Services Web

Présentation

Question : Peut-on envisager le développement d'une application qui recense tous les fichiers se trouvant dans une arborescence et en présente l'inventaire et les caractéristiques dans un fichier tableur (Ms-Excel ou Calc) ?
Réponse : Oui.
Question : Quelles sont les possibilités techniques pour le faire ?
Réponse : Il existe au moins deux manières de procéder:

- **Utiliser les capacités du tableur à être programmé**. On peut, dans ce cas, stocker le programme dans le fichier tableur lui-même et s'appuyer sur la possibilité qu'il offre d'explorer une arborescence de fichiers pour lui faire créer une feuille de calcul à chaque exécution. Celle-ci sera remplie avec les données collectées par l'exploration. Cette possibilité ne nous intéresse pas ici ;
- A partir du moment où le tableur fourni une API (ex. : API LibreOffice) qui le rend accessible par programme, on peut **utiliser l'environnement de programmation de son choix pour peu qu'il soit interfaçable avec cette API**. Ce scenario est particulièrement intéressant puisqu'il permet de mesurer qu'un logiciel destiné à une interaction avec un utilisateur peut, via une API, être aussi conçu pour fonctionner dans une interaction avec un autre programme ;

Définition

Développer un **service Web**, c'est créer une API pour une ressource disponible sur Internet. Autrement dit, c'est **rendre le Web accessible par programme**, sans la nécessité de passer par un navigateur. Ce qui est devenu un besoin fondamental avec le Web 2.0 :

- Prise en compte des équipements mobiles pour lesquels le navigateur n'est pas un bon outil ;
- Besoin d'intégrer des données issues de ressources multiples sur Internet ;
- Besoin de mieux répartir les traitements entre client et serveur ;
- Besoin de mieux prendre en compte la mobilité qui génère des temps hors-connexion ;

Ajax permet déjà cela, mais l'idée ici est de pouvoir le faire sans le **XmlHttpRequest** qui n'est accessible que dans un navigateur. Il y a donc un travail particulier à faire côté serveur et côté protocole de communication. Les premières implémentations de services web étaient très ambitieuses et s'appuyaient sur des protocoles dédiés de type "usine à gaz", utilisant des échanges XML (ex. **SOAP** ou XML-RPC).

L'expérience et le besoin de simplicité ont amené à la généralisation d'un standard nommé **REST** (REpresentational State Transfer) qui n'utilise rien d'autre que le potentiel existant du protocole **HTTP** couplé à des conventions de construction pour un usage homogène et cohérent.

Principes de l'architecture REST

Règles et conventions

- **L'URL adresse la ressource**. Comme une ressource correspond généralement à un ensemble de données de même nature (livres, personnes, lieux, etc.), on exposera cette ressource par un nommage au pluriel. Par ex., **http://web.service.bzh/villes** qui renverra toutes les villes. Alors, l'accès à une ville particulière se fera au travers de la même ressource restreinte à l'identifiant de la ville : **http://web.service.bzh/villes/154**. De même on pourra accéder aux personnes en lien avec une ville en utilisant cette adresse : **http://web.service.bzh/villes/154/personnes**, et ainsi de suite ;

> Les besoins de filtrage et de tri seront réalisés au moyen d'une Query String

- **Les verbes HTTP adressent l'opération**. Au sens des données, il y a quatre opérations à assumer. Les verbes du protocole HTTP seront leur support :
  - Insertion (Create) ⇒ POST
  - Consultation (Read) ⇒ GET
  - Mise à jour (Update) ⇒ PUT
  - Suppression (Delete) ⇒ DELETE

> Ainsi, le verbe POST associé à l'URL http://web.service.bzh/villes permettra l'insertion d'une nouvelle ville dont les caractéristiques seront transmises en Query String.