le serveur : c'est le client qui sait où il est, et où il va aller à l'étape suivante.

REST (*Representational State Transfer*) définit ce style d'architecture logicielle.

JARGON REST et ses acceptions

Le terme REST est d'ailleurs parfois un peu dévoyé et, le succès aidant, certaines architectures sont un peu trop rapidement qualifiées de « REST » alors qu'elles se contentent d'adopter a minima tel ou tel point de ce style d'architecture. C'est au point où parfois, le qualificatif de « REST » n'est utilisé qu'en creux, par opposition à d'autres caractéristiques : « cet outil n'est basé ni sur SOAP, ni sur XML-RPC et comme il utilise HTTP, il est donc REST ! ». Je caricature, mais j'espère montrer dans les pages à venir que REST… c'est bien plus que cela.

REST, un style d'architecture

REST repose sur les standards qui fondent l'infrastructure du Web et c'est là son paradoxe : on ne peut pas pour autant en écrire de spécification précise. REST est un *style* d'architecture, pas une architecture bien précise et concrète.

Les services web conformes à REST (*RESTful*) sont implémentés par dessus un jeu classique et connu de technologies :

- des bibliothèques de code implémentant HTTP,
- des serveurs web pour traiter des requêtes,
- des frameworks pour faciliter l'écriture d'une application,
- des systèmes de « templates » pour générer des vues XML,
- des spécifications de types de contenus,
- etc.

L'examen des différentes constructions logicielles qu'on peut former avec ces différentes briques fait prendre conscience que, parmi toutes ces architectures possibles, certaines ont des propriétés vraiment intéressantes – qui s'inspirent des bonnes pratiques et de la philosophie générale du Web. Bref, REST n'est *que* cela :

1. une famille d'architecture de services web,
2. une façon de représenter des ressources web,
3. une certaine manière de les décliner, de les manipuler,
4. l'usage de quelques principes simples pour parcourir des données dans une application,
5. et, finalement, la décision de voir les services web – ou le Web d'une façon plus large – comme un style spécifique d'écriture d'applications, de structuration de l'information.

Bien sûr, toute architecture vient avec ses présupposés, son domaine d'excellence : REST n'est pas la réponse absolue à tous les problèmes d'architectures de services web, et certains problèmes n'y trouvent pas toujours d'expression élégante (en particulier les questions de transactions, de time-out, de ressources critiques, etc.).

Mais penser une application dans le style REST, c'est reconnaître qu'organiser l'information sur le Web est un travail à part entière ; c'est reconnaître que le Web est plus que la somme de ses parties, plus qu'une collection de standards et de spécifications.

REST est le retour aux fondamentaux du Web.