# TP 8 : systèmes répartis à grande échelle
## Web Services — Communications avec le protocole SOAP

Alexandre DENIS
denis@labri.fr

Systèmes et supports d'exécution — 2006–2007

L'objectif de ce TP est de manipuler *g*SOAP, une implémentation de SOAP qui propose une projection vers le langage C. À titre d'exemple, vous vous connecterez à l'interface *Web Services* de Google, puis vous réaliserez votre propre serveur de calcul.

## 1 Avant de commencer

Nous utiliserons la « Google Web API », sujette à licence particulière. L'utilisation gratuite de ce service est permise à titre de démonstration. Pour se protéger des abus, Google demande donc à ce que les requêtes soient authentifiées par un numéro de licence ; chaque licence permet au maximum 1000 requêtes par jour au service Google Web API.

### 1.1 Obtenir une licence Google Web API

**L'obtention de la licence demande une authentification par échange de mails qui prennent entre 5 et 10 minutes. Demandez immédiatement votre licence avant même de lire la suite du sujet !**

Connectez-vous sur `http://www.google.com/apis/soapsearch/` et cliquez sur « *create a Google Account* » (étape 2). Suivez les étapes jusqu'à l'obtention d'un numéro de licence.

**Remarque :** si vous avez déjà un compte Google (GMail, Google Groups, etc.), vous pouvez réutiliser le même.

### 1.2 Compiler *g*SOAP

Pour passer le temps en attendant la licence Google, téléchargez *g*SOAP et compilez-le. N'oubliez pas de donner un chemin d'installation raisonnable à `./configure` à l'aide de l'option `--prefix`. En cas de doute sur la procédure d'installation, consultez son fichier `README.txt`. La compilation prend seulement 3 minutes.

## 2 Les services web

**Présentation des services web.** Les services web (ou *Web Services* en anglais) sont une architecture de système réparti à grande échelle construite autour du protocole de communication SOAP (*Simple Object Access Protocol*). SOAP est un protocole d'invocation de procédure à distance basé sur un encodage des messages en XML, et un transport par HTTP. Les invocations se font sur des *services*, entités semblables à des objets mais sans état interne. Les différentes problématiques prises en charge sont :



**hétérogénéité** — la représentation des données en XML permet de s'affranchir de toute représentation binaire (32 bits v.s. 64 bits, *little endian* v.s. *big endian*, etc.) ;

**interopérabilité** — la spécification porte sur le protocole « sur le fil ». Deux implémentations différentes faites par des éditeurs différents sont donc capable de se parler et de se comprendre si elle respectent le standard ;

**connectivité** — l'utilisation du protocole de transport HTTP permet à SOAP de franchir les *firewall* en empruntant les proxies existants pour l'accès web ;

**sécurité** — il existe une variante de SOAP sur HTTPS (HTTP sur SSL) qui garantit confidentialité et authentification ;

**localisation** — l'architecture des services web comprend un annuaire (UDDI — *Universal Description, Discovery and Integration*) pour localiser les serveurs. Nous n'utiliserons pas d'annuaire UDDI dans ce TP.

**Description des interfaces en WSDL.** L'interface des services mis à disposition est publiée en format WSDL (*Web Service Definition Language*), qui est un dialecte XML. Ouvrez le fichier `GoogleSearch.wsdl` dans votre éditeur favori[1] et observez les informations qu'il contient :

**définition** du service, avec son nom, les espaces de noms, et les références des standards auxquels le document obéit ;

**les types** manipulés par l'interface décrite. On remarquera au passage que la description des types se fait avec le système XSD qui est déjà utilisé dans les schémas XML ;

**les formats de messages** aussi bien les messages correspondant aux requêtes qu'aux réponses (suffixés en `-Response`) ;

**les opérations** qui associent un nom d'opération, un type de format de requête et un type de format de réponse ;

**l'adresse du service** — en l'occurrence `http://api.google.com/search/beta2`.

Faire une requête SOAP consiste simplement en une requête HTTP de type `POST` contenant le message décrivant la requête en XML ; la réponse à la requête HTTP représente le résultat de la requête SOAP.

### 2.1 Premier contact avec SOAP

L'atout principal de SOAP est qu'il est simple et construit sur des standard (HTTP, XML) pour lesquels il existe déjà de nombreux outils ce qui le rend apte à une intégration rapide dans les programmes.

Cependant, pour notre premier exemple (qui n'a rien de représentatif), nous allons tout faire *à la main*, pour voir ce qui se passe.

**Étape 1 : ouverture d'une connexion.** Puisqu'il s'agit du protocole HTTP, le port à utiliser est le 80. Dans un terminal, tapez :

```
telnet api.google.com 80
```

**Étape 2 : en-têtes HTTP.** Ce sont les en-têtes classiques d'une opération `POST`.

```
POST /search/beta2 HTTP/1.0
Host: api.google.com
Content-Type: text/xml; charset=utf-8
Content-Length: 943
SOAPAction: "urn:GoogleSearchAction"
```

[1] puis tapez `Meta-X sgml-mode` si votre éditeur favori est `emacs`.

**Remarque :** adaptez le contenu du champ `Content-Length` à la taille de votre message (la taille du *corps* du message, caractères de sauts de lignes inclus). Vous pouvez compter large puis compléter avec autant de lignes blanches que nécessaire pour que le serveur accepte votre requête.

**Remarque 2 :** comme il vous faudra plusieurs essais pour que ça marche, préparez le message complet dans un éditeur, et procédez par copier/coller.

**Remarque 3 :** le serveur web ferme la connexion après chaque essai. Répétez l'étape 1 avant chaque tentative !

**Étape 3 : corps du message de la requête.** Le corps de la requête est exprimé en XML. Inutile d'apprendre toute la syntaxe, copiez-collez l'exemple suivant et modifiez le champ `key` pour insérer votre propre numéro de licence. (et éventuellement modifiez le champ `q` qui contient la ***q***uestion, au cas où vous ne vous intéresseriez pas au célèbre groupe de rock californien).

```
<?xml version='1.0' encoding='UTF-8'?>

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doGoogleSearch xmlns:ns1="urn:GoogleSearch"
         SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <key xsi:type="xsd:string">INSEREZ-VOTRE-NUMERO-DE-LICENCE-ICI</key>
      <q xsi:type="xsd:string">toto</q>
      <start xsi:type="xsd:int">0</start>
      <maxResults xsi:type="xsd:int">10</maxResults>
      <filter xsi:type="xsd:boolean">false</filter>
      <restrict xsi:type="xsd:string"></restrict>
      <safeSearch xsi:type="xsd:boolean">false</safeSearch>
      <lr xsi:type="xsd:string"></lr>
      <ie xsi:type="xsd:string">latin1</ie>
      <oe xsi:type="xsd:string">latin1</oe>
    </ns1:doGoogleSearch>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**Remarque :** n'oubliez de laisser une ligne blanche entre les en-têtes HTTP et le corps du message XML (attention : une seule), et au moins deux lignes blanches à la fin du message.

**Debug.** Si la réponse est une erreur de type `The element type "SOAP-ENV :Envelope" must be terminated by the matching end-tag "</SOAP-ENV :Envelope>"`, cela signifie que le message a été tronqué, donc que votre calcul de `Content-Length` n'est pas bon. Si la réponse contient `Invalid authorization key` ou toute autre protestation en rapport avec l'interprétation du champ `<key>`, votre numéro de licence n'est pas bon.

**Résultat.** Au bout d'un nombre fini de tentatives, vous devriez obtenir une réponse qui commence par :

```
HTTP/1.0 200 OK
```

suivi des résultats en format XML, visiblement exploitables par un programme [2].

N'est-ce pas joli ?

Vous devriez commencer à saisir la puissance de l'outil (accéder à des services tels que Google à distance à partir de vos programmes) mais également les limites de l'approche « *à la main* ». Rassurez-vous, c'était la première et la dernière fois que vous écriviez directement les requêtes HTTP et XML.

[2] mais peut-être pas par un étudiant de master 2 :-)



## 2.2 La même chose avec *g*SOAP

Il est pénible d'implémenter le protocole HTTP à la main. Il est encore plus pénible de fabriquer les messages en XML et de parser les résultats.

Pour rendre plus simple l'utilisation de SOAP, il existe des *projections* vers les langages, c'est-à-dire la possibilité d'effectuer une requête SOAP par un simple appel de fonction C, le protocole étant géré intégralement par du code généré automatiquement. C'est ce que fait *g*SOAP. En SOAP, la façon de projeter le WSDL vers un langage de programmation n'est pas standardisée [3]. La façon de faire est spécifique à chaque implémentation.

À l'aide de l'outil `wsdl2h` de *g*SOAP, transformer votre `GoogleSearch.wsdl` en un fichier d'en-tête C. Par défaut, *g*SOAP génère du C++, pensez à utiliser l'option `-c` pour faire du C. Observez le fichier `GoogleSearch.h` ainsi généré. Il décrit la même interface mais en syntaxe C au lieu du WSDL. Intéressez-vous plus particulièrement à l'opération `doGoogleSearch` que vous utiliserez, et au type de ses arguments et valeurs de retour.

Consultez la documentation de *g*SOAP pour plus d'information sur la façon dont les types WSDL sont projetés vers des types C. En particulier, il faut garder à l'esprit les conventions suivantes :
- les types simples sont projetés vers leur équivalent « naturel » ;
- il existe certaines conventions particulières pour les types structurés : par exemple, un champ nommé `__ptr` immédiatement suivi d'un champ nommé `__size` dans une structure représente un tableau de longueur variable ;
- le double souligné (__) sépare un espace de nom (*name space*) d'un symbole ;
- les noms de fonction qui apparaissent dans ce fichier sont ceux utilisés côté serveur. Pour appeler ces fonctions côté client, il faut ajouter le préfixe `soap_call_` et en argument ajouter un pointeur sur une structure `soap` et l'adresse du serveur. Les indications figurent dans le fichier `GoogleSearch.h` généré.

Écrivez votre propre client Google capable de lancer une recherche et d'afficher les résultats de vos requêtes.

**Remarques :**
- générer le code des souches à l'aide de `soapcpp2`
- les en-têtes à inclure dans votre fichier source sont `soapH.h` et `GoogleSearchBinding.nsmap`
- compiler tous les `.c` générés par `soapcpp2`

# 3 Votre propre serveur calcul

Quand on prgramme à la fois le client et le serveur à l'aide de *g*SOAP, il est plus simple d'écrire directement l'en-tête C décrivant l'interface, sans passer par le WSDL, en respectant les conventions de projection. Attention : ce mode de fonctionnement est spécifique à *g*SOAP et n'est pas compatible avec les autres implémentations SOAP.

Notre objectif ici est de simuler un petit serveur de calcul, du type de Ninf ou Netsolve, mais en se restreignant aux vecteurs de réels, aux opérations simples, sans parallélisme ni équilibrage de charge.
- définir un type de vecteur de réels ;
- définir les interfaces des 3 opérations : adition, soustraction, multiplication ;
- implémenter la partie serveur ;
- écrire un client simple.

[3] alors qu'en CORBA, les projections sont standardisées