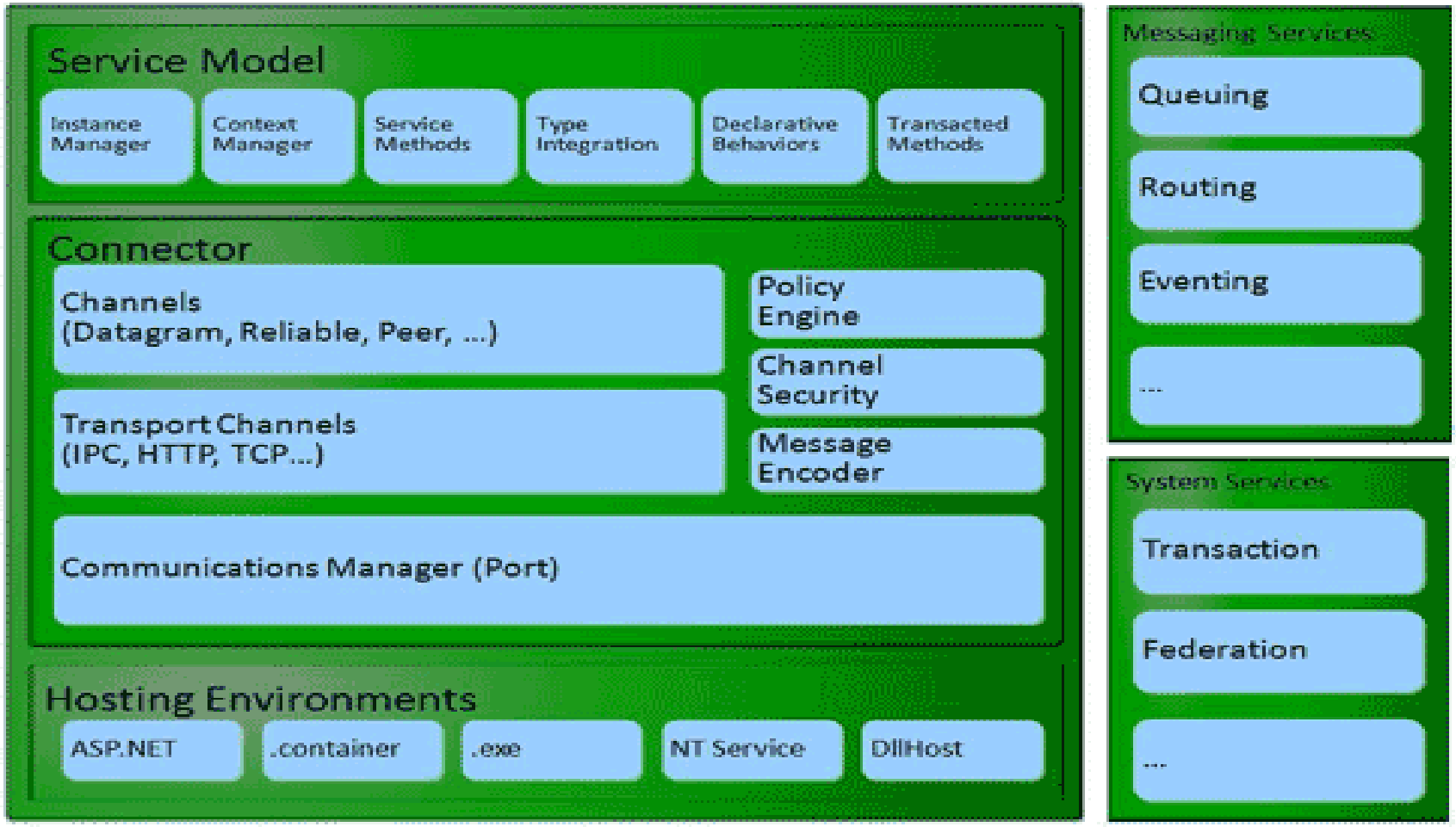
Windows Communication Foundation

1. **Pablo Cibraro, Kurt Claeys, Fabio Cozzolino, Johan Grabner: Profesional WCF 4 Windows Communication Foundation with .NET 4**
2. **John Sharp: Windows Communication Foundation 4, Step by Step**
3. **Juval Lowy: Programming WCF Services**

WCF Architecture



Serviciu ?

- Un serviciu reprezinta unitatea functionala expusa spre a fi utilizata.
- Un serviciu poate fi local sau remote.
- Clientii si serviciile interactioneaza prin trimiterea / primirea de *mesaje*.
- Un serviciu poate fi vazut ca o multime de **endpoints**.
- Un **endpoint** este o resursa pe retea, unde pot fi trimise mesaje.

WCF – mesaje SOAP

FIGURE 1.1 Communication between client and service

WCF

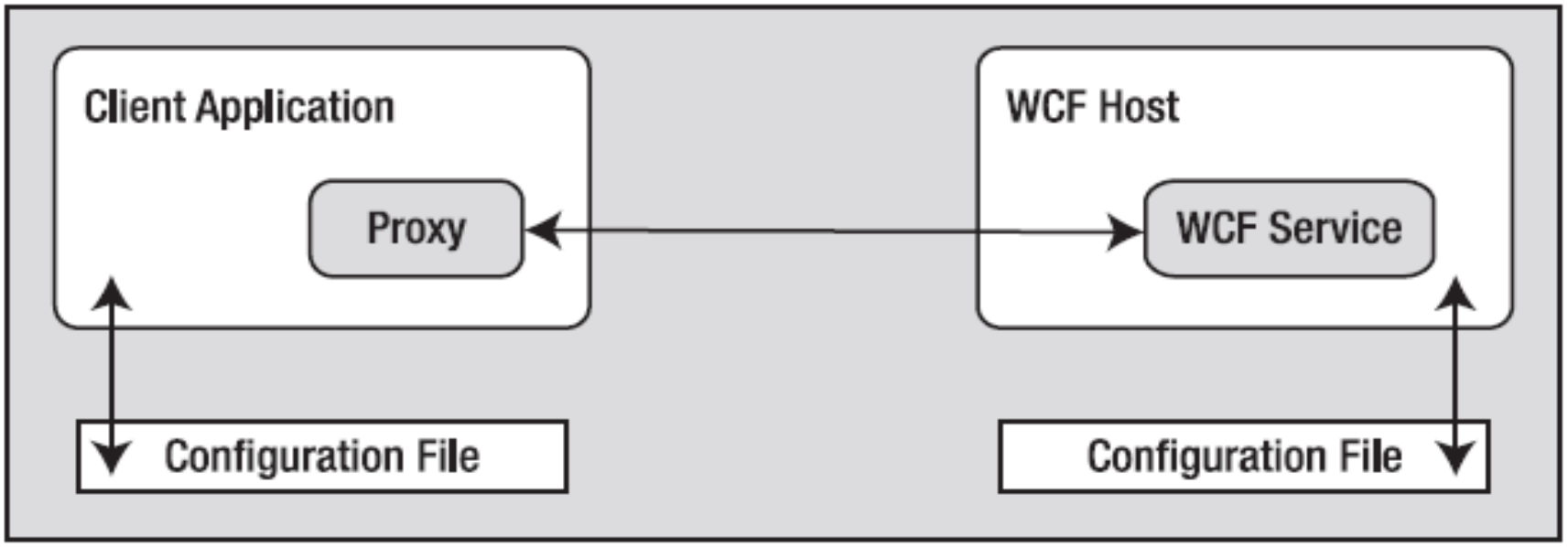


Figure 25-4. *A high-level look at a typical WCF application*

Interfata serviciu

```
using System.ServiceModel;
using WcfHost.DataContract;

namespace WcfHost.Interfaces
{
    [ServiceContract]
    public interface IStockService
    {
        [OperationContract]
        double GetPrice(string _text);

        [OperationContract]
        DataService GetDataService(int id);
    }
}
```

Contract de date

```
using System;
using System.Runtime.Serialization;
namespace WcfHost.DataContract
{
    [DataContract]
    [Serializable]
    public class DataService
    {
        [DataMember(IsRequired = true)]
        public int Id;
    }
}
```

Implementare serviciu

```
using WcfHost.DataContract;
using WcfHost.Interfaces;
namespace WcfHost.Implements
{
   public class StockService : IStockService
   {
      public double GetPrice(string _text)
      {
         return 12.13;
      }

   public DataService GetDataService(int id)
   {
         DataService ds = new DataService();
         ds.Id = id;
         return ds;
   }
  }
}
```

Host serviciu

- App
- Fisiere de configurare

(v pag 8 WCF_Curs)

Client

- Obtinere metadata – svcutil.exe
- Consumare serviciu.

Parti componente serviciu

- A – Address
- B - Binding
- C – Contract
- Endpoint = (A, B, C)

Address

WCF suporta urmatoarele scheme de transport:

- HTTP
- TCP
- Peer network
- IPC (Inter-Process Communication peste pipe-uri cu nume)
- MSMQ

Address

- Adresele au urmatorul format:
- **[base address]/[optional URI]**

unde “**base address”** este totdeauna in formatul:

  - **[transport]://[machine or domain][:optional port]**

Exemplu:

http://localhost:8001/MyService

WCF

Schema transport	Schema URI	Observatie	Port implicit
TCP	net.tcp	Pot fi partajate. net.tcp://localhost:1234/MyService net.tcp://localhost:1234/MyNewService	808
HTTP	http sau https	Pot fi partajate. http://localhost:1234/MyService	80
IPC	net.ipc	Named pipe. Pe aceeasi masina. Nu pot fi partajate. net.ipc://localhost:1234/MyService	

MSMQ	net.msmq	Trebuie specificat numele cozii.
		`net.msmq://localhost/private/MyService`
		`net.msmq://localhost/MyService`	

Peer to Peer	net.p2p		

Binding

- Un ***binding*** contine o multime de alegeri in ceea ce priveste:
  - *protocolul* de transport;
  - *codificarea* mesajelor;
  - pattern-ul de comunicatie;
  - increderea;
  - securitatea;
  - propagarea tranzactiilor si interoperabilitatea.

Binding-uri

- Binding de baza - **BasicHttpBinding**
- Oferit de clasa **BasicHttpBinding**, proiectat pentru a expune un serviciu WCF ca un serviciu web ASMX, deci clientii vechi vor putea folosi acest serviciu.
- Elementul binding: **basicHttpBinding**.

NetTcpBinding

- Binding TCP **- NetTcpBinding**
- Oferit de clasa **NetTcpBinding**. Foloseste comunicatia TCP intre masini pe intranet. Suporta incredere, tranzactii si securitate si este optimizat pentru comunicare WCF –to-WCF. Este necesar ca atat clientul cat si serviciul sa foloseasca WCF.
- Elementul binding: **netTcpBinding**.

Binding-uri …

- **Binding Peer network** Oferit de clasa **NetPeerTcpBinding**. Elementul binding: **netPeerTcpBinding**.
- **Binding IPC** Oferit de clasa **NetNamedPipeBinding**, folosit in comunicarea pe aceeasi masina. Este cel mai sigur. Elementul binding: **netNamedPipeBinding**.
- **Binding Web Service (WS)**
- Oferit de clasa **WSHttpBinding**, foloseste HTTP sau HTTPS pentru transport, si este proiectat sa ofere o varitate de trasaturi: *incredere, tranzactii si securitate in Internet*. Elementul binding : **wsHttpBinding**.
- **Binding Duplex WS** Clasa **WSDualHttpBinding**. Este similar cu binding WS dar in plus suporta comunicarea bidirectionala. Elementul binding: **wsDualHttpBinding**.

Binding

Table 1-1. Transport and encoding for standard bindings (default encoding is in bold)

Name	Transport	Encoding	Interoperable
BasicHttpBinding	HTTP/HTTPS	**Text**, MTOM	Yes
NetTcpBinding	TCP	Binary	No
NetPeerTcpBinding	P2P	Binary	No
NetNamedPipeBinding	IPC	Binary	No
WSHttpBinding	HTTP/HTTPS	**Text,** MTOM	Yes
WSFederationHttpBinding	HTTP/HTTPS	**Text**, MTOM	Yes
WSDualHttpBinding	HTTP	**Text**, MTOM	Yes
NetMsmqBinding	MSMQ	Binary	No
MsmqIntegrationBinding	MSMQ	Binary	Yes

C - Contract

- **Contracte pentru servicii - [ServiceContract]**
- **Contracte de date - [DataContract]**
- **Contracte pentru erori - [FaultContract]**
- **Contracte pentru mesaj - [MessageContract]**

Continuare

- [ServiceContract]
- [OperationContract]

- [DataContract]
- [DataMember]

Host

- Serviciu Windows
- IIS
- App CUI / GUI

Configurare

```
<system.serviceModel>
  <services>
    <service name = "WcfServiceCurs.ServiceTest">
      <endpoint  address  = "http://localhost:8000/MyService/"
          binding  = "wsHttpBinding“
          contract = "WcfServiceCurs.ITestService" />
    </service>
  </services>
</system.serviceModel>
```

(v p 31 wcf_curs)

Metadata Exchange

- Un serviciu are doua optiuni pentru a-si publica metadata :
  - Utilizare protocol HTTP-GET .
  - Utilizare *endpoint* dedicat.

- Interfata IMetadataExchange

HTTP-GET

```
<system.serviceModel>
  <services>
    <service name = "MyService" behaviorConfiguration = "MEXGET">
        <host>
          <baseAddresses>  <add baseAddress = "http://localhost:8000/"/>    </baseAddresses>
       </host>
     </service>
 </services>
<behaviors>
<serviceBehaviors>
    <behavior name = "MEXGET">
      <serviceMetadata httpGetEnabled = "true"/>
    </behavior>
  </serviceBehaviors>
 </behaviors>
</system.serviceModel>
```

MEX – Endpoint dedicat

```
<endpoint  address  = "MEX” binding  = "mexTcpBinding"
   contract = "IMetadataExchange"/>
<endpoint
   address  = "MEX“
   binding  = "mexNamedPipeBinding"
   contract = "IMetadataExchange"   />
<endpoint      address  = "http://localhost:8000/MEX"
   binding  = "mexHttpBinding"
   contract = "IMetadataExchange"   />
```

Clasa ServiceHost

- Clasa **ServiceHost** se foloseste pentru a configura si expune un serviciu.
- Un obiect **ServiceHost** este folosit pentru :
  - Incarcare serviciu.
  - Configurare *endpoints*.
  - Aplicare setari de securitate.
  - Lansare "*listner*" pentru a manipula cererile.

ServiceHost

- Open()
- Close()
- ServiceDescription class
- Add endpoints

Exemplu

```
ServicePost servicePost = new ServicePost();
host = new ServiceHost(servicePost,
                new Uri("http://localhost:8080/httpPost"),
                new Uri("net.tcp://localhost:8081/tcpBlog"));
```

Metadata

- Utilitar svcutil.exe
- Add Service Reference

Tipuri de operatii

- *Operatie cerere-raspuns sincrona*
  - Binding-uri folosite: basicHttpBinding, netTcpBinding
- *Operatie cerere-raspuns asincrona*
  - Bazat pe interfata **IAsyncResult** si metode de tip ***Begin...*** si ***End....***
- *Operatie One-Way : [OperationContract(IsOneWay=true)]*
  - [OperationContract(IsOneWay=true)]
- *Operatie Duplex*
  - **NetNamedPipeBinding** si **NetTcpBinding** suporta comunicarea bidirectionala. Binding-urile care au in nume cuvantul *dual* suporta comunicarea bidirectionala – **wsDualHttpBinding**.

Comunicare bidirectioanala - exemplu

```
interface IMessageCallback
  {
     [OperationContract(IsOneWay = true)]
     void OnMessageAdded(string message, DateTime timestamp);
  }

  [ServiceContract(CallbackContract = typeof(IMessageCallback))]
  public interface IMessage
  {
     [OperationContract]
     void AddMessage(string message);

// cod
  }
```

Setare callback pe partea de client

Clase importante:

- **InstanceContext -** folosita de client.
- **OperationContext -** folosita de serviciu. Metoda importanta aici este **GetCallbackChannel**.

OperationContext

- Clasa **OperationContext** este folosita pe partea de serviciu pentru a accesa referinta callback folosind metoda **GetCallbackChannel**.
- Proxy pentru client va contine o clasa derivata din **DuplexClientBase**

```
public partial class MessageClient :
System.ServiceModel.DuplexClientBase<IMessage>, IMessage
{
   // cod omis
 }
```

Exemplu

- Vezi exemplul de la pagina 61 din curs.
- **Exemplul** complet este compus din trei proiecte.
- Proiect de tip Class Library ce va contine tipurile expuse in server.
  - Interfetele si clasa ce implementeaza aceste interfete
  - Fisierul de configurare
- Proiect de tip Windows Application ce va contine serverul
  - Fisierul de configurare
- Clientul – app CUI
  - Proxy – ce contine clasa MessageClient
  - Fisierul de configurare

Comportare serviciu - Concurenta si instantiere

- Controlarea concurentei in WCF poate fi facuta cu ajutorul urmatoarelor proprietati :
  - InstanceContextMode
  - ConcurrencyMode.

InstanceContextMode

- **InstanceContextMode** specifica modul de creare al instantelor la nivel de server. Valorile posibile sunt :
- **Single** : O instanta a clasei serviciului serveste toate cererile ce vin de la clienti. Acest mod implementeaza un singleton.
- **PerCall** : O instanta a clasei serviciului este creata pentru fiecare cerere.
- **PerSession** : O instanta a clasei serviciului este creata pentru fiecare sesiune a clientului. Cand folosim canale fara sesiune, toate serviciile se comporta ca fiind **PerCall** chiar daca am specificat **PerSession**.

Exemplu

```
[ServiceContract]
interface IStockService
{ // ...}

[ServiceBehavior(InstanceContextMode =
    InstanceContextMode.PerSession)]
public class StockService : IStockService
{
 // implementari metode
}
```

ConcurrencyMode

- **Single**. Numai un fir la un moment dat poate accesa clasa serviciului. Este modul cel mai sigur de executie.
- **Reentrant**. Numai un fir la un moment dat poate accesa clasa serviciului, dar firul poate parasi clasa si poate reveni mai tarziu pentru a-si continua executia.
- **Multiple**. Fire multiple pot accesa clasa serviciului in mod simultan. Clasa trebuie sa fie construita thread-safe.

Exemplu

```
[ServiceContract]
public interface IStockService
{
  //metode
}
[ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Single
    )]
class StockService : IStockService
{
 // cod
}
```

Contracte de date

- DataContractAttribute
- DataMemberAttribute

- Nu facem ierarhia contratului de date.