



Windows Communication Foundation

Johann Oberleitner
Zürich 2007



Overview

- WCF Overview
- Contracts
- EndPoints
- Programming Example



Windows Communication Foundation

- Indigo is Microsoft's new unifying approach for developing and consuming services
 - Web services using SOAP
 - Supporting the WS-* protocols
 - Message Queuing
- In practice this means
 - A Uniform programming model for different distribution technologies



Programming Paradigms

- Contract First
 - start with interfaces first
- WSDL First
 - based on existing WSDL files
 - use WCF tools to create data classes and proxies for the WSDL file
- Code First
 - existing code shall be turned into Web Services



WCF Programming Approaches

- Attribute-based programming
 - Using .NET attributes to mark classes or interfaces (similar to Java @annotations)
- Imperative programming
 - Modify and use the Indigo object model from user code
- Configuration-based programming
 - Behavior is specified in configuration files



Contracts

- Service Contracts
 - Describes the operations a service provides
- Data Contracts
 - Describes data structures supported by service contracts
- Message Contracts

- All contracts: remove coupling of implementation interfaces and service interfaces



WCF Contract Types

- Service Contracts
 - Describes the operations a service provides
- Data Contracts
 - Describes data structures supported by service contracts
- Message Contracts



Service Contracts

- A **service contract** is defined by annotating a .NET interface or a class with the [ServiceContract] attribute
- Each operation included in a service contract requires an **operation contract**
 - specified with the [OperationContract] attribute
- Specifies if a service has request-reply, one-way, duplex behavior
- Specifies Instancing behavior of a service
 - Singleton, per call
- Specifies Session behavior (visibility to clients)
 - Private session, shared session



Service Contract - Sample

```
[ServiceContract]
public interface IGradingService
{
    [OperationContract]
    float GradeStudent(Student,
        string course, int degree);
}
```



Data Contracts

- Describe which **types** are allowed in operation contracts (with [DataContract] attributes)
- Describes which **data elements** of a data structure are transferred in a service request/response operation (with [DataElement] attributes)
- Defines how data types are represented (serialized/deserialized) across the wire
- Implicit and explicit data contracts
 - Implicit are a most built-in types (string,int, ...)
 - Explicit are all other types that shall be used with a service
- Different versions of data contracts supported
 - for service evolution



Data Contracts - Sample

```
[DataContract]
public class Student
{
    [DataMember]
    public string Lastname;
    [DataMember]
    public string Firstname;
    [DataMember]
    public DateTime Birthday;
    [DataMember]
    public string StudentID;
}
```



Message Contract

- Message contracts describe the structure of a message
- Allow precise definitions which information goes into the **message header** and **message body**
 - information contained in the header may be used for message routing



Service Types

- Typed services
 - Parameters and return types can be simple and complex data types
 - Similar to method calls (sample already given above)
- Untyped services
 - Operation contracts use input and output parameters that are of the (WCF) predefined **Message** type
 - All information is put into the message dictionary
 - like a hashtable
- Typed message services
 - User defined messages are used for requests and responses (these message types inherit from the Message class)



Untyped Services - Sample

```
void GradeAllStudents(Message message)
{
    AllStudents item =
        message.GetBody<AllStudents>();
    ...
}
```



Programming Model

- Once contracts are specified
 - Compilation of the interfaces and classes into .NET DLLs – called **assemblies** (similar to Jar files)
- Tool Svcutil.exe
 - Create WSDL and XML Schema files from this contract assembly
 - Create contract classes from WSDL and XML Schema files
 - Create client proxy classes for using services from clients (usually named like ServiceContract+ "Proxy")
 - Create all above via MetadataExchange information via a service running at the service endpoint



Service Interface

```
[ServiceContract]
public interface ISocketService
{
    [OperationContract]
    void Send(string msg);
}
```



Service Code

```
[ServiceBehavior(... some parameters)]
public class SocketServiceImpl:
    ISocketService {

    public void Send(string msg) {
        // do something with msg
    }
}
```



Hosting Options

- Self-Hosting
 - programs hosting services
- Hosting as a Windows Service
 - Windows service <> Web Service
 - Windows service like a driver
- Hosting in IIS
 - IIS6 only HTTP bindings supported
 - IIS7 supports other binding



EndPoint

- Clients and services have to define 1 or more endpoints where a service is hosted or where it can be addressed
- 3 Parameters are required
 - Contract
 - Address
 - Binding



Address

- URL under which service can be reached
- Example
 - <http://localhost:8000/MyGradingService>
 - net.tcp://localhost:9000/MyBestGradingService
- WCF supports also exposing so called MEX endpoints in addition to a service endpoint
 - MEX = Metadata EXchange
 - With this MEX endpoint clients and tools can learn about a service



Bindings / 1

- Define how the communication to an endpoint is accomplished
 - which protocol is used
- WCF predefines 9 standard bindings
 - Each standard binding supports different parameters



Bindings / 2

- BasicProfile
 - Broadest interoperability
 - Compatible with first-generation .NET Web services and next generation Web services
- WSPprofile
 - Supports WS security, WS reliability, and WS transaction
 - Security at transport level (https) and WS security
- NetProfileTcp
 - As WSPprofile but uses a binary encoding



Specifying Endpoints

- Parametrizing endpoints can be done programmatically
- Can also be done via configuration files
 - all information about an endpoint is put into the configuration file



Self-Hosting Example

```
static void Main(string[] args)
{
    Type serviceType = typeof(SocketServiceImpl);
    ServiceHost = new ServiceHost(serviceType);
    // configuration taken from configuration file
    host.Open();
    Console.ReadLine();
    host.Close();
}
```



WCF Configuration File

```
<service name="test1.SocketServiceImpl"
    behaviorConfiguration="test1Behavior">
  <host>
    <baseAddresses>
      <add baseAddress="http://localhost:8080/test1/" />
    </baseAddresses>
  </host>
  <endpoint address="http://localhost:8080/test1/Socket"
    binding="basicHttpBinding"
    contract="SharedInterfaces.ISocketService">
  </endpoint>
</service>
```



Consuming services via WCF

```
static void Main(string[] args)
{
    // generate a ServiceProxy with
    // SvcUtil.exe
    SocketServiceProxy proxy = new
        SocketServiceProxy();
    proxy.Send("xyz");
}
```



Summary

- Unifying Communication Model for applications built with .NET
- Centered on Contracts

- Tools (SvcUtil.exe) can create proxies
- Programming Example