2010-2011

Applications Réparties T.D. 5
Client gSOAP en C++

1 Introduction

La boîte à outils gSOAP est destinée à de multiples plateformes de développement logiciel en C et C++ pour créer des serveurs et des clients de services Web. Les cibles peuvent être Windows en natif, Linux en natif, Mac-OS mais aussi Symbian et autre pour des cibles plus légères.

La boîte à outils gSOAP assure une faible empreinte mémoire. De nombreuses optimisations ont été appliquées pour réduire les besoins en ressources et pour accélérer l'analyse XML.

La boîte à outils gSOAP fournit un serveur Web HTTP(S) autonome (en plus d'un module Apache ou IIS).

Ce TD a donc pour objectif d'introduire les outils C/C++ gSOAP et leur mise en œuvre.

Pour commencer ce TD il vous faut récupérer les outils gSOAP sur SourceForge (http://sourceforge.net/projects/gsoap2), Vous devez télécharger la version stable 2.8.1 (gsoap_2.8.1.zip) et l'installer sur votre machine sous Windows.

2 Création d'un Service Web

Cette section a pour objectif d'introduire les outils C/C++ gsoap pour créer un service web et un client pour y accéder en C.

Pour utiliser les outils gSOAP, il vous faudra ensuite, sous un éditeur de commandes en ligne (ex. cmd.exe ou une terminal sous cygwin), rajouter le chemin vers les exécutable de gSOAP.

Editer dans un nouveau répertoire SOAP2, le fichier service.h suivant:

```
//gsoap ns service style: rpc
//gsoap ns service encoding: literal
//gsoap ns service location: http://localhost:8060
//gsoap ns schema namespace: urn:messg

int ns__test1( int code, char *message, struct ns__noResponse {} *out );
```

Nous allons mettre en place un service Web qui ne possède qu'une méthode dont le but est uniquement d'afficher coté serveur un texte un certain nombre de fois (int code).
Par défaut gSOAP est utiliser pour faire des applications CGI, cependant il est possible de l'utiliser pour construire des web server http « stand-alone » ou encore comme module dans Apache.

gSOAP utilise des directives au début du fichier d'inclusion afin de récupérer certaines informations comme le nom du service (messg), le type du service (rpc), la localisation (http://localhost:8060) ou encore l'espace de nommage utilisé (messg).

Utiliser la commande soapcpp2 pour générer l'ensemble des fichiers. N'oubliez pas les options –n pour forcer l'utilisation d'un espace de nommage, -c pour générer du code source exclusivement en langage C, et enfin –p afin de préciser le préfixe des fichiers générés.

Applications Réparties T.D. 5
Client gSOAP en C++

Un ensemble de fichiers sont alors générés dont :

messgC.c	Cœur de l'encodage/décodage des trames SOAP
messgServer.c	Partie serveur du service web
messgClient.c	Partie cliente du service web
messgServerLib.c	Pour générer une librairie avec un serveur SOAP
messgClientLib.c	Pour générer une librairie avec un client SOAP
messgH.h	Principal fichier d'inclusion du service
Messg.nsmap	Détail de tous les espaces de nommage utilisés dans le service
messg.wsdl	Description du service web dans le langage descritpif WSDL

3 Application serveur

La mise en place d'un serveur SOAP répond à la logique suivante (vous trouverez le détail de ces fonctions dans le document soapdoc2.pdf du répertoire gsoap\doc) :

- Initialisation du contexte SOAP avec *soap_init* (initialisation du runtime)
- Assignation d'un espace de nommage particulier avec *soap_set_namespaces* (nous avons utilisé l'option –n précédemment)
- Création de la socket mère de connexion sur le port 8060 avec *soap_bind*
- Dans une boucle de connexion :
  - On accepte une connexion cliente avec *soap_accept* (dans cette exemple, on rend l'attente non bloquante) ;
  - On traite la requête avec *messg_serve* (nous sommes dans l'espace de nommage « messg »)
  - On désalloue les données temporaires mémoire réservées pour le traitement de la requête avec *soap_destroy*
  - Fermeture de la connexion socket avec *soap_end*
  - On ferme les connexions sockets clientes et serveur et on enlève les fonctions callback avec *soap_done*.

Créez maintenant un fichier serveur.c pour la réception et le traitement des appels ainsi que l'implémentation du corps de la fonction distante ns__test1(…) sur le modèle ci-dessous. Complétez le code et en particulier le corps de la fonction ns__test1(…).

```
#include<stdio.h>
#include<stdlib.h>


// serveur SOAP
#include "messgH.h"
#include "messg.nsmap"

// permet de supporter plusieurs clients et plusieurs serveurs
struct Namespace *namespaces;

int ns__test1(
        struct soap *p_soap,          // contexte d'exécution du service web
        int p_code,                   // entier (code)
```

Applications Réparties T.D. 5
Client gSOAP en C++

```
        char *p_chaine,               // chaine de caractere (message)
        struct ns__noResponse *p_out  // pas de réponse (out)
) {

// A compléter
  }

int main(
      int p_argc,
      char *p_argv[]
) {
        struct soap v_soap;     // contexte du service SOAP

       // on initialise la socket
// A compléter
       // on assigne l'espace de nommage
// A compléter

        // on crée la socket mère de connexion

        if (soap_bind(&v_soap,NULL,8060,100) < 0) { return -1; }

        // on gère une boucle infinie pour recevoir les requêtes
        for (;;) {
                v_soap.accept_timeout = 1; // on rend l'attente non bloquante
                if (soap_accept(&v_soap) < 0) { continue; }
                messg_serve(  &v_soap );
                soap_destroy( &v_soap );  // dealloc data
                soap_end(     &v_soap );  // dealloc data and cleanup
                }
        soap_done( &v_soap );   // détache la strcut soap
  }
```

Sous Visual Studio, créez un projet « win32 application console ». Inclure dans ce projet le fichier serveur.c complété et l'ensemble des fichiers nécessaires à sa compilation (fichiers générés par gsoap spécifiques au service web considéré et les deux fichiers stdsoap2.c et stdsoap2.h de la bibliothèque gsoap/C++).

Soit : messg.nsmap, messgC.c, messgH.h, messgServer.c, messgStub.h, stdsoap2.c, stdsoap2.h, serveur.c
Attention vous aurez peut-être à gérer d'autres dépendances, par exemple avec la libwsock32.a.

Après compilation, exécutez votre projet serveur.exe et vérifier la présence du serveur Web sur http://localhost:8060.
Quel est le contenu du document XML qui vous est retourné ? De quelle erreur s'agit-il et pourquoi ?

4 Application cliente

Passons maintenant à la réalisation de notre client.

Créez et complétez le fichier suivant client.c :

```
#include "messgH.h"
```

Applications Réparties T.D. 5
Client gSOAP en C++

```
#include "messg.nsmap"

struct Namespace *namespaces;

int main(int p_argc,char *argv[]
) {

  struct ns__noResponse out;
  struct soap v_soap;

  soap_init( &v_soap );
  soap_set_namespaces( &v_soap, messg_namespaces );

// Appel de la méthode ns__test1 distante avec la fonction
// int soap_call_ns__test1(struct soap *soap, const char *soap_endpoint, const char
// *soap_action, int code, char *message, struct ns__noResponse *out);

  // A compléter

  if (v_soap.error) { soap_print_fault(&v_soap, stderr); }
  soap_end( &v_soap );
  return(0);
  }
```

Remarquons que l'appel à la fonction distante se fait au travers la fonction préfixée par soap_call_.... .
Le traitement des erreurs se fait au travers la variable v_soap.error et la fonction soap_print_fault.

Compilez le projet et exécutez *client.exe 3 « Bonjour »* . Que remarquez vous sur le serveur ?

5 Génération statique du Proxy avec gSOAP

Il faut commencer par récupérer le fichier de description WSDL du service web que nous allons utiliser pour les exercices suivants.

Il s'agit d'un service web gratuit de WebserviceX.NET : ConversionRate dont vous pouvez charger le fichier wsdl à l'url : __. Sauvegardez-le dans un fichier CurrencyConvertor.wsdl et dans un répertoire SOAP.

Générez à partir du WSDL du web service dans le répertoire SOAP du projet, le header WSCurrencyConv.h correspondant, grâce à la commande :

```
wsdl2h.exe -o WSCurrencyConv.h CurrencyConvertor.wsdl
```

Ce fichier n'est qu'un fichier intermédiaire qui sera utilisé pour générer tous les fichiers nécessaires à la classe proxy du service web.

Générez ensuite les fichiers nécessaires à la classe CurrencyConvertorSoap (Cf. pour un client, le fichier soapCurrencyConvertorSoapProxy.h et pour un serveur, le fichier soapCurrencyConvertorSoapObjet.h), grâce à la commande :

Applications Réparties T.D. 5
Client gSOAP en C++

```
soapcpp2 -I"……\gsoap\import" WSCurrencyConv.h
```

6 Déploiement en C++ dans Visual Studio

6.1 Un projet C++/MFC/DialogBox d'intégration :

Pour tester l'accès ce service web depuis un client gsoap, il nous faut créer un projet C++ /MFC/DialogBox.
L'objet de ce TD n'étant pas de développer une telle interface graphique, vous trouverez le code correspondant dans le fichier MyClient_Gsoap_C++_TD.zip, sous le répertoire CurrencyConvertor_TD. Il s'agit du projet CurrencyConvertor.dsp.

Compilez-le et testez-le en l'état.

6.2 Intégration du proxy vers le service Web :

Pour intégrer le proxy du service web, soapCurrencyConvertorSoapProxy.h, il faut inclure dans le projet l'ensemble de ses dépendances.

Il s'agit des fichiers suivants générés par gSOAP spécifiques au service web considéré :

CurrencyConvertorSoap.nsmap
soapC.cpp
soapClient.cpp
soapCurrencyConvertorSoapProxy.h
soapH.h
soapStub.h,

et des deux fichiers suivants des bilbiothèques gsoap/C++ :

stdsoap2.cpp
stdsoap2.h.

6.3 Classe CMyCurrencyConvertor

Dans la suite nous allons interconnecter le service web à l'interface graphique au travers une classe CMyCurrencyConvertor dont les fichiers se trouvent dans le répertoire « Classe CMyCurrencyConvertor » de MyClient_Gsoap_C++_TD.zip.

Il s'agit des fichiers MyCurrencyConvertor.h et MyCurrencyConvertor.cpp à compléter (Cf. commentaire dans les fichiers).

Les utilisations de cette classe dans l'interface graphique ont été commentées dans les fichiers CurrencyConvertorDlg.h et CurrencyConvertorDlg.cpp.

Décommentez pour tester votre application.

```
// App Rep SI4 TD4 GSOAP : TO ADD  //
//=====================================================================
```