



Dotnet France
Technologies Sharepoint, SQL Server & .NET

Association Dotnet France

Création et consommation de services WCF

Version 1.1

Jean DALAT



James RAVAILLE

<http://blogs.dotnet-france.com/jamesr>

Sommaire

1	Introduction.....	3
1.1	Présentation	3
1.2	Pré-requis	3
2	Les bases fondamentales de WCF	4
2.1	Les étapes de mise en œuvre	4
2.2	Les éléments A B C	4
2.3	Espaces de nom du Framework .NET	4
3	Création d'un service WCF	5
3.1	Définition du contrat de service.....	5
3.1.1	ServiceContract	5
3.1.2	OperationContract.....	5
3.1.3	DataMember	5
3.2	Mise en œuvre.....	5
3.2.2	Implémentation du contrat de service	7
3.2.3	Utilisation de DataMember	8
4	Hébergement d'un service WCF.....	11
4.1	Présentation	11
4.2	Hébergement du service <i>Médiathèque</i>	11
4.2.1	Le fichier de configuration du service WCF	11
5	Consommer un service WCF	15
5.1.1	La classe proxy	15
5.1.2	Comment générer une classe proxy pour consommer un service WCF ?.....	15
5.2	Ecriture d'un client WCF.....	22
5.2.1	Ecriture d'un client winform pour le service <i>Médiathèque</i>	22
6	Conclusion	26

1 Introduction

1.1 Présentation

WCF est sorti en Novembre 2007. C'est l'un des trois piliers du .Net Framework 3.0 qui n'est autre qu'une « simple » extension du Framework .NET 2.0, auquel il ajoute principalement trois fonctionnalités :

- **WWF** (Windows Workflow Foundation) : apport de fonctionnalités concernant la gestion des Workflows avec une interface de création en mode Graphique dédiée.
- **WPF** (Windows Présentation Foundation) : nouveau moteur de rendu pour les clients lourds Windows voué à remplacer les applications Winform.
- **WCF** (Windows Communication Foundation) : l'objet de ce cours.
- **WCS** (Windows Card Space) : système de gestion d'identités par authentification unique pour le système d'exploitation Windows Vista.

WCF est une technologie qui permet de faciliter la mise en place des applications distribuées en servant de socle commun aux architectures orientées services (SOA : Service Oriented Architecture). WCF se veut être un modèle uniforme de distribution de services :

- Faiblement couplé pour plus de souplesse ;
- Adapté à toutes situations et besoins de communication :
 - o Disponibilité ;
 - o Interopérabilité ;
 - o Sécurité.

1.2 Pré-requis

Pour vous familiariser avec WCF et les architectures SOA, nous vous recommandons de lire le chapitre d'introduction à WCF publié sur Dotnet-France.

De manière à comprendre les exemples de mise en œuvre de WCF, nous vous recommandons aussi de maîtriser un langage .NET (C# ou VB .NET). Pour ce faire, des cours sont aussi à votre disposition sur Dotnet-France.

2 Les bases fondamentales de WCF

2.1 Les étapes de mise en œuvre

Les étapes de mise en œuvre de création d'un service WCF sont les suivantes :

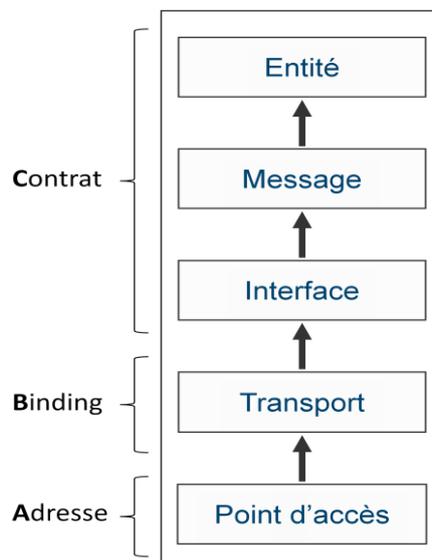
- Première étape : définition du contrat. On définit les signatures des méthodes composant notre service WCF, et du format des données à échanger.
- Seconde étape : implémentation du contrat. On implémente les services.
- Troisième étape : configuration du service. On définit les endPoints, autrement dit les points d'accès au service.
- Quatrième étape : hébergement des services dans une application .NET.

Puis, il ne reste plus qu'à créer une application cliente (quelque soit sa forme, Windows, Web, ou tout autre composant) qui pourra consommer le service WCF.

2.2 Les éléments A B C

WCF repose sur trois éléments :

- Une **A**dresse : adresse à laquelle le client doit se connecter pour utiliser le service.
- Un **B**inding : protocole à utiliser par le client pour communiquer avec le service.
- Un **C**ontrat : infos échangées entre le serveur et le client afin que ce dernier sache comment utiliser le service.



2.3 Espaces de nom du Framework .NET

La mise en œuvre de WCF repose principalement sur 2 espaces de nom du Framework .NET :

- **System.ServiceModel**
- **System.Runtime.Serialization**

3 Création d'un service WCF

3.1 Définition du contrat de service

Comme nous l'avons vu précédemment, le contrat de service est l'entité qui va :

- Etre échangée entre le serveur et le client ;
- Permettre au client de savoir quelles sont les méthodes proposées par le service et comment les appeler.

L'élaboration d'un contrat de service s'effectue au travers des 3 métadonnées suivantes :

- ServiceContract
- OperationContract
- DataMember

3.1.1 ServiceContract

Cette métadonnée est attachée à la définition d'une classe ou, d'une interface. Il sert à indiquer que la classe ou l'interface est un contrat de service.

3.1.2 OperationContract

Cette métadonnée est attachée aux méthodes que l'on souhaite exposer au travers du service WCF. Ainsi, il est techniquement possible de l'exposer au client que certaines méthodes d'une classe.

3.1.3 DataMember

Cet attribut se place avant les propriétés des classes qui définissent les objets que l'on va devoir passer en paramètre au service, ou que celui-ci va devoir nous retourner.

3.2 Mise en œuvre

Soit un service qui permet de réaliser des opérations d'ajout/modification/suppression dans une base de données, représentant la base de données d'une petite médiathèque fictive de films. Elle ne contient que 2 tables : « film » et « dvd ». On considère qu'un DVD peut contenir plusieurs films et qu'un film se trouve sur un seul DVD.

La couche d'accès aux données est encapsulée dans un projet de bibliothèque de classes séparé. Afin de simplifier au maximum les exemples, on accède à cette couche en instanciant un objet de type « *MediathequeDAO* ». Les deux classes issues de la base de données ont été générés par LINQ to SQL, qui a aussi servi à écrire les requêtes permet de lire et gérer les données contenues dans la base de données.

3.2.1.1 Contrat de service défini dans une interface

Voici l'interface de notre service WCF, que nous allons utiliser comme contrat :



```
// C#

using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
using Mediatheque;
namespace MediathequeService
{
    [ServiceContract]
    public interface IMediathequeService
    {
        [OperationContract]
        List<film> GetFilmsList();

        [OperationContract]
        bool AddFilm(string Titre, string Url, int DVDId);

        [OperationContract]
        List<dvd> GetAllDVD();

        [OperationContract]
        bool AddDVD(dvd UnDVD);

        [OperationContract]
        bool ModifyDVD(dvd unDVD);

        [OperationContract]
        bool RemoveDVD(dvd UnDVD);
    }
}
```

```
' VB

Imports Mediatheque

<ServiceContract()> _
Public Interface IMediathequeService

    <OperationContract()> _
    Function ListFilm() As List(Of film)

    <OperationContract()> _
    Function AddFilm(ByVal Nom As String, ByVal url As String, ByVal id
As Integer) As Boolean

    <OperationContract()> _
    Function GetAllDVD() As List(Of dvd)

    <OperationContract()> _
    Function AddDVD(ByVal undvd As dvd) As Boolean

    <OperationContract()> _
    Function ModifyDVD(ByVal undvd As dvd) As Boolean

    <OperationContract()> _
    Function RemoveDVD(ByVal undvd As dvd) As Boolean

End Interface
```

3.2.2 Implémentation du contrat de service

Une fois l'écriture la définition de notre contrat réalisée, nous devons créer une classe implémentant cette interface. Cette classe contiendra l'implémentation de chaque méthode. Ainsi, la classe en elle-même n'a pas « conscience » d'être en fait l'implémentation d'un service WCF. Elle l'est devenue uniquement grâce aux métadonnées utilisées dans l'interface qu'elle implémente.

```
public class MediathequeServiceImpl : IMediathequeService
{
    MediathequeDAO DAO = new MediathequeDAO();

    public List<film> GetFilmsList()
    {
        return DAO.GetAllFilms();
    }

    public bool AddFilm(string Titre, string Url, int DVDId)
    {
        film NouveauFilm = new film();

        NouveauFilm.filmTitre = Titre;
        NouveauFilm.filmURL = Url;
        NouveauFilm.dvdID = DVDId;

        return DAO.AddFilm(NouveauFilm);
    }

    public List<dvd> GetAllDVD()
    {
        return DAO.GetAllDVD();
    }

    public bool AddDVD(dvd UnDVD)
    {
        return DAO.AddDVD(UnDVD);
    }

    public bool ModifyDVD(dvd unDVD)
    {
        return DAO.ModifyDVD(unDVD);
    }

    public bool RemoveDVD(dvd UnDVD)
    {
        return DAO.RemoveDVD(UnDVD);
    }
}
```

```
' VB

Public Class MediathequeServiceImpl
    Implements IMediathequeService

    Private dao As New MediathequeDAO

    Public Function GetAllDVD() As List(Of dvd) Implements
IMediathequeService.GetAllDVD
        Return dao.GetAllDVD()
    End Function

    Public Function AddDVD(ByVal undvd As dvd) As Boolean Implements
IMediathequeService.AddDVD
        Return dao.AddDVD(undvd)
    End Function

    Public Function AddFilm(ByVal titre As String, ByVal url As String,
ByVal dvdid As Integer) As Boolean Implements IMediathequeService.AddFilm

        Dim film As New film

        film.filmTitre = titre
        film.filmURL = url
        film.dvdID = dvdid

        Return dao.AddFilm(film)
    End Function

    Public Function ListFilm() As List(Of film) Implements
IMediathequeService.ListFilm
        Return dao.GetAllFilms()
    End Function

    Public Function ModifyDVD(ByVal undvd As dvd) As Boolean Implements
IMediathequeService.ModifyDVD
        Return dao.ModifyDVD(undvd)
    End Function

    Public Function RemoveDVD(ByVal undvd As dvd) As Boolean Implements
IMediathequeService.RemoveDVD
        Return dao.RemoveDVD(undvd)
    End Function

End Class
```

3.2.3 Utilisation de DataMember

Comme indiqué précédemment, les classes Film et DVD ont été générées en amont de ce projet (exemple grâce à la technologie LINQ to SQL). Par défaut, les classes ainsi générées ne sont pas utilisables avec WCF car leurs propriétés ne comportent pas l'attribut *DataMember* nécessaire à WCF pour savoir quelles propriétés des objets doivent être sérialisées et sont donc utilisables dans un service WCF.

Afin de rendre ces classes « compatibles » avec WCF, il faut dire à LINQ d'ajouter l'attribut « DataMember » à toutes les propriétés, on a placé la propriété **SérialisationMode** du diagramme Linq To SQL de classes LINQ à **Unidirectionnal**.

3.2.3.1 Exemple de code « compatible WCF » généré par LINQ avec les attributs *DataMember* et *DataContract*

```
// C#

[Table (Name="dbo.dvd" )
  [DataContract ()]
public partial class dvd : INotifyPropertyChanging,
INotifyPropertyChanged
  {
  ...

      [Column (Storage="_dvdID", DbType="Int NOT NULL",
IsPrimaryKey=true)]
      [DataMember (Order=1)]
      public int dvdID
      {
          get
          {
              return this._dvdID;
          }
          set
          {
              if ((this._dvdID != value))
              {
                  this.OndvdIDChanging (value);
                  this.SendPropertyChanging ();
                  this._dvdID = value;
                  this.SendPropertyChanged ("dvdID");
                  this.OndvdIDChanged ();
              }
          }
      }
  }
}
```

```
' VB .NET

<Table (Name:="dbo.dvd"), _
  DataContract()>
Partial Public Class dvd
  Implements System.ComponentModel.INotifyPropertyChanging,
  System.ComponentModel.INotifyPropertyChanged

  Private Shared emptyChangingEventArgs As PropertyChangingEventArgs
= New PropertyChangingEventArgs (String.Empty)

  Private _dvdID As Integer

  Private _dvdDateCreation As Date

  Private _dvdDescription As String

  Private _film As EntitySet(Of film)

  Public Sub New()
    MyBase.New
    Me.Initialize
  End Sub

  <Column (Storage:="_dvdID", DbType:"Int NOT NULL",
  IsPrimaryKey:=true),
  DataMember (Order:=1)>
  Public Property dvdID() As Integer
    Get
      Return Me._dvdID
    End Get
    Set
      If ((Me._dvdID = value)
        = false) Then
        Me.OndvdIDChanging (value)
        Me.SendPropertyChanging
        Me._dvdID = value
        Me.SendPropertyChanged ("dvdID")
        Me.OndvdIDChanged
      End If
    End Set
  End Property
End Class
```

Note: Pour pouvoir être sérialisée, un attribut doit posséder un accesseur en lecture (get) et en écriture (set). En effet, il faut pouvoir lire la propriété au moment de la sérialiser et lui assigner une valeur au moment de désérialiser. Ainsi on ne peut pas sérialiser et utiliser en WCF des objets aillant des propriétés en lecture seule ou en écriture seule.

4 Hébergement d'un service WCF

Maintenant que le service WCF est créé, nous allons l'héberger dans une application.

4.1 Présentation

Un service WCF peut être hébergé dans diverses applications :

- Dans une application cliente (Winform, service Windows...)
- Dans une application ASP .NET, elle-même hébergée sur un serveur IIS

En fonction du type d'application possède ses avantages et ses inconvénients notamment en ce qui concerne les protocoles utilisables pour héberger le service. Ainsi, par exemple, si un service est hébergé sous IIS 6.0, on ne pourra utiliser que les protocoles HTTP / HTTPS. Nous verrons plus en détail les contraintes liées aux technologies d'hébergement dans les chapitres suivants.

4.2 Hébergement du service *Médiathèque*

Pour cette simple introduction à WCF nous allons juste chercher à héberger notre service sur **WCF Service Host**. WSH est le nom de l'hébergeur intégré à Visual Studio 2008. C'est lui qui héberge par défaut un service WCF créé avec le « template » WCF Service Library.

4.2.1 Le fichier de configuration du service WCF

La configuration d'un service WCF est stockée dans un fichier XML d'extension *.config* (App.config ou Web.config selon le type d'application). La configuration spécifique à WCF se trouve entre les balises `<system.serviceModel>` et `</system.serviceModel>`. On retrouve ici le nom de l'espace de nom spécifique à WCF.

4.2.1.1 Notion de Endpoint

La notion d'endpoint est un point clé dans la configuration de l'hébergement d'un service WCF. Un *endpoint* est un point de connexion auquel un client va pouvoir se connecter pour utiliser le service. Un service peut être exposé à travers un ou plusieurs endpoints. L'utilisation de plusieurs endpoints est pertinente pour assurer la compatibilité avec un maximum de clients possibles.

4.2.1.2 Notion de Behavior (comportement)

Nous venons de survoler très rapidement la notion d'endpoint, pour être complet sur le contenu d'un fichier de base d'un fichier de configuration de service, voyons à présent une autre notion de base des services WCF : les comportements.

Un **Behavior** est un paramètre qui va affecter le fonctionnement local du service. De ce fait, les paramètres de comportement sont propres au serveur et au(x) client(s) et ne sont jamais échangés sur le réseau.

Un des Behavior les plus simples est « **ServiceDebug.IncludeExceptionDetailInFaults** » qui permet de dire au service si oui ou non il doit renvoyer au client des détails si une exception est levée de son côté ou s'il doit se contenter d'un message d'erreur « standard ».

Comme expliqué plus haut, ce Behavior affecte le comportement local du serveur lorsque qu'une exception est levée dans le code (doit-il oui ou non renvoyer des détails sur l'exception ?) mais cela n'accepte pas directement le(s) client(s) potentiel(s).

4.2.1.3 Exemple : mise en place du fichier de configuration du service Mediatheque



```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>

  <system.web>
    <compilation debug="true" />
  </system.web>

  <!-- When deploying the service library project, the content of the
  config file must be added to the host's
  app.config file. System.Configuration does not support config files for
  libraries. -->

  <system.serviceModel>

    <services>
      <service
behaviorConfiguration="MediathequeService.MediathequeServiceBehavior"
        name="MediathequeService.MediathequeServiceImpl">

        <endpoint address="" binding="wsHttpBinding"
contract="MediathequeService.IMediathequeService">
          <identity>
            <dns value="localhost" />
          </identity>
        </endpoint>
        <endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange" />

        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8731/MediathequeService"
/>
          </baseAddresses>
        </host>

      </service>
    </services>
    <behaviors>

      <serviceBehaviors>
        <behavior name="MediathequeService.MediathequeServiceBehavior">
          <!-- To avoid disclosing metadata information,
          set the value below to false and remove the metadata endpoint
          above before deployment -->
          <serviceMetadata httpGetEnabled="True"/>
          <!-- To receive exception details in faults for debugging
          purposes,
          set the value below to true. Set to false before deployment
          to avoid disclosing exception information -->
          <serviceDebug includeExceptionDetailInFaults="False" />
        </behavior>
      </serviceBehaviors>

    </behaviors>
  </system.serviceModel>
</configuration>

```

4.2.1.3.1 Analyse rapide du fichier de configuration de l'exemple

Dans le fichier de configuration de notre service Médiathèque présenté ci-dessus, nous retrouvons les éléments importants de la configuration d'un service WCF :

Les éléments de configuration du service se placent entre les balises `<System.ServiceModel>` et `</System.ServiceModel>`

Les paramètres de l'hôte (l'adresse de base) :

Tout d'abord, intéressons nous à la configuration de l'adresse de base du service, celle que le client devra interroger pour accéder au service.

```
<baseAddresses>
  <add baseAddress="http://localhost:8731/MediathequeService"/>
</baseAddresses>
```

Il s'agit de l'adresse de base du service. Les adresses indiquées dans les endpoints sont relatifs à cette adresse de base.

Les Endpoints et leurs trois éléments de configuration :

- L'adresse : `endpoint address=""`
- Le binding: `binding="wsHttpBinding"`
- Le contract: `contract="MediathequeService.IMediathequeService">`.

Concernant l'adresse, un champ `address` vide signifie que le service est disponible « à la racine » de l'adresse de base.

Concernant le contrat, on constate aussi que ce qui est partagé est bien l'interface (`IMediathequeService`) du service et non pas son implémentation (`MediathequeServiceImpl`). C'est l'application du 3eme grand principe de la SOA : Le serveur et le(s) client(s) ne partagent que des contrats (les interfaces) et non pas du code (implémentation).

En étudiant plus en détail ce fichier de configuration, on se rend compte qu'il y a deux endpoints configurés : l'endpoint d'adresse « mex » a une fonction spéciale qui sera étudiée en détail dans les chapitres suivants : il sert à exposer aux clients les opérations disponibles sur le service. Il est généré automatiquement par le système et dispose d'un binding propre, « `MexhttpBinding` », et de son propre contrat `IMetadataExchange` .

D'après son attribut `address` cet endpoint est situé dans le répertoire `mex`. Selon les principes évoqués plus haut, pour y accéder, le client devra donc interroger l'adresse :

`http://localhost:8731/MediathequeService/mex`

Les Behaviors :

```
<serviceDebug includeExceptionDetailInFaults="False" />
```

```
<serviceMetadata httpGetEnabled="True"/> (gestion de la publication des opérations du service)
```

5 Consommer un service WCF

Après avoir vu sommairement la création et l'hébergement d'un service WCF, nous allons maintenant voir comment le consommer dans une application .NET.

5.1.1 La classe proxy

Nous avons vu que les échanges entre un service WCF et ces clients se font généralement par un mécanisme de sérialisation/désérialisation d'objets dans un langage dérivé du XML. De plus nous avons dit aussi que les clients obtiennent des informations sur un service donné grâce à l'appel d'un fichier d'extension .WSDL et que celui-ci contient la liste des méthodes exposées par le service ainsi que comment les utiliser (quels sont éventuellement les paramètres à fournir).

La combinaison de toutes les informations contenues dans le fichier WSDL va permettre à Visual Studio de générer ce que l'on appelle une « **classe proxy** ». Le rôle de cette classe proxy va être de masquer la complexité des mécanismes de communication entre le service web WCF et le client afin que le programmeur n'ait en réalité qu'un seul objet à utiliser (plus éventuellement les types complexes utilisés par le service qui seront alors eux aussi récupérés automatiquement). Cet objet exposera « localement » les méthodes proposées par le service distant. Comme son nom de « proxy » l'indique, le rôle de cette classe est de servir d'intermédiaire entre le client et le service.

5.1.2 Comment générer une classe proxy pour consommer un service WCF ?

Il existe deux méthodes pour générer une classe proxy :

- En ajoutant une référence de service dans le projet consommant le service WCF
- En utilisant l'utilitaire *svcutil.exe* (méthode manuelle).

Nous allons brièvement voir comment utiliser *svcutil.exe* pour générer une classe proxy permettant de consommer notre service Médiathèque.

5.1.2.1 Générer le proxy du service Médiathèque à l'aide de *svcutil.exe*

Svcutil.exe est un utilitaire en ligne de commande permettant de générer des proxys pour les clients WCF en se connectant au service à consommer et en analysant le fichier WSDL de celui-ci. Il est fourni dans le SDK Windows et se trouve par défaut à l'emplacement : **C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin**

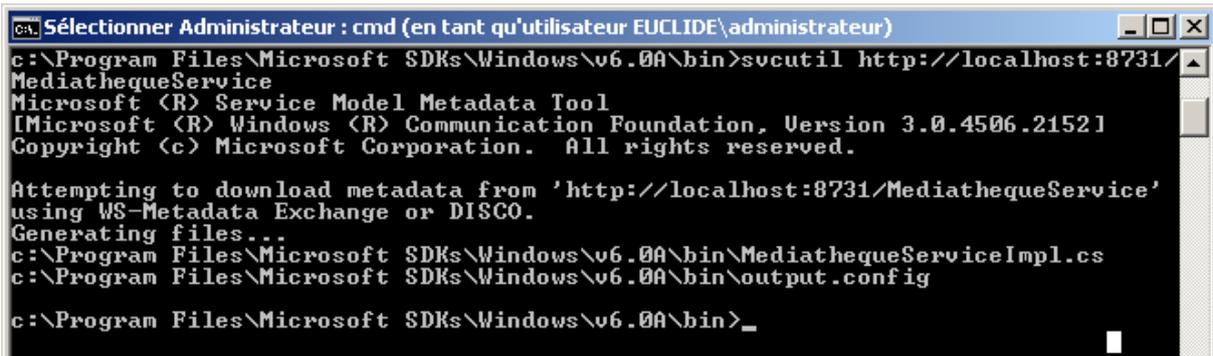
Pour commencer, on lance le service WCF *MediathequeService*, afin de s'assurer qu'il fonctionne correctement.

Ensuite, on exécute *svcutil.exe* dans une fenêtre de commandes DOS pour Visual Studio 2008. La syntaxe de base est la suivante :

Svcutil.exe <adresse du service>

Dans notre cas, d'après les informations du fichier de configuration du service étudié précédemment, nous allons donc taper la commande suivante :

Svcutil http://localhost:8731/MediathequeService



```
cmd Sélectionner Administrateur : cmd (en tant qu'utilisateur EUCLIDE\administrateur)
c:\Program Files\Microsoft SDKs\Windows\v6.0A\bin>svcutil http://localhost:8731/
MediathequeService
Microsoft (R) Service Model Metadata Tool
[Microsoft (R) Windows (R) Communication Foundation, Version 3.0.4506.2152]
Copyright (c) Microsoft Corporation. All rights reserved.

Attempting to download metadata from 'http://localhost:8731/MediathequeService'
using WS-Metadata Exchange or DISCO.
Generating files...
c:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\MediathequeServiceImpl.cs
c:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\output.config
c:\Program Files\Microsoft SDKs\Windows\v6.0A\bin>_
```

L'utilitaire est allé interroger les « Metadatas » du service c'est-à-dire les infos WSDL et à la suite de cela générer deux fichiers.

- MediathequeServiceImpl : ce fichier contient la classe proxy en elle-même ainsi que les types complexes utilisés par le service : les classes « film » et « DVD »
- Output.config : Fichier de configuration du client, qu'il conviendra par la suite de renommer en « App.config »
-

5.1.2.1.1 Survol rapide du fichier proxy généré.

Dans ce fichier proxy, nous retrouvons notamment 3 classes :

- **Film** : la « traduction » du type complexe du même nom exposé par le service :



```
//C#
public partial class film : object,
System.Runtime.Serialization.IExtensibleDataObject
{
    private System.Runtime.Serialization.ExtensionDataObject
extensionDataField;

    private int filmIDField;

    private string filmTitreField;

    private string filmURLField;

    private int dvdIDField;

[...]
```

```
[System.Runtime.Serialization.DataMemberAttribute()]
public int filmID
{
    get
    {
        return this.filmIDField;
    }
    set
    {
        this.filmIDField = value;
    }
}

[System.Runtime.Serialization.DataMemberAttribute()]
public string filmTitre
{
    get
    {
        return this.filmTitreField;
    }
    set
    {
        this.filmTitreField = value;
    }
}

[System.Runtime.Serialization.DataMemberAttribute()]
public string filmURL
{
    get
    {
        return this.filmURLField;
    }
    set
    {
        this.filmURLField = value;
    }
}
```



```
\vb

Partial Public Class film
    Inherits Object
    Implements System.Runtime.Serialization.IExtensibleDataObject,
System.ComponentModel.INotifyPropertyChanged

[...]

    <System.Runtime.Serialization.DataMemberAttribute()> _
    Public Property filmID() As Integer
        Get
            Return Me.filmIDField
        End Get
        Set
            If (Me.filmIDField.Equals(value) <> true) Then
                Me.filmIDField = value
                Me.RaisePropertyChanged("filmID")
            End If
        End Set
    End Property

    <System.Runtime.Serialization.DataMemberAttribute()> _
    Public Property filmTitre() As String
        Get
            Return Me.filmTitreField
        End Get
        Set
            If (Object.ReferenceEquals(Me.filmTitreField, value) <>
true) Then
                Me.filmTitreField = value
                Me.RaisePropertyChanged("filmTitre")
            End If
        End Set
    End Property

    <System.Runtime.Serialization.DataMemberAttribute()> _
    Public Property filmURL() As String
        Get
            Return Me.filmURLField
        End Get
        Set
            If (Object.ReferenceEquals(Me.filmURLField, value) <>
true) Then
                Me.filmURLField = value
                Me.RaisePropertyChanged("filmURL")
            End If
        End Set
    End Property

    <System.Runtime.Serialization.DataMemberAttribute(Order:=3)> _
    Public Property dvdID() As Integer
        Get
            Return Me.dvdIDField
        End Get
        Set
            If (Me.dvdIDField.Equals(value) <> true) Then
                Me.dvdIDField = value
                Me.RaisePropertyChanged("dvdID")
            End If
        End Set
    End Property
End Class
```

Nous retrouvons les propriétés de la classe « film » exposées coté serveur, avec pour chacune d'entre elles, les accesseurs get et set correspondants. Ce code est en fait à peu de choses près le même que celui généré par LINQ coté serveur.

DVD : la « traduction » du type complexe du même nom exposé par le service. La démarche étant la même que pour le type *film* le code n'est pas présenté ici.

MediathequeServiceClient : C'est la classe proxy en elle-même, celle que le client développeur devra instancier dans le code source du client pour interagir avec le service WCF

Nous retrouvons dans cette classe, l'ensemble des méthodes exposées par le service ainsi que les paramètres de type complexe qu'elles prennent éventuellement en argument et dont nous venons de voir l'implémentation coté client.

```
//C#
public partial class MediathequeServiceClient :
System.ServiceModel.ClientBase<IMediathequeService>, IMediathequeService
{

    public MediathequeServiceClient ()
    {
    }

    ... [Le code masqué concerne 4 autres surcharges du constructeur, elles
seront étudiées plus tard dans ce cours.]

    public ClientMediathequeService.film[] GetFilmsList ()
    {
        return base.Channel.GetFilmsList ();
    }

    public bool AddFilm (string Titre, string Url, int DVDId)
    {
        return base.Channel.AddFilm (Titre, Url, DVDId);
    }

    public ClientMediathequeService.dvd[] GetAllDVD ()
    {
        return base.Channel.GetAllDVD ();
    }

    public bool AddDVD (ClientMediathequeService.dvd UnDVD)
    {
        return base.Channel.AddDVD (UnDVD);
    }

    public bool ModifyDVD (ClientMediathequeService.dvd unDVD)
    {
        return base.Channel.ModifyDVD (unDVD);
    }

    public bool RemoveDVD (ClientMediathequeService.dvd UnDVD)
    {
        return base.Channel.RemoveDVD (UnDVD);
    }
}
```

```
' VB .NET

Partial Public Class MediathequeServiceClient
    Inherits System.ServiceModel.ClientBase(Of
MediathequeService.IMediathequeService)
    Implements MediathequeService.IMediathequeService

    Public Sub New()
        MyBase.New
    End Sub

[...

    Public Function GetFilmsList() As MediathequeService.film()
Implements MediathequeService.IMediathequeService.GetFilmsList
        Return MyBase.Channel.GetFilmsList
    End Function

    Public Function AddFilm(ByVal Titre As String, ByVal Url As
String, ByVal DVDId As Integer) As Boolean Implements
MediathequeService.IMediathequeService.AddFilm
        Return MyBase.Channel.AddFilm(Titre, Url, DVDId)
    End Function

    Public Function GetAllDVD() As MediathequeService.dvd()
Implements MediathequeService.IMediathequeService.GetAllDVD
        Return MyBase.Channel.GetAllDVD
    End Function

    Public Function AddDVD(ByVal UnDVD As MediathequeService.dvd) As
Boolean Implements MediathequeService.IMediathequeService.AddDVD
        Return MyBase.Channel.AddDVD(UnDVD)
    End Function

    Public Function ModifyDVD(ByVal unDVD As MediathequeService.dvd)
As Boolean Implements MediathequeService.IMediathequeService.ModifyDVD
        Return MyBase.Channel.ModifyDVD(unDVD)
    End Function

    Public Function RemoveDVD(ByVal UnDVD As MediathequeService.dvd)
As Boolean Implements MediathequeService.IMediathequeService.RemoveDVD
        Return MyBase.Channel.RemoveDVD(UnDVD)
    End Function
End Class
```

5.1.2.1.2 Survol rapide du fichier de configuration généré (output.config)

Le deuxième fichier généré est le fichier output.config. C'est le fichier de configuration XML du client. Tout comme pour le fichier précédent, c'est en analysant le fichier WSDL que *svcutil* a généré ce fichier.

De plus, des paramètres par défaut ont été ajoutés pour la gestion des timeouts et les tailles des buffers. On peut bien sûr éditer ce fichier à la main afin de modifier ces valeurs si on travail sur un réseau particulièrement lent ou avec des volumes de données particulièrement importants.

Enfin, on retrouve ici aussi tous les éléments (Adresse, Binding et Contract) qui caractérisent l'endpoint sur lequel on s'est connecté.

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <bindings>
      <wsHttpBinding>
        <binding name="WSHttpBinding_IMediathequeService"
closeTimeout="00:01:00"
openTimeout="00:01:00" receiveTimeout="00:10:00"
sendTimeout="00:01:00"
bypassProxyOnLocal="false" transactionFlow="false"
hostNameComparisonMode="StrongWildcard"
maxBufferSize="524288"
maxReceivedMessageSize="65536"
messageEncoding="Text" textEncoding="utf-8"
useDefaultWebProxy="true"
allowCookies="false">
          <readerQuotas maxDepth="32"
maxStringLength="8192" maxArrayLength="16384"
maxBytesPerRead="4096"
maxNameTableCharCount="16384" />
          <reliableSession ordered="true"
inactivityTimeout="00:10:00"
enabled="false" />
          <security mode="Message">
            <transport clientCredentialType="Windows"
proxyCredentialType="None"
realm="" />
            <message clientCredentialType="Windows"
negotiateServiceCredential="true"
algorithmSuite="Default"
establishSecurityContext="true" />
          </security>
        </binding>
      </wsHttpBinding>
    </bindings>
    <client>
      <endpoint address="http://localhost:8731/MediathequeService"
binding="wsHttpBinding"
bindingConfiguration="WSHttpBinding_IMediathequeService"
contract="IMediathequeService"
name="WSHttpBinding_IMediathequeService">
        <identity>
          <dns value="localhost" />
        </identity>
      </endpoint>
    </client>
  </system.serviceModel>
</configuration>

```

5.1.2.1.3 Survol rapide du fichier de configuration généré (output.config)

Le deuxième fichier généré est le fichier *output.config*. C'est le fichier de configuration XML du client. Tout comme pour le fichier précédent, c'est en analysant le fichier WSDL que *svcutil* a généré ce fichier.

De plus, des paramètres par défaut ont été ajoutés pour la gestion des timeouts et les tailles des buffers. On peut bien sûr éditer ce fichier à la main afin de modifier ces valeurs si on travail sur un réseau particulièrement lent ou avec des volumes de données particulièrement importants.

Enfin, on retrouve ici aussi tous les éléments (Adresse, Binding et Contract) qui caractérisent l'endpoint par lequel on s'est connecté.

5.2 Ecriture d'un client WCF

Pour finir cette présentation de WCF, nous allons voir comment écrire un client qui utilise les fichiers générés par *svcutil.exe*, afin d'utiliser véritablement notre premier service WCF.

5.2.1 Ecriture d'un client winform pour le service Médiathèque

Nous allons coder assez rapidement un client winform qui consommera notre service Médiathèque : pour simplifier le client n'utilisera réellement que trois des méthodes proposées par le service. Ce projet sera inclus dans la même solution que le reste du code et sera lancé en même temps que le service WCF lors du débogage.

- **GetFilmsList()** : Renvoie la liste de tous les films disponibles dans la médiathèque
- **GetAllDVD()** : Renvoie la liste de tous les DVD
- **AddFilm()** : Ajoute un film dans la liste

Enfin, on notera que par soucis de synthèse, seules seront détaillées les parties du code relevant directement du cours WCF.

5.2.1.1 IHM

L'IHM de cet exemple est une simple application Winform avec 2 fenêtres :

La fenêtre principale avec une barre de menu et une grille de données ancrée sur la fenêtre parent. Elle est destinée à recevoir la liste brute de tous les films.

Une petite fenêtre supplémentaire composée de deux zones de saisies et d'une ListBox permet d'ajouter un film dans la médiathèque. La ListBox permet de sélectionner simplement le DVD qui contient le nouveau film.



5.2.1.2 Ecriture de la partie client WCF de l'application.

Les différentes étapes sont les suivantes :

- On commence par ajouter une application Winform à la solution et on construit l'IHM
- On importe les deux fichiers générés précédemment qui se trouvent dans le répertoire de l'utilitaire svcutil.exe (clic droit sur le projet → ajouter un élément existant)
- On renomme ensuite le fichier de code en MediathequeServiceClient.cs et le fichier output.config en App.config
- On modifie le namespace du fichier MediathequeServiceClient.cs afin qu'il soit identique à celui du projet Winform que l'on vient de créer
- On rajoute les références à System.ServiceModel et System.Runtime.Serialization au projet.

5.2.1.2.1 Instanciation et utilisation de la classe proxy.

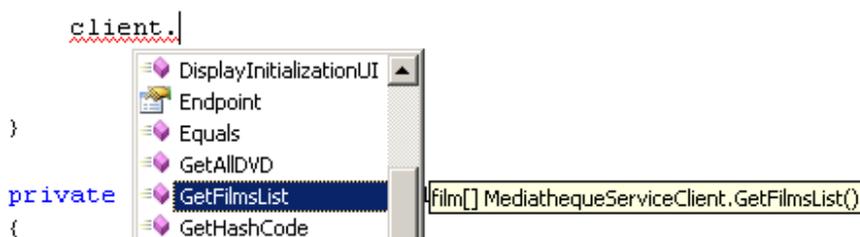
Maintenant que l'IHM est prête et que les réglages préliminaires sont faits, concentrons-nous sur l'utilisation de l'objet proxy que l'on a généré précédemment :

Ajoutons une instance de la classe proxy MediathequeServiceClient. Pour cet exemple nous utilisons le constructeur par défaut qui ne prend pas d'argument en paramètre.

```
//C#
private MediathequeServiceClient client = new MediathequeServiceClient();
```

```
\vb
Private client As New MediathequeService.MediathequeServiceClient()
```

Regardons maintenant ce que l'Intellisense nous propose comme méthodes sur notre objet client pour récupérer dans un premier temps la liste de tous les films :



On constate que le client propose directement une méthode GetFilmsList() retournant un tableau d'objet de type « film ». On peut difficilement faire plus simple ! Par extension, on comprend rapidement que pour peupler la DataGridView de la fenêtre principale il suffit juste d'écrire quelque chose comme :

```
//C#
private void ChargerFilms ()
{
    DGFilms.DataSource = client.GetFilmsList();
}
```

```
' VB .NET

Private Sub ChargerFilms ()
    DGFilms.DataSource = client.GetFilmsList ()
End Sub
```

Il suffira ensuite d'appeler `chargerFilms()` dans le code de l'événement clic de l'option « Charger les données » du menu « Fichier ».

De la même manière, on remplit la liste des DVD disponibles dans la fenêtre d'ajout de films grâce à la méthode `GetAllDVD()` de l'objet client à l'aide d'un code comme :

```
//c#

private void FrmAjoutFilm_Load(object sender, EventArgs e)
{
    foreach (dvd d in client.GetAllDVD())
        lstDVD.Items.Add(d);
}
```

```
' VB .NET

Private Sub FrmAjoutFilm_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    For Each f As MediathequeService.film In client.GetAllDVD()
        lstdvd.Items.Add(f)
    Next

End Sub
```

Enfin, pour l'ajout de films on utilise la méthode `Addfilm()` toujours depuis notre objet client :

```
client.AddFilm(|
bool MediathequeServiceClient.AddFilm (string Titre, string Url, int DVDId)
```

Cette méthode prend trois paramètres :

- Le titre du film (chaîne de caractères)
- Une URL pour consulter le résumé (chaîne de caractères)
- Une référence à l'ID du DVD dans lequel il est stocké (entier)

Le type de retour est un booléen qui fera ici très sommairement office de système de gestion d'erreur (la gestion des erreurs en WCF sera étudiée en détail dans d'autres chapitres de ce cours).

Cette méthode renvoie *true* si tout s'est bien passé et que le film a pu être ajouté ou *false* dans le cas contraire.

Voici pour terminer un exemple de code permettant l'enregistrement d'un nouveau Film dans la base de données.

```
//C#  
  
private void btnEnregistrer_Click(object sender, EventArgs e)  
{  
    film nouveaufilm = new film();  
    //prepare la recuperation de l'ID du dvd selectionné  
    dvd dvdselectionne = (dvd) lstDVD.SelectedItem;  
    //construit un objet film (facultatif à la vue du type de //parametres a fournir)  
    nouveaufilm.dvdID = dvdselectionne.dvdID;  
    nouveaufilm.filmTitre = txtNom.Text;  
    nouveaufilm.filmURL = txtpageInternet.Text;  
    //Envoi des parametres et vérification du bon déroulement de //l'operation  
  
    if(client.AddFilm(nouveaufilm.filmTitre,nouveaufilm.filmURL,nouveaufilm.dvdID)  
        MessageBox.Show("Le nouveau film a bien été enregistré","Travail  
terminé",MessageBoxButtons.OK,MessageBoxIcon.Information);  
        else  
            MessageBox.Show("Erreur lors de l'enregistrement", "Travail  
terminé", MessageBoxButtons.OK, MessageBoxIcon.Stop);  
}
```

```
'VB.NET
```

```
Private Sub btnenregistrer_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles btnenregistrer.Click  
  
    Dim unfilm As New MediathequeService.film()  
  
    unfilm.filmTitre = txttitre.Text  
    unfilm.filmURL = txturl.Text  
    unfilm.dvdID = CType(lstdvd.SelectedItem, MediathequeService.dvd).dvdID  
  
    If client.AddFilm(unfilm.filmTitre, unfilm.filmURL, unfilm.dvdID) Then  
        MsgBox("Le nouveau film a bien ete enregistré",  
MsgBoxStyle.Information, "Travail terminé")  
    Else  
        MsgBox("Erreur lors de l'enregistrement", MsgBoxStyle.Exclamation,  
"travail terminé")  
    End If  
  
End Sub
```

6 Conclusion

Ce chapitre d'introduction à WCF est à présent terminé. Nous avons traité le concept d'application distribuée et d'architecture orientée service. Nous avons ensuite vu les notions fondamentales nécessaires à la compréhension et à la création de services WCF.

Après avoir lu ce chapitre, vous devriez être en mesure de :

- ✓ Comprendre la notion de notion de contrat de service
- ✓ Comprendre la notion de Endpoint
- ✓ Cerner la notion de comportement (behavior)
- ✓ Savoir comment créer, héberger et consommer un petit service de démonstration.

Il est tout a fait normal que certaines notions semblent encore floues. Il faut garder à l'esprit que ce chapitre est un chapitre d'introduction a une technologie très riche en matière d'exposition de service et de distribution de données . Le but premier de ce chapitre est de présenter WCF et ses fondamentaux, avec un premier cas pratique. Chacun des points importants seront repris dans des chapitres dédiés.