Windows Communication Foundation

Intergiciels à objets et services web

Université de Montpellier – Faculté des sciences

Mars 2016

1. Introduction
2. Les contrats
3. Les bindings
4. Hébergement
5. Consommation
6. WCF et remoting
7. Sessions, instantiations et concurrence
8. Sécurité
9. Conclusion

Sommaire

1. Introduction
2. Les contrats
3. Les bindings
4. Hébergement
5. Consommation
6. WCF et remoting
7. Sessions, instantiations et concurrence
8. Sécurité
9. Conclusion

WCF

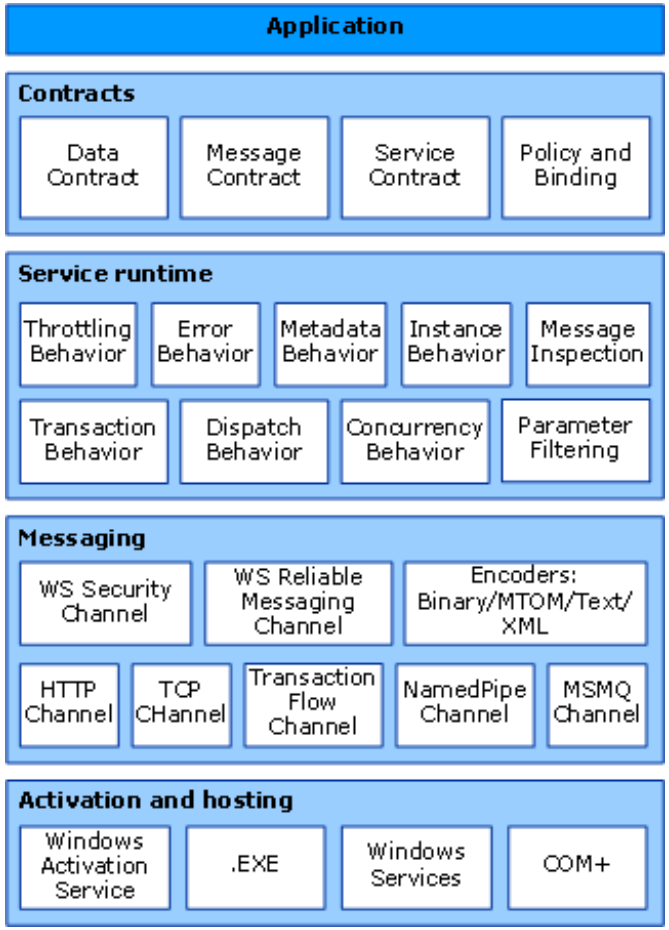
- Runtime + outils + ensemble d'APIs pour créer des systèmes qui envoient des messages entre des services et des clients
- Pour la distribution, l'interopérabilité, et le support direct de l'orienté service
- Unification de pratiques .net existantes (MSMQ, COM+, services web asp.net, WSE, remoting, enterprise services (ou serviced components), ...)

Principes : A, B, C

Ou plutôt C, B, A

- C : contrats de service. Défini via des classes et des interfaces + des attributs (annotations)
- B : bindings. Association au protocole.
- A : Adresse. Adresse à laquelle le service est disponible.

Architecture



- Contracts & Descriptions. Contrats : permet de décrire ce qui sera échangé. Policies & bindings : modalités de communication
- Service Runtime. Comportements du service à l'exé. Throttling : contrôle le nb de messages traités. Error behavior : ce qui se passe qd erreur interne du service. Metadata : quelles métadonnées sont dispos de l'extérieur. Instances : combien d'instances peuvent être utilisées. Transaction : permet le rollback. Dispatch : comment le message est traité par l'infrastructure WCF.
- Messaging. Channel : traite un message de la manière souhaitée. Transport (RW messages sur le réseau, http, tcp, MSMQ) ou Protocole (protocoles de traitement des messages, comme WSS).
- Hosting and Activation. Le service est un programme et peut dc être lancé dans un exe (self-hosted service). Peut aussi être hébergé par un agent externe comme IIS ou WAS.

Développer avec WCF

1. Définir le contrat de service. Définit la signature des services, les données échangées.
2. Implémenter le contrat.
3. Configurer le service : spécification des endpoints.
4. Héberger les services dans une application.
5. Construire une application cliente.

Sommaire

1. Introduction
2. Les contrats
3. Les bindings
4. Hébergement
5. Consommation
6. WCF et remoting
7. Sessions, instantiations et concurrence
8. Sécurité
9. Conclusion

Contrats de service

Un contrat est une promesse sur :

- Le groupements d'opérations dans un service
- La signature des opérations en terme de messages échangés
- Les types de données de ces messages
- La localisation de ces opérations
- Les protocoles et formats de sérialisation utilisés pour communiquer avec le service

Cela devrait vous rappeler quelque chose ...

Contrats, WSDL et XSD

- WSDL et XSD sont bien adaptés pour décrire des services mais sont difficiles à écrire directement et ne sont que des descriptions, pas des contrats à implémenter
- WCF utilise les structures habituelles OO (classes, interfaces, attributs) pour définir la structure d'un service et l'implémenter
- Les contrats WCF peuvent être exportés vers WSDL et XSD

Contrats en WCF

- Contrat $\cong$ opérations + regroupement des opérations
- Opération = méthode (C#, VB.NET, ...) avec attribut `OperationContractAttribute [OperationContract]`
- Contrat de service = Classe ou interface (avec opérations) avec attribut `ServiceContractAttribute [ServiceContract]`

Classes ou interfaces ?

- Plus logique d'utiliser des interfaces
  - Les interfaces de contrats peuvent étendre autant d'autres interfaces de contrats que souhaité
  - Une même classe peut implémenter autant de contrats de services que souhaité
  - On peut modifier l'implémentation d'un contrat de service sans toucher au contrat lui-même
  - Gestion de version : possibilité d'implémenter à la fois le vieux contrat et le nouveau

Paramètres et types de retour

- Attention : pas de passage de référence, que des copies
- Tout type manipulé doit donc être sérialisable

Les contrats de données (`Data Contract`)

- Définition des types de données échangées
- Contrat de données = Classe avec attribut DataContractAttribute `[DataContract]`
- Membre du contrat de données = membre ou propriété avec attribut DataMemberAttribute `[DataMember]`
- Attention, pas de prise en compte de l'accessibilité lors de la sérialisation. Un membre privé sera sérialisé.
- On peut aussi simplement rendre les données sérialisables.

Paramètres et valeurs de retour vers MEP

MEP Message Exchange Pattern

- Request/Reply. Par défaut. Un message de requête qui arrive au service, un message de réponse qui est émis vers le client. Y compris si le type de retour est void (alors le message de retour est vide ou permet de transmettre une erreur).
- One-way. Pas de message de réponse. Possible uniquement si le type de retour est void (sinon : InvalideOperationException). Pas de faute transmise en retour si pb sur le serveur.

```
[OperationContractAttribute(IsOneWay=true)]
void Hello(string greeting);
```

- Duplex. Le service peut invoquer en retour un “service” du client. Nécessite l'introduction d'une interface contenant la déclaration des méthodes appelées sur le client.

Paramètres out et ref

- Rappel out. Permet de déclarer un paramètre de sortie.

```
public void Foo(out int x){
  x=1900;
}
```

- Rappel ref. Permet de passer par référence un paramètre de type simple.

```
public void Echanger(ref int a,ref int b){
  int temp=a;
  a=b;
  b=temp;
}
```

- Leur présence implique un Request/Reply même si type de retour void

Spécification du niveau de protection des messages

- Nécessaire si le binding associé a le mode sécurité des messages activé
- Niveaux de protection d'un message : signé, signé&crypté, ni signature ni encryption
- Peut être spécifié au niveau du service, d'une opération, d'un message de l'opération, d'une partie (part) d'un message.
- Les protections spécifiées à un niveau (scope) deviennent les valeurs par défauts pour tous les éléments de la portée.
- Si rien n'est précisé c'est la configuration du binding qui contrôle la sécurité.

Aparté : activation / désactivation de la sécurité au niveau du binding

System.ServiceModel.SecurityMode

Membre de SecurityMode	Description
Message	Security fournie en utilisant SOAP message security
None	Sécurité désactivée
Transport	Sécurité fournie en utilisant un transport sécurisé (ex HTTPS)
TransportWithMessageCredential	Transport sécurisé + SOAP message security

Exemple

```
[ServiceContract]
public interface IExplicitProtectionLevelSampleService
{
  [OperationContractAttribute]
  public string GetString();

  [OperationContractAttribute(ProtectionLevel=ProtectionLevel.None)]
  public int GetInt();
  [OperationContractAttribute(ProtectionLevel=ProtectionLevel.EncryptAndSign)]
  public int GetGuid();
}
```

Si cet exemple est implémenté par un service dont le binding est WShttpBinding (SecurityMode par défaut : Message) :

- Messages de GetString encryptés et signés
- Messages de GetInt ni encrytés ni signés
- Messages de GetGuid encryptés et signés

Enumération System.Net.Security.ProtectionLevel

Membre	Description
EncryptAndSign	Encrypte et signe les données ; assure confidentialité et intérité des données transmises
Sign	Signe les données ; assure l'intégrité des données transmises
None	Seulement Authentification

Sommaire

1. Introduction
2. Les contrats
3. Les bindings
4. Hébergement
5. Consommation
6. WCF et remoting
7. Sessions, instantiations et concurrence
8. Sécurité
9. Conclusion

Binding

Un binding définit :

- des protocoles : par exemple les mécanismes de sécurité à utiliser
- un encodage : par exemple Texte ou binaire
- le transport : par exemple TCP ou HTTP

Au minimum : le transport.

Des bindings pré-définis

Par exemple :

- BasicHttpBinding : http, peut être utilisé pour connexion à WS suivant WS-I (ex : asmx WS)
- WSHttpBinding : WS-* protocols
- NetMsmqBinding : utilise des files de messages

Les bindings prédéfinis (v3)

Binding	Configuration Element	Description
BasicHttpBinding [http://msdn2.microsoft.com/en-us/library/ms405774(printer).aspx]	basicHttpBinding Element [http://msdn2.microsoft.com/en-us/library/ms731361(printer).aspx]	A binding that is suitable for communicating with WS-Basic Profile conformant Web Services, for example, ASMX-based services. This binding uses HTTP as the transport and Text/XML as the default message encoding.
WSHttpBinding [http://msdn2.microsoft.com/en-us/library/ms586935(printer).aspx]	wsHttpBinding Element [http://msdn2.microsoft.com/en-us/library/ms731299(printer).aspx]	A secure and interoperable binding that is suitable for non-duplex service contracts.
WSDualHttpBinding [http://msdn2.microsoft.com/en-us/library/ms586909(printer).aspx]	wsDualHttpBinding Element [http://msdn2.microsoft.com/en-us/library/ms731821(printer).aspx]	A secure and interoperable binding that is suitable for duplex service contracts or communication through SOAP intermediaries.
WSFederationHttpBinding [http://msdn2.microsoft.com/en-us/library/ms586925(printer).aspx]	wsFederationHttpBinding element [http://msdn2.microsoft.com/en-us/library/ms731374(printer).aspx]	A secure and interoperable binding that supports the WS-Federation protocol, enabling organizations that are in a federation to efficiently authenticate and authorize users.
NetTcpBinding [http://msdn2.microsoft.com/en-us/library/ms576421(printer).aspx]	netTcpBinding Element [http://msdn2.microsoft.com/en-us/library/ms731343(printer).aspx]	A secure and optimized binding suitable for cross-machine communication between WCF applications.
NetNamedPipeBinding [http://msdn2.microsoft.com/en-us/library/ms576406(printer).aspx]	netNamedPipeBinding Element [http://msdn2.microsoft.com/en-us/library/ms731291(printer).aspx]	A secure, reliable, optimized binding that is suitable for on-machine communication between WCF applications.
NetMsmqBinding [http://msdn2.microsoft.com/en-us/library/ms576397(printer).aspx]	netMsmqBinding Element [http://msdn2.microsoft.com/en-us/library/ms731380(printer).aspx]	A queued binding that is suitable for cross-machine communication between WCF applications.
NetPeerTcpBinding [http://msdn2.microsoft.com/en-us/library/ms576415(printer).aspx]	netPeerTcpBinding element [http://msdn2.microsoft.com/en-us/library/ms731318(printer).aspx]	A binding that enables secure, multi-machine communication.
MsmqIntegrationBinding [http://msdn2.microsoft.com/en-us/library/ms600424(printer).aspx]	msmqIntegrationBinding element [http://msdn2.microsoft.com/en-us/library/ms731792(printer).aspx]	A binding that is suitable for cross-machine communication between an WCF application and existing MSMQ applications.

.net v4 : http ://msdn.microsoft.com/en-us/library/ms730879.aspx

Les bindings prédéfinis : caractéristiques (v3)

Binding	Interoperability	Security (Default)	Session (Default)	Transactions	Duplex	Encoding (Default)	Streaming (Default)
BasicHttpBinding	Basic Profile 1.1	(None), Transport, Message, Mixed	(None)	(None)	n/a	Text, (MTOM)	Yes (Buffered)
WSHttpBinding	WS	Transport, (Message), Mixed	(None), Transport, Reliable Session	(None), Yes	n/a	Text, (MTOM)	No
WSDualHttpBinding	WS	(Message), None	(Reliable Session)	(None), Yes	Yes	Text, (MTOM)	No
WSFederationHttpBinding	WS-Federation	(Message), Mixed, None	(None), Reliable Session	(None), Yes	No	Text, (MTOM)	No
NetTcpBinding	.NET	(Transport), Message, None, Mixed	(Transport), Reliable Session,	(None), Yes	Yes	Binary	Yes (Buffered)
NetNamedPipeBinding	.NET	(Transport), None	None, (Transport)	(None), Yes	Yes	Binary	Yes (Buffered)
NetMsmqBinding	.NET	Message, (Transport), Both	(None)	(None), Yes	No		No
NetPeerTcpBinding	Peer	(Transport)	(None)	(None)	Yes		No
MsmqIntegrationBinding	MSMQ	(Transport)	(None)	(None), Yes	n/a		No

Définition des bindings

- Via du code
- Via un fichier de configuration (XML)

Configuration des bindings

```
<configuration>
    <system.serviceModel>
        <bindings>
        </bindings>
        <services>
        </services>
        <behaviors>
        </behaviors>
    </system.serviceModel>
</configuration>
```

Un assistant de création de tels fichiers existe (intégré à VS)

Exemple de binding

```
<configuration>
  <system.serviceModel>
    <services>
      <service name="Service.CalculatorService"
                    behaviorConfiguration="metadataSupport">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:5000/monService" />
          </baseAddresses>
        </host>
        <endpoint address="" binding="basicHttpBinding"
                  contract="Service.ICalculator"/>
        <endpoint address="mex" binding="mexHttpBinding"
                  contract="IMetadataExchange"/>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="metadataSupport">
          <serviceMetadata httpGetEnabled="true" httpGetUrl=""/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Sommaire

1. Introduction
2. Les contrats
3. Les bindings
4. Hébergement
5. Consommation
6. WCF et remoting
7. Sessions, instantiations et concurrence
8. Sécurité
9. Conclusion

Hébergement

- Internet Information Services (IIS) – http uniquement
- Windows Process Activation Service (WAS) (+IIS) – tout protocole
- Windows Service (service autonome géré par le système)
- managed application (au sein d'une appli .net classique)

Self-hosting (dans une appli gérée)

```
using (ServiceHost serviceHost = new ServiceHost(typeof(CalculatorService)))
{
  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();
  }
}
```

Sommaire

1. Introduction
2. Les contrats
3. Les bindings
4. Hébergement
5. Consommation
6. WCF et remoting
7. Sessions, instantiations et concurrence
8. Sécurité
9. Conclusion

Un Client

1. Obtenir des informations sur le contrat de service, les liaisons (bindings), et l'adresse pour un point de terminaison (endpoint) de service.
2. Créer un client WCF à partir de ces informations
3. Appeler les operations souhaitées.
4. (Détruire l'objet du client WCF).

Récupérer un proxy sur le service WCF

- Deux solutions :
  - Le service est en marche et publie son WSDL (publication de métadonnées) $\rightarrow$ le proxy est créé à partir du WSDL par l'outil `svcutil.exe` (mode statique)
  - Vous ne voulez pas passer par le WDSL mais par une dll partagée contenant les (interfaces de) contrats $\rightarrow$ utilisation de ChannelFactory (mode dynamique)
- Dans tous les cas, on récupère un objet local au client qui transmettra les appels au service WCF.

Client utilisant la génération de proxy à partir des métadonnées

Le contrat du service, qui publie des métadonnées

```
// Define a service contract.
[ServiceContract(Namespace="http://Microsoft.ServiceModel.Samples")]
public interface ICalculator
{
    [OperationContract]
    double Add(double n1, double n2);
    // Other methods are not shown here.
}
```

Client utilisant la génération de proxy à partir des métadonnées

Le client généré par `svcutil.exe`

```
public partial class CalculatorClient :
      System.ServiceModel.ClientBase<ICalculator>, ICalculator
{
    public CalculatorClient(){}

    public CalculatorClient(string configurationName) :
            base(configurationName)
    {}

    public CalculatorClient(System.ServiceModel.Binding binding) :
            base(binding)
    {}

    public CalculatorClient(System.ServiceModel.EndpointAddress address,
    System.ServiceModel.Binding binding) :
            base(address, binding)
    {}

    public double Add(double n1, double n2)
    {
        return base.InnerChannel.Add(n1, n2);
    }
}
```

Client utilisant la génération de proxy à partir des métadonnées

Le code d'appel du service chez le client

```
// Create a client object with the given client endpoint configuration.
CalculatorClient calcClient = new CalculatorClient("CalculatorEndpoint"));
// Call the Add service operation.
double value1 = 100.00D;
double value2 = 15.99D;
double result = calcClient.Add(value1, value2);
Console.WriteLine("Add({0},{1}) = {2}", value1, value2, result);
```

Client utilisant un ChannelFactory

```
ICalculator serviceProxy =new ChannelFactory<ICalculator>
    ("ServiceConfiguration").CreateChannel();
// Call the Add service operation.
double value1 = 100.00D;
double value2 = 15.99D;
double result = serviceProxy.Add(value1, value2);
Console.WriteLine("Add({0},{1}) = {2}", value1, value2, result);
```

Sommaire

1. Introduction
2. Les contrats
3. Les bindings
4. Hébergement
5. Consommation
6. WCF et remoting
7. Sessions, instantiations et concurrence
8. Sécurité
9. Conclusion

Intégrer ou migrer ?

- .NET remoting et WCF peuvent cohabiter
- WCF est basé service, pas basé objets distribués ...

Migration : cas simple (1)

Une classe publiée, avec méthodes qui retournent des types simples ou sérialisables

- Rendre les contrats explicites : rajouter des attributs `[ServiceContract]` et `[OperationContract]`
- Créer un fichier de config (avec par exemple, `binding="netTcpBinding"`)
- Modifier la classe en charge de la publication : la transformer en appli hôte

Migration : cas simple (2)

Une classe publiée, avec méthodes qui retournent des types simples ou sérialisables

- Côté client :
  - Un fichier de config avec

```
<system.serviceModel>
  <client>
    <endpoint name=... address=... binding=... contract=.../>
  </client>
<system.serviceModel>
```

  - plus d'Activator.getObject ni de new : passer par un ChannelFactory

```
ChannelFactory<IMonContrat> factory=
  new ChannelFactory<IMonContrat>("leNomDuService");
IMonContrat proxy=factory.CreateChannel();
proxy.Mamethode();
```

Gestion des fautes

- Rajout de `[ServiceBehavior(ReturnUnknownExceptionsAsFaults=true)]` à la classe d'implémentation du service
- Cela permet de transmettre les `CommunicationException` en `FaultException`

Optionnel : Créer des contrats de données

- Pour les classes transmises par sérialisation
- Rajout de `[DataContract]`, `[DataMember]`
- Suppression ou pas de `[Serializable]`

Plus compliqué

- Pas de passage facile de référence distante
- Orientation service plus qu'orientation objets
- http ://msdn2.microsoft.com/en-us/library/aa730857(VS.80).aspx

Sommaire

1. Introduction
2. Les contrats
3. Les bindings
4. Hébergement
5. Consommation
6. WCF et remoting
7. Sessions, instantiations et concurrence
8. Sécurité
9. Conclusion

Sessions, instantiations et concurrence

- Session = corrélation entre tous les messages entre 2 endpoints
- Une session donne un général des garanties de délivrance de messages
- Instantiation : contrôle de la ligne de vie des contextes d'instances (System.ServiceModel.InstanceContext) et des objets de service qui implémentent les opérations.
- Concurrence : contrôle du nombre de threads qui exécutent un InstanceContext à un instant t.

Sessions

- Dans un contrat de service, on peut positionner l'attribut System.ServiceModel.ServiceContractAttribute.SessionMode à System.ServiceModel.SessionMode.Required.
- Dans ce cas, tous les messages sous-jacents aux appels au service doivent faire partie de la même conversation.
- Sessions WCF :
  - Explicitement initiées et terminées par l'appelant
  - Les messages délivrés dans une même session sont traités dans l'ordre dans lequel ils sont reçus
  - Les sessions sont abstraites ; un binding qui implémente une session peut garantir diverses caractéristiques
  - Pas de dépot de données général associé à une session WCF

Instantiation

- Le comportement d'instantiation est défini par la propriété System.ServiceModel.ServiceBehaviorAttribute.InstanceContextMode
- Il contrôle comment les InstanceContext sont créés en réponse aux messages qui arrivent.
- Par défaut chaque InstanceContext est associé à un objet de service (défini par l'utilisateur)
- Donc par défaut, la propriété InstanceContextMode contrôle aussi l'instantiation des objets de service.

Les modes d'instantiation

- PerCall : un nouvel InstanceContext (et donc un nouvel objet de service) est créé à chaque requête du client.
- PerSession : un nouvel InstanceContext (et donc un nouvel objet de service) est créé à chaque nouvelle session du client et est maintenu au long de la session (seulement si binding avec session bien sûr).
- Single : un seul InstanceContext (et donc un seul objet de service) gère toutes les requêtes des clients pendant la durée de l'application

Si ça ne vous rappelle rien, il y a un problème ...

```
[ServiceBehavior(InstanceContextMode=InstanceContextMode.PerSession)]
public class CalculatorService : ICalculatorInstance {
    ...
}
```

Single

- Il est possible de créer un objet de service et de le désigner à l'hébergeur comme l'objet à utiliser
- Dans l'hébergeur : construction du ServiceHost par le constructeur :

```
public ServiceHost (
  Object singletonInstance,
  params Uri[] baseAddresses
)
```

```
CalculatorService service = new CalculatorService();
ServiceHost serviceHost = new ServiceHost(service, baseAddress);
```

Concurrence

- Contrôlé par System.ServiceModel.ServiceBehaviorAttribute.ConcurrencyMode
- 3 modes :
  - Single : chaque instance de contexte est autorisée à avoir au maximum un thread à la fois qui traite les messages. Les autres threads qui veulent utiliser le même contexte d'instance sont bloqués en attendant que le thread courant termine
  - Multiple : Chaque instance de service peut avoir plusieurs threads qui traitent les messages concurremment.
  - Reentrant : Un seul message à la fois mais accepte les opérations ré-entrantes.

Sommaire

1. Introduction
2. Les contrats
3. Les bindings
4. Hébergement
5. Consommation
6. WCF et remoting
7. Sessions, instantiations et concurrence
8. Sécurité
9. Conclusion

Sécurité

- Similaire à WSE
- Authentification par Username / Password
- Certificats X.509, Tickets Kerberos

Sommaire

1. Introduction
2. Les contrats
3. Les bindings
4. Hébergement
5. Consommation
6. WCF et remoting
7. Sessions, instantiations et concurrence
8. Sécurité
9. Conclusion

Conlusion

- Explicitation des services via des constructions OO ($\cong$ WSDL)
- Portage facile des services vers différents protocoles de transport
- Orientation service