Debreceni Egyetem
Informatikai Kar
Információ Technológia tanszék

XML alapú szolgáltatások

Témavezető:
Dr. Adamkó Attila
egyetemi adjunktus

Készítette:
Pogány Tamás
programtervező informatikus

Debrecen
2010

Tartalomjegyzék

1. Mi a webszolgáltatás?

A webszolgáltatás alkalmazások közötti adatcserére szolgáló protokollok és szabványok gyűjteménye. Különböző programnyelveken megírt alkalmazások kommunikálnak egymással valamilyen hálózaton (leggyakrabban Internet, vagy akár Intranet) keresztül, melyek akár különböző platformokon is futhatnak. A webszolgáltatások nyílt szabványokat használnak, a fejlesztésért főként a W3C[1] felel.

A webszolgáltatás kliens-szerver alapú. A szerver az általa publikált szolgáltatást (ami nem más, mint meghívható metódusok összessége) nyújtja a külvilág felé, melyet a kliensek valamilyen formában meghívnak, majd feldolgozzák a szerver válaszát.

Minden kommunikáció XML[2] segítségével történik, így biztosítható a platformok közötti átjárhatóság, illetve szükség esetén egyszerűsödik a hibakeresés is. Az XML az esetek többségében SOAP[3] vagy XML-RPC[4] üzenet.

A hálózaton történő továbbítás lehetséges HTTP[5], FTP[6], SMTP[7], vagy XMPP[8] protokollok segítségével. Azonban az alkalmazások túlnyomó többsége a HTTP kapcsolatot használja, egész egyszerűen azért, mert így biztosítható, hogy a szolgáltatás tűzfalakon keresztül is elérhető legyen.

A szerver által nyújtott szolgáltatásokat egy speciális XML fájl, az úgynevezett WSDL (Web Service Definition Language) írja le. A WSDL fájlt (és ezáltal a szolgáltatást magát) az UDDI (Universal Description, Discovery, and Integration) protokoll szerint propagálják a külvilág felé, a biztonságról pedig a WS-Security protokoll szerint gondoskodnak.

1 World Wide Web Consortium

2 Extensible Markup Language

3 Simple Object Access Protocol

4 XML Remote Procedure Call

5 HyperText Transfer Protocol

6 File Transer Protocol

7 Simple Mail Transfer Protocol

8 Extensible Messaging and Presence Protocol

1.1. A webszolgáltatás előnyei

- A szolgáltatás egésze szempontjából teljesen lényegtelen, hogy a kliens és a szerver milyen nyelven íródtak, milyen platformon futnak, vagy éppen fizikailag hol vannak
- A felhasznált szabványok nyíltak
- A HTTP használatával biztosítható a tűzfalakon keresztüljutás
- Egy-egy szolgáltatást több, különböző feladatot ellátó kliens is használhat

1.2. A webszolgáltatás hátrányai

- Figyelni kell a biztonságra, a HTTP miatt főleg az adatbiztonságra
- A hálózati továbbítás miatt sok esetben lassú, a kliensnek akár másodperceket is várnia kell a válaszra
- Nem támogatja a tranzakciókezelést

2. Az ős-webszolgáltatások

2.1. XML-RPC

Az XML-RPC minden XML alapú RPC[9] szolgáltatás őse, 1998 óta létezik. Lényege, hogy egy XML dokumentumot küldenek, általában HTTP POST módszerrel a szervernek, amely a kérést feldolgozva szintén XML-ben küldi a választ. A szolgáltatás minimalista, a küldött és a kapott XML fájlok rendkívül egyszerűek, könnyen kezelhetőek. Összesen kilenc, egyszerű adattípus használható vele, ezek: array, base64, boolean, datetime, double, integer, string, struct és nil. Az egyszerűségből fakad legnagyobb hátránya: egész egyszerűen nincs elég elérhető típus (tipikusan kollekcióknál) a magasszintű programozási nyelvek objektumainak szerializálására. Egy tipikus XML-RPC kérés így néz ki:

```
<?xml version="1.0"?>
<methodCall>
  <methodName>pelda.Osszead</methodName>
  <params>
    <param><value><int>1</int></value></param>
    <param><value><int>2</int></value></param>
  </params>
</methodCall>
```

1. kód
XML RPC kérés

Látható, hogy meg kell adni a meghívandó metódus nevét, és a paramétereit. Megjegyzendő, hogy a <params> node akkor is kell, ha nincsenek paraméterek. Egy erre a hívásra érkezett válasz így néz ki:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param><value><int>3</int></value></param>
  </params>
</methodResponse>
```

2. kód
XML RPC válasz

9 Remote Procedure Call

Amennyiben a szerver nem tudja végrehajtani a kérést, úgy egy speciális fault XML-t küld, amely tartalmazza a hiba kódját, és szöveges leírását is.

Kérdés, hogy a kliens mi módon szerezhet tudomást a szerveren meghívható metódusokról, azok specifikációjáról, leírásáról. Erre három beépített függvénye van az XML-RPC-nek:

Függvény	Jelentés
system.listMethods	Egy sztring-tömbben (array típus) visszaadja a meghívható metódusokat
system.methodSignature	Megadja a paraméterként átadott nevű metódus specifikációját, vagy specifikációit (szintén array)
system.methodHelp	Ha elérhető a paraméterként megadott metódus dokumentációja, akkor visszatér azzal, ha nem, akkor üres sztringgel

1. táblázat
XML RPC függvények metódus lekérdezéshez

2.2. SOAP

2.2.1. Bevezetés

Az XML-RPC továbbfejlesztéseként még 1998-ban megjelent, 2003-tól pedig W3C ajánlás lett (1.1-es verzió), jelenleg az 1.2-es a legfrissebb. Nem kizárólagosan RPC-k kezelésére készült, hanem általános üzenetküldésre. Előnye, hogy a HTTP mellett HTTPS[10], vagy akár SMTP protokollal is használható, platform-és nyelvfüggetlen. Számos nyelv beépített támogatást tartalmaz SOAP üzenetek kezelésére. Hátránya, hogy lassú (összehasonlítva például a CORBA[11]-val), különösen nagy méretű üzeneteknél, valamint nem minden nyelv támogatja megfelelően (pl. a PHP[12]).

10 HTTP Secure

11 Common Object Request Broker Architecture

12 Hypertext Preprocessor

Maga a SOAP üzenet egy XML fájl, és SOAP Envelope-nak (SOAP boríték) hívjuk. A boríték egy fejlécből és egy üzenettörzsből áll. Minden egyes elem névtérben található, amely az emberi olvasást nehézkessé teszi. Ugyanakkor a névterek használatának előnye a névütközések elkerülése. A SOAP belső működéséhez a Schema-t használja, egy elem teljes névtere megegyezés szerint az elemet leíró Schema fájl.

Egy SOAP kérés a következőképpen nézhet ki:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
  http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Body>
  <req:echo xmlns:req =
       "http://localhost:8080/axis2/services/MyService/">
    <req:category>classifieds</req:category>
  </req:echo>
</soapenv:Body>
</soapenv:Envelope>
```

3. kód
SOAP kérés

Erre érkezett válasz:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
  http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Header>
  <wsa:ReplyTo>
    <wsa:Address>
  http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:Address>
  </wsa:ReplyTo>
  <wsa:From>
    <wsa:Address>
      http://localhost:8080/axis2/services/MyService
    </wsa:Address>
  </wsa:From>
  <wsa:MessageID>
    ECE5B3F187F29D28BC11433905662036
```

```
  </wsa:MessageID>
</soapenv:Header>
<soapenv:Body>
  <req:echo xmlns:req =
       "http://localhost:8080/axis2/services/MyService/">
    <req:category>classifieds</req:category>
  </req:echo>
</soapenv:Body>
</soapenv:Envelope>
```

4. kód
SOAP válasz

Látható, hogy a SOAP borítékon belül négyféle névtér létezik, valamint hogy egy egyszerű kérés is milyen mennyiségű fizikai adatot jelent.

2.2.2. A http://schemas.xmlsoap.org/soap/envelope/ névtér

Ez a névtér definiálja a SOAP boríték felépítését, vagyis az Envelope, Body, Fault, és Header elemeket, azok egymáshoz viszonyított helyzetét. Legkülső elem az Envelope, ennek különálló gyerekelemei a Header, Body, és Fault node-ok.

2.2.3. A http://www.w3.org/2001/12/soap-encoding névtér

Az encodingStyle attribútum az XML dokumentumban használt adattípusok definiálására való. Bármely elemen megjelenhet, lefelé terjed. Már nem támogatott.

2.2.4. A http://www.w3.org/2001/XMLSchema névtér

Ez a névtér definiálja az XML-ben elérhető különböző primitív (pl. boolean) és komplex (pl. group) típusokat, az elérhető elemeket, attribútumokat, ezek alapértelmezett értékét és egymáshoz viszonyított sorrendjét.

2.2.5. A http://www.w3.org/2001/XMLSchema-instance névtér

Ez a névtér együtt használatos az XMLSchema névtérrel. Attribútumokat definiál, melyek segítségével típust adhatunk meg (xsi:type), null elemet definiálhatunk (xsi:nil), vagy a schema dokumentum helyét (xsi:schemaLocation) tudjuk megadni.

2.2.6. Összefoglalás

A SOAP jóval kifinomultabb hibakezeléssel rendelkezik, mint az XML-RPC. A hibaüzenetben rendelkezésre áll a hibakód, a hiba kiváltója, valamint egy részletes hibaleírás (pl. melyik sorban történt a hiba).

A SOAP adattípusait tekintve is gazdagabb, támogatott típusai: boolean, integer, double, sztring, tömb, asszociatív tömb, objektum, kevert típus. Ezek segítségével már alkalmas objektumok továbbítására a hálózaton.

2.3. WSDL

A WSDL egy XML alapú webszolgáltatás-leíró nyelv, és szorosan kötődik a SOAP-hoz, a webszolgáltatás nyilvános felületét írja le. Egy webszolgáltatás igénybe venni kívánó kliens a WSDL fájlból tudja meg, milyen metódusok érhetőek el a szerveren, és hogyan. 2000 óta létezik, a WSDL 2.0 pedig 2007 óta W3C ajánlás. A használt speciális adattípusok a WSDL fájlba vannak ágyazva Schema formátumban. Ugyanúgy névtereket használ, mint a SOAP, és igen terjedelmes XML-t jelent, egyszerű szolgáltatás esetén is. A nyelvi implementációk automatikus WSDL generálási lehetőséggel rendelkeznek.

```
<?xml version="1.0"?>
<wsdl:definitions name="StockQuote"
targetNamespace="http://example.com/stockquote.wsdl"
xmlns:tns="http://example.com/stockquote.wsdl"
xmlns:xsd1="http://example.com/stockquote.xsd"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <xsd:schema targetNamespace="http://example.com/stockquote.xsd"
        xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
      <xsd:element name="TradePriceRequest">
        <xsd:complexType>
          <xsd:all>
            <xsd:element name="tickerSymbol" type="string"/>
          </xsd:all>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="TradePrice">
        <xsd:complexType>
          <xsd:all>
```

```
            <xsd:element name="price" type="float"/>
          </xsd:all>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="GetLastTradePriceInput">
    <wsdl:part name="body" element="xsd1:TradePriceRequest"/>
  </wsdl:message>
  <wsdl:message name="GetLastTradePriceOutput">
    <wsdl:part name="body" element="xsd1:TradePrice"/>
  </wsdl:message>
  <wsdl:portType name="StockQuotePortType">
    <wsdl:operation name="GetLastTradePrice">
      <wsdl:input message="tns:GetLastTradePriceInput"/>
      <wsdl:output message="tns:GetLastTradePriceInput"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="StockQuoteSoapBinding"
      type="tns:StockQuotePortType">
      <soap:binding style="document"
           transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="GetLastTradePrice">
      <soap:operation
           soapAction="http://example.com/GetLastTradePrice"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="StockQuoteService">
    <wsdl:documentation>My first service</documentation>
    <wsdl:port name="StockQuotePort"
         binding="tns:StockQuoteBinding">
      <soap:address location="http://example.com/stockquote"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

5. kód
Egy egyszerű WSDL fájl

2.4. UDDI

Az UDDI egy platformfüggetlen, XML alapú nyilvántartó rendszer, a webszolgáltatások harmadik alapvető eleme (a SOAP és a WSDL mellett), szintén 2000 óta létezik. SOAP üzenetekkel kérdezhető le, és a WSDL dokumentumokhoz biztosít hozzáférést. Az UDDI-t mint egy brókert képzelték el a webszolgáltatások világában. Ha egy cég új webszolgáltatást vezet be, akkor azt be kell jegyeznie a brókernél, a kliensek pedig a brókertől tudják meg, hogy ilyen szolgáltatás egyáltalán létezik, és veszik fel a kapcsolatot a szolgáltatóval. Az UDDI ilyen módon kritikus komponense a webszolgáltatásnak, különösen a pénzért igénybe vehetőknek.

1. ábra
Az UDDI ahogy elképzelték

2.5. WS-Security

A WS-Security egy olyan kommunikációs protokoll, mely eszközöket tartalmaz a webszolgáltatások biztonságossá tételéhez. Viszonylag későn, 2004-ben jelent meg, 1.1-es verzióját 2006-ban érte el. A WSS leírja, hogyan használjunk webszolgáltatásokban SAML és Kerberos titkosítást, vagy éppen tanúsítványokat. Leírást tartalmaz a SOAP üzenetek aláírásáról, és titkosításáról. Fontos megjegyezni, hogy meglévő, jól bevált szabványokra épít, mint az XML DigitalSignature, XML Encryption, vagy X.509.

Szükségszerűen jött létre, hiszen a SOAP kommunikáció miatt tudnunk kell autentikálni a szolgáltatást igénybe venni akaró felet, aláírni az üzenetet, és titkosítani azt. Ilyen módon ismertté válik a hívó, aki egyben képes lesz ellenőrizni, hogy a kommunikáció során harmadik fél nem változtatta-e meg az üzenetet, illetve harmadik fél nem férhet hozzá illetéktelenül a küldött üzenethez.

Elemeit tekintve, a WS-SecurityPolicy a titkosítási követelményeket írja le, a WS-Interoperability azért felelős hogy a különböző megvalósítások kompatibilisek legyenek egymással. A WS-SecureConversation leírást tartalmaz, hogy a titkosítási kulcsok igénylése vagy éppen generálása hogyan történik. A WS-Trust a biztonsági tokenek kibocsátásáról, megújításáról, ellenőrzéséről rendelkezik, a WS-RM a megbízható üzenetküldésről, a WS-Authorization pedig az autorizációs kérdésekről.

A WS-Security üzenetszintű titkosítást jelent, az átviteli közeggel nem foglalkozik. Ha biztonságos átvitelt szeretnénk, akkor a WSS mellett / helyett Transport Layer Security-t (TLS) kell használnunk, ez webszolgáltatás esetén például HTTPS-t jelent.

Folyamatát tekintve először a hívó azonosítása zajlik le egy UserNameToken, Kerberos ticket, vagy éppen X.509 tanúsítvánnyal. Természetesen a UserNameToken-ben a jelszó hash-elve van. A szolgáltatás ellenőrzi, hogy a hívó jogosult-e meghívni az adott metódust, majd előállítja a nyers választ. Következik a nyers üzenet aláírása, ami persze önmagában nem elég, hogy harmadik fél illetéktelenül hozzáférjen a tartalmához, csak az üzenet érintetlenségének megállapítására jó. X.509 esetén a tanúsítvány segítségével ír alá, Kerberos esetén a session kulccsal, UserNameToken esetén pedig a jelszóval. Következik az aláírt üzenet titkosítása, amely lehet szimmetrikus, és aszimmetrikus. Szimmetrikus kódolásnál mindkét félnek rendelkeznie kell ugyanazzal a titkosítási kulccsal, ezt adott esetben problémás megoldani. Alternatíva, ha már az autentikációt tanúsítvánnyal oldjuk meg, akkor aszimmetrikus kódolással dolgozzunk. Megjegyzendő, hogy csak a SOAP üzenet Body része kerül titkosításra, a fejléc nem.

3. A Microsoft válasza: WCF

A korai webszolgáltatások nagy hátránya, hogy elsajátításuk nehéz, nagy szakértelmet kíván. Más technológiát kell alkalmaznunk IPC[13] esetén, szöveges, vagy bináris átvitelnél. Változik a kód, a programot újra kell fordítani.

A WCF[14] a 3.0-s .NET Framework-ben jelent meg először, 2006-ban. Szolgáltatás orientált alkalmazásokat lehet a segítségével létrehozni, és a korábbiaktól eltérően egységes modellt nyújt a fejlesztőknek. A kód változtatása nélkül, egy konfigurációs fájl szerkesztésével alkalmas bináris, szöveges, IPC, vagy akár P2P[15] kommunikációra. Számos szolgáltatása van biztonság, tranzakció kezelés, és biztonságos átvitel terén. Teljes körű diagnosztikai funkciókkal rendelkezik, úgymint: naplózás, nyomkövetés, teljesítményszámlálók.

3.1. A WCF alapfogalmai

- **Endpoint**: A szerver végpontokon keresztül nyújt szolgáltatásokat a kliensnek. Minden endpoint három részből áll: Address, Binding, Contract, erre szoktak ABC néven hivatkozni.
- **Address**: A végpont címe, egy URI[16], ahol a szolgáltatás elérhető.
- **Binding**: A kommunikáció módját adja meg. Lehet a felhasználó által definiált, de számos beépített binding áll rendelkezésre, saját használatára ritkán van szükség.
- **Contract**: A szolgáltatás interfészének leírása, vagyis hogy a szolgáltatás milyen műveleteket támogat.

13 Inter-process communication

14 Windows Communication Foundation

15 Peer to peer

16 Uniform Resource Identifier

3.2. A WCF beépített kötései (binding)

- **BasicHttpBinding**: HTTP protokollon alapuló, szöveges XML formátumú küldést definiál.
- **WsHttpBinding**: Biztonságos, egyirányú üzenetküldést definiál.
- **WsDualHttpBinding**: Biztonságos, duplex üzenettovábbítást és SOAP alapú kommunikációt definiál.
- **NetTcpBinding**: Bináris, WCF alkalmazások közötti gép-gép kapcsolaton megvalósított megbízható kommunikációt definiál. Csak .NET-es alkalmazások között használható.
- **NetNamedPipeBinding**: Egy gépen futó, IPC kommunikációt definiál. Csak Windows platformon használható.
- **NetMsmqBinding**: Várakozási soron alapuló kötés. Előnye, hogy ha a szerver ideiglenesen nem érhető el, a várakozási sor bizonyos időközönként újra próbálkozik a küldéssel. Csak Windows platformon használható.
- **NetPeerTcpBinding**: P2P kommunikációt definiál.

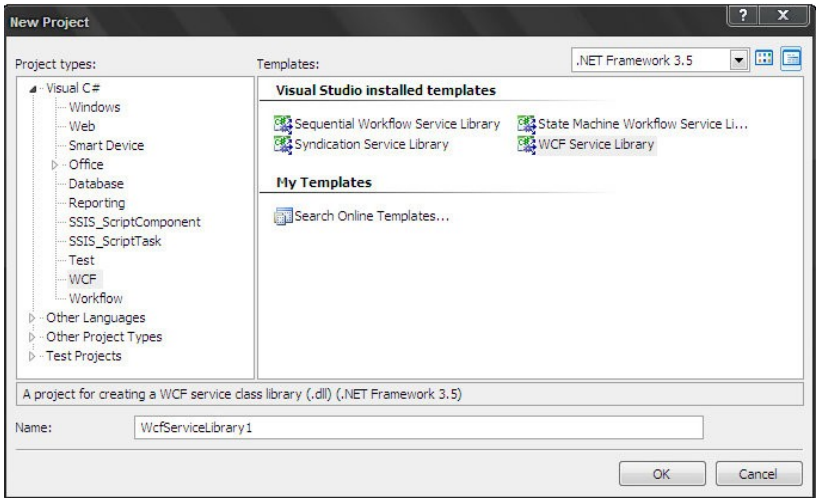
Látható, hogy ha a kliens és a szerver egyaránt .NET-es alkalmazás, akkor számos lehetőség adott a nekünk megfelelő kötés kiválasztására. De ha a kliens vagy a szerver nem .NET-es, akkor is bőven marad lehetőség, amiből választhatunk. Természetesen mindig az adott feladathoz leginkább illeszkedő kötést érdemes választani. A `basicHttpBinding` nem támogat semmilyen biztonságot, ellenben van néhány szolgáltatás (tipikusan a régi webszolgáltatások), amelyek egyszerűen nem képesek többre. Leggyakrabban használt kötés a `wsHttpBinding`, biztonságos, és széles körben használt. Ha a kliens és a szerver ugyanazon a gépen van, akkor jelentős sebességnövekedés érhető el a `netNamedPipeBinding` használatával, ugyanis kevesebb hálózati rétegen megy át az üzenet. Nagyszámú adat küldésénél a `netTcpBinding` jóval gyorsabb a kevesebb átküldendő adat miatt, mint az összes többi. Ha kiemelkedően fontos, hogy az üzenet akkor is

célba érjen, ha a szerver ideiglenesen nem elérhető, akkor az egyetlen lehetőség a `netMsmqBinding` használata. Illetve ha a kommunikáció peer to peer kapcsolatot igényel, akkor a `netPeerTcpBinding`-ot kell választanunk.

4. Egyszerű WCF szolgáltatás készítése

4.1. Előkészületek

A példa összeállításához Visual Studio 2008 SP1-et, illetve .NET Framework 3.5 SP1-et fogok használni. A korábbi verziókban volt néhány bosszantó hiba, amit már javítottak. WCF projekt létrehozásához a Visual Studio-ban a File/New/Project-ben a WCF kategória alatt a WCF Service Library-t kell választani (1. ábra), majd az elkészült projektben törölni az automatikusan létrehozott IService1.cs és Service1.cs fájlokat.



2. ábra
WCF projekt létrehozása

4.2. A tényleges kódok

WCF szolgáltatás készítéséhez szükség van egy interfészre, és egy osztályra, ami azt implementálja. Az interfészt el kell látni `ServiceContract` attribútummal, ami a `System.ServiceModel` névtérben van. Szükség van még a metódusok fejlécének `OperationContract` attribútummal ellátására. Egy minimális ilyen interfész:

```
using System.ServiceModel;
namespace DiplomamunkaLib
{
  [ServiceContract]
  public interface IDiplomamunka
  {
    [OperationContract]
    string echo(string input);
  }
}
```

6. kód
WCF interfész

Szükség van még egy osztályra, ami ezt az interfészt implementálja. Itt már nincs szükség attribútumokra.

```
using System;
namespace DiplomamunkaLib
{
  public class Diplomamunka:IDiplomamunka
  {
    public string echo(string input)
    {
      Console.WriteLine(input);
      return input;
    }
  }
}
```

7. kód
WCF osztály

Ez a szolgáltatás nem fog mást csinálni, mint szerver oldalon kiírja azt, amit a kliens küldött, és visszaadja a kliensnek.

4.3. Végpontok beállítása

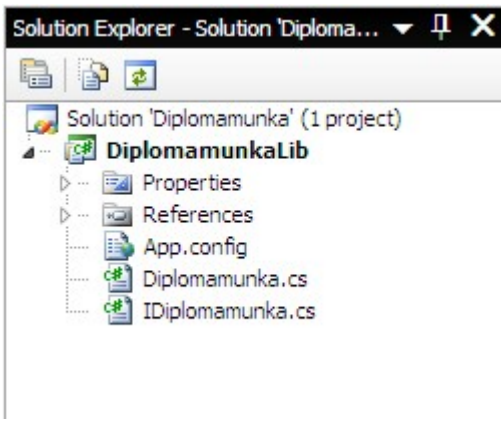
A Solution Explorerben (2. ábra) az App.config fájlon kattintva érhető el a szerkesztő, Edit WCF Configuration alatt. A beállítási lehetőségek közül egyedül a Services pont az érdekes. Az endpoint konfiguráció nevének végét át kell nevezni `Service1`-ről `Diplomamunkára`. A `DiplomamunkaLib.Diplomamunkában` a Host-ra kattintva (3. ábra) be kell állítani a kiindulási címét a szolgáltatásnak. Erre tökéletes a `http://localhost:8731/` , és

innentől kezdve a végpontoknak már csak relatív címet kell adni. Fontos megjegyezni, hogy Vistánál és az újabb Windows-okon nincs automatikus portnyitási engedélye a felhasználónak, azt explicit engedélyezni kell a következő paranccsal:

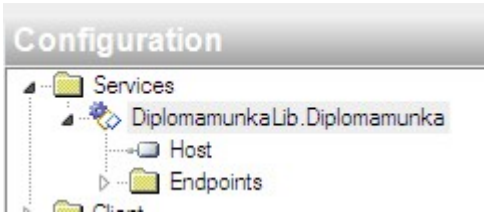
```
netsh http add urlacl url=http://+:8731/ user=domain\user.
```

A végpontok tényleges beállítása következik. Az első végpontnál névnek tetszőleges sztring beállítható, cél a könnyű azonosíthatóság. Binding-nak egyelőre `basicHttpBinding`-ot használok, Contract-nak pedig a 4.2 pontban létrehozott interfészt kell megadni (4. ábra).

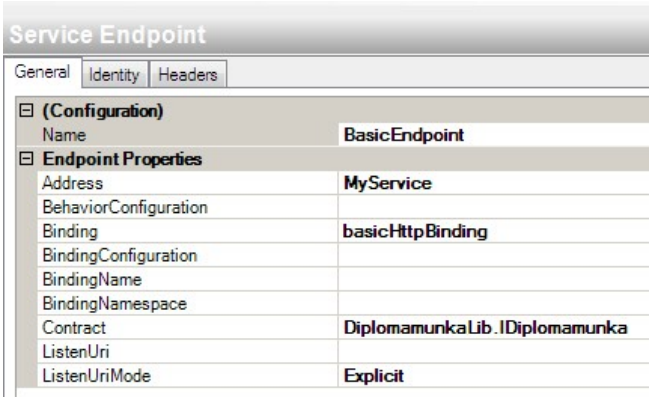
A másik endpoint speciális, a szolgáltatás propagálására való, címe mindig mex, binding-ja valamelyik mex-es binding (http kapcsolat esetén `mexHttpBinding`), contract-ja pedig `IMetadataExchange` (5. ábra). A konfigurációt menteni kell, majd bezárni a szerkesztőt.



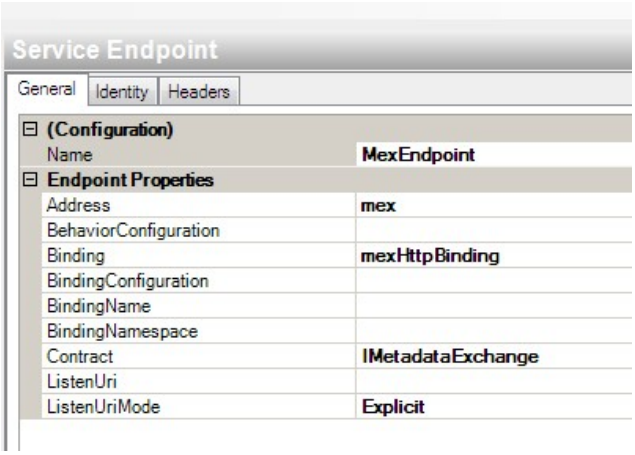
3. ábra
Solution Explorer



4. ábra
Host beállítása



5. ábra
Endpoint beállításai

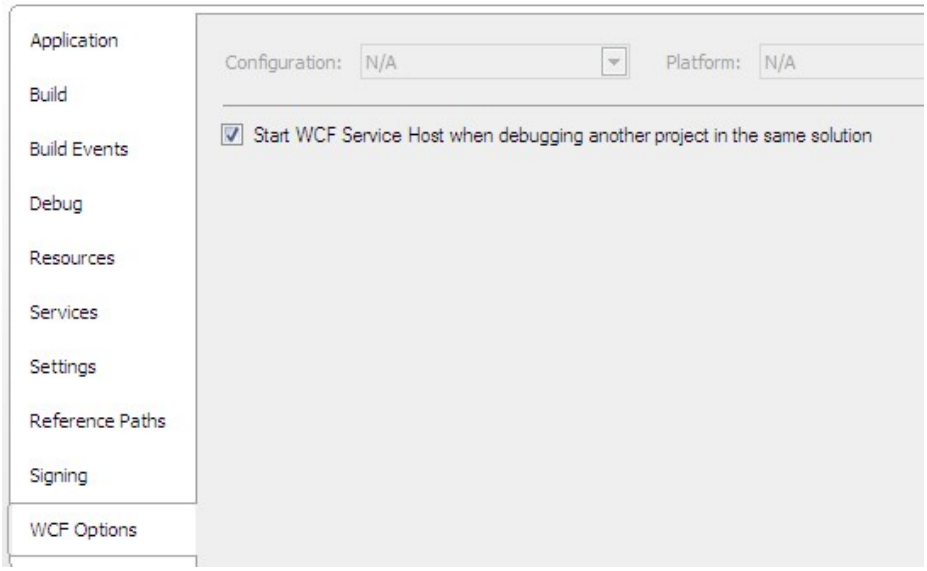


6. ábra
Mex endpoint

4.4. Host alkalmazás

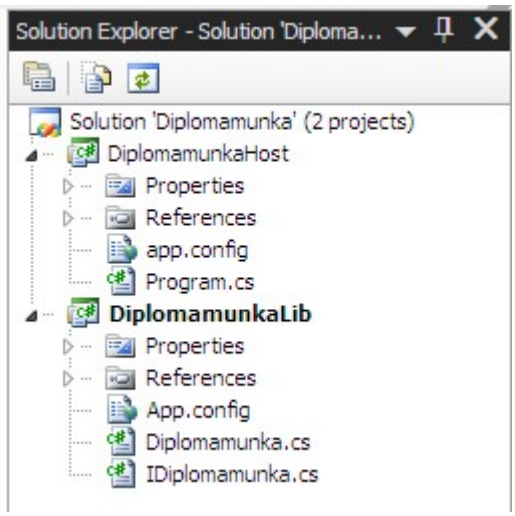
A fent elkészített alkalmazást lefordítva egy dll-t kapunk, ami önmagában még nem elég. Szükség van egy másik alkalmazásra, ami hosztolja. Ez akár IIS[17] is lehet, a gyakorlatban azonban egy Windows Service. Én az egyszerűség kedvéért egy ConsoleApplication-t fogok használni. Előkészületként le kell tiltani a dll automatikus hosztolását, ehhez a Solution Explorer-ben a projekten kattintva, a properties ablak WCF Options részében ki kell kapcsolni a Start WCF Service Host... részt (6. ábra).

17 Internet Information Service



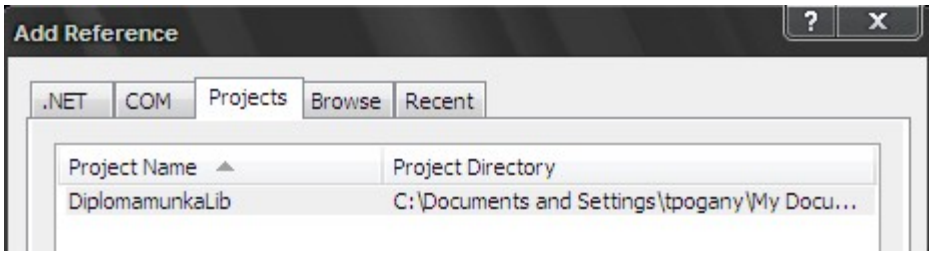
7. ábra
WCF Service host kikapcsolása

Ezután hozzá kell adni a solution-höz egy új ConsoleApplication projektet, majd a WCF Service Library-ben lévő, korábban elkészített App.config fájlt átmásolni az új projektbe (7. ábra).



8. ábra
A hosztoló projekt hozzáadva

A hosztolást a `ServiceHost` osztály fogja végezni, ami a `System.ServiceModel` névtérben található, ezt referenciaként hozzá kell adni a projekthez. Hozzá kell még adni referenciaként a korábban elkészített WCF Service Library projektet (8. ábra).



9. ábra
WCF Service Library hozzáadása

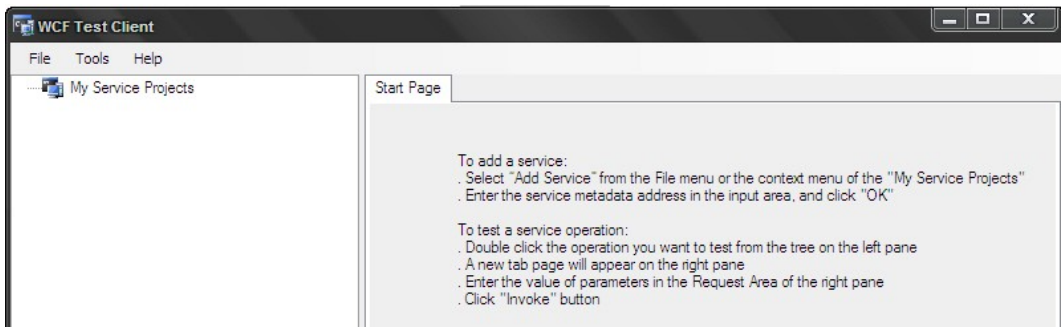
```
using System;
using System.ServiceModel;
using DiplomamunkaLib;
namespace DiplomamunkaHost
{
  class Program
  {
    static void Main(string[] args)
    {
      try
      {
        ServiceHost host = new ServiceHost(typeof(Diplomamunka));
        host.Open();
        Console.WriteLine("Elindult");
        Console.ReadLine();
        host.Close();
      }
      catch (Exception ex)
      {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
      }
      }
  }
}
```

8. kód
WCF host

A solution beállításánál még be kell állítani, hogy csak a hoszt projekt induljon el, és a szerver résszel készen vagyunk.

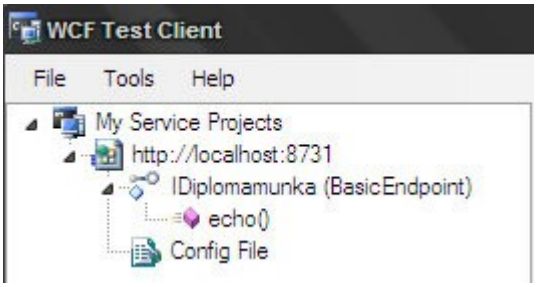
4.5. Beépített kliens használata

A Visual Studio rendelkezik beépített WCF klienssel. Kiválóan alkalmas a szolgáltatást tesztelni, képes például megjeleníteni a szervernek küldött, és az onnan kapott XML üzeneteket. A kliens a Visual Studio Command Prompt-ból indítható `WcfTestClient` paranccsal (9. ábra).

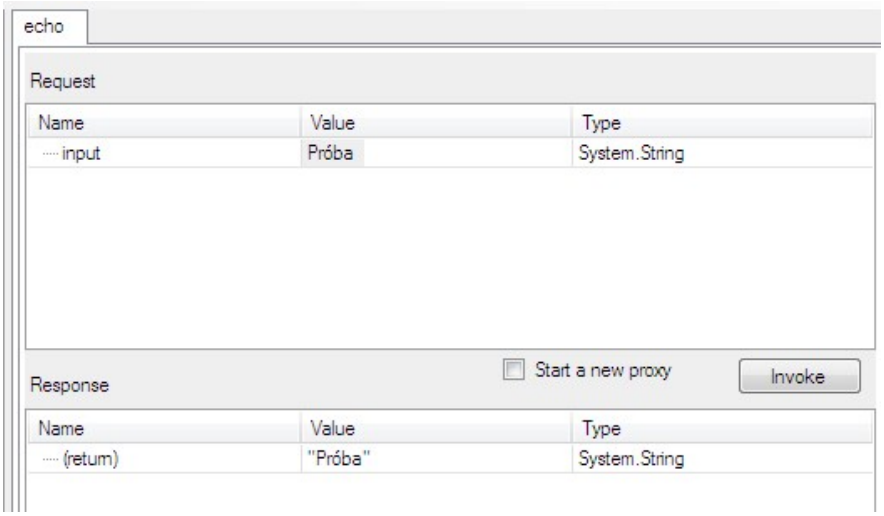


10. ábra
Beépített WCF kliens

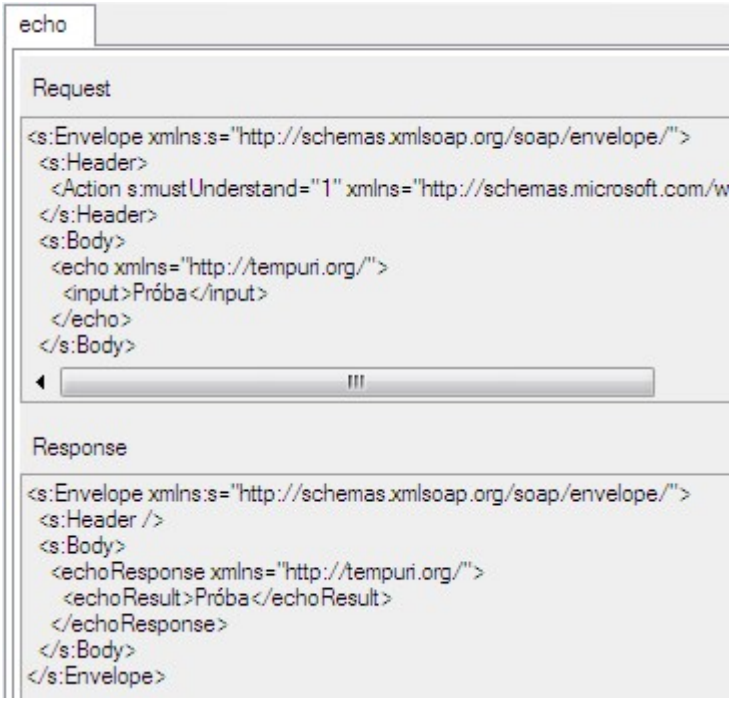
Hozzá kell adni a szervert a szolgáltatások listájához, miután Visual Studio-ban elindítottuk. Ehhez a File/Add service menüjében meg kell adni a szolgáltatás címét (`http://localhost:8731/`). A mex endpoint segítségével automatikusan letöltődik és láthatóvá válik az elérhető metódusok listája (10. ábra), majd a metóduson kattintva, és a Value mezőt kitöltve meghívható a metódus az Invoke gombbal (11. ábra), és kis idő elteltével látható az eredmény is. Alul XML nézetbe átkapcsolva látható, hogy a tényleges üzenet SOAP-ra képződött le, és a válasz is SOAP üzenet. A szerver rész ablakában szintén láthatóak az elküldött értékek.



11. ábra
Az elérhető metódusok



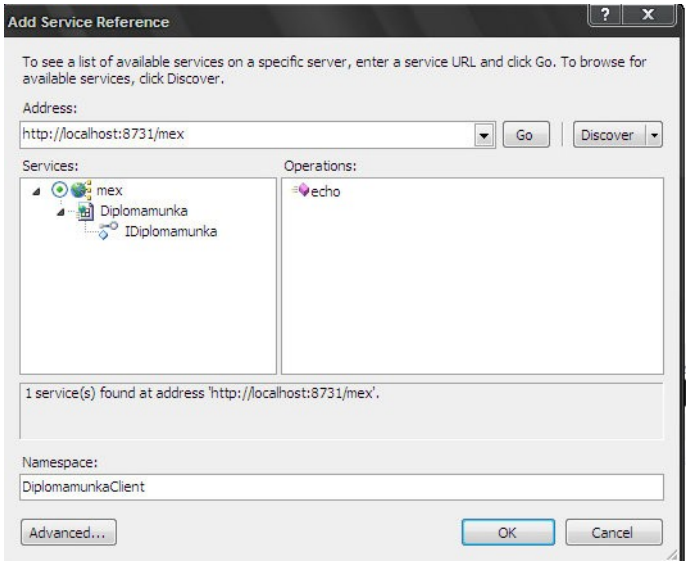
12. ábra
Paraméterezés, és metódus meghívása



13. ábra
A küldött és kapott XML

4.6. Saját kliens használata

Saját kliens használatához a solution-ben hozzá kell adni egy új projektet, illetve be kell állítani hogy a szerverrel együtt ez is induljon el. A References-en kattintva hozzá kell adni a korábban létrehozott WCF Service-t, mint Service Reference-t. A Discover gombra kattintva automatikusan megtalálja a WCF alkalmazást, és felépíti a hozzá tartozó proxy-t (13. ábra).



14. ábra
Referencia hozzáadása

Ezek után egy egyszerű WCF kliens így nézhet ki:

```
using System;
namespace TestClient
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.ReadLine();
      try
      {
        DiplomamunkaClient.DiplomamunkaClient client =
        new TestClient.DiplomamunkaClient.DiplomamunkaClient();
        client.Open();
        Console.WriteLine(client.echo("Próba"));
        client.Close();
      }
      catch (Exception ex)
```

```
      {
        Console.WriteLine(ex.Message);
      }
      finally
      {
        Console.ReadLine();
      }
    }
  }
}
```

9. kód
WCF kliens

Egyedül a kód elején lévő `Console.ReadLine` szorul magyarázatra. Előfordulhat, hogy az egyszerre történő indításkor a kliens hamarabb indul el, mint a szerver, ekkor természetesen kivételt kapnánk, ezért lett beiktatva a várakozás.

4.7. Összefoglalás

Látható, hogy egy egyszerű kliens és szerver összeállítása egyáltalán nem nehéz, a kód megírásán kívül csak a végpontok összeállítása igényel egy kis odafigyelést, minden más kvázi automatikus. Másik átviteli mód választásánál csak a végpont beállításán kell módosítani, a kódot egyáltalán nem érinti, és a kliensben kell frissíteni a Service Reference-t, ami fájlrendszer szinten szintén egy XML fájl.

5. Adatok átküldése a hálózaton

5.1. Általános kérdések

Akármilyen adatküldési módot is választunk a végpont összeállítása során, a háttérben a tényleges küldés előtt szerializáció történik. A választott kötés függvényében ez XML vagy bináris szerializáció, így követni kell a .NET szerializációs szabályait. Fontos ismételten megjegyezni, hogy bináris szerializáció csak .NET szerver és kliens között támogatott.

5.2. Egyszerű adattípusok

Az egyszerű adattípusokkal nincs tennivaló, mind a két szerializációt további beállítás nélkül támogatják. Egyszerű adattípusok közé tartozik a bool, az int, a string, illetve ezek tömbjei. Típusjelölés csak a tömböknél van, egyéb esetben a típuskonverzió a WSDL fájl alapján történik. Nagyobb mennyiségű adatok esetén érdemes bináris szerializációval dolgozó kötést választani, ez kevesebb átküldendő adatot generál.

5.3. Objektumok

XML szerializáció (SOAP) csak olyan osztály objektumaira kérhető, amely publikus, rendelkezik default konstruktorral, az objektumnak körmentesnek kell lennie, és csak a publikus adattagok kerülnek szerializálásra. Az osztályt el kell látni `DataContract` attribútummal, az adattagokat pedig `DataMember` attribútummal. Ezek után az objektum átküldhető a hálózaton. Természetesen a megfelelő osztálynak a kliens oldalon is léteznie kell (erős, típusos kötés). Egy helyesen összeállított osztály például:

```
using System.Runtime.Serialization;
namespace DiplomamunkaLib
{
  [DataContract]
  public class People
  {
     [DataMember]
     public string Name { get; set; }
     [DataMember]
     public int Age { get; set; }
     public People(){}
     public People(string name, int age)
     {
       this.Name = name;
       this.Age = age;
     }
  }
}
```

10. kód
Egy helyes osztály WCF küldéshez

Abban az esetben, ha sok, már korábban meglévő osztályunk van, akkor körülményes lehet azokat attribútumokkal ellátni. Lehetőségünk van a `ServiceKnownType` attribútumot használni az interfészben, így az osztályok átemelése automatikus lesz.

```
[ServiceKnownType(typeof(People))]
[ServiceContract]
public interface Idiplomamunka
```

11. kód
ServiceKnownType attribútum

Bináris szerializációnál tetszőleges osztályt használhatunk, a privát tagokat is sorosítja, az osztályt `Serializable` attribútummal kell ellátni, természetesen a `DataContract` mellett.

Lehetőség van akár saját szerializációra is, ekkor a sorosított objektumot mint sztringet (vagy bináris adatot) küldjük át a hálózaton, és a kliens oldalon újra felépítjük. Ennek a megoldásnak az előnye, hogy teljes mértékben kézben tartható a szerializálási folyamat, szükség esetén akár titkosíthatjuk is az objektumot. Egyéni szerializálásnál elhagyhatóak a `DataContract` és `DataMember` attribútumok, a szerializációs szabályokat viszont követni kell.

```
public string createPeople(string name, int age)
{
  People p = new People(name, age);
  XmlSerializer ser = new XmlSerializer(typeof(People));
  StringWriter output = new StringWriter(new StringBuilder());
  ser.Serialize(output, p);
  return (output.ToString());
}
```

12. kód
Egy megfelelő saját szerializálás szerver oldalon

```
client.Open();
string toDeserialize = client.createPeople("John", 23);
XmlSerializer ser = new XmlSerializer(typeof(People));
People p = (People)ser.Deserialize(new
StringReader(toDeserialize));
Console.WriteLine(p);
client.Close();
```

13. kód
Objektum visszanyerése kliens oldalon

```
<?xml version="1.0" encoding="utf-16"?>
<People xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<Name>John</Name>
<Age>23</Age>
</People>
```

14. kód
A szerializált objektum

6. Kivételkezelés

WCF webszolgáltatásokban a kivételkezelés a `FaultException` segítségével történik. .NET-ben ez az egyetlen olyan kivétel, amely nem a `System.Exception`-ből öröklődik. Az osztály a `System.ServiceModel` névtérben található. Különlegessége, hogy generikus kivétel, a generikus típus a ténylegesen történt kivétel típusa. Használatához az interfészben el kell látnunk a szükséges metódust `FaultContract` attribútummal, paraméterének pedig a tényleges kivételt mint típust megadni. Az attribútumot annyiszor kell felsorolni, ahány különböző kivétel lehetséges a metódusban (vagy használható az `Exception` is, mint általános kivétel).

```
[OperationContract]
[FaultContract(typeof(DivideByZeroException))]
double divide(double a, double b);
```

15. kód
Interfész metódus specifikáció, FaultContract attribútummal

Az interfészt implementáló WCF osztályban már nem kell attribútumot használni, csupán a metódusban kiváltani a szükséges kivételt, az előző példánál maradva a `DivideByZeroException`-t, és megindokolni, hogy miért történt a hiba (`FaultReason`):

```
public double divide(double a, double b)
{
  if (b == 0)
     throw new FaultException<DivideByZeroException>
     (new DivideByZeroException(),
     new FaultReason("Nullával osztás"));
  return a / b;
}
```

16. kód
A WCF osztályban a szükséges kivétel kiváltása

A kliens osztályban lehetőségünk van elérni a `FaultReason`-ben átadott üzenetet, illetve a tényleges kivétel üzenetét is:

```
try
{
 DiplomamunkaClient.DiplomamunkaClient client =
 new TestClient.DiplomamunkaClient.DiplomamunkaClient();
 client.Open();
 Console.WriteLine(client.divide(5, 0));
 client.Close();
}
catch (FaultException<DivideByZeroException> ex)
{
 Console.WriteLine(ex.Message);
 Console.WriteLine(ex.Detail.Message);
}
```

17. kód
A FaultException kezelése kliens oldalon

Fontos megjegyezni ismét, hogy a `FaultException` nem a `System.Exception` leszármazottja, egy `catch(Exception ex)` ág nem fogja meg ezt a kivételt. A kód lefuttatása előtt ki kell kapcsolnunk egy beállítást a kliens oldali kivétel helyes kezeléséhez. A Tools / Options / Debugging / General lapon ki kell kapcsolni az Enable Just My Code beállítást. Futtatáskor láthatjuk a "Nullával osztás" hibát, illetve az "Attempted to divide by zero" hibát. Az első a `FaultReason`, a második a `DivideByZeroException` üzenete.

Természetesen lehetőség van saját kivételek használatára is, ehhez az ősosztállyal nem rendelkező osztályt el kell látni az 5.3-as fejezetben ismertetett `DataContract` és `DataMember` attribútumokokal:

```
[DataContract]
public class MyException
{
 [DataMember]
 public string Message { get; set; }
 public MyException(){}
 public MyException(string msg)
 {
   this.Message = msg;
 }
}
```

18. kód
Saját kivétel WCF-ben

Ezek után a kivétel elérhető lesz kliens oldalon is (ServiceReference frissítés után), és az előző bekezdésekben bemutatott módon használható:

```
catch (FaultException<DiplomamunkaClient.MyException> ex)
```

19. kód
Saját kivétel catch ága kliens oldalon

7. Autentikáció és autorizáció

7.1. Tanúsítvány létrehozása

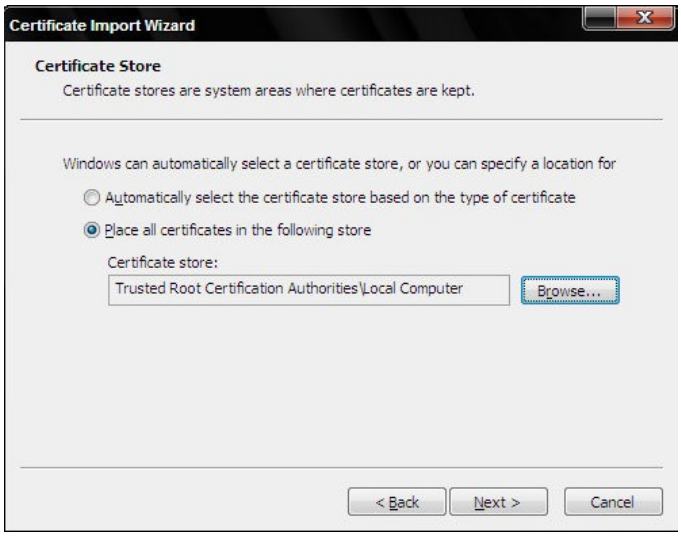
Ahhoz, hogy WCF webszolgáltatásban autentikációt és/vagy autorizációt használjunk, egy tanúsítványra van szükség, melynek természetesen meg kell lennie szerver oldalon (a privát részének), illetve kliens oldalon is (a publikus részének). Tanúsítvány vásárolható nevesebb cégektől, mint a VeriSign vagy a NetLock, ezeknek előnye, hogy a Windows automatikusan megbízhatónak tekinti őket, könnyen kezelhetőek, hátrányuk viszont az áruk. Ha nem kívánunk évi 70 dollár körüli összeget költeni erre, akkor magunknak kell tanúsítványt generálnunk.

Először egy gyökértanúsítványt (root certificate) kell generálnunk, ehhez a Visual Studio parancssorában kell kiadni ezt a parancsot:

```
makecert -n CN=DiplomaMunka -cy authority -r c:\rootcert.cer
```

20. kód
Gyökértanúsítvány létrehozása

A `DiplomaMunka` szöveg szabadon változtatható. A generált `rootcert.cer` fájlt fel kell telepíteni mind a kliens mind a szerver gépre, a Trusted Root Certification Authorities \ Local Computer tárolóba, így válik a gyökértanúsítvány a Windows számára megbízhatóvá.



15. ábra
Tanúsítvány importálása

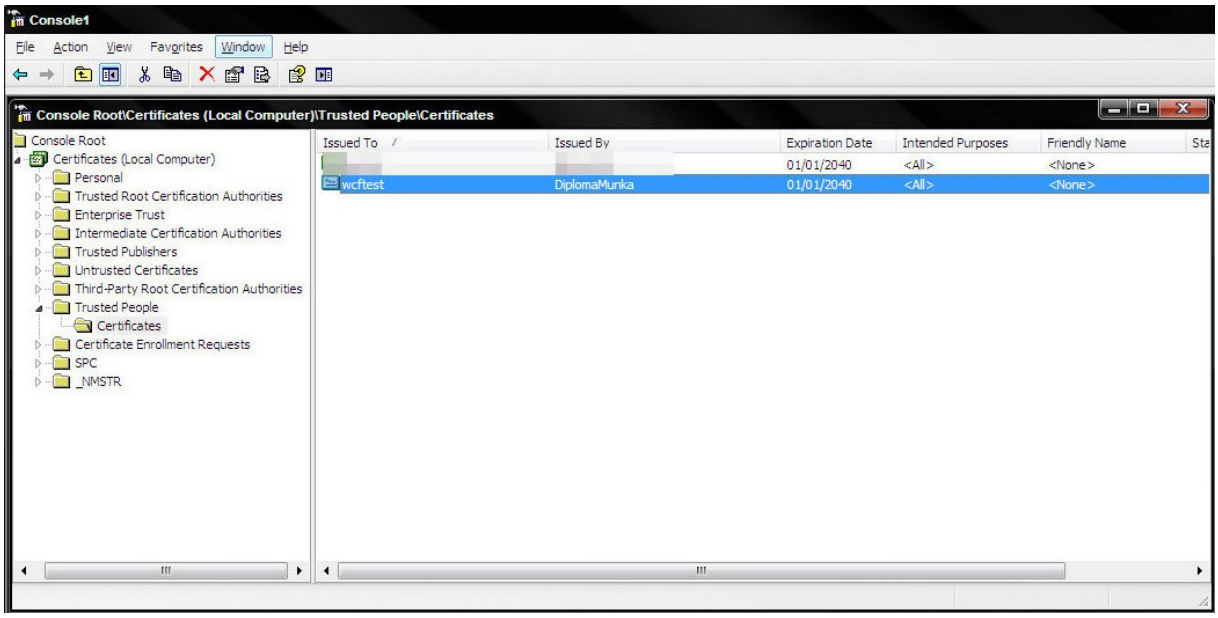
Most létre kell hozni egy szervertanúsítványt, amit az előzőleg létrehozott gyökértanúsítvánnyal írunk alá:

```
makecert -pe -n CN=wcftest -sky exchange -in DiplomaMunka
c:\server.cer -sv c:\server.pvk
pvk2pfx.exe -pvk c:\server.pvk -spc c:\server.cer -pfx
c:\server.pfx
```

21. kód
Szervertanúsítvány létrehozása

A generált tanúsítványt (a pfx fájlt) a szerver oldalon kell feltelepíteni, ez tartalmazza ugyanis a publikus és a privát részt is. A Trusted People \ Local Computer tárolóba kell telepíteni. Figyelni kell arra, hogy Windows-ban a tanúsítványoknak is vannak jogosultságaik, ezért vagy az telepítse ezt a tanúsítványt, akinek a nevében a szerver futni fog, vagy a `winhttpcertcfg` eszközzel utólag be kell állítani a jogosultságot.

Következő lépésként (ha a kliens és a szerver külön gépen van) ki kell exportálnunk a szervertanúsítvány publikus részét, ehhez el kell indítanunk az `mmc` programot, majd betölteni a Certificate beépülőt, a Local Computer tanúsítványokkal, majd megkeresni a Trusted People tárolóban az előbb feltelepített tanúsítványt.



16. ábra
A szervertanúsítvány

A tanúsítványon jobb kattintással előjövő menüben az All task \ Export pontot választva elmenthetjük a publikus részt, amit a kliens gépen szintén a Trusted People \ Local Computer tárolóba kell importálnunk, a jogosultságra figyelve. Vagy ha a még rendelkezésre áll a generált `server.cer`, azt is telepíthetjük.

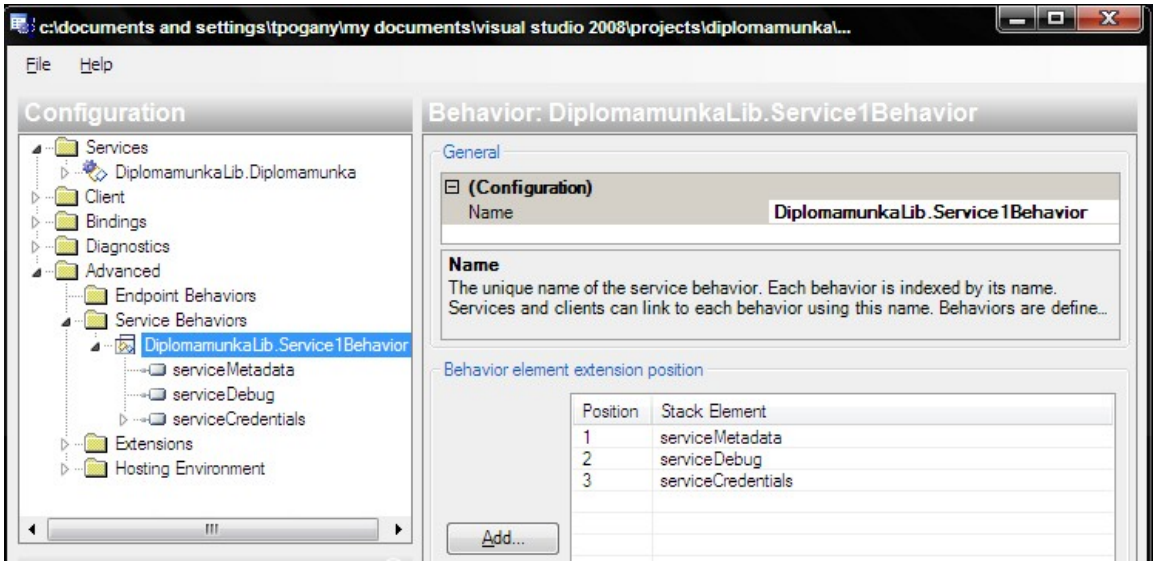
7.2. Autentikáció szerver oldal

Autentikáció létrehozásához először készíteni kell egy validátor osztályt, amely a `System.IdentityModel.Selector` névtérben található `UserNamePasswordValidator` osztályból öröklődik, és annak a `Validate` metódusát definiálja felül. Itt lehet ellenőrizni a felhasználói nevet és jelszót, és amennyiben nem megfelelőek, akkor `SecurityTokenException` kivételt kell dobni. Az alábbi példa a „diploma” kezdetű felhasználói neveket és bármilyen 5 karakteres jelszót fogad el:

```
using System.IdentityModel.Selectors;
using System.IdentityModel.Tokens;
namespace DiplomamunkaLib
{
  public class MyValidator: UserNamePasswordValidator
  {
    public override void Validate(string userName, string password)
    {
      if(!userName.StartsWith("diploma") || password.Length!=5)
      throw new SecurityTokenException("Failed to authenticate");
    }
  }
}
```

22. kód
Saját validátor osztály

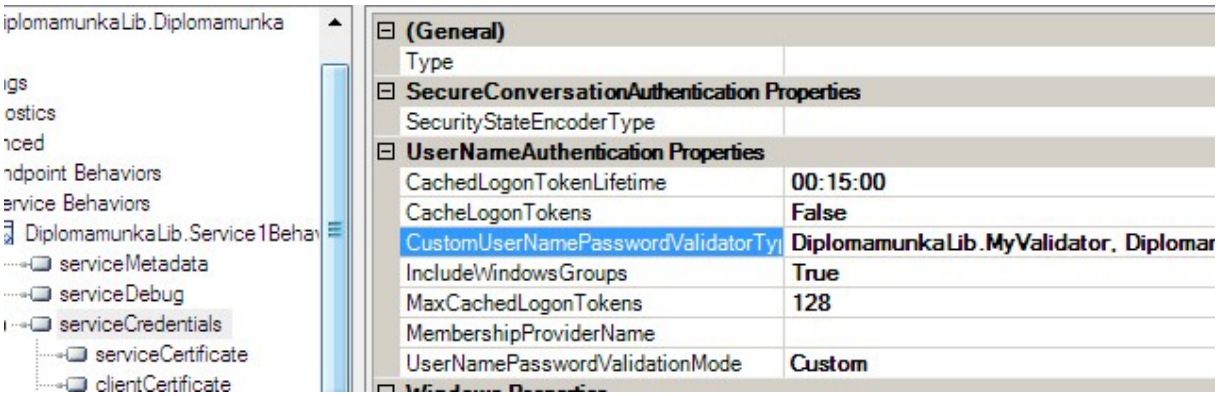
Ezután be kell állítani, hogy a webszolgáltatás használja is ezt a validátor osztályt, ehhez meg kell nyitni az App.config fájlt a WCF konfigurációszerkesztőben, majd az Advanced \ Service Behaviours-ban hozzá kell adni a serviceCredentials pontot a listához.



17. ábra
ServiceCredentials hozzáadása

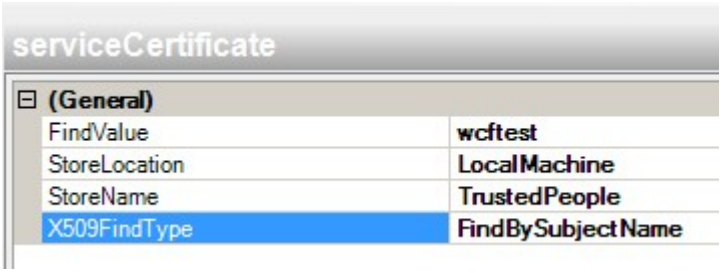
Ezután a serviceCredentials-ben CustomUserNamePasswordValidationType-nak be kell állítani a létrehozott validátor osztályt, ennek alakja: névtér.osztály, dll. Az én esetemben: `DiplomamunkaLib.MyValidator, Diplomamunka`.

UserNamePasswordValidationMode pedig Custom legyen!



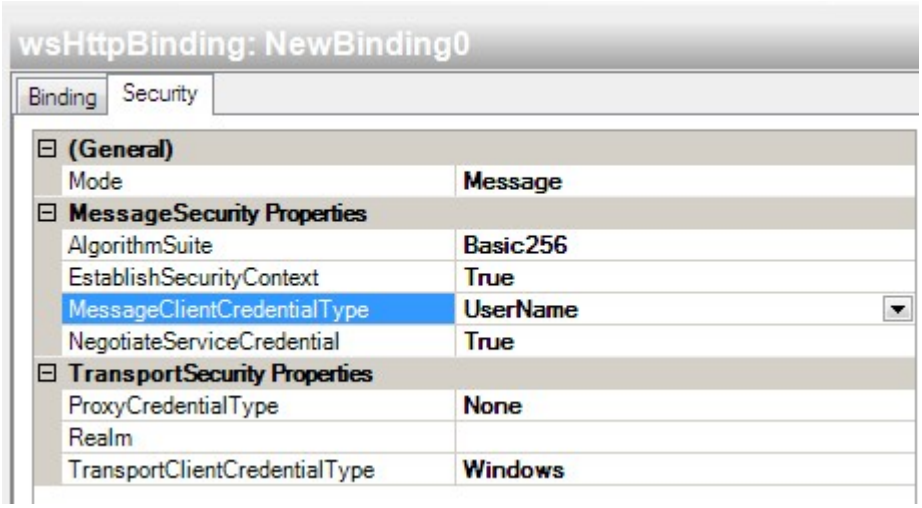
18. ábra
ServiceCredentials beállítása

A serviceCredentials menüt kibontva a serviceCertificate-et a következő értékekre kell állítani:



19. ábra
ServiceCertificate beállítása

Validátor osztályt wsHttpBinding alatt tudunk használni (vagy más biztonságos átvitelt támogató kötés alatt). A binding beállításainál a Security fülön MessageClientCredential-nak UserName-et kell megadni, módnak pedig üzenettitkosítást (Message).



20. ábra
WsHttpBinding beállítása validációhoz

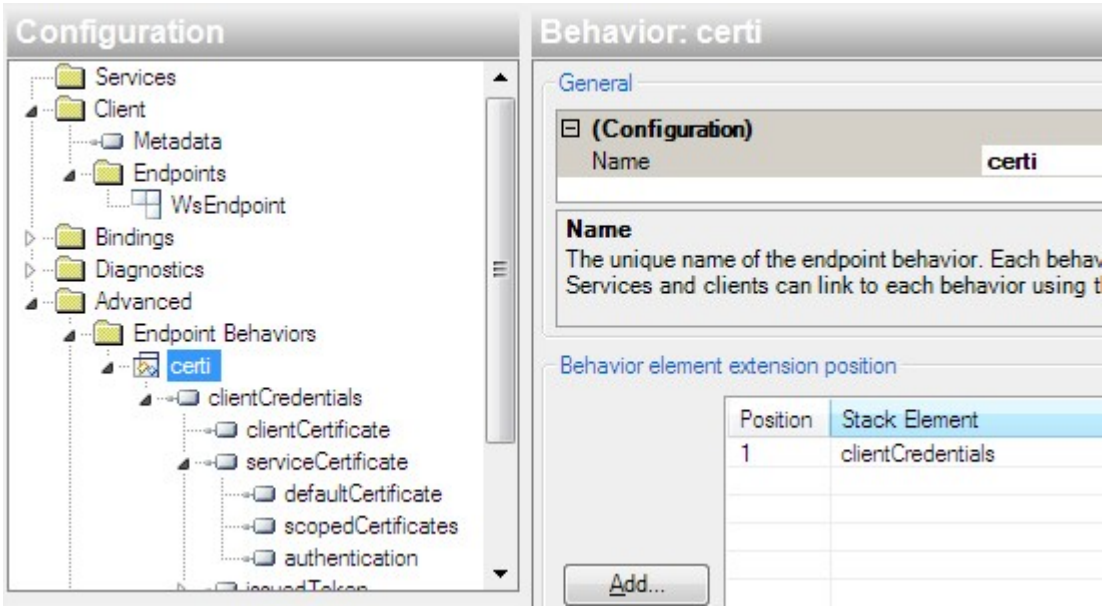
7.3. Autentikáció kliens oldal

A 7.2 fejezetben leírtak után a kliensben frissíteni kell a referenciát a szerverre. A kliens app.config-ját megnyitva az identity résznél a dns beállítást le kell cserélni a tanúsítvány CN nevére. Ez az én esetemben `wcftest`.

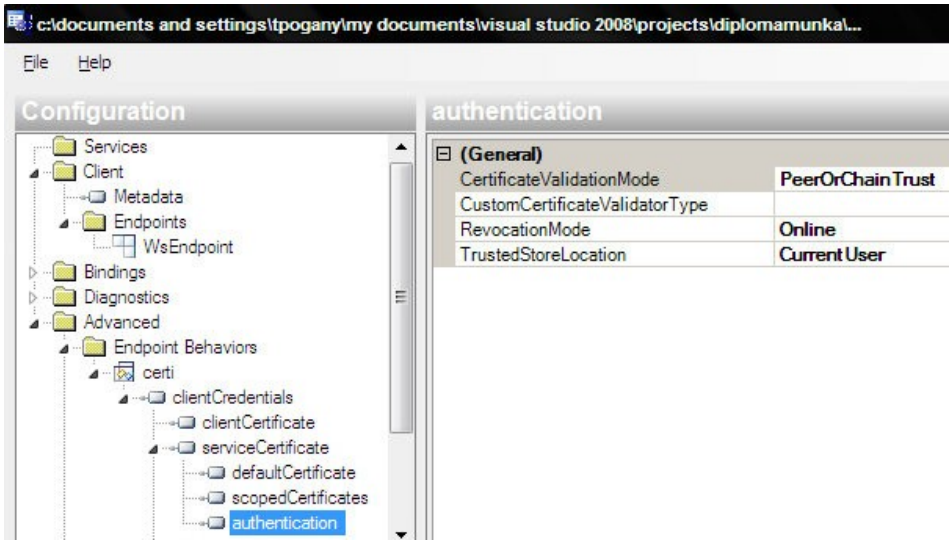
```
<identity>
  <dns value="wcftest" />
</identity>
```

23. kód
Identity beállítása

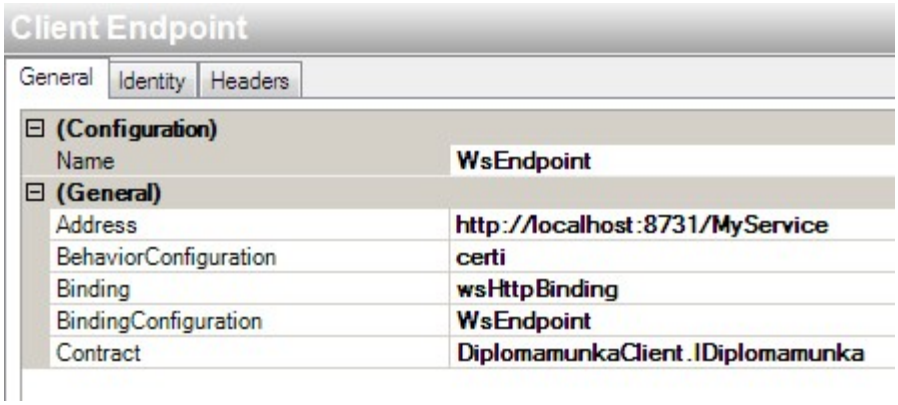
Ha nem magunk által generált tanúsítványt használunk, akkor a konfigurációs fájl szerkesztésével készen vagyunk. Egyébként meg kell nyitni a WCF konfigurációszerkesztőben, és az Advanced \ Endpoint Behaviours-ban létre kell hozni egy új konfigurációt, amihez hozzá kell adni clientCredentials részt (20. ábra), majd annak a serviceCertificate \ authentication ágában PeerOrChainTrust-ot beállítani (21. ábra). És természetesen be kell állítani az endpoint-ot, hogy ezt a behaviour-t használja (22. ábra).



21. ábra
Behaviour megadása



22. ábra
PeerOrChainTrust megadása



23. ábra
Végpont beállítása

Meg kell adnunk a kódban a felhasználói nevet és a jelszót, amivel a szerverre kívánunk csatlakozni. Ha ezek nem megfelelőek, akkor kivételt kapunk.

```
DiplomamunkaClient.DiplomamunkaClient client = new
TestClient.DiplomamunkaClient.DiplomamunkaClient();
client.ClientCredentials.UserName.UserName = "diploma2";
client.ClientCredentials.UserName.Password = "12345";
client.Open();
Console.WriteLine(client.divide(5, 2));
client.Close();
```

24. kód
Felhasználói név és jelszó megadása

7.4. Autorizáció

Ha sikeresen authentikált a kliens, az authorizáció onnantól gyerekjáték. A `OperationContext.Current.ServiceSecurityContext.PrimaryIdentity.Name` sztringben elérjük a meghíváskor megadott felhasználói nevet, és így könnyedén korlátozható a metódusok meghívása, például az elejükön egy `if` vizsgálattal.

```
If(OperationContext.Current.ServiceSecurityContext.PrimaryIdentity.
Name.Equals("admin")
{
  ...
}
```

25. kód
Autorizáció példakód

8. Egyéb biztonsági kérdések

8.1. Tűzfalak

Vállalati környezetben tűzfalak megfelelő konfigurálásával korlátozhatjuk a webszolgáltatáshoz való hozzáférést. A webszolgáltatás valamilyen porton figyel, ehhez a porthoz való hozzáférést korlátozhatjuk bizonyos gépekre. Hátránya, hogy nehezen karbantartható, körülményes. Helyette inkább ajánlott autentikációt bevezetni a webszolgáltatásban, vagy a 8.2-es pontban ismertetett módszert alkalmazni.

8.2. Mex endpoint kikapcsolása

Kevésbé kézenfekvő, ám adott esetben hatékonyan alkalmazható eljárás. A szerver végpontjainak címét véletlen módon generált sztringekre állítjuk, legeneráltatjuk a kliensben a referencia XML-t, majd a szerveren kikapcsoljuk a mex végpontot. Így a szolgáltatás nem kerül propagálásra, csak az férhet hozzá, aki pontosan tudja hol van (ezért volt szükség a véletlen nevekre, hogy kitalálni se lehessen), vagyis birtokában van a legenerált referencia XML kliens oldalon. Egyéb átviteli titkosítást nem támogató binding-ok (pl. `basicHttpBinding`) esetén ez, illetve a tűzfalas technika alkalmazható csak.

8.3. Message security

A WS-Security protokollt követve titkosítható a SOAP üzenet. Előnye, hogy nem pont-pont kapcsolat (közbeékelt harmadik fél) esetén is működik, titkosítható az üzenet egy része is, többféle transzport típussal (TCP[18], named pipe) működik. Hátrány a lassúsága, nem támogatja a stream-elést, illetve nem mindenhol elérhető.

18 Transmission Control Protocol

8.4. Transport security

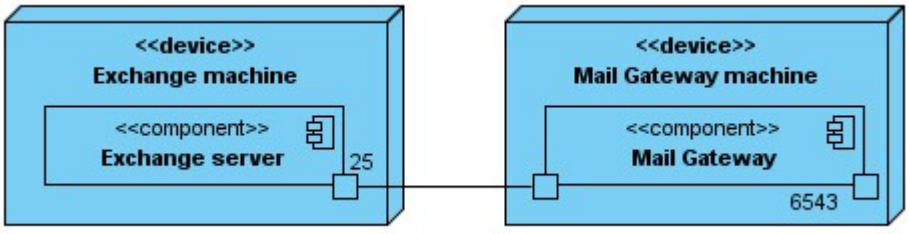
A transzport rétegben biztosít biztonságos átvitelt, HTTP (`wsHttpBindig`) esetén ez HTTPS, TCP (`netTcpBinding`) esetén SSL[19] a TCP felett. SSL tanúsítvány kell a használatához, nem működik közbeékelt harmadik fél esetén, viszont gyors, és támogatja a stream-elést. Sokszor együtt használják a message security-val.

19 Secure Socket Layer

9. Egy konkrét példa: Mail Gateway

9.1. Specifikáció

X.Y cég szoftverfejlesztéssel foglalkozik. Egyes termékeikhez email funkcionalitást kívánnak használni. Levelezőszervernek Microsoft Exchange-et használnak, melyet nem kívánnak megnyitni a külvilág felé. Elhatározzák, hogy a szolgáltatást WCF webszolgáltatással valósítják meg. Az Exchange szervert csak a webszolgáltatást futtató szerver felé nyitják meg, a külvilág pedig a webszolgáltatással fog kommunikálni. Természetesen autentikációt is bevezetnek, hogy csak az általuk írt szoftverek tudjanak emailt küldeni.



24. ábra
Architektúra

9.2. WCF interfész

A szolgáltatáshoz egyetlen sendMail metódusra lesz szükség. A metódus paraméterül kapja az email kódolását (pl. ISO-8859-2), az Exchange szerver címét (a szolgáltatást felkészítjük arra, hogy bármilyen szervert tudjon használni, amit elér), a használandó portot (általában 25), html formátumban van-e a levél, a feladó email címét, a címzett email címét, a levél tárgyát és a levél törzsét.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
namespace MailService
{
  [ServiceContract]
  public interface ImailService
  {
     [OperationContract]
     string SendMail(string encoding, string server, int port,
     bool htmlmail, string sender, string recipient,string subject,
     string body);
  }
}
```

26. kód
A WCF interfész

9.3. A WCF osztály

A WCF osztályban nem foglalkozok a `FaultContract` lehetőségeivel, a kód vagy „success” eredménnyel fog visszatérni, vagy a hiba szövegével, így a kliensben nem kell a különböző kivételekkel foglalkozni.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
using System.Net.Sockets;
using System.IO;
namespace MailService
{
  public class SMailService : ImailService
  {
     //A négy általunk figyelt szerver válaszkód
     private enum SMTPResponse : int {
        CONNECT_SUCCESS = 220,
        GENERIC_SUCCESS = 250,
        DATA_SUCCESS = 354,
        QUIT_SUCCESS = 221
     }
     //Újsor karakter
```

```
private string CRLF = "\r\n";
//Maximum ennyi másodpercet várunk a szerver válaszára
private int timeout = 10;

public string SendMail(string encoding, string server,
int port, bool htmlmail, string sender, string recipient,
string subject, string body)
{
  try
  {
    //TCP kliens nyitása a szerver felé
    TcpClient SmtpServ = new TcpClient();
    SmtpServ.Connect(server, port);
    NetworkStream nstSMTP = SmtpServ.GetStream();
    //Író és olvasó stream-ek létrehozása
    StreamWriter stwSMTP = new StreamWriter(nstSMTP);
    StreamReader strSMTP = new StreamReader(nstSMTP);
    //Kész-e a szerver SMTP kérést fogadni
    if (!Check_Response(nstSMTP, strSMTP,
    SMTPResponse.CONNECT_SUCCESS))
      throw new FaultException<MailError>
      (new MailError { Text = "Server is not ready" });

    //HELO küldése
    Senddata(stwSMTP, "HELO server");
    //Fogadta-e a szerver a HELO-t
    if (!Check_Response(nstSMTP, strSMTP,
    SMTPResponse.GENERIC_SUCCESS))
      throw new FaultException<MailError>
      (new MailError {
        Text = "Server did not respond 250 after HELO"
      });

    //Feladó beállítása
    Senddata(stwSMTP, string.Format("MAIL From: <{0}>",
    sender));
    //Rendben volt-e a feladó beállítása
    if (!Check_Response(nstSMTP, strSMTP,
    SMTPResponse.GENERIC_SUCCESS))
      throw new FaultException<MailError>
      (new MailError {
        Text = "Server did not respond 250 after MAIL FROM"
      });

    //Címzett beállítása
    Senddata(stwSMTP, string.Format("RCPT TO: <{0}>",
    recipient));
    //Rendben volt-e a címzett beállítása
    if (!Check_Response(nstSMTP, strSMTP,
```

```
        SMTPResponse.GENERIC_SUCCESS))
          throw new FaultException<MailError>
          (new MailError {
            Text = "Server did not respond 250 after RCPT TO"
          });

        //Levéltörzs küldésének indítása
        Senddata(stwSMTP, "DATA");
        //Kész-e a szerver fogadni a levél törzsét
        if (!Check_Response(nstSMTP, strSMTP,
        SMTPResponse.DATA_SUCCESS))
          throw new FaultException<MailError>
          (new MailError {
            Text = "Server did not respond 354 after DATA"
          });
        //Levél törzsének küldése, fejlécek
        Senddata(stwSMTP, "From: " + sender);
        Senddata(stwSMTP, "To: " + recipient);
        Senddata(stwSMTP, string.Format("Subject: =?{0}?B?{1}?=",
        encoding, base64Encode(subject)));
        string MsgBody = body;
        if (htmlmail)
          Senddata(stwSMTP, "Content-Type: text/html; charset=" +
          encoding + ";");
        else
          Senddata(stwSMTP, "Content-Type: text/plain; charset=" +
          encoding + ";");
        Senddata(stwSMTP, "Content-Transfer-Encoding:base64"+CRLF);
        //Base64 kódolt törzs küldése
        Senddata(stwSMTP, base64Encode(MsgBody));
        //Adatküldés vége
        Senddata(stwSMTP, ".");
        //Rendben volt-e a törzs fogadása
        if (!Check_Response(nstSMTP, strSMTP,
        SMTPResponse.GENERIC_SUCCESS))
          throw new FaultException<MailError>
          (new MailError {
            Text = "Server did not respond 250 after sending DATA"
          });

        //Kapcsolat bontása a szerverrel
        Senddata(stwSMTP, "QUIT");
        //A szerver bontási üzenete, nem érdekel minket a tartalma
        Check_Response(nstSMTP, strSMTP,
        SMTPResponse.QUIT_SUCCESS);

        //Kapcsolatok lezárása
        stwSMTP.Close();
        strSMTP.Close();
```

```
      nstSMTP.Close();
      SmtpServ.Close();
    }
    catch (FaultException<MailError> ex)
      {
        return ex.Detail.Text;
      }
      catch (FaultException<GeneralError> ex)
      {
        return ex.Detail.Text;
      }
      catch (IOException ex)
      {
        return "IO error, message is: " + ex.Message;
      }
      catch (Exception ex)
      {
        return "General error, message is: " + ex.Message;
      }
      return "success";
    }
  }

  //Szöveg base64 kódolása küldéshez
  private string base64Encode(string data)
  {
    try
    {
      byte[] encData_byte = new byte[data.Length];
      encData_byte = System.Text.Encoding.Default.GetBytes(data);
      string encodedData = Convert.ToBase64String(encData_byte);
      return encodedData;
    }
    catch (Exception ex)
    {
      throw new FaultException<GeneralError>
      (new GeneralError {
        Text = "Error in base64Encode, message is: " + ex.Message
      });
    }
  }

  //Szerver válaszának fogadása, timeout funkcióval
  private bool Check_Response(NetworkStream n, StreamReader s,
  SMTPResponse response_expected)
  {
    DateTime timeoutCheck = DateTime.Now;
    TimeSpan tsp = DateTime.Now - timeoutCheck;
    while (tsp.Seconds < timeout)
```

```
      {
        if (!n.DataAvailable)
        {
          tsp = DateTime.Now - timeoutCheck;
          continue;
        }
        string response = s.ReadLine();
        string expected = ((int)response_expected).ToString();
        if (response.Substring(0,expected.Length).Equals(expected))
          return true;
        return false;
      }
      return false;
    }

    //Adat küldése a szervernek
    private void Senddata(StreamWriter s, string msg)
    {
      s.WriteLine(msg);
      s.Flush();
    }
  }
  public class MailError
  {
    public string Text { get; set; }
  }
  public class GeneralError
  {
    public string Text { get; set; }
  }
}
```

27. kód
A WCF osztály

9.4. A webszolgáltatás összeállítása

A 4.3 fejezetben ismertetett módon létre kell hozni a végpontokat a szolgáltatáshoz, majd a 7. fejezetben ismertetett módszerekkel autentikációt készíteni. A webszolgáltatáshoz hozzáférés természetesen tovább korlátozható tűzfalak használatával. Választani kell egy host alkalmazás-típust, kézenfekvő a Windows Service használata. A kliens alkalmazást a 4.6 és a 7.3 fejezetek szerint kell elkészíteni, és a szolgáltatás használható.

9.5. A kód használata

A webszolgáltatás elég általános ahhoz, hogy tetszőleges típusú SMTP szerverrel képes legyen működni, bármelyik porton. Sok erőforrást nem igényel, ellenben figyelni kell arra, hogy esetenként nagyszámú socket-et foglal, hiszen ő maga is nyit az SMTP szerver felé, illetve a rákapcsolódó kliensek is. Nem érdemes olyan szerverre telepíteni, ahol nagyszámú helyi socket-et igénylő szolgáltatás fut (pl. MSSQL), ilyenkor a szolgáltatás gyakran le fog állni, szabad socket hiányában.

A webszolgáltatás kiválóan használható különböző monitorozó alkalmazásokhoz. Alkalmas arra, hogy monitorozó szolgáltatásokat email küldési funkcióval egészítse ki. Így a rendszergazdának nem kell az operátor konzol előtt ülnie, hanem automatikusan kaphat email értesítéseket ha egy szerver leáll, kritikus szint alá csökken a szabad memóriája, vagy valaki éppen megpróbálja feltörni. Használható hibajegyek automatikus generálására, akár olyan szinten is, ha a hibát megadott időkereten belül nem javítják ki / javul meg magától, akkor értesítse a megfelelő személyeket. Felhasználható olyan értesítések generálására, amire a levelezőszerver alapból nem képes (pl. naptárbejegyzések száma elért egy bizonyos limitet).

10. Összefoglalás

Webszolgáltatások informatikai léptékben mérve régen léteznek. Kezdetben igencsak gyerekcipőben jártak, sok megvalósításuk létezett, és az olyan fontos kérdésekben, mint a biztonság, sokáig nem létezett szabvány vagy ajánlás. A Microsoft ezt a meglévő sokszínűséget próbálta 2006-ban standardizálni a WCF kiadásával. Szemléletmódjuknak megfelelően sikerült egy könnyen használható, gyorsan megtanulható eszközt adniuk a programozók kezébe. Hiszen a fejlesztők meglévő .NET tudásukat használva írhattak webszolgáltatást, csupán néhány új fogalmat és eszközt kellett megismerniük. Ügyeltek arra is, hogy a hosztolás a lehető legegyszerűbb legyen, így mind az IIS, mind a létrehozott `ServiceHost` osztály transzparens módon tudja hosztolni a megírt szolgáltatást. A konfiguráció egy XML fájl segítségével történik, így biztosítva a kényelmet, hogy ha más jellegű átvitelre van szükség, akkor a meglévő webalkalmazást ne kelljen újrafordítani.

A Microsoft piaci helyzetét kihasználva hamar elérte, hogy a többi nyelv is támogassa, és használja a WCF-et. Számos standard kötést építettek be, melyeket ma a Java-tól a PHP-ig szinte minden magasszintű nyelv támogat. Az ezek közti kommunikáció, még az objektumok küldése is viszonylag jól megoldott. Ügyeltek ugyanakkor arra, hogy kedvezzenek a saját platformjukon fejlesztőknek, így olyan kötéseket is beleépítettek a WCF-be, amelyek csak .NET-es kliens-szerver felállásban működnek, és adott körülmények között sokkal gyorsabbá teszik a kommunikációt. Biztonság terén amit lehetett beleintegráltak, így a fejlesztő a végletekig testre szabhatja alkalmazását.

Összességében elmondhatjuk, hogy bár a webszolgáltatás még korántsem egy elterjedt, és széles körben használt technológia [24] [25] (ahogy azt létrehozásakor elképzelték), de a WCF-hez hasonló megoldásoknak köszönhetően nagyon hamar azzá válhat.

11. Irodalomjegyzék

[1] Juval Lowy: Programming WCF. O'Reilly Media, 2007.

[2] Scott Klein: Professional WCF Programming. Wiley India, 2007.

[3] Craig McMurty [et al.]: Windows Communication Foundation 3.5 Unleashed. Sams, 2008.

[4] Bruce Johnson, Peter Madziak, Sara Morgan: MCTS Self-Paced Training Kit (Exam 70-503): Microsoft .NET Framework 3.5-Windows Communication Foundation: Microsoft .Net Framework 3.5 Windows Communication Foundation. Microsoft Press, 2008.

[5] Webszolgáltatás. In http://hu.wikipedia.org/wiki/Webszolg%C3%A1ltat%C3%A1s . Accessed 2010.01.10.

[6] Web service. In http://en.wikipedia.org/wiki/Web_service . Accessed 2010.01.10.

[7] XML-RPC Home Page. In http://www.xmlrpc.com/ . Accessed 2010.01.10.

[8] XML-RPC: In http://en.wikipedia.org/wiki/XML-RPC . Accessed 2010.01.10.

[9] XML-RPC: Együttműködés távoli szolgáltatásokkal. 2004.08.10. In http://weblabor.hu/cikkek/xmlrpc . Accessed 2010.01.10.

[10] SOAP Tutorial. In http://www.w3schools.com/soap/default.asp . Accessed 2010.01.10.

[11] SOAP: Együttműködés távoli szolgáltatásokkal. 2004.09.07. In http://weblabor.hu/cikkek/soap . Accessed 2010.01.10.

[12] SOAP. In http://en.wikipedia.org/wiki/SOAP . Accessed 2010.01.10.

[13] SOAP Data Types. In http://old.apisnetworks.com/soap-data-types.php . Accessed 2010.01.10.

[14] Webszolgáltatás-leíró nyelv. In http://hu.wikipedia.org/wiki/Webszolg%C3%A1ltat%C3%A1s-le%C3%ADr%C3%B3_nyelv . Accessed 2010.01.10.

[15] UDDI. In http://hu.wikipedia.org/wiki/UDDI . Accessed 2010.01.10.

[16] Universal Description Discovery and Integration. In http://en.wikipedia.org/wiki/Universal_Description_Discovery_and_Integration . Accessed 2010.01.10.

[17] WS-Security. In http://en.wikipedia.org/wiki/WS-Security . Accessed 2010.01.10.

[18] WS-Security. In https://sauron.inf.mit.bme.hu/Edu/IIM/Iim08.nsf/a141f6e0f01d3349c1256e4c005bc395/0ecffab7d0aa6a21c125744e003f0b4b/$FILE/WS-Security.pdf . Accessed 2010.01.10.

[19] Understanding WS-Security. 2002.10. In http://msdn.microsoft.com/en-us/library/ms977327.aspx . Accessed 2010.01.10.

[20] Turóczy Attila: Windows Communication Foundation - Elméleti háttér. 2008.05.13. In http://msportal.hu/blogs/turczy_attila/archive/2008/05/13/windows-communication-foundation-elm-233-leti-h-225-tt-233-r.aspx . Accessed 2010.01.10.

[21] Claus Konrad: WCF and Custom Exception Handling (custom exceptions). 2008.06.24. In http://blog.clauskonrad.net/2008/06/wcf-and-custom-exceptions.html . Accessed 2010.01.10.

[22] Tóth László: WCF service ASP.Net hitelesítéssel. 2008.06.27. In http://tudastar.netacademia.net/default.aspx?upid=22808 . Accessed 2010.01.10.

[23] Budai Péter: Webszolgáltatások a vadonban. In http://download.microsoft.com/download/8/f/7/8f75bdd7-a0f9-4f53-a0ad-9d9aae86e21e/28-29.pdf . Accessed 2010.01.10.

[24] Web Services Jobs. In http://www.itjobswatch.co.uk/jobs/uk/webservices.do . Accessed 2010.01.10.

[25] Web service Job Trends. In http://www.indeed.com/jobtrends?q=web+service&l= . Accessed 2010.01.10.