Windows Communication Foundation

1 Prise en main de WCF

1.1 Création d'une bibliothèque WCF simple (et fournie par VS !)

— Dans une nouvelle solution, créer un nouveau projet WCF "bibliothèque de service WCF"
— Vous obtenez : une interface de contrats de service avec 2 services, l'un n'utilisant que des types simples, et l'autre utilisant des types complexes, avec des DataContract. Vous obtenez également un fichier de configuration App.config.

1.2 Hébergement et test en utilisant les outils externes `WcfSvcHost.exe` et `WcfTestClient.exe`

Visual Studio 2008 a introduit 2 outils facilitant la mise au point de services WCF. `WcfSvcHost` permet d'héberger un service WCF sans avoir à créer une assembly dédiée à cet hébergement. `WcfTestClient` fournit une interface de test du service WCF, et évite ainsi de créer une assembly dédiée au test.

Nous allons utiliser conjointement ces deux outils pour tester notre bibliothèque WCF précédemment créée.

Dans Visual Studio 2008 : Allez dans les propriétés du projet de la bibliothèque WCF, et allez sur l'onglet Déboguer. Choisissez dans *Action de démarrage* l'option *Démarrer le programme externe*, et sélectionnez alors `WcfSvcHost.exe`. Dans les options de démarrage, passez comme arguments de la ligne de commande :

```
/service:"nomAssemblyService.dll"
/../..config:App.config
/client:"WcfTestClient.exe"
```

Sauvegardez les propriétés. Vous pouvez maintenant exécuter votre projet (bien qu'il génère une assembly de type `dll`). L'exécution lance tout d'abord l'hôte de services, puis le client de test.

Dans Visual Studio 2010 : Exécutez votre projet (bien qu'il génère une assembly de type `dll`). L'exécution lance tout d'abord l'hôte de services, puis le client de test.

1.3 Hébergement sans utiliser les outils externes

— Créer un projet de type *Application Console*.
— Ajoutez comme référence `System.ServiceModel`.
— Ajoutez une référence au projet de bibliothèque de service.
— Dans le `Main` de votre classe principale, ajoutez des lignes telles que les suivantes :

```
using (ServiceHost serviceHost = new ServiceHost(typeof(Service1)))
    {
        try
        {
            // Open the ServiceHost to start listening for messages.
            serviceHost.Open();
            // The service can now be accessed.
            Console.WriteLine("The service is ready.");
            Console.WriteLine("Press <ENTER> to terminate service.");
```

```
            Console.ReadLine();
            // Close the ServiceHost.
            serviceHost.Close();
        }
        catch (TimeoutException timeProblem)
        {
            Console.WriteLine(timeProblem.Message);
            Console.ReadLine();
        }
        catch (CommunicationException commProblem)
        {
            Console.WriteLine(commProblem.Message);
            Console.ReadLine();
        }
    }
```

— Ajoutez également les using adéquates en début de fichier.
— Créez dans votre projet un fichier de configuration App.config. Vous pouvez ensuite utiliser l'éditeur de configuration WCF pour renseigner les champs de configuration nécessaire. Passez par une configuration simple via basicHttpConfig ou NetTcpBinding (il se peut que par http cela ne fonctionne pas pour des raisons de sécurité insolubles faute de droits suffisants).
— Lancez votre projet : votre service est maintenant hébergé.

1.4 Créer un client WCF en passant par ChannelFactory

— Créer un nouveau projet application console. Référencez le projet de bibliothèque de service et l'assembly System.ServiceModel.
— Créer un fichier de configuration en vous basant sur celui du service.
— Dans le main du client, récupérez un proxy sur le service et utilisez-le avec des lignes de ce style :

```
IService1 serviceProxy =new ChannelFactory<IService1>("ServiceConfiguration").CreateChannel();
Console.WriteLine(serviceProxy.GetData(1));
```

Le nom de la configuration donnée en paramètre de `ChannelFactory` doit être le même que celui de votre fichier de configuration.

1.5 Publication de métadonnées et création d'un client

— Créez un nouvel hébergement (dans un autre projet) en rendant visibles les métadonnées via un deuxième endpoint de ce style :
`<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>`
— Lancez cet hébergement.
— Ouvrez un navigateur à l'adresse du service. Vous pouvez accéder au wsdl généré.
— Créez un client. Pour cela, l'hébergement doit être lancé (pas via Visual Studio car alors on ne peut plus modifier la solution). Ajoutez une référence de service. Entrez l'adresse de votre service. Un proxy est alors généré. Vous pouvez écrire votre client en utilisant ce proxy via ce genre de lignes :

```
Service1Client proxy = new Service1Client();
String s=proxy.GetData(0);
Console.WriteLine(s);
```

Cela nécessite bien sûr de rajouter le `using` adéquate en début de fichier. Vous devez fournir un fichier de configuration `App.config` configurant le client (par exemple en le construisant à partir de celui de l'hébergement, et en modifiant ensuite l'espace de nom du contrat pour mettre celui du proxy).

2 Gestion de recettes

L'objectif du TP est de créer un service WCF et un client WCF pour une application de gestion de recettes. On ne publiera pas de métadonnées côté serveur. Dans une nouvelles solution, on créera 4 projets :

- Projet share : contient l'interface du service et les DataContract ; sera connue du client et du service
- Projet ServiceRecettes : contient l'implémentation du service
- Projet HebergementService : héberge le service dans une application gérée
- Projet Client

Dans le projet share, définir les classes et interfaces nécessaires. Une recette a juste un titre et une liste d'ingrédients (sans quantités associées). Un ingrédient a juste un nom.. Le service doit permettre :

- de rechercher la liste des recettes dans la composition desquelles entre un ingrédient de nom donné,
- de mémoriser le résultat de la dernière recherche comme la sélection courante du client,
- de supprimer une recette donnée de la sélection courante,
- de récupérer la sélection courante,
- d'ajouter une recette donnée à la base de recettes connue du service.

Dans le projet Hebergement, on utilisera un binding `netTcpBinding`.
Dans le projet Client, on récupèrera un proxy sur le service via le code suivant :

```
IServiceRecettes recetteProxy =
new ChannelFactory<IServiceRecettes>("RecetteServiceConfiguration").CreateChannel();
```

qui suppose d'avoir une configuration de ce style :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <client>
      <endpoint name="RecetteServiceConfiguration"
        address="net.tcp://localhost:8001/ServiceRecettes"
        binding="netTcpBinding"
        contract="share.IServiceRecettes"/>
    </client>
  </system.serviceModel>
</configuration>
```

Dans le projet ServiceRecettes, on utilisera des sessions adéquates pour permettre la gestion de la sélection courante.