

**Hochschule für Technik und Wirtschaft HTW Chur**

Telekommunikation/Elektrotechnik

# **Verteilte Systeme**

Windows Communication Foundation

**Silvan Weber**

8. Juni 2010

# Windows Communication Foundation

Dieser Bericht wurde im Rahmen des Bachelor-Studienganges Telekommunikation / Elektrotechnik an der Hochschule für Technik und Wirtschaft HTW Chur erstellt.

## **Studierender**

Silvan Weber  
Ackerbühlstr. 8  
7000 Chur

## **Dozent**

Martin Studer, Dipl. Inf.-Ing. ETH  
HTW Chur  
Ringstrasse/Pulvermühlestrasse 57  
7004 Chur

## Inhaltsverzeichnis

1	Einführung .....	4
2	Komponenten .....	5
2.1	Service Endpoint.....	5
2.1.1	Deklaration .....	5
2.2	A wie Address.....	5
2.3	B wie Binding .....	6
2.3.1	Sicherheit .....	7
2.4	C wie Contract .....	7
2.4.1	ServiceContract und OperationContract.....	8
2.4.2	DataContract.....	8
2.5	Client Endpoint .....	9
3	Vergleich mit anderen Technologien.....	9
3.1	RMI .....	9
3.2	CORBA .....	10
3.3	Sockets .....	10
4	Quellenverzeichnis .....	11
5	Anhang 1: Source-Code des Calculators .....	12
5.1	App.config (Server).....	12
5.2	App.config (Client) .....	12
5.3	ServiceHost .....	12
5.4	ServiceClient.....	13
6	Anhang 2: SOAP Meldungen von Calculator .....	15
7	Anhang 3: Unterstützte Protokolle.....	17

# 1 Einführung

Die *Windows Communication Foundation (WCF)* (früherer Codename: *Indigo*) ist eine serviceorientierte Kommunikationsplattform für die Entwicklung und Modellierung von verteilten Anwendungen auf der Basis von Microsoft .NET und seit Version 3.0 fester Bestandteil dessen [Wiki10]. Dazu ein Auszug aus dem .NET Framework Developer Center von Microsoft:

*„Windows Communication Foundation ist ein Satz von .NET-Technologien zum Erstellen und Ausführen vernetzter Systeme. Es handelt sich dabei um eine neue Klasse der Kommunikationsinfrastruktur, die für die Webdienstarchitektur erstellt wurde.“ [MSDN1]*

Die WCF hat eine so genannte serviceorientierte Architektur (SOA) [Wiki10].

Es vereint bzw. ersetzt mehrere bereits bestehende Microsoft-Technologien ([Unif10], Unterkapitel *Unification of Microsoft Distributed Computing Technologies*; [Vid06] ab 04:00):

- ASP.NET Web-Dienste (ASMX)
- .NET Framework-Remoting
- Enterprise Services (DCOM / COM+)
- WSE (Web Services Enhancement)
- Microsoft Message Queuing (MSMQ)

Sinn und Zweck von WCF ist die Vereinfachung bzw. Zusammenfassung all dieser verschiedenen Technologien, so dass es bei Microsoft in Zukunft nur noch WCF gibt ([Vid06] ab 04:00). Die Programmierung von WCF ist in der Tat ziemlich einfach, denn die Implementierung eines kleinen Services erfordert wenig Programmieraufwand (dazu mehr später).

Ein weiterer Vorteil von WCF ist die relativ einfache Konfiguration. Diese kann zentral in einer XML-basierten Datei (*App.config* bzw. *Web.config*) getätigt werden.

Ausserdem gibt es, gegeben durch .NET 3.0 (früher *WinFX*), eine Interoperabilität mit anderen Microsoft-Anwendungen, namentlich Windows Presentation Foundation für Oberflächen mit XAML, Windows CardSpace für das Identitätsmanagement und Windows Workflow Foundation für die Modellierung von Workflows ([.NET3], [Vid06] ab 01:50). Diese drei Technologien lassen sich verknüpfen bzw. koppeln mit WCF ([Vid06] ab 01:10:40). Auf das wird hier aber nicht eingegangen.

Die Web Services von WCF sind in diesem Sinne plattform- und programmiersprachenunabhängig, dass das Übertragungsprotokoll und Format frei wählbar sind. Es gibt z.B. die Möglichkeit, WCF in der Kombination XML mit HTTP (also SOAP) zu verwenden, welche beide standardisiert, also plattform- und programmiersprachenunabhängig, sind.

Programmiert werden kann WCF in allen .NET-Sprachen, namentlich Visual Basic, C#, C++ und teilweise selbst mit F# und der Microsoft-JavaScript-Variante JScript [Impl10].

## 2 Komponenten

### 2.1 Service Endpoint

Ein Service Endpoint ist eine eindeutige Kennzeichnung eines Services auf einem Host. Er besteht aus folgenden Elementen:

- Address (*Wo* befindet sich der Service?)
- Binding (*Wie* werden die Daten übertragen?)
- Contract (*Was* wird übertragen?)

#### 2.1.1 Deklaration

Endpoints können in der Anwendungskonfigurationsdatei (App.config) auf dem Server (Service Endpoint) und dem Client (Client Endpoint) deklariert werden, wie in Abbildung 1 gezeigt.

```
<services>
  <service name="WCFService.Calculate">
    <endpoint
      address="http://localhost:2310/services/Calculator"
      binding="basicHttpBinding"
      contract="WCFService.ICalculate"
    />
  </service>
</services>
```

**Abbildung 1: Service Endpoint Deklaration in XML-Form (File: App.config)**

Dieses Beispiel soll nachfolgend analysiert werden.

### 2.2 A wie Address

Eine WCF-Address entspricht einem normalen URI. Ein Beispiel hierzu wäre: `http://localhost:1234/Service`.

Es kann aber, je nach Binding (s. Kapitel *B wie Binding*), andere Gestalten annehmen, wie z.B.:

- `net.tcp://example.com:7777/TCPService`
- `net.msmq://195.242.14.88:6969/MsgQ`
- `net.pipe://localhost/NamedPipe`

Eine Address entspricht also einfach der Adresse von einem Service auf einem Host. Aber Achtung: Ein Service („exe-File“) kann mehrere Endpunkte mit verschiedenen Eigenschaften (Format & Protokoll) haben.

Wir können nun aus dem in Abbildung 1 gezeigten Endpoint bereits etwas interpretieren. Nämlich er befindet sich auf dem localhost im Verzeichnis `/services/Calculator` und ist über Port 2310 erreichbar. Das Übertragungsprotokoll muss wohl HTTP sein, dieses wird aber erst im Binding definitiv beschrieben.

## 2.3 B wie Binding

Mit Bindings können folgende Funktionen („Mehrwertfeatures“) definiert werden ([Vid06] ab 11:30 & 19:00; [Bind10]):

- Encoding (Beispiele: binär, XML)
- Transport (TCP, HTTP, HTTPS, Named Pipes, Message Queues)
- Kommunikationsmuster / Message Exchange Pattern (Request-Reply, Duplex)
- Security (Authentifizierung, Autorisierung, Verschlüsselung) (optional als Erweiterung)
- Reliability (Zuverlässigkeit) (optional)
- Verteile Transaktionen nach dem ACID-Prinzip (Transaction Flow) (optional)

Es gibt bereits vorgefertigte Bindings wie *basicHttpBinding* (kombiniert HTTP 1.1, XML 1.0 / SOAP 1.1, per default keine Security aktiv) [BHB10], *netTcpBinding* (TCP, binär, Transport Security) [NTB10] (beide: [Vid06] ab 20:20). Die wichtigste Binding ist *wsHttpBinding* (Web Service Binding). Es ist wie *basicHttpBinding*, stellt zusätzlich noch Nachrichtensicherheit, Transaktionen und zuverlässiges Messaging bereit.

Diese Bindings können aber auch noch zusätzlich über App.config konfiguriert werden mit: `bindingConfiguration = "bindingConf"` und unter dem tatsächlichen Binding (z.B. `<basicHttpBinding> <binding name = "bindingConf">...`) die nötigen Konfigurationen angeben.

Der oben definierte Endpoint kann also noch weiter interpretiert werden. Dank dem Attribut `binding` mit dem Wert `basicHttpBinding` wissen wir, dass er mit HTTP übermittelt wird. Wenn wir unter [BHB10] nachschauen, stellen wir fest, dass die Message mit SOAP 1.1 also XML encoded wird.

Streams sind auch möglich dank *CustomBindings*, also benutzerdefinierten Bindings ([Vid06] ab 57:50), wie es eine in Abbildung 2 dargestellt ist.

```
<?xml version="1.0" encoding="utf-8" ?>
<bindings>
  <customBinding>
    <binding name="tcpStreamingBinding">
      <compositeDuplex />
      <binaryMessageEncoding />
      <tcpTransport transferMode="Streamed" maxMessageSize="100000"/>
    </binding>
  </customBinding>
</bindings>
```

**Abbildung 2: CustomBinding für TCP-Streaming (App.config)**

Das Encoding erfolgt in diesem Beispiel binär (`<binaryMessageEncoding />`), TCP wird als Transportprotokoll verwendet (`<tcpTransport />`) und die Übertragung findet in beide Richtungen, also duplex (`<compositeDuplex />`), statt. Dass die Daten gestreamt werden, wird mit dem Attribut `transferMode = "Streamed"` angegeben.

### 2.3.1 Sicherheit

Die Sicherheit ist in jedem System ein wichtiger Faktor, deshalb wird sie hier kurz erläutert.

In WCF ist die gesamte Security konfigurationsbasiert, also deklarativ wiederum im File App.config, erfordert somit keinen Programmieraufwand ([Vid06] ab 01:09:20).

Asymmetrische (z.B. RSA) und symmetrische (z.B. 3DES) Verschlüsselung werden unterstützt ([Alg10], Unterkapitel *Properties*).

Grundsätzlich gibt es in WCF drei *Security Modes*, wie sie in Tabelle 1 beschrieben sind ([BaS10], Unterkapitel *Binding Choices*; [Sec10], Unterkapitel *Security Modes*).

Mode	Beschreibung
Transport	Der Transportweg selbst ist mit SSL/TLS verschlüsselt. Die Funktionen Integrität, Vertraulichkeit und gegenseitige Authentifizierung sind also durch das Transportprotokoll (z.B. HTTPS) gegeben.
Message	Die Nachrichten für sich sind verschlüsselt. Integrität, Vertraulichkeit und gegenseitiger Authentifizierung sind gegeben durch SOAP-Meldungen, die WS-Sicherheit (Web Service Security [WSS02]) verwenden. Auf WS-Sicherheit wird hier nicht weiter eingegangen.
TransportWithMessageCredential	Eine Mischung der beiden. Es wird die Transportsicherheit zur Gewährleistung von Integrität, Vertraulichkeit sowie für die Server-Authentifizierung verwendet. Für die Client-Authentifizierung wird die WS-Sicherheit eingesetzt.

**Tabelle 1: Security Modes**

Eine Kombination von Transport und Message ist auch möglich, ist aber nur für das Message Queueing (MSMQ) zulässig ([Sec10], Unterkapitel *Security Modes*).

Der Mode wird mit der Variable `<security mode = "{None | Message | Transport | TransportWithMessageCredential}">` in der Binding definiert. Gleiches gilt übrigens auch für die **Zuverlässigkeit**. Dann lautet das Attribut: `<reliableSession enabled = "true" />`. Wie besprochen funktioniert es nur bei *wsHttpBinding*.

## 2.4 C wie Contract

Im Contract wird vereinbart, welche Daten (samt Datentypen & Methoden) zum Server und zurück zum Client übertragen werden. Der Client und der Server müssen dieselben Datentypen bei derselben Operation austauschen. Der Contract ist ein Interface in der jeweiligen Entwicklungsprogrammiersprache und vergleichbar mit Stubs auf dem Client und Skeletons auf dem Server, bekannt aus anderen verteilten Systemen.

Es ist im Übrigen nicht unbedingt nötig, dass der Client in seinem Interface *alle* Operationen des Servers deklariert hat, wenn also „zu wenig“ Operationen im Client definiert sind. Umgekehrt dürfen aber nicht „zu viele“ Operationen im Client definiert bzw. verwendet sein.

In der Referenz bzw. im Verweis System.ServiceModel der .NET-Klassenbibliothek liegen etwa 90% der WCF Funktionalitäten ([Vid06], ab 36:05).

### 2.4.1 ServiceContract und OperationContract

Ein ServiceContract (*der* Contract) besteht aus einem oder mehreren OperationContracts (angebotene Operationen), welche wiederum aus DataContracts (Datenstruktur für in und out) bestehen können ([Vid06] ab 18:10). In Abbildung 3 sind der ServiceContract und zwei der OperationContracts eines kleinen Kalkulators abgebildet (vollständiger Code s. Anhang).

```
[ServiceContract]
public interface ICalculate
{
    [OperationContract]
    double Add(double a, double b);

    [OperationContract]
    double Subtract(double a, double b);
}
```

**Abbildung 3: Code (C#) für ServiceContract mit OperationContracts (Calculator)**

Wie man sieht, muss immer explizit angegeben werden, was man veröffentlichen will (z.B. mit dem Attribut `[OperationContract]`) ([Vid06] ab 28:55).

### 2.4.2 DataContract

DataContracts werden dann gebraucht, wenn komplexere Datentypen übermittelt also serialisiert werden müssen. Für die einfachen Datentypen wie int, boolean, string etc. braucht es keine separaten DataContracts bzw. diese sind bereits mit Standarddatenverträgen ausgestattet. Das Serialisieren übernimmt die Klasse DataContractSerializer aus dem Namespace System.Runtime.Serialization [Dat10]. In Abbildung 4 ist die Standard-DataContract des Visual Studio-Templates *WCF-Dienstanwendung* dargestellt.

```
[DataContract]
public class CompositeType
{
    bool boolValue = true;
    string stringValue = "Hello ";

    [DataMember]
    public bool BoolValue
    {
        get { return boolValue; }
        set { boolValue = value; }
    }

    [DataMember]
    public string StringValue
    {
        get { return stringValue; }
        set { stringValue = value; }
    }
}
```

**Abbildung 4: Default-DataContract in Visual Studio 2008**

Der Kommentar dazu ist:

*„Verwenden Sie einen Datenvertrag, wie im folgenden Beispiel dargestellt, um Dienstvorgängen zusammengesetzte Typen hinzuzufügen.“*

Die Zusammenfassung für die Klasse DataContract lautet:

*„Gibt an, dass der Typ einen Datenvertrag definiert oder implementiert und mit einem Serialisierer wie dem System.Runtime.Serialization.DataContractSerializer serialisierbar ist. Um ihren Typ serialisierbar zu machen, müssen die Autoren hierfür einen Datenvertrag definieren.“*

## 2.5 Client Endpoint

Im Client muss dasselbe Interface und derselbe Endpoint deklariert werden wie im Service (Abbildung 3). Auch hier bitte wieder auf die richtigen Angaben für A, B und C im App.config (Abbildung 1) achten.

## 3 Vergleich mit anderen Technologien

Grundsätzlich kann man bei all den nachfolgenden verteilten Systemen sagen, dass sie Client-Server-Technologien sind. Es wird also ein Server benötigt, bevor Clients sinnvoll ausführbar sind. Im Unterricht wurden diese ausführlich behandelt, deshalb wird hier ein Vergleich aufgeführt.

### 3.1 RMI

RMI ist seit Version 1.5 in Java integriert, ebenso ist WCF seit Version 3.0 in Microsoft .NET integriert. Beide können so einfach ohne zusätzliche Software in der jeweiligen Programmiersprache implementiert werden.

WCF erlaubt ebenfalls eine Zugriffstransparenz wie RMI.

RMI kann nur in Java implementiert werden, wobei WCF in mehreren Sprachen programmiert werden kann.

Die Meldungen können bei WCF (fast) beliebig encoded werden, wobei Java-RMI nur Byte-Streams unterstützt.

WCF hat keinen Registry-Server wie RMI.

RMI übernimmt den Gedanken der Remote Object Reference (verteiltes Objektmodell) im Sinne der Objektorientierung, wobei WCF einen serviceorientierten Ansatz à la Webservice verfolgt ([Vid06] ab 07:15).

## 3.2 CORBA

Auch in CORBA hat man analog zu den Contracts gemeinsame Interfaces (dort mit der Schnittstellenbeschreibungssprache IDL) definiert.

CORBA ist komplett programmiersprachenunabhängig und WCF kann nur in den .NET-Sprachen implementiert werden.

Ausserdem verlangt CORBA wie RMI eine Art Registry-Server, den so genannten Naming-Service, wovon WCF das nicht benötigt.

Das Übertragungsprotokoll von CORBA ist IIOIP (Internet Inter-ORB Protocol; [IIOIP09]), wobei WCF mehrere Protokollmöglichkeiten des entfernten Methodenaufrufs bietet.

## 3.3 Sockets

Bei Sockets kann die Kommunikation nur direkt über UDP oder TCP erfolgen.

Das Encoding bei Datagram und Transport Sockets erfolgt in Byte-Arrays, also binär.

## 4 Quellenverzeichnis

- [.NET3] .NET Framework 3.0, Wikipedia, die freie Enzyklopädie [[http://de.wikipedia.org/w/index.php?title=.NET&oldid=74045094#.NET\\_Framework\\_3.0](http://de.wikipedia.org/w/index.php?title=.NET&oldid=74045094#.NET_Framework_3.0)], 6. Mai 2010 (Zuletzt besucht: 24.05.10)
- [Alg10] SecurityAlgorithmSuite Member, Microsoft Corporation [[http://msdn.microsoft.com/en-us/library/system.servicemodel.security.securityalgorithmsuite\\_members.aspx](http://msdn.microsoft.com/en-us/library/system.servicemodel.security.securityalgorithmsuite_members.aspx)], 2010 (Zuletzt besucht: 30.05.10)
- [BaS10] Bindings and Security, Microsoft Corporation [<http://msdn.microsoft.com/en-us/library/ms731172.aspx>], 2010 (Zuletzt besucht: 27.05.10)
- [BHB10] BasicHttpBinding Class, Microsoft Corporation [<http://msdn.microsoft.com/en-us/library/system.servicemodel.basichttpbinding.aspx>], 2010 (Zuletzt besucht: 24.05.10)
- [Bind10] Binding Class, Microsoft Corporation [<http://msdn.microsoft.com/en-us/library/system.servicemodel.channels.binding.aspx>], 2010 (Zuletzt besucht: 24.05.10)
- [Dat10] Using Data Contracts, Microsoft Corporation [<http://msdn.microsoft.com/en-us/library/ms733127.aspx>], 2010 (Zuletzt besucht 30.05.10)
- [IIOp09] IIOp, Wikipedia, die freie Enzyklopädie [<http://de.wikipedia.org/w/index.php?title=IIOp&oldid=60315260>], 22.05.09 (Zuletzt besucht 29.05.10)
- [Impl10] Implementing Service Contracts, Microsoft Corporation [<http://msdn.microsoft.com/en-us/library/ms733764.aspx>], 2010 (Zuletzt besucht: 24.05.10)
- [MSDN1] .NET Framework Developer Center - Windows Communication Foundation (WCF) | MSDN Online. Microsoft Corporation [<http://msdn.microsoft.com/de-de/netframework/aa663324.aspx>], 2010. (Zuletzt besucht: 27.03.10).
- [NTB10] NetTcpBinding Class, Microsoft Corporation [<http://msdn.microsoft.com/en-us/library/system.servicemodel.nettcpbinding.aspx>], 2010 (Zuletzt besucht: 24.05.10)
- [Sec10] Distributed Application Security, Microsoft Corporation [<http://msdn.microsoft.com/en-us/library/ms731204.aspx>], 2010 (Zuletzt besucht: 27.05.10)
- [Tut07] WCF Tutorial. Michael Voggenreiter. [<http://www.michis-blog.de/wp-content/uploads/2007/05/wcf-tutorial.pdf>], 2007 (Zuletzt besucht: 16.04.2010)
- [Unif10] What Is Windows Communication Foundation?, Microsoft Corporation [[http://msdn.microsoft.com/en-us/library/ms731082\(v=VS.90\).aspx](http://msdn.microsoft.com/en-us/library/ms731082(v=VS.90).aspx)], 2010 (Zuletzt besucht: 12.05.10)
- [Vid06] Windows Communication Foundation mit Experte: Christian Weyer. dotnetpro.tv. [<http://www.dotnetpro.de/community/downloads/dnptv16cd.wmv>], 2006 (Zuletzt besucht: 12.05.10)
- [WHB10] <wsHttpBinding>, Microsoft Corporation [<http://msdn.microsoft.com/en-us/library/ms731299.aspx>], 2010 (Zuletzt besucht: 27.05.10)
- [Wiki10] Windows Communication Foundation, Wikipedia, die freie Enzyklopädie [[http://de.wikipedia.org/w/index.php?title=Windows\\_Communication\\_Foundation&oldid=69349463](http://de.wikipedia.org/w/index.php?title=Windows_Communication_Foundation&oldid=69349463)], 2010 (Zuletzt besucht: 12.05.2010)
- [WSS02] Web Services Security (WS-Security), Microsoft Corporation [<http://msdn.microsoft.com/en-us/library/ms951257.aspx>], 05.04.02 (Zuletzt besucht: 27.05.10)

## 5 Anhang 1: Source-Code des Calculators

Der nachfolgende Quellcode wurde unter Microsoft Visual Studio 2008 mit dem .NET Framework 3.5 erstellt.

Die lauffähigen Programme sind in C# geschrieben.

### 5.1 App.config (Server)

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="WCFService.Calculate">
        <endpoint name="Calc"
          address="http://localhost:2310/services/Calculator"
          binding="basicHttpBinding"
          contract="WCFService.ICalculate"
        />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

### 5.2 App.config (Client)

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <client>
      <endpoint name="Calc"
        address="http://localhost:2310/services/Calculator"
        binding="basicHttpBinding"
        contract="WCFClient.Program+ICalculate"
      />
    </client>
  </system.serviceModel>
</configuration>
```

### 5.3 ServiceHost

```
using System;
using System.ServiceModel;

namespace WCFService
{
  class Program
  {
    static void Main(string[] args)
    {
      ServiceHost calcService = new ServiceHost(typeof(Calculate));
      calcService.Open();
      Console.WriteLine("CalculatorService is up and running...");
      Console.WriteLine("Press return to close...");
      Console.ReadLine();
      calcService.Close();
    }
  }
}
```

```
[ServiceContract]
public interface ICalculate
{
    [OperationContract]
    double Add(double a, double b);

    [OperationContract]
    double Subtract(double a, double b);

    [OperationContract]
    double Multiply(double a, double b);

    [OperationContract]
    double Divide(double a, double b);
}

public class Calculate : ICalculate
{
    public double Add(double a, double b)
    {
        return a + b;
    }
    public double Subtract(double a, double b)
    {
        return a - b;
    }
    public double Multiply(double a, double b)
    {
        return a * b;
    }
    public double Divide(double a, double b)
    {
        return a / b;
    }
}
```

Ergibt:

**CalculatorService is up and running...**  
**Press return to close...**

-

## 5.4 ServiceClient

```
using System;
using System.ServiceModel;

namespace WCFClient
{
    class Program
    {
        [ServiceContract]
        public interface ICalculate
        {
            [OperationContract]
            double Add(double a, double b);

            [OperationContract]
            double Subtract(double a, double b);

            [OperationContract]
            double Multiply(double a, double b);

            [OperationContract]
            double Divide(double a, double b);
        }
    }
}
```

```
static void Main(string[] args)
{
    ChannelFactory<ICalculate> factory = new ChannelFactory<ICalculate>("Calc");
    ICalculate channel = factory.CreateChannel();
    double a = 3;
    double b = 4;
    double result = channel.Add(a, b);
    Console.WriteLine("Result Add:\n" + a + " + " + b + " = " + result);

    a = 73;
    b = 4;
    result = channel.Subtract(73, 4);
    Console.WriteLine("\nResult Subtract:\n" + a + " - " + b + " = " + result);

    a = Math.PI;
    b = Math.Sin(0.456);
    result = channel.Multiply(a, b);
    Console.WriteLine("\nResult Multiply:\n" + a + " * " + b + " = " + result);

    a = Math.Log10(2.15);
    b = Math.Cos(0.456);
    result = channel.Divide(a, b);
    Console.WriteLine("\nResult Divide:\n" + a + " / " + b + " = " + result);

    Console.WriteLine("Press return to exit...");
    Console.ReadLine();
}
}
```

Ergibt:

**Result Add:**  
**3 + 4 = 7**

**Result Subtract:**  
**73 - 4 = 69**

**Result Multiply:**  
**3.14159265358979 \* 0.440360354953183 = 1.38343285605311**

**Result Divide:**  
**0.332438459915605 / 0.897821116807522 = 0.370272489354775**  
**Press return to exit...**

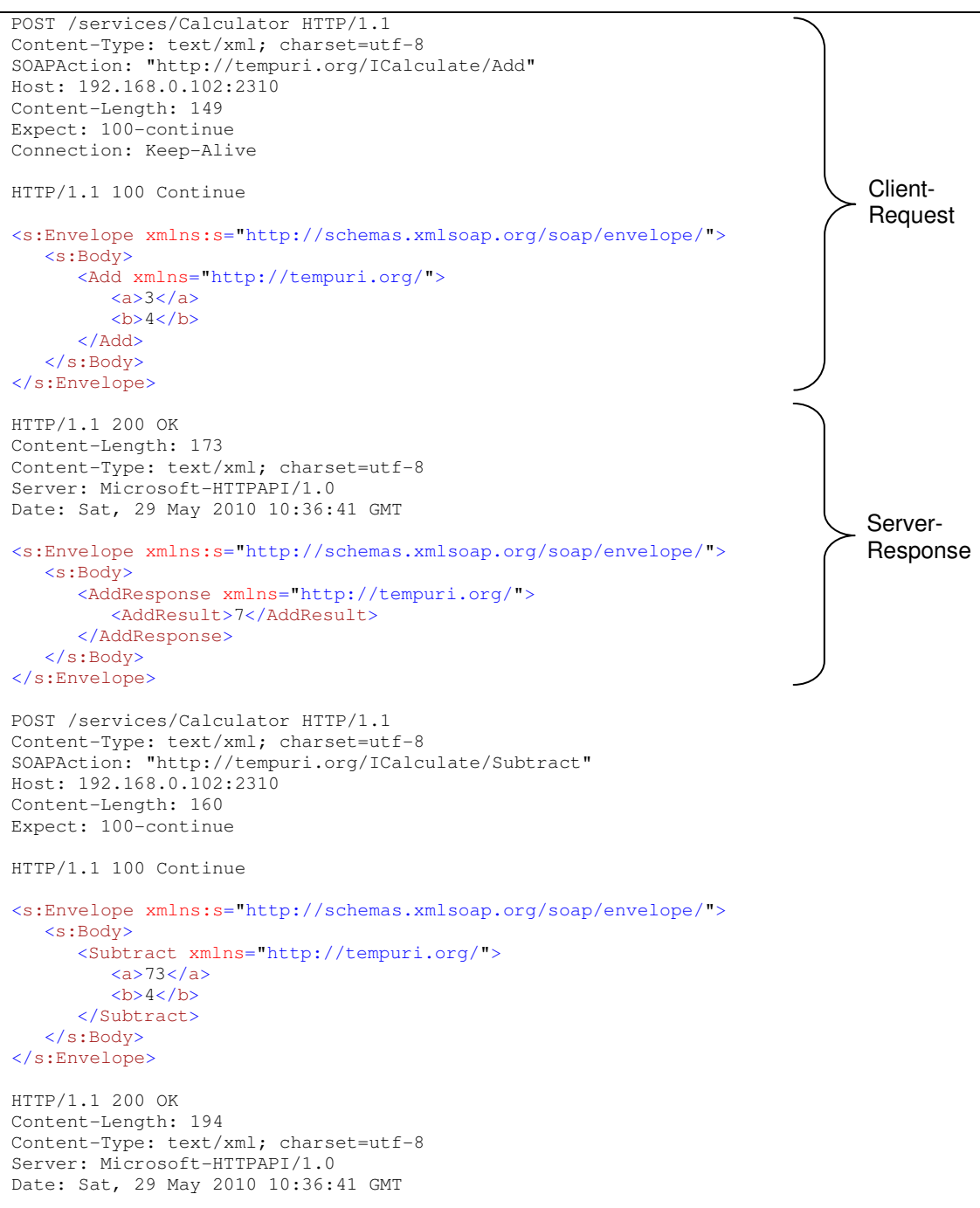
=

## 6 Anhang 2: SOAP Meldungen von Calculator

Folgende Abbildung zeigt die SOAP-Meldungen zwischen Server und Client anhand des behandelten Quellcodes mit *basicHttpBinding*.

Die Meldungen wurden mit Wireshark aufgezeichnet. Um diese Darstellung zu erhalten, benutzt man den Wireshark-Befehl *Follow TCP Stream*. Formatiert und eingefärbt wurden die Meldungen dann mit Visual Studio; zu den Werten in der SOAP Meldung (z.B. a, b, Reponse), vgl. Source-Code.

Hier ist auch die Plattformunabhängigkeit zu erkennen, denn im Payload der Meldungen werden keine herstellerspezifischen Protokolle verwendet.



```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>
    <SubtractResponse xmlns="http://tempuri.org/">
      <SubtractResult>69</SubtractResult>
    </SubtractResponse>
  </s:Body>
</s:Envelope>
```

POST /services/Calculator HTTP/1.1  
Content-Type: text/xml; charset=utf-8  
SOAPAction: "http://tempuri.org/ICalculate/Multiply"  
Host: 192.168.0.102:2310  
Content-Length: 194  
Expect: 100-continue

HTTP/1.1 100 Continue

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>
    <Multiply xmlns="http://tempuri.org/">
      <a>3.1415926535897931</a>
      <b>0.44036035495318304</b>
    </Multiply>
  </s:Body>
</s:Envelope>
```

HTTP/1.1 200 OK  
Content-Length: 210  
Content-Type: text/xml; charset=utf-8  
Server: Microsoft-HTTPAPI/1.0  
Date: Sat, 29 May 2010 10:36:41 GMT

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>
    <MultiplyResponse xmlns="http://tempuri.org/">
      <MultiplyResult>1.3834328560531135</MultiplyResult>
    </MultiplyResponse>
  </s:Body>
</s:Envelope>
```

POST /services/Calculator HTTP/1.1  
Content-Type: text/xml; charset=utf-8  
SOAPAction: "http://tempuri.org/ICalculate/Divide"  
Host: 192.168.0.102:2310  
Content-Length: 190  
Expect: 100-continue

HTTP/1.1 100 Continue

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>
    <Divide xmlns="http://tempuri.org/">
      <a>0.33243845991560533</a>
      <b>0.8978211168075223</b>
    </Divide>
  </s:Body>
</s:Envelope>
```

HTTP/1.1 200 OK  
Content-Length: 202  
Content-Type: text/xml; charset=utf-8  
Server: Microsoft-HTTPAPI/1.0  
Date: Sat, 29 May 2010 10:36:41 GMT

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Body>
    <DivideResponse xmlns="http://tempuri.org/">
      <DivideResult>0.3702724893547748</DivideResult>
    </DivideResponse>
  </s:Body>
</s:Envelope>
```

## 7 Anhang 3: Unterstützte Protokolle

- ATOM
- COM+
- HTTP
- MSMQ
- Named Pipes
- SOAP
- UDP
- XML
- AJAX
- DCOM
- JSON
- MTOM
- REST
- TCP
- WSDL