

Slide 0

Les Web Services

Stéphane Bortzmeyer

<bortzmeyer@nic.fr>

Novembre 2003

Slide 0

Ce document est distribué sous les termes de la GNU Free Documentation License

<http://www.gnu.org/licenses/licenses.html#FDL>. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

Slide 1

Pourquoi utiliser le réseau

- C'est l'autre machine qui a les données
- C'est l'autre machine qui va vite
- C'est l'autre machine qui a les bons logiciels

Slide 2

Avant les *Web Services* , les bricolages

- On copiait les données et les logiciels (cohérence),
- On analysait les pages Web (*Welcome to Hell*),
- On formatait en texte puis on re-analysait (whois...).

Slide 3

Programmation réseau sérieuse

- On faisait tout à la main (définir un protocole, peut-être avec BEEP, écrire les clients et les serveurs)
- On avait des solutions spécifiques à un langage (RMI)
- Corba <http://www.corba.org/>
- ONC-RPC

Slide 4

Les *Web Services* arrivent

<http://www.w3.org/2002/ws/>

Ils s'appuient sur le succès du Web :

- Disponibilité de HTTP,
- Web, donc bon,
- Et on passe les coupe-feux !

Voir quand même RFC 3205.

Slide 5

Les *Web Services* spécifient :

- Un encodage (toujours XML),
- Un transport (souvent HTTP),
- Une organisation des requêtes et réponses (RPC, par exemple).

Slide 6

XML-RPC, le plus simple

<http://www.xml-rpc.com>

Principe : la bibliothèque client encode les paramètres en XML et la bibliothèque serveur les décode.

Le programmeur ne voit **jamais** de XML.

On ne fait que des appels de procédure.

Transport en HTTP seulement.

Bibliothèques pour tous : Perl, C, Python, Ruby, Java, VisualBasic/.NET, PHP et même Emacs-Lisp.

Slide 7

XML-RPC, exemple Java

```
// server has been created above
Vector params = new Vector();
params.addElement(new Integer(5));
params.addElement(new Integer(3));
// Call the server, and get our result.
Hashtable result =
    (Hashtable) server.execute("sample.sumAndDifference", params);
// We cannot use the procedure name directly (a limit of Java)
int sum = ((Integer) result.get("sum")).intValue();
int difference = ((Integer) result.get("difference")).intValue();
```

Slide 8

XML-RPC, exemple Python

```
server = xmlrpclib.Server('http://whois.eureg.org:8080/RPC2')
# Call the server, and get our result.
result = server.sample.sumAndDifference(3, 5);
sum = result["sum"]
difference = result["difference"]
```

Exemples : Meerkat

Un service d'informations en ligne http://www.oreillynet.com/pub/a/rss/2000/11/14/meerkat_xmlrpc.html, accessible en XML-RPC.

Slide 9

```
<?php
    $server_url = '/meerkat/xml-rpc/server.php';
    $msg = new xmlrpcmsg('meerkat.getCategories', array());
    $client = new xmlrpc_client($server_url, "www.oreillynet.com", 80);

    # Send our XML-RPC message to the server and receive a response in return
    $response = $client->send($msg);
    $value = $response->value();

    # And convert it to a PHP data structure
    $categories = xmlrpc_decode($value);

    # Iterate over the results, printing each category's title
```

Slide 10

```
while( list($k, $v) = each( $categories ) ) {
    print $v['title'] . "<br />\n";
}

?>
```

Slide 11

Exemples : Adam's Names

Une interface d'accès au registre :

<http://www.adamsnames.tc/api/xmlrpc.html>

Permet par exemple un whois moderne :

```
my $rpc = Frontier::Client->new( url =>
    'http://www.adamsnames.tc/api/xmlrpc' );
my $status = $rpc->call('domquery', 'xmlrpcdemo.tc');
my $dumper = Data::Dumper->new([ $status ])->Terse(1)->Indent(1);
my $txt = $dumper->Dump;
```

Slide 12

XML-RPC, détails

Types de paramètres :

- entiers, dates, booléens, chaînes
- *structs* (tableaux associatifs)
- tableaux

Les erreurs sont signalées par des exceptions.

XML-RPC, le serveur

Avec le “registry” de XML-RPC (pour l’introspection).

```
self.registry.add_method('registry.queryDomain',
                        self.domquery,
                        [[STRUCT, STRING, STRUCT]])
...
def domquery (self, domain, credentials):
    """Queries the registry for a domain's attributes"""
    if credentials.has_key('name'):
        raise Unauthorized
    self.cursor.execute("""
        SELECT name,
...
def call (self, methodName, params):
    """Use our registry to find and call the appropriate method."""
    try:
        return self.registry.dispatch_call(methodName, params)
    except Unauthorized:
```

Slide 13

```
raise xmlrpclib.Fault(403, 'Unauthorized')
```

Slide 14

domquery est une procédure normale, sans aucune connaissance de XML-RPC (par exemple, elle lève des exceptions normales).

call connaît le protocole et lève donc des exceptions spécifiques.

XML-RPC, sur le câble

Uniquement si vous voulez écrire une bibliothèque

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (NetBSD)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181
```

Slide 15

```
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value><i4>41</i4></value>
    </param>
  </params>
</methodCall>
```

XML-RPC, lectures

Slide 16

Un O'Reilly très bien

Le HOWTO [http:](http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto.html)

[//xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto.html](http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto.html)

XML-RPC, limites

Slide 17

- Unicode ? (uniquement ASCII, en standard)
- Introspection ? <http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto-api-introspection.html>

Slide 18

REST, le Web original

Representational State Transfer. Pas un protocole ou un format, c'est une architecture, c'est même l'architecture originale du Web.

- L'URI est important : connaître l'URI doit suffire pour accéder à la ressource
- HTTP fournit toutes les opérations nécessaires (GET, PUT et DELETE, essentiellement).
- Chaque opération est auto-suffisante : il n'y a pas d'état.
- En outre, et c'est un point où le sens moderne de REST dévie de l'architecture originale du Web, les requêtes et surtout les réponses sont aujourd'hui typiquement encodées en XML

Slide 19

Exemple REST en biologie

XEMBL <http://www.xml.com/pub/a/ws/2002/05/14/biows.html>
interface REST à EMBL.

```
import urllib
import sys
from xml.dom.ext.reader import Sax2
from xml import xpath

def extract(element, dom):
    match = xpath.Evaluate("/INSDSet/INSDSeq/%s/text()" % element, dom)
    return match[0].nodeValue

def format(element, dom):
    return "%s: %s" % (element, extract(element, dom))

id = sys.argv[1]
url = 'http://www.ebi.ac.uk/cgi-bin/dbfetch?db=embl&id=%s&format=insdxml'
result = urllib.urlopen(url % id)
```

Slide 20

```
reader = Sax2.Reader()
doc = reader.fromStream(result)
print format("INSDSeq_locus", doc)
print format("INSDSeq_moltype", doc)
print format("INSDSeq_definition", doc)
```

Slide 21

SOAP, le plus vendu

<http://www.software.org/> <http://www.w3.org/TR/SOAP/>

Simple Object Access Protocol

Le meilleur marketing (W3C et Microsoft)

Slide 22

SOAP, les principes

Très proche de XML-RPC.

La bibliothèque client encode les paramètres en XML et la bibliothèque serveur les décode.

Le programmeur ne voit **jamais** de XML.

On fait des appels de procédure ou de l'asynchrone.

Transport en HTTP, BEEP, etc.

Bibliothèques pour tous : Perl, C, C#, Python, Ruby, Java, VisualBasic/.NET, PHP, Ada.

Slide 23

SOAP, un exemple Perl

```
SOAP::Lite http://www.soaplite.com/
```

```
# Utilise l'AUTOLOAD de Perl
use SOAP::Lite +autodispatch =>
    uri => 'http://www.soaplite.com/Temperatures',
    proxy => 'http://services.soaplite.com/temper.cgi';
print f2c(100), "\n"; # Appelle une procédure distante
```

Slide 24

SOAP, un exemple Python

```
SOAPpy http://pywebsvcs.sourceforge.net/  
server = SOAP.SOAPProxy('http://api.google.com/search/beta2',  
                        namespace='urn:GoogleSearch')  
result = server.doGoogleSearch('Zls0Q7uAt2Lrcd7BHjai...zWJj7',  
                               'python wsdl', ...);  
print result['estimatedTotalResultsCount']
```

La chaîne incompréhensible est la clé de la licence Google.

Slide 25

SOAP, détails

Types de paramètres :

- entiers, dates, booléens, chaînes, etc (tout ce qu'on peut écrire avec les schémas)
- *structs* (tableaux associatifs)
- tableaux

Les erreurs sont signalées par des exceptions (*faults*).

Slide 26

SOAP, le serveur

```
use SOAP::Transport::HTTP;
my $daemon = SOAP::Transport::HTTP::Daemon
  -> new (LocalAddr => 'localhost', LocalPort => 8080)
  -> dispatch_to('Handler');
$daemon->handle;

package Handler;
sub hi {
    return "hello, world";
}
sub bye {
    return "goodbye, cruel world";
}
```

Slide 27

Serveur SOAP plus détaillé

On veut créer un service indiquant la disponibilité d'un nom de domaine.

```
% whois -h whois.sidn.nl 'is bortzmeyer.nl'  
bortzmeyer.nl is free  
% whois -h whois.sidn.nl 'is ripe.nl'  
ripe.nl is active
```

Slide 28

Méticiel

On développe une bibliothèque “métier”, assez indépendante de SOAP.

```
package Meticiel;  
sub is_available () {  
  if ($domain !~ /\.fr$/) {  
    return "We only register domains in \".fr\"";  
  }  
  if (&registered($domain)) {  
    return "Domain $domain already registered";  
  }  
  return "Domain $domain is available. Buy it soon!";  
}
```

Logiciel réseau

Puis on écrit la colle SOAP autour.

```
use SOAP::Transport::HTTP;
$hostname = 'hostname';
chomp $hostname;
my $daemon = SOAP::Transport::HTTP::Daemon
    -> new (LocalAddr => $hostname, LocalPort => 8080)
    -> dispatch_to(undef, Meticiel,
                 undef, undef);
print "Contact to SOAP server at ", $daemon->url, "\n";
$daemon->handle;
```

Slide 29

On peut alors développer les clients.

```
$result = SOAP::Lite
    -> uri('http://does-not-exist-really-just-a-URI.nic.fr/Meticiel')
    -> proxy("http://$hostname:8080/")
    -> is_available($domain);
unless ($result->fault) {
```

```
    print $result->result();
    print "\n";
} else {
    print join ', ',
        $result->faultcode,
        $result->faultstring,
        $result->faultdetail;
}
```

Slide 30

Slide 31

SOAP, sur le câble

Uniquement si vous voulez écrire une bibliothèque

SOAP s'appuie sur les schémas XML.

SOAP permet de transmettre du XML brut (à analyser soi-même).

```
POST /StockQuote HTTP/1.1
```

```
Content-Type: text/xml; charset="utf-8"
```

```
Content-Length: 2456
```

```
SOAPAction: "http://electrocommerce.org/abc#MyMessage"
```

Slide 32

SOAP, sur le câble, suite

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <myapp:GetLastTradePrice xmlns:myapp="Some-URI">
      <symbol>AFNIC</symbol>
    </myapp:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Les *namespaces* (ici, “myapp”) permettent de définir ses propres éléments, sans risque de collision.

Slide 33

SOAP, lectures

Un O'Reilly assez médiocre

Plein de “SOAP pour les nuls”.

Slide 34

SOAP, les problèmes

- Usine à gaz
- Peu interopérable

What's wrong with SOAP?

```
wc soap-spec.txt
 1519  10671  79445 soap-spec.txt
wc xmlrpc-spec.txt
   315    1657  15838 xmlrpc-spec.txt
```

Slide 35

Exemples : Google

Google a un accès SOAP
<http://www.google.com/apis/index.html>, décrit en WSDL
(inscription gratuite et obligatoire)

Slide 36

Exemple, Amazon

<http://www.amazon.com/webservices>

IL existe des paquetages tout faits, au dessus de SOAP.

```
% perl amazon-by-keyword.pl ipv6
IPv6 Essentials - $27.97
Understanding Ipv6 - $20.99
IP Addressing and Subnetting, Including IPv6 - $41.97
...
```

Slide 37

UDDI, l'annuaire universel

<http://www.uddi.org/> Oasis et d'autres. Permet d'enregistrer les Web Services, afin de les retrouver. ("The CPAN of Web Services")

Un protocole + plusieurs registres (encore du travail pour les gérants de registre).

Un registre UDDI peut être accédé en SOAP mais aussi en XML-RPC ou Corba.

Documentation difficile à aborder (Oasis...). L'information est très structurée, avec beaucoup de niveaux.

Et la documentation n'est pas en hyper-texte :-)

Slide 38

UDDI, exemple

```
use UDDI::Lite +autodispatch =>
    proxy => 'http://uddi.microsoft.com/inquire';

$info = find_business(name => 'amazon')
    -> businessInfos->businessInfo->serviceInfos->serviceInfo;
print $info->name, "\n";
```

UDDI, plein pot

```
# find_* : "fuzzy" searches
# get_* : exact searches, with the key
$mybusinessList = find_business(name => 'ama');
$mybusinessInfos = $mybusinessList->businessInfos;
@mybusinessInfo = $mybusinessInfos->businessInfo;
for $mybusinessInfo (@mybusinessInfo) {

    print $mybusinessInfo->name, "\n";
    print $mybusinessInfo->businessKey, "\n\n";

    $myserviceInfos = $mybusinessInfo->serviceInfos;
    @myserviceInfo = $myserviceInfos->serviceInfo;

    for $myserviceInfo (@myserviceInfo) {
    print "    ", $myserviceInfo->name, "\n";
    print "    ", $myserviceInfo->serviceKey, "\n";
    @myserviceDetails = get_serviceDetail
        (serviceKey => $myserviceInfo->serviceKey);
```

Slide 39

```
for $myserviceDetail (@myserviceDetails) {
    print "        ", $myserviceDetail->name, "\n";
    print "        ", $myserviceDetail->description, "\n";
    $mybindingTemplate = $myserviceDetail->bindingTemplates->bindingTemplate; # Actually, several
    print "        ", $mybindingTemplate->description, "\n";
    print "        ", $mybindingTemplate->accessPoint, "\n";
}
print "\n";
}

    print "\n\n";
}
```

Slide 40

Slide 41

WSDL, méta-informations

WSDL est un langage (du W3C <http://www.w3c.org/TR/wsdl>) pour décrire les API (surtout pour SOAP)

Slide 42

WSDL, exemple

```
my $google = SOAP::Lite->service('http://api.google.com/GoogleSearch.wsdl');  
my $result = $google->doGoogleSearch(  
    $key, $query, 0, 10, 'false', '', 'false', '', 'latin1', 'latin1');
```

Slide 43

Utilisateurs : quelques conseils

1. Suivre un tutoriel,
2. Lire la spécification (formelle avec WSDL ou informelle),
3. Écrire le client (choix du langage, de la bibliothèque).

Slide 44

Créateurs : quelques conseils

1. Choix importants (paiement, qualité de service, conditions d'accès, conditions d'usage, **sécurité**),
2. Écrire la spécification (formelle avec WSDL ou informelle),
3. Écrire la partie métier (si pas encore fait),
4. Écrire le serveur.

Slide 45

L'avenir est aux *Web Services* ?

- Vers une migration technique vers les *Web Services* ?
- Vers une vraie ouverture des systèmes d'information ? La fin des *semantic firewall* ?