Programming WCF Clients

The ABC's of programming WCF clients

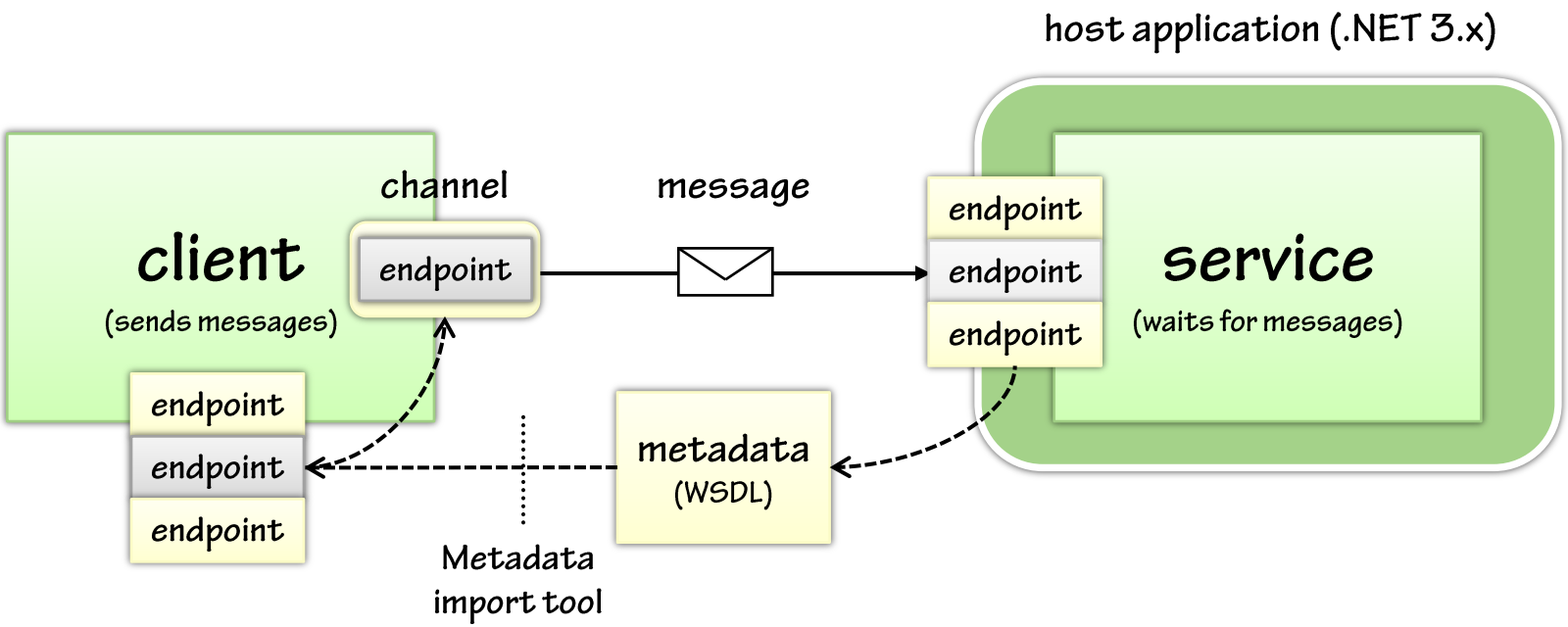
pluralsight
see what you can learn

Overview

- WCF client-side architecture
- Client-side programming experience
- Channel lifetime & exceptions
- Asynchronous invocations
- Sharing contract assemblies
- Programming MEX

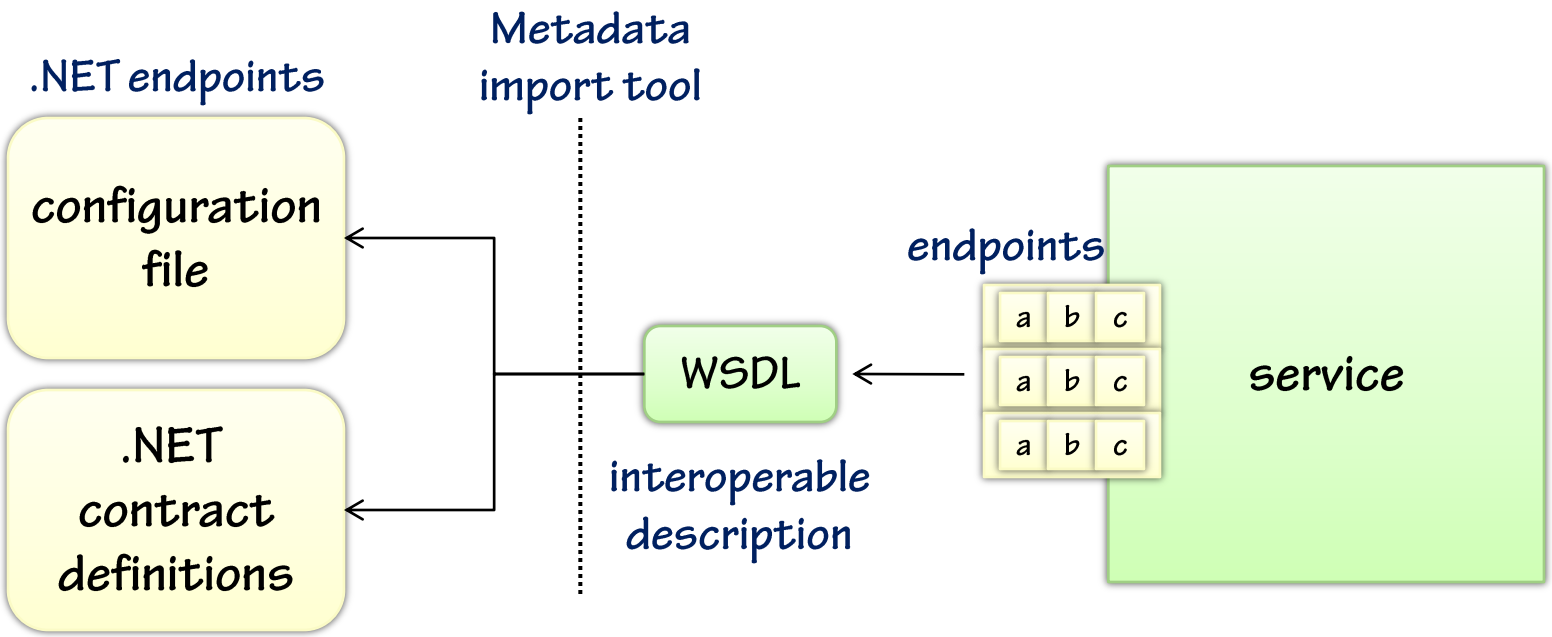
The WCF client architecture

You create channels based on endpoints



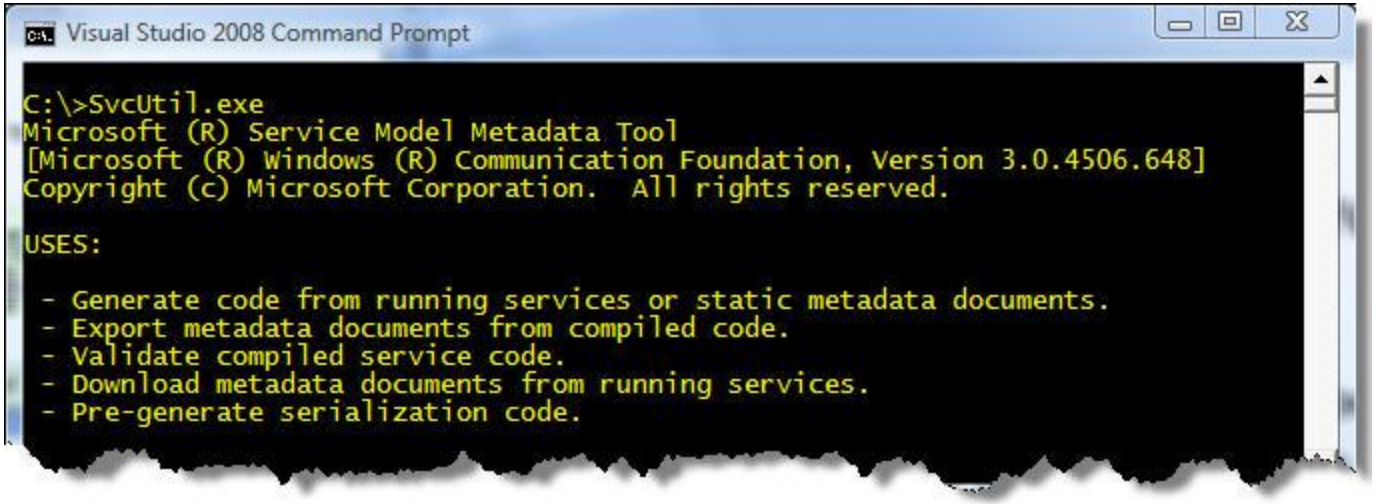
Retrieving endpoint definitions

- **Clients can automatically generate endpoint definitions from WSDL**
  - Generates equivalent contract definitions (code)
  - Generates equivalent endpoint definitions (configuration)



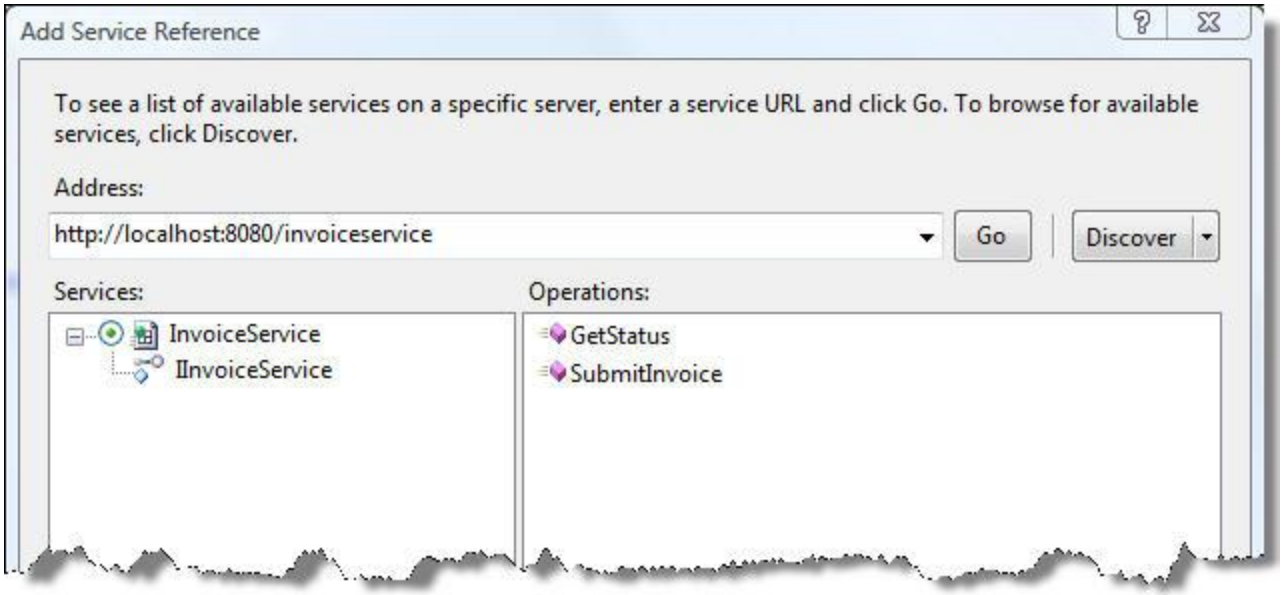
SvcUtil.exe

- **SvcUtil.exe is a metadata import tool for producing WCF client code**
  - Ships with Visual Studio 2008 and the Windows Vista SDK
  - Downloads WSDL and generates WCF code and configuration
  - Provides numerous command-line options (see tool usage)
  - Supports MEX retrieval over HTTP, TCP, pipes



Visual Studio 2008

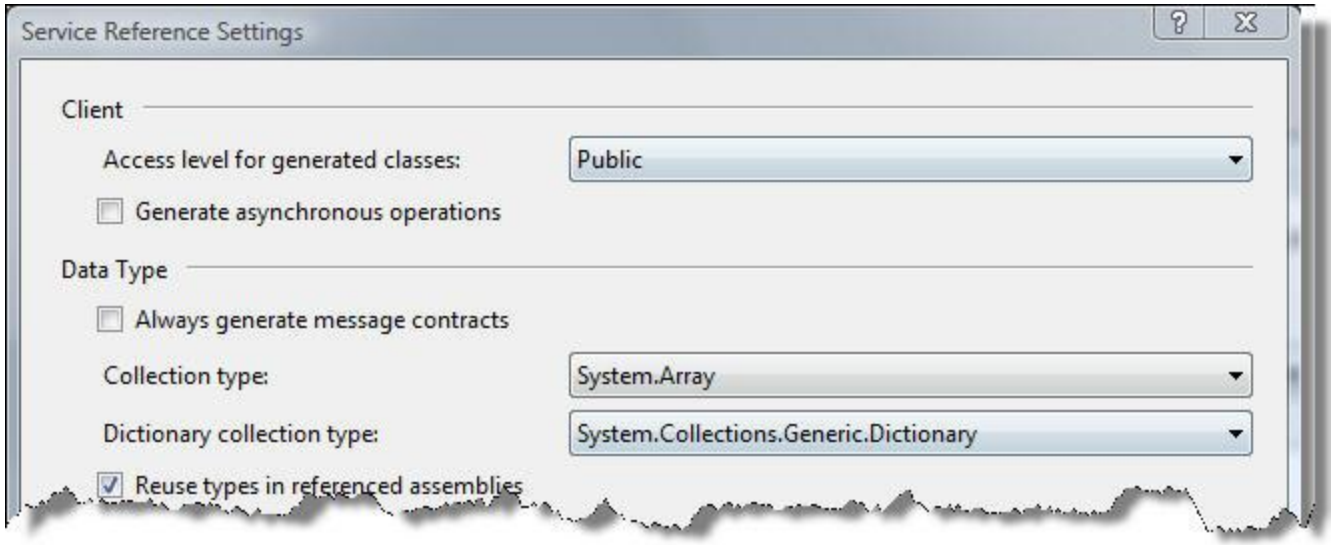
- **Visual Studio 2008 provides the Add Service Reference feature**
  - So you don't have to run SvcUtil.exe directly in most cases
  - Provides most of the same functionality
  - Generates code/configuration and adds it to the project



Service References

- **Service references provide several developer productivity features**
  - You can update ("refresh") a service reference whenever needed
  - You can browse the types within a service reference
  - You can re-configure the code-generation settings as needed

provides access to many of the advanced SvcUtil features



Programming WCF channels

- Programming WCF channels consists of the following steps:

1. Create and configure a ChannelFactory
2. Create a channel
3. Make method calls through channel
4. Close or Abort the channel

Creating a ChannelFactory

- **Use ChannelFactory<T> to create a channel that targets an endpoint**
  - Specify the service contract type for T
  - Supply the target endpoint in the constructor

specify service contract *specify endpoint*

```
ChannelFactory<IInvoiceService> cf =
    new ChannelFactory<IInvoiceService>(endpoint);
...
```

Creates channels that implement IInvoiceService, compatible with specified endpoint

Specifying client-side endpoints

- You can specify client endpoints in either code or configuration

specify endpoint in code

```
ChannelFactory<IInvoiceService> factory =
    new ChannelFactory<IInvoiceService>(
        new BasicHttpBinding(),
        new EndpointAddress("http://server/invoiceservice"));
...
```

specify endpoint in configuration

```
ChannelFactory<IInvoiceService> factory =
    new ChannelFactory<IInvoiceService>("httpEndpoint");
...
```

name of endpoint in config file

Client-side endpoint definitions

WCF client section

```
<configuration>
  <system.serviceModel>
    <client>
      <endpoint name="httpEndpoint"
        address="http://server/invoiceservice"
        binding="basicHttpBinding"
        contract="IInvoiceService"/>
      <endpoint name="tcpEndpoint"
        address="net.tcp://server:8081/invoiceservice"
        binding="netTcpBinding"
        contract="IInvoiceService"/>
    </client>
  </system.serviceModel>
</configuration>
```

refer to this name in code

You can also have binding/behavior configurations

Channel lifecycle

- **Call CreateChannel to create a new channel**
  - Then, simply make method calls (on service contract) to invoke service
  - Call Close or Abort (on IClientChannel) to close the channel

```
ChannelFactory<IInvoiceService> factory =
    new ChannelFactory<IInvoiceService>("tcpEndpoint");

IInvoiceService channel = factory.CreateChannel();

channel.SubmitInvoice(invoice);

((IClientChannel)channel).Close();
```

create channel →

call methods →

gracefully close channel →

Close vs. Abort

- **Close performs a graceful shutdown of the client channel**
  - Waits for in-progress calls to complete before closing
  - Can be a lengthy process (async version exists)
  - Closes underlying networking resources
  - Can throw CommunicationException & TimeoutException
- **Abort tears-down the client channel immediately**
  - Aborts in-progress calls and closes channel immediately
- **You must call Abort on "faulted" channels**
  - Typical after certain types of exceptions

IClientChannel

- **All channels implement IClientChannel for lifecycle management**
  - But you must cast the channel object to IClientChannel to use it
- **Generated code includes an interface derived from IClientChannel**
  - Let's you avoid casting when used with ChannelFactory<T>

```
public interface IInvoiceServiceChannel :
    IInvoiceService, System.ServiceModel.IClientChannel {
}
```

```
ChannelFactory<IInvoiceServiceChannel> factory =
    new ChannelFactory<IInvoiceServiceChannel>("tcpEndpoint");
IInvoiceServiceChannel channel = factory.CreateChannel();
channel.SubmitInvoice(invoice);
channel.Close();
```

casting no longer necessary

Avoiding ChannelFactory<T>

- **The import tools generate a proxy class for each service contract type**
  - Proxy class name = service contract name (minus the "I") + "Client"
- **Proxy class simplifies the client-side programming experience**
  - Implements the service contract type & provides channel lifecycle methods

```
public partial class InvoiceServiceClient :
    ClientBase<IInvoiceService>, IInvoiceService
{
    public InvoiceServiceClient() { }
    public InvoiceServiceClient(string endpointName) :
            base(endpointName)
    {
    }
    ... // you're service contract methods will be here
```

Using the generated proxy class

create proxy →

create invoice →

invoke operation →

gracefully close channel →

```
...
InvoiceServiceClient client =
    new InvoiceServiceClient("httpEndpoint");

Invoice invoice = new Invoice();
invoice.CustomerName = "Acme, Inc";
invoice.Amount = 100.00;
invoice.InvoiceDate = DateTime.Now;

client.SubmitInvoice(invoice);

client.Close();
...
```

Configuring client channels

- **Client channels can be configured via the binding & behaviors**
- **Key things to configure on the binding:**
  - Send timeout value
  - Transport/protocol specific settings
- **Client behaviors are called "endpoint" behaviors in WCF**
  - ClientViaBehavior is one example for auto-routing

Configuring client bindings

```
<configuration>
    <system.serviceModel>
        <client>
            <endpoint name="httpEndpoint"
                address="http://server/invoiceservice"
                binding="basicHttpBinding"
                bindingConfiguration="MyConfiguration"
                contract="InvoiceServiceReference.IInvoiceService" />
        </client>
        <bindings>
            <basicHttpBinding>
                <binding name="MyConfiguration" sendTimeout="00:05:00">
                    <security mode="Transport">
                      <transport clientCredentialType="Basic"/>
                    </security>
                </binding>
            </basicHttpBinding>
        </bindings>
    </system.serviceModel>
</configuration>
```

new binding configuration

increase send timeout limit

configures for basic auth over SSL

Configuring client behaviors

```
<configuration>
    <system.serviceModel>
        <client>
            <endpoint name="httpEndpoint"
                address="http://server/invoiceservice"
                binding="basicHttpBinding"
                behaviorConfiguration="viaBehavior"
                contract="InvoiceServiceReference.IInvoiceService" />
        </client>
        <behaviors>
            <endpointBehaviors>
                <behavior name="viaBehavior">
                    <clientVia viaUri="http://router/invoiceservice"/>
                </behavior>
            </endpointBehaviors>
        </behaviors>
...
```

New endpoint behavior

Handling exceptions

- Client channels need to be prepared for two main exception types

CommunicationException	• Various runtime communication errors • FaultException's thrown by operations
TimeoutException	• Send timeout limit exceeded • Thrown by underlying transport channel

You may need to call "Abort" after certain exceptions (more on this later)

Handling exceptions

invoke operation and call Close for gracefull shutdown

handle service-thrown FaultExceptions

handle other WCF runtime errors

handle timeout errors

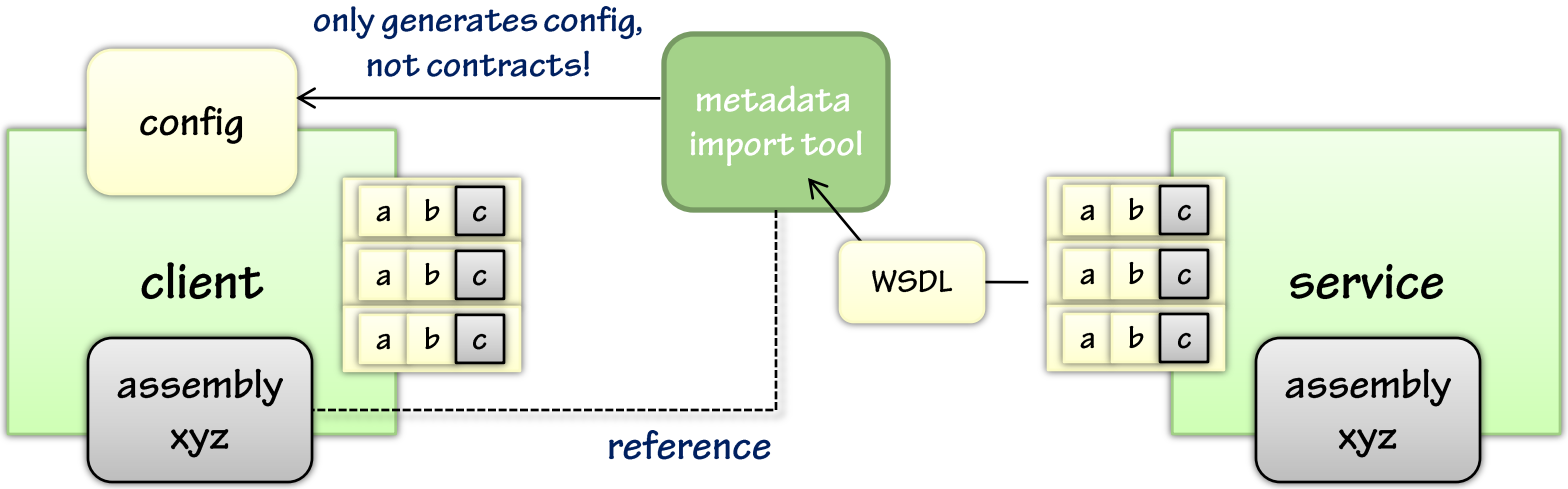
```
InvoiceServiceClient client =
    new InvoiceServiceClient("httpEndpoint");
Invoice invoice = ... // create invoice
try {
    client.SubmitInvoice(invoice);
    client.Close();
}
catch (FaultException fe) {
    Console.WriteLine(fe);
    client.Abort();
}
catch (CommunicationException ce) {
    Console.WriteLine(ce);
    client.Abort();
}
catch (TimeoutException te) {
    Console.WriteLine(te);
    client.Abort();
}
...
```

Invoking services asynchronously

- **WCF also makes it possible to invoke service asynchronously**
  - Use the /async switch when generating the client code
  - Then you'll find Begin/End invocation methods for each operation
- **WCF 3.5 added a simplified asynchronous programming model**
  - Use /async /targetClientVersion:Version35 when generating code
  - This produces an XXXAsync method + an event for each operation
  - Simply hookup event and call the asynchronous method

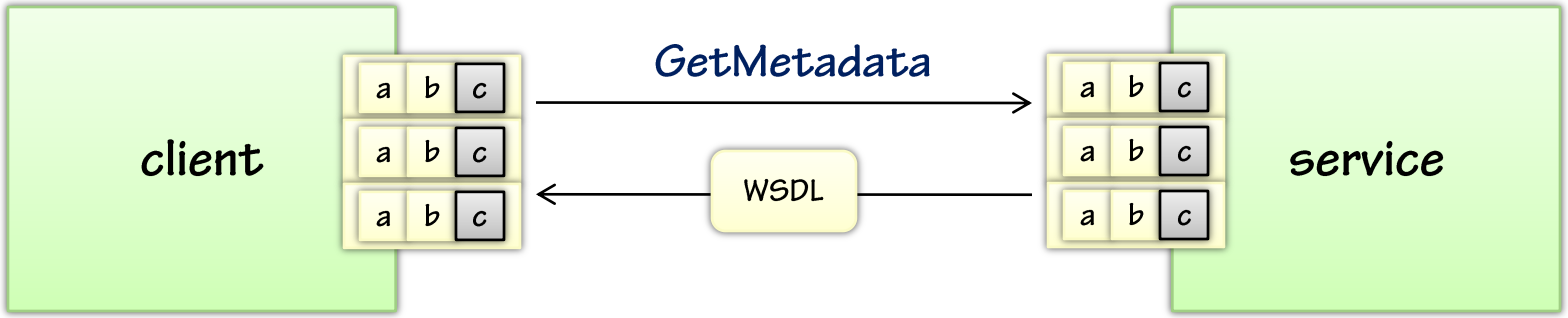
Sharing service contract assemblies

- **The client-side programming model is symmetric to the service model**
  - Allows clients to share contract assemblies when using WCF on both sides
  - Bypassing code-gen preserves constructors, properties, helpers, etc
- **You can tell import tools to reuse types in referenced assemblies**
  - Use /reference (SvcUtil.exe) or the Service Reference Settings



Programming MEX

- **WCF provides official support for MEX in the programming model**
  - Use MetadataResolver and/or MetadataExchangeClient
  - Allows you to programmatically retrieve metadata at runtime
  - Clients can dynamically choose endpoints to use



Summary

- **The WCF client architecture is symmetric with service architecture**
  - Clients must first retrieve endpoint definitions from the service
- **Then clients program against channels following these steps:**
  - Create and configure a ChannelFactory
  - Create a channel
  - Make method calls through channel
  - Close or Abort the channel
- **The client model offers numerous more advanced features**

References

- **The ABC's of Programming WCF, Skonnard**
  - http://msdn.microsoft.com/msdnmag/issues/06/02/WindowsCommunicationFoundation/default.aspx
- **WCF Essentials, Lowy**
  - http://www.code-magazine.com/Article.aspx?quickid=0605051
- **Accessing Services using a WCF Client**
  - http://msdn2.microsoft.com/en-us/library/ms734691.aspx
- **Difference between Close & Abort**
  - http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=855018
- **Pluralsight's WCF Wiki**
  - http://pluralsight.com/wiki/default.aspx/Aaron/WindowsCommunicationFoundationWiki.html