5 Application Client/Serveur avec AMFPHP

5.1 Pré requis

Pour cet exemple, il est nécessaire:

- D'installer **Apache avec un module PHP5** ou tout autre serveur web équivalent.
- Télécharger **AMFPHP 1.9** et déposer le contenu de l'archive dans ***{HTDOCS}/amfphp***

5.2 Création d'un service AMFPHP

Dans *{HTDOCS}/amfphp/services*, créer un dossier myservice dans lequel on ajoute le script *MyService.php*

MyService.php

```
<?php
class MyService{
    public function hello(){
        return "Salut !";
    }
}
?>
```

Il est possible de tester l'appel de la méthode en accédant à l'application Flex livrée avec AMFPHP : ***{SERVER URL}/amfphp/browser***

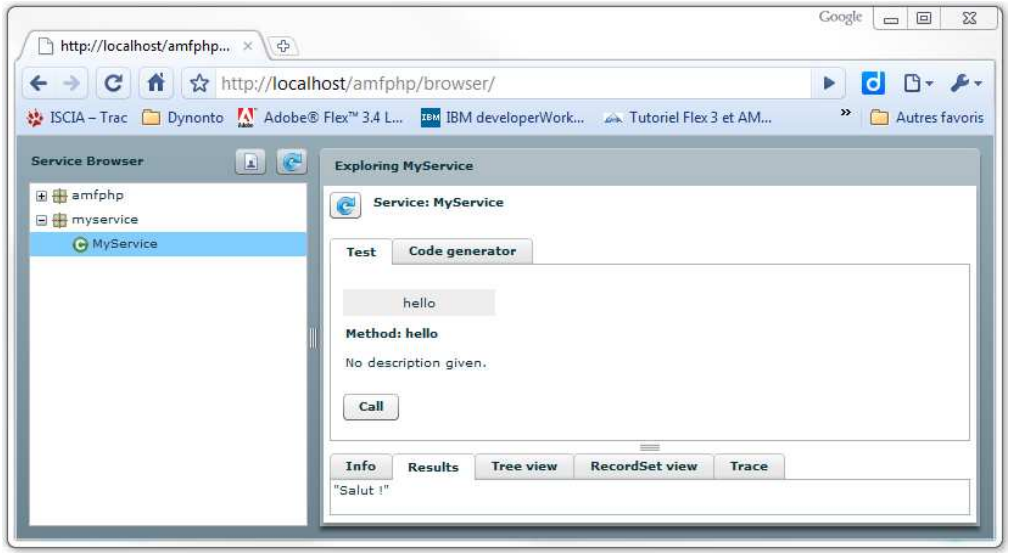

Figure 3 - Interface de test AMFPHP

5.3 Création du projet Flex

Créer un projet **ClientServeurAMFPHP** en spécifiant le répertoire de sortie à ***{HTDOCS} /clientserveur_amf***

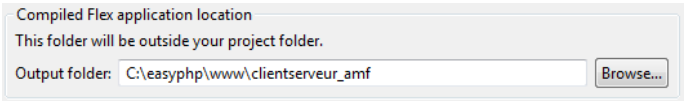

Figure 4 - Répertoire de sortie

La première étape consiste à créer un fichier *services-config.xml* à la racine du projet permettant de faire le lien entre le back-end AMFPHP et le front-end Flex.

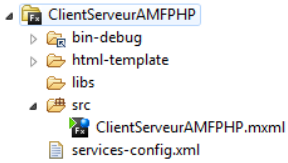

Figure 5 - Fichier services-config.xml

services-config.xml

```
<?xml version="1.0" encoding="utf-8"?>
<services-config>
    <services>
        <service
            id="amfphp-remoting"
            class="flex.messaging.services.RemotingService"
            messageTypes="flex.messaging.messages.RemotingMessage">
            <destination id="amfphp">
                <channels>
                    <channel ref="my-amfphp"/>
                </channels>
                <properties>
                    <source>*</source>
                </properties>
            </destination>
        </service>
    </services>
    <channels>
        <channel-definition
            id="my-amfphp"
            class="mx.messaging.channels.AMFChannel">
            <endpoint
                uri="http://localhost/amfphp/gateway.php"
                class="flex.messaging.endpoints.AMFEndpoint"/>
        </channel-definition>
    </channels>
</services-config>
```

`amfphp` : Nom identifiant la destination
`http://localhost/amfphp/gateway.php` : URL du gateway AMFPHP

Pour que le fichier XML soit pris en considération, il est nécessaire d'ajouter un argument dans ***Project properties → Compilateur Flex → Arguments de compilateur supplémentaires***.

Argument à ajouter

```
-services "{Path_to_FlexBuilderWorkspace}/ClientServeurAMFPHP/services-config.xml"
```

{Path_to_FlexBuilderWorkspace} correspond au chemin absolu vers le workspace en cours d'utilisation par Flash Builder.

5.4 Appel d'un service sans argument

Pour appeler le service AMFPHP précédemment créé, nous utilisons le composant Flex ***RemoteObject***. Cet objet fait référence à la destination définie dans le fichier *services-config.xml* et détermine les méthodes utilisées dans l'application.

Déclaration d'un RemoteObject

```
<fx:Declarations>
    <s:RemoteObject
        id="backendService"
        showBusyCursor="true"
        destination="amfphp"
        source="myservice.MyService">
        <s:method
            name="hello"
            result="trace(event.result as String)"/>
    </s:RemoteObject>
</fx:Declarations>
```

Invocation de méthode distante

```
<s:Button
    label="Call hello"
    click="backendService.getOperation('hello').send()"/>
```

Pour tester le fonctionnement, démarrez l'application en mode **debug**, puis cliquez sur le bouton. Si tout se passe comme prévu, vous devriez voir *« Salut ! »* dans la console.

5.5 Création et appel d'un service avec argument

Nous allons maintenant créer un nouveau service côté serveur prenant un *username* ainsi qu'un *password* en argument et retournant vrai si le couple de valeurs correspond à « *root/1234* ».

Dans le fichier MyService.php, ajouter la fonction suivante

```
public function authenticate($user, $pass){
    return strcmp($user, "root") == 0 && strcmp($pass, "1234") == 0;
}
```

Ajouter la méthode suivante au RemoteObject du fichier ClientServeurAMFPHP.mxml

```
<s:method
    name="authenticate"
    result="trace(event.result as Boolean)">
    <s:arguments>
        <user>root</user>
        <pass>1234</pass>
    </s:arguments>
</s:method>
```

Ajouter le bouton suivant à l'application

```
<s:Button
    label="Check password"
    click="backendService.getOperation('authenticate').send();"/>
```

Le passage d'un paramètre s'effectue à l'aide des balises `<s:arguments>`

5.6 Exercice

Modifier le code de « *Exemple* 3, Application Login » afin d'implémenter une authentification au niveau du service amfphp précédemment mis en place. L'authentification se fait de manière statique (vérifier que le « *Username = root* » et le « *Password = 1234* ») mais obligatoirement par un service amfphp.

5.7 Mapping de classes

Jusqu'à maintenant, nous avons échangé entre le client et le serveur uniquement des objets de type primitif. Si nous voulons transmettre des objets personnalisés, il est intéressant de mapper les classes afin d'éviter de devoir reconstruire les objets manuellement à partir de tableaux ou d'un flux XML.

Commençons par créer un dossier *vo/myservice* dans *{HTDOCS}/amfphp/services.*

Créer un fichier Person.php dans ce nouveau dossier

```
<?php
class Person
{
    var $firstName;
    var $lastName;
    var $_explicitType = "myservice.Person";

    function getFirstName(){
        return $this->firstName;
    }
}
?>
```

Inclure le fichier Person.php dans le fichier MyService.php

```
require_once "../vo/myservice/Person.php";
```

Toujours dans le fichier MyService.php, ajouter les fonctions suivantes

```
public function getFirstName($p){
    return $p->getFirstName();
}

public function getPerson(){
    $p = new Person();
    $p->firstName = "myFirstName";
    $p->lastName = "myName";
    return $p;
}
```

Dans le projet Flex, créer un fichier *src/myservice/Person.as*.

Person.as

```
package myservice
{
    [RemoteClass(alias="myservice.Person")]
    public class Person
    {
        public var firstName:String;
        public var lastName:String;
    }
}
```

Modifier l'application pour appeler les nouveaux services.

ClientServeurAMFPHP.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx"
               minWidth="955" minHeight="600"
               creationComplete="init()">
    <fx:Declarations>
        <!-- Services distants -->
        <s:RemoteObject
            id="backendService"
            showBusyCursor="true"
            destination="amfphp"
            source="myservice.MyService">
            <s:method
                name="hello"
                result="trace(event.result as String)"/>
            <s:method
                name="authenticate"
                result="trace(event.result as Boolean)">
                <s:arguments>
                      <user>root</user>
                      <pass>1234</pass>
                </s:arguments>
            </s:method>
            <s:method
                name="getPerson"
                result="trace(event.result as Person)"/>
            <s:method
                name="getFirstName"
                result="trace(event.result as String)">
                <s:arguments>
                      <arg>{person}</arg>
                </s:arguments>
            </s:method>
        </s:RemoteObject>
    </fx:Declarations>
    <fx:Script>
        <![CDATA[

        import myclient.Person;
        import mx.controls.Alert;

        [Bindable]
```

```
        public var person:Person = new Person();

        public function init():void
        {
            person.firstName = "Foo";
            person.lastName = "Bar";
        }

        ]]>
    </fx:Script>

    <s:layout>
        <s:VerticalLayout horizontalAlign="center"/>
    </s:layout>

    <!-- Boutons utilisés pour invoquer les services -->
    <s:Button
        label="Call hello"
        click="backendService.getOperation('hello').send()"/>
    <s:Button
        label="Check password"
        click="backendService.getOperation('authenticate').send();"/>
    <s:Button
        label="Get person"
        click="backendService.getOperation('getPerson').send();"/>
    <s:Button
        label="Get first name"
        click="backendService.getOperation('getFirstName').send();"/>
</s:Application>
```

5.8 Exercice récapitulatif

Sur la base des réalisations précédentes autour du scénario login/register (tutoriel n°1), il s'agit de créer la RIA complète y correspondant, avec dialogue client/serveur. Le formulaire d'enregistrement sera aussi plus étoffé, avec : nom, prenom, date de naissance, login, password, et email. Ces champs sont tous obligatoires et doivent être validés (règles de validation à définir).

- La RIA doit interagir avec des services, pour l'authentification et pour l'enregistrement. Pour simplifier, la base des utilisateurs est stockée sous forme de fichier type CSV (on ignore les problèmes d'accès concurrentiel).
- L'opération de login affiche comme résulat le message approprié, et l'enregistrement affiche un rappel du login/password nouvellement créé.

Remarque: l'idée de cette réalisation est de mettre en place un échange client/serveur basé sur AMFPHP et utilisant un accès aux services par le composant RemoteObject.