PHP-Ajax

Jusqu'à présent, les scripts que nous avons réalisés retournent l'intégralité d'une page web. Impossible de ne retourner qu'une partie (c'est que l'on appelle le Web 1.0). Ainsi, lors d'interactions soutenues avec un utilisateur, des lenteurs peuvent apparaître du fait de la lourdeur des échanges, ce qui nuit rend pénible l'usage de certaines pages. La technologie AJAX (qui n'est pas nouvelle) a pour objectif de permettre un rafraichissement partiel de données d'une page web, et uniquement de certaines données (c'est que l'on appelle le Web 1.0). AJAX est l'acronyme de « Asynchronous JavaScript and XML », bien qu'il ne soit en rien lié au XML. AJAX est basé sur l'objet *XMLHttpRequest* qui permet de faire une requête via Javascript à un serveur http (jusque là rien de nouveau !) et d'attendre le retour en ne rafraichissant que certaines données contenues dans le code HTML retourné initialement

Même si dans l'acronyme d'AJAX il est mentionné le mot asynchrone, il n'est pas nécessaire que ce le soit (rappel : un appel synchrone => on attend la réponse du serveur pour continuer/terminer ; asynchrone, => on n'attend pas !). Le choix entre synchrone et asynchrone se fait lors de l'instanciation de l'objet *XMLHttpRequest* avec dans le dernier paramètre *true* pour asynchrone, *false* pour synchrone.

La contrepartie à l'utilisation de la méthode asynchrone est qu'il n'est pas possible de prédire le moment où le serveur va répondre. Lorsque l'objet *XMLHttpRequest* change d'état, il lève un événement (*onreadystatechange*) que l'on va associer à une fonction. Cet événement est levé dès que l'objet *readyState* est modifié. Il peut prendre les valeurs : 0 non initialisée, 1 en chargement, 2 chargée, 3 en cours de traitement, 4 terminée.

Comme précédemment dit, le mode asynchrone permet de ne pas bloquer le navigateur client pendant le chargement de la page. L'exemple qui suit permet d'afficher un message d'attente durant un traitement quelconque, puis l'affichage du résultat de ce traitement. Cela va se faire en deux temps : d'une part afficher un message lors de l'appel initial, puis le retirer lorsque notre *onreadystatechange* passe à 4 (terminé).
On va utiliser la balise HTML DIV qui permet de diviser le document en section. C'est justement une de ces sections que va mettre à jour notre script.

Page PHP qui réalise une requête…

```
<?php
header('Content-Type: text/xml');

//on connect
$dbhost="iparla.iutbayonne.univ-pau.fr";
$dbuser="roose";
$dbpass="drop64";
$ddb="roose";

$dblink=mysql_connect($dbhost,$dbuser,$dbpass);
mysql_select_db($ddb);

//on lance la requete
$query = "SELECT * FROM bourse";
$result = mysql_query($query,$dblink) or die (mysql_error($dblink));
```

```
sleep(2);

//On boucle sur le resultat
echo "<?xml version=\"1.0\"?>\n";
echo "<exemple>\n";

while ($row = mysql_fetch_array($result))
{
        echo "<donnee> $row[0] </donnee>\n";
}
echo "</exemple>\n";

?>
```

Code « Ajax »

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
        <title>Exemple d'attente</title>
<script type="text/javascript">

function ajax()
{
   var xhr=null;

   if (window.XMLHttpRequest) {
      xhr = new XMLHttpRequest();
   }
   else if (window.ActiveXObject)
   {
      xhr = new ActiveXObject("Microsoft.XMLHTTP");
   }
   //on définit l'appel de la fonction au retour serveur
   xhr.onreadystatechange = function() { alert_ajax(xhr); };

   //on affiche le message d'accueil
   //document.getElementById("message").className="msg";
        document.getElementById('zonetraitement').innerHTML = "Recherche des villes";
   //on appelle le fichier php
        xhr.open("GET", "http://iparla.iutbayonne.univ-pau.fr/~roose/ajax/attente.php", true);
   xhr.send(null);
```

```
}

function alert_ajax(xhr)
{
   if (xhr.readyState==4)
   {
         var docXML= xhr.responseXML;
         var items = docXML.getElementsByTagName("donnee")

                  //une boucle sur chaque element "donnee" trouvé
         var Table = '<table border="1">';
                  for (i=0;i<items.length;i++) {
                           Table += '<tr>';
                     Table += '<td>' + items.item(i).firstChild.data + '</td>';
                  }
                  Table += '</tr>';
                  Table += '</table>';

                  document.getElementById("zonetraitement").innerHTML=Table;

   }
}
</script>
</head>
<body>
<p>
<a href="javascript:ajax();">Go...</a>
</p>

<div id="zonetraitement">Veuillez patienter...</div>
<p>
...suite de la page HTML...
</p>
</body>
</html>
```

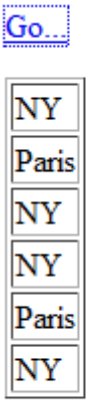
Ce qui donne :

Go...

Veuillez patienter...

...suite de la page HTML...

Puis une fois un clic sur « go » et l'attente de la requête…



Si dans le cas présent, la communication (asynchrone) se déclenche sur le « clic » sur « Go… », il est également possible réaliser des interactions plus intéressantes comme une aide à la saisie par exemple :

BDAjax.php

```
<?php
$hote = 'localhost';
$base = 'roose';
$user = 'roose';
$pass = 'drop64';
$cnx = mysql_connect ($hote, $user, $pass) or die (mysql_error ());
$ret = mysql_select_db ($base) or die (mysql_error ());

/* Vérification */
$qer = mysql_query("select ville from bourse where ville='".$_GET["ville"]."'");
if(mysql_num_rows($qer)>=1)
echo "occupe";
else
echo "libre";
?>
```

Formulaire.html

```
<html> <head>
<title>EssayeAjax</title>Request

<script type="text/javascript">

function writediv(texte)
{
```

```
document.getElementById('zonetraitement').innerHTML = texte;
}

function verificationville(ville)
{
if(window.ActiveXObject) // IE
	xhr_object = new ActiveXObject("Microsoft.XMLHTTP");
else
	if(window.XMLHttpRequest) // FIREFOX
		xhr_object = new XMLHttpRequest();

fichier = "http://iparla.iutbayonne.univ-pau.fr/~roose/ajax/BDAjax.php?ville="+ville;

xhr_object.open("GET", fichier, false);
xhr_object.send(null);

if(xhr_object.readyState == 4) {
	texte = xhr_object.responseText;
	}

if(texte != '')
{
	if(texte == "occupe"){
	writediv(ville+' : est un Nom de ville est occup&eacute; !');
	}
	else if(texte == "libre") {
	writediv(ville+' : est un nom de ville libre vous pouvez l\'ajouter a la BD');
	}
}
	else
	writediv(texte);

}



</script>
</head>
<body>
<form name="formville" action="" methode="GET">
<input name="ville" type="text" onKeyUp="verificationville(this.value)" >
<!-- onKeyUp : c est un evenement lance la fonction js 'verificationville'
this.value : ce qui est tapé ds la zone de texte, ici alias de ville (nom d'onglet)
-->
```

```
<div id="zonetraitement"></div>
</form>

</body>
</html>
```

1 Webservices & SOAP

Un service web, ou webservice – en anglais ça fait plus chic - voire même SOA (Service Oriented Architecture – ça fait plus pro) permet à des applications de conceptions et de réalisations différentes de communiquer entre elles, et qui plus est, sans avoir à se soucier de l'implémentation. Ce couplage entre applications, appelé 'faible', permet ainsi de réaliser de « nouvelles' applications par assemblages et/ou appels de services fournis par d'autres.

Il existe plusieurs méthodes de communication, nous retiendrons ici uniquement le protocole SOAP, issu du RPC (*Remote Procedure Call*) puis du XML-RPC (le même mais avec des appels en XML pour masquer encore plus l'hétérogénéité des implémentations). Le protocole SOAP (*Simple Object Access Protocol)* permet à des objets d'en appeler d'autres distants (comme avec Java/RMI par exemple) mais en utilisant le protocole HTTP comme protocole de communication (il est possible de faire entre autre également du SMTP) sans se soucier de leur implémentation, et avec une uniformité de représentation des données. Il autorise ainsi la communication et l'échange de messages/données entre objets distants.

Le mécanisme est le suivant :

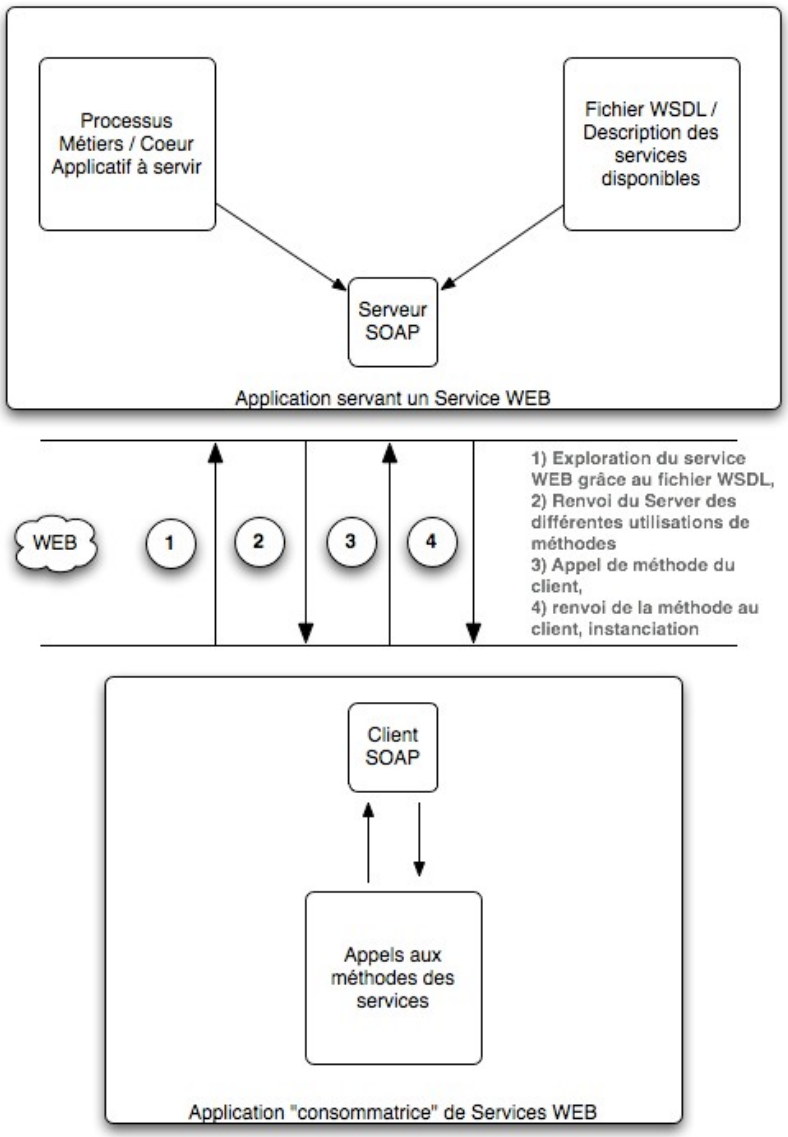


Figure 1:http://vivien-brissat.developpez.com/tutoriels/php/soap/#LI

Lors d'un échange SOAP, le message transmis se décompose en 2 parties :

1. L'enveloppe, qui contient toutes les informations relatives au contenu du message ;
2. Le contenu lui-même qui est formé de données structurées (méthodes formatées, interrogation structurée d'après les besoins du serveur, *etc*.).

Afin de savoir comment utiliser un service web, ce dernier est décrit selon dans un langage de description de service web basé sur XML : le WSDL (*Web Services Description Language)*. Il donne une définition abstraite des services, le détail des types de données échangées, les opérations possibles, le protocole à utiliser ainsi que l'adresse (URL) du service. Le fichier WSDL de chaque webservice peut être publié

dans un annuaire. Le fichier WSDL associé à un webservice représente en quelque sorte sa notice d'utilisation.

Dans sa version 5, le PHP intègre en natif une gestion du protocole SOAP.

Fichier wsdl

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- WSDL file generated by Zend Studio. -->
<definitions name="exemple" targetNamespace="urn:exemple" xmlns:typens="urn:exemple"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:typens0="http://iparla.iutbayonne.univ-pau.fr/~roose/wsdl/moteur.php">
	<message name="retourDate"/>
	<message name="retourDateResponse">
		<part name="retourDateReturn"/>
	</message>
	<portType name="essai_instancePortType">
		<operation name="retourDate">
			<input message="typens:retourDate"/>
			<output message="typens:retourDateResponse"/>
		</operation>
	</portType>
	<binding name="essai_instanceBinding" type="typens:essai_instancePortType">
		<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
		<operation name="retourDate">
			<soap:operation soapAction="urn:essai_instanceAction"/>
			<input>
				<soap:body namespace="urn:exemple" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
			</input>
			<output>
				<soap:body namespace="urn:exemple" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
			</output>
		</operation>
	</binding>
	<service name="exempleService">
		<port name="essai_instancePort" binding="typens:essai_instanceBinding">
			<soap:address
```

```
location="http://iparla.iutbayonne.univ-pau.fr/~roose/wsdl/moteur.php"/>
                </port>
        </service>
</definitions>
```

Moteur.php (serveur)

```
<?php
class DateServer{

//On déclare notre méthode qui renverra la date et la signature du serveur dans un tableau associatif...
function retourDate(){
        $tab = array(
                'serveur' => $_SERVER['SERVER_SIGNATURE'],
                'date' => date("d/m/Y"),
                'auteur' => "service web appele" //attention, unaccent => erreur
                );
        return $tab;
}
}

//Cette option du fichier php.ini permet de ne pas stocker en cache le fichier WSDL, afin de pouvoir faire nos tests
//Car le cache se renouvelle toutes les 24 heures, ce qui n'est pas idéal pour le développement
ini_set('soap.wsdl_cache_enabled', 0);

//Instanciation du SoapServer
$serversoap=new SoapServer("http://iparla.iutbayonne.univ-pau.fr/~roose/wsdl/exemple.wsdl");

// on peut aussi déclarer plus simplement des fonctions
//par l'instruction addFunction() : $serversoap->addFunction("retourDate"); à ce moment-là nous ne faisons pas de classe.

//Noter le style employé pour la déclaration : le nom de la classe est passé en argument de type String, et non pas de variable...
$serversoap->setClass("DateServer");

//Ici, on dit très simplement que maintenant c'est à PHP de prendre la main pour servir le Service WEB : il s'occupera de l'encodage XML, des
//Enveloppes SOAP, de gérer les demandes clientes, etc. Bref, on en a fini avec le serveur SOAP !!!!
$serversoap->handle();
?>
```

Client.php

```
<?php
//Cette option permet d'éviter la mise en cache du WSDL, qui se renouvelle toutes les 24 heures... Pour le développement, ce n'est pas génial !!!
ini_set('soap.wsdl_cache_enabled', 0);

//On doit passer le fichier WSDL du Service en paramètre de l'objet SoapClient
$service=new SoapClient("http://iparla.iutbayonne.univ-pau.fr/~roose/wsdl/exemple.wsdl");

//On accède à la méthode de notre classe DateServeur, déclaré dans notre SoapServer
$taballservices=$service->retourDate();

//On renvoie le résutat de notre méthode, pour voir...
print_r($taballservices); // affiche les éléments d'un tableau
?>
```