Étude de faisabilité d'une application SOAP avec un système embarqué

Titre du projet
I04_05s01
Chaîne du froid: Communication par SOAP

Numéro du projet
53

École
HE-ARC ingénierie informatique

Temps imparti
Travail de semestre 2004-2005

Filière concernée par le projet
Téléinformatique

Chef de projet
François Tièche

Date
Dimanche 20 février

Candidat
Batiste Bieler

Résumé

Cette étude propose des solutions pour créer des services de communication Web avec un système embarqué fonctionnant avec le système d'exploitation GNU/Linux. L'étude devra apporter des réponses quant à la faisabilité d'une application utilisant le protocole SOAP sur ce système embarqué. Les logiciels choisis sont des logiciels libres sous licence GPL. Le langage C est requis pour le client alors que le langage Java est demandé pour le serveur.

Abstract

This study approaches solutions to create Web communication services with an embedded system running under GNU/Linux Operating System. The study will bring answers on the applicability of SOAP as protocol under an embedded system. On the one hand the server will use Java language on the other hand the client (embedded system) will use the C language. Selected software are under GPL licence.

Table des matières

1. Introduction

1.1. Généralitées sur le protocole SOAP

SOAP (Simple Object Access Protocol) est un protocole de communication provenant du monde XML. Il est utilisé dans le cadre des services Web.

Un service SOAP est une fonction que l'on peut appeller depuis un client sur un serveur distant. Cette fonction peut en théorie accepter n'importe quelle structure de donnée et donner une réponse tout aussi complexe. Il est également envisageable qu'il n'y ait pas de réponse.

SOAP ne fonctionne pas tout seul, il s'appuie sur un autre protocole pour fonctionner. À priori, n'importe quel protocole de communication peut faire l'affaire (HTTP, SMTP, FTP). Dans les faits, le protocole HTTP est le plus utilisé car il correspond bien aux besoins et aux architectures en place. Son fonctionnement avec le protocole SOAP est normalisé par le W3C ce qui en fait un standard solide.

SOAP implique souvent un dialogue qui se compose de l'appel à une procédure et d'une réponse de cette dernière. Des services sans dialogues sont également possibles.

Dans le cas où la programmation orientée objet est utilisée, l'architecture est parfois appellée instanciation à distance (*Remote Invocation*). L'idée qui se cache la derrière est de pouvoir accéder à des objets persistants ou non qui existent réellement sur le serveur distant et cela quel que soit le langage, l'implémentation ou l'architecture.

1.2. Pourquoi SOAP ?

Avec la multiplication des systèmes hétérogènes, l'interopérabilité devient un problème préoccupant. SOAP répond à ce problème en offrant un standard ouvert d'échange d'information pour l'informatique distribuée.

SOAP a été initialement défini par Microsoft et IBM, puis est devenu depuis une recommandation du W3C. Le soutien de ce protocole par les grands acteurs de l'informatique laisse à penser que SOAP va prendre de l'importance dans les années à venir.

1.3. But de ce rapport

Comme il n'existait pas de bon tutoriel en français sur le sujet, j'ai voulu créer une introduction pas à pas à l'utilisation du protocole SOAP et des outils client-serveur choisis.

Toutefois le contenu reste technique. Sans connaissances informatiques, on ne saisira

pas les exemples de code de ce rapport.

1.4. Structure du rapport

Pour commencer, vous trouverez une brève introduction aux messages SOAP HTTP ponctué d'exemples. Les types simples SOAP seront abordés ainsi que le langage WSDL (Web Service Deployment Descriptor).

Vous trouverez ensuite explication du rôle et de l'interaction du serveur *Apache Tomcat* et du *servlet Apache Axis*. Ceci sera suivi d'une procédure pas à pas pour l'installation de ces deux logiciels sur un système Linux.

Un service simple de calcul mathématique sera implémenté en Java et déployé sur Apache Axis. Un client C en GSOAP sera implémenté pour ce service mathématique afin d'illustrer le fonctionnement de cette boite-à-outils et son interaction avec les fichies WSDL.

Fort de l'expérience acquise, nous verrons comment réaliser un petit logiciel de rapatriement de fichiers binaires. Nous utiliserons ensuite des données complexes avec un service. Enfin, nous verrons comment réaliser un service et un client de sauvegarde de mesure.

Nous conclurons par les résultats de déploiement et de faisabilité de tels logiciels avec des systèmes embarqués

2. Introduction à SOAP avec le protocole HTTP

2.1. Le service MathSevice

Examinons le code Java suivant qui représente un service fort simple :

```
/**
* exemple MathService
*/
public class MathService
{
  /**
  * addition de deux entiers
  */
  public int add(int a, int b) {
    return (a + b);
  }

  /**
  * soustraction de deux entiers
  */
  public int subtract(int a, int b) {
    return (a - b);
  }
}
```

2.2. Forme d'un message SOAP

Tout message SOAP valide doit contenir l'élément `Envelope` qui est l'élément racine du message. Le corps `Body` est également incontournable dans un message SOAP. Le corps contient les informations utiles. On y trouve l'attribut `encodingStyle` qui spécifie les règles d'encodage utilisées pour sérialiser et désérialiser les données (transformer un flux paralllèle de donnée en flux série et inversement. Dans notre cas sérialiser signifie transformer les paramètres d'une fonction en un flux XML). En effet il n'existe pas une seule manière d'encoder les données dans un message SOAP. Cet attribut a donc son importance.

2.2.1. Requète d'un client SOAP en HTTP

Voici un message SOAP HTTP qui conviendrait pour l'appel de la fonction `add` de *`MathService`* avec comme paramètres 2 et 3 :

```
POST /MathService HTTP/1.0
Host: localhost
Content-Type: text/xml; charset="iso-8859-1"
Content-Length: 344
SOAPAction: ""

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

```
  <SOAP-ENV:Body>
    <m:add xmlns:m="MathService">
      <m:a type="xsd:int">2</a>
      <m:b type="xsd:int">3</b>
    </m:add>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

La méthode `add` fait partie de l'espace de nom `MathService`.

2.2.2. Les types de données en SOAP

Il est intéressant de noter que les paramètres de la fonction add sont de type entier. C'est un des types simple de SOAP. Il existe un grand nombre de types simples prédéfinis. Le tableau de correspondance suivant énumère quelque un des types en SOAP, Java, et le langage naturel :

SOAP	**Java**	**langage naturel**
xsd:base64Binary	byte[]	flux binaire
xsd:dateTime	java.util.Calendar	date
xsd:double	double	nombre réel à double précision
xsd:float	float	nombre réel
xsd:int	int	nombre entier
xsd:string	java.lang.String	chaîne de caractères

Si les types simples ne suffisent pas, il est possible de créer de nouveaux types. Ces types complexes sont les tableaux et les structures. Voici deux fragments de XML correspondants :

```
<nombres xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="xsd:int[3]">
  <nombre>1</nombre>
  <nombre>2</nombre>
  <nombre>3</nombre>
</nombres>

<lampe id="lampeDesign">
  <couleur type="xsd:string">noir</nocouleur>
  <modele type="xsd:string">thorg erkison</modele>
  <numeroProduit type="xsd:int">14987</numeroProduit>
</lampe>

<lampe href="lampeDesign"/>
```

L'attribut `id` permet d'identifier la lampe dans le message ce qui permet d'y faire référence. Cela peut réduire la redondance ou répondre à des besoins plus spécifiques comme la sérialisation d'un graphe.

2.2.3. Réponse du serveur SOAP en HTTP

En réponse au message d'interrogation du client SOAP, le serveur renverrait une réponse de ce type :

```
HTTP/1.0 200 OK
Content-Type: text/xml; charset="iso-8859-1"
Content-Length: 344

<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <addResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <addReturn xsi:type="xsd:int">2</addReturn>
    </addResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Ou alors en cas d'erreur de traitement du serveur :

```
HTTP/1.0 500 Internal Server Error
Content-Type: text/xml; charset="iso-8859-1"
Content-Length: 521

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Server</faultcode>
      <faultstring>Je suis trop bête pour effectuer ce
calcul</faultstring>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Ce message d'erreur indique que la faute provient d'un problème interne au serveur. Mais si par exemple le message du client avait été mal formé, la valeur de `faultcode` aurait été *Client* au lieu *Server*.

3. WSDL (Web Service Description Language)

Le language WSDL est une partie importante des services Web. C'est une grammaire XML qui décrit précisément quelles méthodes du service sont accessibles et quels sont les types des paramètres. Grâce à ce fichier, le développement de client est facilité.

Axis peut créer un tel fichier automatiquement grâce au logiciel `JAVA2WSDL`. Il existe un logiciel qui permet de réaliser l'opération inverse. Sans surprise, il s'agit de `WSDL2JAVA`.

WSDL est souvent utilisé pour le déploiement au niveau client. Nous verrons plus loin comment GSOAP utilise ce type de fichier afin de créer un canevas de programmation. Mais d'abords, étudions la forme d'un fichier WSDL qui décrit le service *`MathService`* :

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://localhost:8080/axis/services/MathService"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://localhost:8080/axis/services/MathService"
  xmlns:intf="http://localhost:8080/axis/services/MathService"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:message name="addResponse">
  <wsdl:part name="addReturn" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="subtractRequest">
  <wsdl:part name="in0" type="xsd:int"/>
  <wsdl:part name="in1" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="subtractResponse">
  <wsdl:part name="subtractReturn" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="addRequest">
  <wsdl:part name="in0" type="xsd:int"/>
  <wsdl:part name="in1" type="xsd:int"/>
</wsdl:message>
<wsdl:portType name="MathService">
  <wsdl:operation name="add" parameterOrder="in0 in1">
  <wsdl:input message="impl:addRequest" name="addRequest"/>
  <wsdl:output message="impl:addResponse" name="addResponse"/>
  </wsdl:operation>
  <wsdl:operation name="subtract" parameterOrder="in0 in1">
  <wsdl:input message="impl:subtractRequest" name="subtractRequest"/>
  <wsdl:output message="impl:subtractResponse" name="subtractResponse"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="MathServiceSoapBinding" type="impl:MathService">
  <wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="add">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="addRequest">
```

```
      <wsdlsoap:body
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="http://DefaultNamespace"
      use="encoded"/>
    </wsdl:input>
    <wsdl:output name="addResponse">
      <wsdlsoap:body
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="http://localhost:8080/axis/services/MathService"
      use="encoded"/>
    </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="subtract">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="subtractRequest">
      <wsdlsoap:body
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="http://DefaultNamespace"
      use="encoded"/>
    </wsdl:input>
    <wsdl:output name="subtractResponse">
      <wsdlsoap:body
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="http://localhost:8080/axis/services/MathService"
      use="encoded"/>
    </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="MathServiceService">
    <wsdl:port binding="impl:MathServiceSoapBinding" name="MathService">
    <wsdlsoap:address
  location="http://localhost:8080/axis/services/MathService"/>
    </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

Les fichier WDSL sont assez verbeux et peu agréables à lire. Bien qu'il y ait peu de besoin d'en réaliser manuellement, il est utile de savoir les comprendre afin d'imaginer comment fonctionne le service associé.

L'élément racine se nomme `definitions`. WSDL définit les services comme des collections de points de destinations sur le réseau. Ces points de destination sont designés comme des `ports` WSDL sépare ces `ports` et leurs messages. Les principales sections du document sont définies comme suit :

types

Défini des types de données supplémentaires si les types de bases ne suffisent pas.

message

Définit les types de données qui sont échangées. Ce sont les paramètres de sorties et d'entrée des fonctions SOAP.

operation

Un regroupement des paramètres de sorties et d'entrées qui forment une fonction.

portType

Regroupe les opérations qui sont supportées par un ou plusieurs ports de communication

binding

Définit le format des données pour un type de port particulier

port

Point de destination qui est une liaison entre le *binding* et l'adresse réseau

service

L'ensemble des points de destination

4. Choix des logiciels serveur

4.1. Apache Jakarta Tomcat, Apache SOAP et Apache Axis

Je me suis tourné vers les grands outils standards comme *Apache Jakarta Tomcat* qui est un serveur d'applications Web.

Plus précisemment, Tomcat est un serveur de *servlet*. Une *servlet* est une application qui s'exécute dans un environnement confiné. Cet environnement se charge de l'interaction avec le client via le protocole HTTP. Une *servlet* doit respecter une interface et en contrepartie elle est déchargée de toute la gestion du protocole HTTP. On peut voir Tomcat comme un *framework* orienté application Web.

Il existe deux implémentations de SOAP pour le serveur Tomcat: Apache SOAP et Apache Axis. Après un détour rapide par Apache SOAP, j'ai été convaincu par la simplicité de déploiement des services SOAP avec Axis ainsi que par La documentation abondante du projet. Axis a une réputation de logiciel moderne. Il est disponible dans le langage Java pour le serveur Tomcat et en C++ comme module du serveur Apache.

En d'autre termes, Axis est une *servlet* Tomcat qui permet de développer facilement des services SOAP en Java ou en *C*++ (dans le cas du module apache).

5. Installation de Apache Jakarta Tomcat et de Apache Axis sur Linux

Un des points ennuyeux de Apache Axis est son installation car le logiciel est basée sur une grande diversité de logiciels différents. J'ai voulu produire une procédure d'installation pas à pas afin de pouvoir la réutiliser.

5.1. Version des Logiciels utilisés

Version Java
j2sdk 1.5

Distributions Linux testées
Archlinux 0.7, Debian Woody, Mandrake

Version Apache Jakarta Tomcat
5.5.7

Version Apache Axis
1.1

Version *JavaBeans Activation Framework*
1.0.2

5.2. Installation de Apache Jakarta Tomcat

1. Installer le `j2dk` de Java et contrôler que la variable d'environnement `JAVA_HOME` pointe bien sur le dossier du `j2dk`.
2. Créer le répertoire où vous désirez Installer Tomcat. Le répertoire par défaut indiqué dans ce document est le répertoire factice `/home/me/tomcat`.
3. *Télécharger la version stable courante de Apache Jakarta Tomcat* dans le répertoire d'installation de Tomcat et décompresser le fichier ainsi obtenu :

   ```
   gunzip jakarta-tomcat-5.5.7.tar.gz
   tar -xvf jakarta-tomcat-5.5.7.tar
   ```

   Ou plus simplement :

   ```
   tar -xzvf jakarta-tomcat-5.5.7.tar.gz
   ```

4. L'installation est maintenant terminée. Lancer le serveur Tomcat par la commande suivante :

   ```
   /home/me/tomcat/jakarta-tomcat-5.5.7/bin/startup.sh
   ```

   Le serveur devrait être accessible à l'adresse suivante : *http://localhost:8080/*.

Afin de contrôler son bon fonctionnement, essayer d'exécuter les exemples disponibles sur cette page.

5.3. Installation de Apache Axis sur Apache Jakarta Tomcat

1. *Télécharger la version Java stable courante de Apache Axis* dans le répertoire d'installation de Tomcat et décompresser le fichier ainsi obtenu :

   ```
   tar -xzvf axis-1_1-src.tar.gz
   ```

2. Copier le dossier Axis se trouvant dans `axis-1_1/webapps/` dans `jakarta-tomcat-5.5.7/webapps/` :

   ```
   cp -R axis-1_1/webapps/axis/ jakarta-tomcat-5.5.7/webapps/
   ```

3. *Télécharger le JavaBeans Activation Framework* dans le répertoire d'installation de Tomcat et décompresser le fichier ainsi obtenu :

   ```
   unzip jaf-1.0.2.zip
   ```

4. Copier le fichier `jaf-1-0.2/activation.jar` dans `jakarta-tomcat-5.5.X/webapps/axis/WEB-INF/lib` et dans le dossier `/jre/ext` de votre jdk :

   ```
   cp jaf-1-0.2/activation.jar
   jakarta-tomcat-5.5.X/webapps/axis/WEB-INF/lib
   cp jaf-1-0.2/activation.jar $JAVA_HOME/jre/ext
   ```

   La copie du fichier dans `$JAVA_HOME/jre/ext` nécessite les droits administrateur. Cette manipulation est faite pour éviter un bogue. Il est possible que vous n'ayez pas à faire cette manipulation.

5. Créer un scripte de démarrage "start.sh" de Tomcat tel que celui-ci :

   ```
   #! /bin/bash
   echo tomcat and axis startup file

   HOME=/home/me/tomcat
   TOMCAT_HOME=${HOME}/jakarta-tomcat-5.5.7

   echo '  *set classpath for axis'
   AXIS_HOME=$TOMCAT_HOME/webapps/axis/WEB-INF
   AXIS_LIB=$AXIS_HOME/lib
   AXISCP=$AXIS_LIB/activation.jar
   AXISCP=${AXISCP}=$AXIS_LIB/axis.jar
   AXISCP=${AXISCP}=$AXIS_LIB/commons-discovery.jar
   AXISCP=${AXISCP}:$AXIS_LIB/commons-logging.jar
   AXISCP=${AXISCP}:$AXIS_LIB/wsdl4j.jar
   AXISCP=${AXISCP}:$AXIS_LIB/jaxrpc.jar
   AXISCP=${AXISCP}:$AXIS_LIB/saaj.jar
   AXISCP=${AXISCP}:$AXIS_LIB/log4j-1.2.8.jar
   AXISCP=${AXISCP}:$AXIS_LIB/xml-apis.jar
   ```

```
AXISCP=${AXISCP}:$AXIS_LIB/xercesImpl.jar:./
AXISCLASSPATH=${AXISCP}
export AXIS_HOME; export AXIS_LIB; export AXISCLASSPATH

if [ $TOMCAT_STARTED ];
then
echo '  *tomcat shutdown'
$TOMCAT_HOME/bin/shutdown.sh
fi
echo '  *tomcat startup'
$TOMCAT_HOME/bin/startup.sh

TOMCAT_STARTED='true'
export TOMCAT_STARTED;
```

Ce fichier permet de relancer le serveur en l'arrêtant et défini la variable AXISCLASSPATH que nous allons utiliser plus loin pour lancer des logiciels Axis.

6. Exécuter le script :

```
chmod +x start.sh
. ./start.sh
```

7. Contrôler que l'installation s'est correctement déroulée en suivant ces adresses :

- *http://localhost:8080/axis/*: la page d'accueil du serveur Axis;
- *http://localhost:8080/axis/happyaxis.jsp*: contrôle que toutes les bibliothèques nécessaires sont présentes.; Certains composants optionnels peuvent être manquants. Vous pouvez les installer en suivant les liens associés mais ils ne seront pas nécessaires pour faire fonctionner les exemples figurants dans ce document.

- *http://localhost:8080/axis/services/Version?method=getVersion*: affiche la version de votre serveur Axis.

6. Déploiement d'un service sur Apache Axis

6.1. Déploiement rapide

Apache Axis offre un service de déploiement rapide. Ce type de déploiement utilise du code source Java qui est compilé à la volée. Pour l'utiliser il suffit de renommer votre classe en JWS(Java Web Service) et de le copier dans le répertoire racine de Apache Axis :

```
cp MathService.java $AXIS_HOME/MathService.jws
```

Faites un essai avec *MathService*. Vous pouvez ensuite directement tester votre service grâce au URL suivantes :

- Le service est il bien reconnu ? *http:///localhost:8080/axis/MathService.jws*
- Le fichier WSDL(Web Descritpion Language) de mon service : *http://localhost:8080/axis/MathService.jws?wsdl*. Ce fichier WSDL est généré directement à partir du code Java grâce au logiciel JAVA2WSDL.
- Test de la fonction add : *http://localhost:8080/axis/MathService.jws?method=add&a=3&b=2*

Ce type de déploiement reste toutefois déconseillé en production.

6.2. Déploiement standard

6.2.1. WSDD (Web Service Deployment Descriptor)

Pour déployer un service de manière standard, suivez les étapes suivantes :

1. Il faut écrire un fichier WSDD (Web Service Deployment Descriptor) pour le service *`MathService`* :

   ```
   <deployment xmlns="http://xml.apache.org/axis/wsdd/"
   xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
   <service name="MathService" provider="java:RPC">
     <parameter name="className" value="MathService"/>
     <parameter name="allowedMethods" value="*"/>
   </service>
   </deployment>
   ```

   Ce fichier définit le nom de notre service, la classe associée à ce service ainsi que les fonctions disponibles depuis l'extérieur.

   Enregistrer ce fichier sous le nom "MathService.wsdd"

2. Compiler la classe de `MathService` et copier la dans le dossier `$AXIS_HOME/classes` :

```
javac -classpath $AXISCLASSPATH *.java
cp *.class $AXIS_HOME/classes
```

3. Vous pouvez lister les services déployés sur le serveur grâce à cette commande :

   ```
   java -classpath $AXISCLASSPATH org.apache.axis.client.AdminClient
   list
   ```

   Cette sortie est en faite la copie du fichier `server-config.wsdd` qui se trouve dans le répertoire `$AXIS_HOME/WEB-INF` du *servlet Axis*.

4. Pour déployer le service `MathService` exécuter la commande suivante :

   ```
   java -classpath $AXISCLASSPATH org.apache.axis.client.AdminClient
   MathService.wsdd
   ```

   En exécutant le programme AdminClient vous allez modifier le fichier `server-config.wsdd` et y ajouter votre service. Si les classes sont disponibles au bon endroit, la commande va également lancer votre service. L'interêt de cette commande et de pouvoir déployer des services en toute sécurité sans redémarrer le serveur. C'est utile quand le serveur est en production.

   Il est également possible de copier manuellement le contenu du fichier WSDD dans le fichier `server-config.wsdd`. Il faut alors redémarrer le serveur pour que le service devienne disponible.

5. Assurez vous ensuite que ces fonctions fonctionnent :

- La liste des services: *http://localhost:8080/axis/servlet/AxisServlet*
- Le service est il bien là: *http://localhost:8080/axis/services/MathService*
- Le fichier WSDL de notre service: *http://localhost:8080/axis/services/MathService?wsdl*
- Utilisons la méthode add de notre service avec les valeurs 2 et 3: *http://localhost:8080/axis/services/MathService?method=add&a=2&b=3*

7. CSOAP

CSOAP est la première bibliothèque C sur laquelle je me suis penché. Elle est simple d'utilisation et utilise LibXML2 qui est une bibliothèque de traitement XML. LibXML2 est de grande taille ce qui ne correspond pas au contraintes des systèmes embarqués qui demandent de pouvoir obtenir un logiciel aussi petit que possible.

LibXML2 semblait fort complexe à compiler pour les systèmes embarqués. Le problème se révélant rapidemment délicat et la bibliothèque *CSOAP* ne correspondant pas aux contraintes du projet, j'ai décidé d'abandonner. Je donne tout de même plus loin l'implémentation d'un client *CSOAP* pour `MathService`.

Les détails de cette bibliothèque ne seront pas abordés dans ce document mais uniquement sur une petite implémentation qui permet d'utiliser la méthode `add` de notre service `MathService`.

7.1. Implémenter un client CSOAP pour MathService

Afin de donner un exemple, je livre ici un client CSOAP simple qui demande au *service MathService* d'additionner 5 et 3 :

```
#include <libcsoap/soap-client.h>

static const char *url =
"http://localhost:8080/axis/services/MathService";
static const char *urn = "urn:MathService";
static const char *method = "add";

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

SoapCtx *ctx, *ctx2;
herror_t err;

/* initialisation de l'environnement */
err = soap_client_init_args(argc, argv);
if (err != H_OK) {
  log_error4("[%d] %s(): %s ", herror_code(err), herror_func(err),
herror_message(err));
  herror_release(err);
  return 1;
}

/* création de la structure de méthode */
err = soap_ctx_new_with_method(urn, method, &ctx);
if (err != H_OK) {
  log_error4("[%d] %s(): %s ", herror_code(err), herror_func(err),
herror_message(err));
  herror_release(err);
  return 1;
}

/* ajout des paramètres a et b à la méthode */
soap_env_add_item(ctx->env, "xsd:int", "a", "5");
```

```
soap_env_add_item(ctx->env, "xsd:int", "b", "3");

/* appel de la fonction add */
err = soap_client_invoke(ctx, &ctx2, url, "");
if (err != H_OK) {
  log_error4("[%d] %s(): %s ", herror_code(err), herror_func(err),
herror_message(err));
  herror_release(err);
  soap_ctx_free(ctx);
  return 1;
}

/* affichage du message reçu */
soap_xml_doc_print(ctx2->env->root->doc);

/* libération de la mémoire */
soap_ctx_free(ctx2);
soap_ctx_free(ctx);
soap_client_destroy();

return 0;
}
```

La réponse du serveur est la suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
  <soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
        <addResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <addReturn xsi:type="xsd:int">8</addReturn>
    </addResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

8. GSOAP

GSOAP est la deuxième bibliothèque sur laquelle je me suis penché. Elle a comme principaux avantages d'avoir une faible utilisation mémoire et une compilation aisée car elle se base sur un mode de fonctionnement statique. La documentation est abondante et de bonne qualité.

Le mode de fonctionnement de GSOAP n'est pas vraiment celui d'une bibliothèque de fonctions. on parle plus volontiers de boîte à outils (*toolkit*).

Elle est relativement simple d'utilisation, mais le processus de mise en place est plus long qu'avec *CSOAP*. Elle permet de réaliser des programmes en *C* ou en *C++*.

Le processus de développement est basé sur les fichiers WSDL qui décrivent précisemment les services et leurs messages. À l'aide de ces fichiers, les outils GSOAP vont générer une série de modules qui s'occcuperont de générer et de traduire les messages SOAP. Les services sont alors accessibles directement par le programmeur via des fonctions. **On nomme ces fonctions des *stubs***.

Du côté serveur, vos fonctions sont appelées par le *servlet* Axis. Le pont entre vos fonctions et Axis est appelé *ties*.

8.1. Installation de la bibliothèque GSOAP

Version du logiciel GSOAP
2.7

Voici un petite procédure pour l'installation de GSOAP :

1. Télécharger le fichier `gsoap-2.7.tar.gz` à cette adresse: *http://sourceforge.net/project/showfiles.php?group_id=52781*.
2. Décompresser l'archive ainsi obtenue :

   ```
   tar -xzfv  gsoap-2.7.tar.gz
   ```

3. Compiler et installer la bibliothèque :

   ```
   ./configure && make
   make install
   ```

4. Contrôlez que vous avez accès aux programmes `wsdl2h` et `soapcpp2` car nous allons les utiliser immédiatement.

9. Faire fonctionner GSOAP et Axis ensemble

9.1. Implémenter un client pour MathService

Voilà enfin le moment de créer un client pour notre fameux service.

1. Enregistrer *le fichier MathService.wsdl* dans le dossier de votre client.
2. Créer le fichier *header* `MathService.h` de notre application en utilisant la commande suivante :

   ```
   wsdl2h -c MathService.wsdl
   ```

   Ce programme va parcourir le fichier WSDL et créer les structures de données nécessaires ainsi que les prototypes de fonctions et stocker cela dans un fichier *header* `MathService.h`. Le paramètre `-c` indique que nous voulons travailler avec le langage C plutôt qu' avec C++.
3. À partir du fichier `MathService.h`, `soapcpp2` va créer les fonctions de sérialisation et de désérialisation nécessaires :

   ```
   soapcpp2 -c MathService.h
   ```
4. Il nous faut également le fichier "stdsoap2.c" qui se trouve dans le dossier de GSOAP dans le sous-dossier "soapcpp2". Copier le dans le dossier de votre client.

Nous voilà prêt pour réaliser notre premier client en GSOAP . Étudions le fichier `MathService.h`. Intéressons nous en particulier à la fonction `add` :

```
/// Operation response struct "ns1__addResponse" of service binding
"MathServiceSoapBinding" operation "ns1__add"
struct ns1__addResponse
{
  int  _addReturn;
};
```

La structure `ns1__addResponse` est la structure de réponse de notre méthode `add`. Sans surprise, elle contient un entier. La suite de l'information nous renseigne sur la forme de la fonction de *stub*. Un *stub* est une interface de service qui utilise la structure GSOAP pour créer un message SOAP et pour récupérer les informations. **Si vous avez un stub vous n'avez donc pas à vous soucier des détails du message SOAP**.

Un peu plus loin on peut trouver en commentaire le prototype de la fonction `stub`

```
int soap_call_ns1__add(struct SOAP *soap, NULL, NULL, int in0, int in1,
struct ns1__addResponse*);
```

Voici le code d'un client qui utilise la fonction de *stub* `soap_call_ns1__add` :

```
#include "soapH.h"
#include "MathServiceSoapBinding.nsmap"

const char server[] = "http://localhost:8080/axis/services/MathService";

int main(int argc, char **argv)
{
  if( argc<3 )
  {
    printf("add function prototype: add(int,int)\r\n");
    return 0;
  }

  int a,b;
  // contrôle des paramètres d'entrées
  if( sscanf(argv[1],"%d",&a)==0 )
  {
    printf("add first param not found\r\n");
    return 0;
  }
  if( sscanf(argv[2], "%d", &b)==0 )
  {
    printf("add second param not found\r\n");
    return 0;
  }

  struct SOAP soap;
  struct ns1__addResponse addResponse;
  soap_init(&soap);

  // appel de la fonction de "stub" "add"
  soap_call_ns1__add(&soap, server, "", a, b, &addResponse);
  if (soap.error)
  {
    soap_print_fault(&soap, stderr);
    return 0;
  }
  else
  {
    printf("result: add(%d,%d)=%d\r\n",a, b, addResponse._addReturn);
  }
}
```

Créer le fichier `mathClient.c` avec le code ci-dessus.

Compilons maintenant ce client :

```
gcc -o mathClient mathClient.c stdsoap2.c soapC.c soapClient.c
```

Il ne reste plus qu'à tester notre client mathématique pour vérifier que tout fonctionne :

```
./mathClient 5 6
```

La réponse attendue est :

```
result: add(5,6)=11
```

9.2. Transférer un fichier binaire

Une des fonctionnalité désirée était de pouvoir transférer des patchs binaires.

Il existe un type SOAP de base qui permet de transférer de tels fichiers. Il s'agit du type `base64Binary` que l'on voit dans *la petite table de correspondance des type SOAP et Java*.

Comme son nom l'indique, le code binaire est encodé en *base64* qui permet d'optenir des chaînes de caractères directement exploitable en XML. Cet encodage transforme un flux binaire en un flux texte avec seulement 64 caractères de base. Il faut garder à l'esprit que la taille du binaire encodé en *base64* et d'un tiers plus grande que l'originale.

Avec Axis, il suffit simplement d'utiliser le type Java `byte[]` et le travail sera fait pour vous :

```
/**
* service de lecture de fichier binaire
*/

import java.io.FileInputStream;

public class GetFileService
{
  public byte[] getFile(String fileName)
  {
    byte[] buffer;
    /* gestion des exceptions pour l'accès au fichier */
    try{
      FileInputStream fileInput = new FileInputStream(fileName);
      buffer = new byte[fileInput.available()];
      fileInput.read(buffer);
      return buffer;
    }
    catch(Exception e)
    {
      return new byte[0];
    }
  }
}
```

Du coté du client, il suffit de procéder comme pour `mathClient`. GSOAP va créer une structure permettant de récupérer ce contenu binaire :

```
struct xsd__base64Binary
{
  unsigned char *__ptr;
  int __size;
};

struct ns1__getFileResponse
{
  xsd__base64Binary _getFileReturn;
};
```

Il suffit alors de réaliser un client GSOAP pour récupérer un fichier sur le serveur et le sauvegarder sur le système embarqué :

```
#include "soapH.h"
#include "GetFileServiceSoapBinding.nsmap"
```

```
const char service[] =
"http://localhost:8080/axis/services/GetFileService";

int main(int argc, char **argv)
{
  struct SOAP soap;
  soap_init(&soap);
  // gestion sommaire des arguments
  if( argc<3 )
  {
    printf("not enough argument\r\n");
    return 0;
  }
  // structure de réponse
  struct ns1__getFileResponse fileResponse;
  // fichier sur le serveur
  char * serverFileName = argv[1];
   fichier sur le client
  char * clientFileName = argv[2];
  // appel de la fonction de "stub"
  soap_call_ns1__getFile(&soap, service, "", serverFileName,
&fileResponse);
  if (soap.error)
  {
    soap_print_fault(&soap, stderr);
    return 1;
  }
  else
  {
    // écriture du fichier binaire
    FILE* f = fopen(clientFileName,"wb+");
    if( f!=NULL )
    {
      fwrite(fileResponse._getFileReturn.__ptr,
fileResponse._getFileReturn.__size,1, f);
      fclose(f);
      printf("%s was writen successfully\r\n", clientFileName,
fileResponse._getFileReturn.__size);
      return 0;
    }
    else
    {
      printf("could not write %s\r\n",clientFileName);
      return 1;
    }
  }
}
```

À l'utilisation, vous devez indiquer un chemin de fichier que le serveur a le droit d'atteindre et le chemin où vous désirez écrire sur le client. Typiquement si tout fonctionne sur une machine locale vous pouvez saisir les commandes suivantes :

```
gcc -o getFile getFile.c stdsoap2.c soapC.c soapClient.c
./getFile /home/me/img.png img.png
```

Vous devriez obtenir l'image désirée dans le dossier de votre client.

9.3. Les types complexes

Les types complexes interviennent quand les types simples ne suffisent plus. C'est le cas quand on veut échanger des structures de données et des tableaux.

9.3.1. Avec Apache Axis

Afin d'illustrer les types complexes, voici un exemple aussi simple que possible. Pour créer un type complexe SOAP avec Axis, il faut créer une classe qui sera la représentation de la structure de donnée. Cette structure doit respecter une des propriétés des *javabeans*. Il faut des méthodes pour accéder et définir les attributs de l'objet afin que Axis puisse les utiliser et les sérialiser. Considérons l'exemple suivant :

```
/**
  structure de donnée complexe
*/
public class ComplexType
{
  private int i;
  private String s;
  private int[] t;

  public void setI(int ii){ i=ii; }
  public int getI(){ return i; }

  public void setS(String si){ s=si; }
  public String getS(){ return s; }

  public void setT(int[] ti){ t=ti; }
  public int[] getT(){ return t; }
}
```

Il est important de bien définir les fonctions d'accès et de définition de données. Les accès sur une variable se font par le mots-clé `get` et les définitions par le mots-clé `set`. Il faut ensuite implémenter un service qui utilise cette structure de donnée :

Voyons maintenant la classe de service utilisant notre type complexe :

```
/**
  class renvoyant un type complexe
*/
public class Complex{
  public ComplexType getComplex()
  {
    ComplexType c = new ComplexType();
    c.setI(2005);
    c.setS("hello world");
    int[] tab=new int[3];
    tab[0]=0; tab[1]=1; tab[2]=2;
    c.setT(tab);
    return c;
  }
}
```

Le déploiement d'une application utilisant un type complexe est légèrement différent que précédemment. Il faut ajouter un *beanMapping* sur le type complexe afin qu'il soit reconnu comme une structure de donnée sérialisable et désérialisable. Le fichier de déploiement de notre service devient alors :

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="Complex" provider="java:RPC">
    <parameter name="className" value="Complex"/>
    <parameter name="allowedMethods" value="*"/>
    <beanMapping qname="ns:ComplexType" xmlns:ns="urn:Complex"
languageSpecificType="java:ComplexType"/>
  </service>
</deployment>
```

Déployer le service comme précédemment en utilisant la commande `AdminClient`. Malgré un message positif, il est possible que l'opération ne fonctionne pas. Pour remédier à cela, copier le contenu du WSDD manuellement dans le fichier `$AXIS_HOME/WEB-INF/server-config.wsdd`.

Une fois le serveur relancé, j'ai pu obtenir le resultat suivant le lien *http://localhost:8080/axis/services/Complex?method=getComplex* :

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <getComplexResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <getComplexReturn xsi:type="ns1:ComplexType"
xmlns:ns1="urn:Complex">
        <i xsi:type="xsd:int">2005</i>
        <s xsi:type="xsd:string">hello world</s>
        <t xsi:type="soapenc:Array" soapenc:arrayType="xsd:int[3]"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
          <item>0</item>
          <item>1</item>
          <item>3</item>
        </t>
      </getComplexReturn>
    </getComplexResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Il est intéressant d'étudier le fichier WSDL afin de voir comment sont représentés les types complexes :

```
<wsdl:types>
  <schema
  targetNamespace="http://localhost:8080/axis/services/Complex"
  xmlns="http://www.w3.org/2001/XMLSchema">
    <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
    <complexType name="ArrayOf_xsd_int">
      <complexContent>
        <restriction base="soapenc:Array">
          <attribute ref="soapenc:arrayType"
wsdl:arrayType="xsd:int[]"/>
        </restriction>
      </complexContent>
    </complexType>
  </schema>
  <schema targetNamespace="urn:Complex"
xmlns="http://www.w3.org/2001/XMLSchema">
```

```
    <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
    <complexType name="ComplexType">
      <sequence>
        <element name="i" type="xsd:int"/>
        <element name="s" nillable="true" type="xsd:string"/>
        <element name="t" nillable="true" type="impl:ArrayOf_xsd_int"/>
      </sequence>
    </complexType>
  </schema>
</wsdl:types>
```

Les éléments SOAP avec l'attribut `nillable` sont facultatifs.

9.3.2. Utilisation avec GSOAP

La procédure est identique que précédemment. Si tous c'est bien passé, une structure est présente dans le fichier `Complex.h` :

```
struct ns3__ComplexType
{
  int i
  char* s
  struct ArrayOf_USCORExsd_USCOREint*  t
};
```

La structure de tableau d'entier ressemble à ceci :

```
struct ArrayOf_USCORExsd_USCOREint
{
  int *__ptr;
  int __size;
};
```

Cette structure correspond bien à notre type complexe. La structure de réponse est la suivante :

```
struct ns1__getComplexResponse
{
  struct ns3__ComplexType* getComplexReturn;
};
```

Le prototype de notre fonction *stub* doit ressembler à ceci :

```
int soap_call_ns1__getComplex(struct SOAP *soap, NULL, NULL, struct
ns1__getComplexResponse* );
```

Voyons maintenant le code de l'application :

```
#include "soapH.h"
#include "ComplexSoapBinding.nsmap"

const char server[] = "http://localhost:8080/axis/services/Complex";

int main(int argc, char **argv)
{
  struct SOAP soap;
  struct ns1__getComplexResponse complexResponse;
  struct ArrayOf_USCORExsd_USCOREint *tab;
```

```
  soap_init(&soap);
  soap_call_ns1__getComplex(&soap, server, NULL, &complexResponse);
  if (soap.error)
    soap_print_fault(&soap, stderr);
  else
  {
      printf("i=%d;\r\n", complexResponse.getComplexReturn->i);
      printf("s=%s;\r\n", complexResponse.getComplexReturn->s);
      tab=complexResponse.getComplexReturn->t;
      int i;
      for(i=0; i<tab->__size; i++)
      {
        printf("tab[%d]=%d\r\n",i,tab->__ptr[i]);
      }
    }
  return 0;
}
```

À l'exécution du programme, vous devriez obtenir :

```
i=2005;
s=hello world;
tab[0]=0
tab[1]=1
tab[2]=3
```

À partir de là, toutes les combinaisons avec des types complexes et simples sont possibles.

10. Un exemple d'application pour l'envoi de mesure de température

Voici une petite application qui permet d'envoyer une série de mesures depuis le client jusqu'au serveur. Le serveur enregistre simplement ces informations dans un fichier.

Étudions la forme du type complexe qui nous permet de stocker une série de mesures :

```
/**
  classe de mesure
*/
import java.util.Calendar;

public class Mesure
{
  private float valeur;
  private Calendar date;

  public void setValeur(float vi){valeur=vi;}
  public float getValeur(){return valeur;}

  public void setDate(Calendar di){date=di;}
  public Calendar getDate(){return date;}
}
```

La mesure est associée à une date. Les dates sont représentées en SOAP par le type simple `xsd:dateTime`. Examinons maintenant le code du service :

```
/**
  classe d'enregistrement de mesure
*/
import java.util.Calendar;
import java.io.FileOutputStream;

public class EregMesure
{
  // adresse MAC, numéro de sonde, tableau de mesure
  public boolean eregMesure( String macAddr, int nbSondes, Mesure[]
mesures)
  {
    // enregistrement du fichier
    try{
      FileOutputStream fileOutput = new
FileOutputStream("/home/me/mesure-"+macAddr+".xml", true);
      String buffer = macAddr+"\r\n";
      fileOutput.write(buffer.getBytes());
      buffer = nbSondes+"\r\n";
      fileOutput.write(buffer.getBytes());
      // lecture du tableau et écriture
      for( int i=0; i<mesures.length; i++)
      {
        buffer =
mesures[i].getValeur()+","+mesures[i].getDate().getTime().getTime()+"\r\n";
        fileOutput.write(buffer.getBytes());
      }
      return true;
```

```
    }
    catch(Exception e)
    {
      return false;
    }
  }
}
```

Le service reçoit l'adresse *MAC* de l' *AccessBox*, le numéro de la sonde de température ainsi qu'un tableau de mesures. Changez le chemin par défaut pour qu'il corresponde au dossier où vous désirez enregistrer les mesures.

Une fois le service déployé, le fichier WSDL produit, et les commandes de création de canevas GSOAP executée; l'aspect des structures de données sont les suivantes :

```
struct ns2__Mesure
{
  /// Element date of type xs:dateTime
  time_t* date;  /// Nullable pointer
  /// Element valeur of type xs:float
  float valeur;  /// Required element
};
```

Les mesures utilisent le type standard `time_t` qui est un entier non signé. Le temps est donné en secondes par rapport au premier janvier 1970. Examinons maintenant la structure représentant le tableau de mesures :

```
struct ArrayOf_USCOREtns1_USCOREMesure
{
/// Pointer to an array of struct ns2__Mesure*
  struct ns2__Mesure** __ptr;
/// Size of the dynamic array
  int __size;
};
```

L'attribut `__ptr` est un tableau de pointeur de mesures. L'attribut `__size` indique le nombre de pointeurs de mesures que contient le tableau.

Le code du client vaut la peine que l'on s'y attarde car la gestion des pointeurs n'est pas triviale :

```
#include "soapH.h"
#include "EregMesureSoapBinding.nsmap"
#include "time.h"

const char server[] = "http://localhost:8080/axis/services/EregMesure";

int main(int argc, char **argv)
{
  struct SOAP soap;
  soap_init(&soap);
  // la structure de réponse
  struct ns1__eregMesureResponse mesureResponse;
  // le type complexe
  struct ArrayOf_USCOREtns1_USCOREMesure tabMesure;
  int i;
  int nbSonde=2;
  int nbMesure=5;
```

```
  tabMesure.__size=nbMesure;
  // le tableau de pointeur de mesures
  struct ns2__Mesure* mesuresPtr[nbMesure];
  // le tableau de mesures
  struct ns2__Mesure mesures[nbMesure];
  // le tableau de temps
  time_t times[nbMesure];
  tabMesure.__ptr=mesuresPtr;

  // remplissage du tableau de mesures
  for( i=0; i<nbMesure; i++ )
  {
    time(&times[i]);
    mesures[i].date=&times[i];
    mesures[i].valeur=i;
    mesuresPtr[i]=&mesures[i];
  }

  // appel de la fonction de stub
  soap_call_ns1__eregMesure(&soap, server, "", "00-0C-6E-F1-66-92",
nbSonde, &tabMesure, &mesureResponse);
  if (soap.error)
  {
    soap_print_fault(&soap, stderr);
    return 0;
  }
  else
  {
    if( mesureResponse._eregMesureReturn==0 )
        printf("EregMesure cannot write file\r\n");
    else
        printf("EregMesure write file sucessfully\r\n");
    return 0;
  }
}
```

En exécutant le service, vous devriez obtenir :

```
EregMesure write file sucessfully
```

11. Porter une application GSOAP sur la carte Axis (système embarqué)

Le portage se fait en utilisant l'environnement de développement de la carte Axis. Commencer par installer l'environnement de développement: *http://developer.axis.com/*

Copier votre application dans le répertoire `apps` de votre environnement tout en créant un `Makefile` en vous inspirant des exemples fournis dans ce dossier. Pour le reste de la procédure, référez-vous à la documentation en ligne: *http://developer.axis.com/doc/index.html*.

12. Conlusion

12.1. Utilisation mémoire de GSOAP

La bibliothèque GSOAP est présentée comme une solution efficace pour une faible utilisation mémoire.

« Web service and client executables can be as small as 90K on Linux with a 150K total memory footprint. »

— http://www.cs.fsu.edu/~engelen/soap.html

Mes tests indiquent que l'utilisation mémoire d'une petite application comme celles présentées dans ce document est de l'ordre du demi-méga-octets. Cela semble satisfaisant sachant que la carte Axis (le système embarqué) dispose de 16 mégas-octets. Le système en consomme une partie à l'exécution mais il en reste suffisament pour réaliser des applications GSOAP complexes.

GSOAP est une solution relativement peu gourmande en mémoire pour traiter des messages SOAP dans le langage C. Elle est particulièrement bien adaptée au développement de logiciels embarqués.

12.2. Espace disque utilisé par les applications GSOAP

L'espace disque utilisé par une application GSOAP simple est d'environ 150 kilo-octets. C'est encore un fois satisfaisant et compatible avec le système embarqué. Il faut toutefois garder à l'esprit que GSOAP est compilé en statique, c'est à dire que chaque nouveau programme GSOAP utilisera de l'espace disque inutilement. Dans le cas d'un système embarqué, Il est préférable de rassembler un maximum de fonctionnalitées dans un seul logiciel ou alors d'utiliser des bibliothèques avec lesquelles les liaisons dynamiques sont possibles.

13. Bibliographie

- ENGLANDER (Robert), *Java et SOAP*, O'Reilly, Paris

- VAN ENGELEN (Robert), *GSOAP User Guide*, http://www.cs.fsu.edu/~engelen/soapdoc2.html
- THE APACHE SOFTWARE FOUNDATION (), *Axis User's Guide*, http://ws.apache.org/axis/java/user-guide.html
- AXIS COMMUNICATION (), *Axis Developer Documentation (ETRAX 100LX)*, http://developer.axis.com/doc/index.html