Emiatt a .NET 3.0 előtt nehéz volt elosztott API-kat egyszerűen csatlakoztatni (plug and play) anélkül, hogy tekintélyes mennyiségű egyedi infrastruktúrát ne kellett volna létrehozni. Ha például .NET-remoting-API-kal készítjük a rendszerünket, és később úgy döntünk, hogy az XML-webszolgáltatások megfelelőbbek lennének, újra kell írni a kódalapot.

A WCF a .NET 3.0-ban bevezetett egy elosztott eszközrendszert, amely integrálja ezeket a korábban független elosztott technológiákat egy áramvonalas API-ban, amelyet elsősorban a `System.ServiceModel` névtér képvisel. A WCF révén a szolgáltatásokat a legkülönfélébb technikák segítségével elérhetővé lehet tenni a hívók számára. Ha például olyan házon belüli alkalmazást készítünk, ahol minden kapcsolódó számítógép Windows-alapú, többféle TCP protokoll használata biztosíthatja a lehető legjobb teljesítményt. Ugyanezt a szolgáltatást XML-webszolgáltatás-alapú protokollal elérhetővé tehetjük úgy, hogy külső hívók is birtokba vehessék a funkcióit a programozási nyelvtől vagy az operációs rendszertől függetlenül.

Mivel a WCF lehetővé teszi a feladatnak megfelelő protokoll kiválasztását (egy közös programozási modellel), elég könnyű lesz az elosztott alkalmazásunk alapjául szolgáló összeköttetéseket beállítani. Legtöbb esetben ezt a kliens/szerver szoftver újraindítása vagy újratelepítése nélkül meg lehet tenni, ugyanis a „piszkos" részleteket gyakran az alkalmazás konfigurációs fájljai veszik át (úgy, mint a régebbi .NET-remoting-API-knál).

A WCF-funkciók áttekintése

A különböző API-k együttműködése és integrációja a WCF-nek csak két (nagyon fontos) aspektusa. Ezenkívül a WCF gazdag szoftveranyagot ad az elérhető remotingtechnológiák kiegészítésére. Nézzük meg a következő fő WCF-jellegzetességeket:

- Erősen típusos *és* típus nélküli üzenetek támogatása jellemzi. Ezzel a megközelítéssel a .NET-alkalmazások hatékonyan megoszthatják egyedi típusaikat, az egyéb platformokon készült szoftverek (pl. J2EE) feldolgozhatják a gyengén tipizált XML-folyamokat.
- A többféle *kötés* (nyers HTTP, TCP, MSMQ és named pipe-ok) támogatása lehetővé teszi, hogy a legmegfelelőbb összeköttetést válasszuk üzenetek ide-oda szállítására.

- A legújabb és a legjobb webszolgáltatások specifikációjának (WS-*) támogatása.
- Teljesen integrált biztonsági modell, amely a Win32/.NET natív biztonsági protokolljait és sok más semleges, webszolgáltatási szabványra épülő biztonsági módszert magában foglal.
- Munkamenetszerű állapotkezelési módszerek és egyirányú, állapot nélküli üzenetek támogatása.

Bármilyen lenyűgöző is ez a lista, valójában csak a WCF-nyújtotta funkcionalitás felszínét mutatja. A WCF nyomkövető és naplózó eszközöket, teljesítménymérőket, „közzétesz és előfizet" (publish/subscribe) eseménymodellt, tranzakciós támogatást és más egyéb funkciókat is tartalmaz.

A szolgáltatásorientált architektúra áttekintése

A WCF egy másik előnye, hogy a *szolgáltatásorientált architektúra* (SOA) által megalapozott tervezési elveken alapul. A SOA felkapott kifejezés az iparban, és mint a legtöbb ilyen, többféleképpen definiálható. A SOA egyszerűen egy olyan módszer elosztott rendszerek építésére, ahol több autonóm *szolgáltatás* működik együtt úgy, hogy *üzeneteket* küldenek határaikon túlra (hálózatba kötött gépek között vagy csak két folyamat között egyazon gépen belül) jól definiált *interfészek* segítségével.

A WCF világában ezek a „jól definiált interfészek" általában CLR-interfésztípusokkal készülnek (lásd az előző kötet 9. fejezetét). Általánosabban egy szolgáltatás interfésze egyszerűen csak leírja a külső hívók által hívható tagok halmazát.

A WCF tervezése során a WCF-csoport felállította a SOA tervezési módszer négy alapelvét. Bár ezeket az alapelveket automatikusan tiszteletben tartjuk már azzal is, hogy WCF-alkalmazást készítünk, a SOA négy fő tervezési szabályának megértése segíthet abban, hogy a WCF-et kellő megvilágításba helyezzük. A következő részek egy-egy alapelv rövid áttekintését tartalmazzák.

1. alapelv: A határok explicitek

Ez az alapelv megerősíti azt, hogy a WCF-szolgáltatás funkcionalitása jól definiált interfészeken keresztül jelenik meg (pl. minden tagjának, a paramétereinek és a visszatérési értékének leírása). A külső hívó egyedül az interfészen keresztül tud a WCF-szolgáltatással kommunikálni, ráadásul a külső hívó nem tud az alapul szolgáló implementációs részletekről.

2. alapelv: A szolgáltatások autonómok

Ha „autonóm" entitásként beszélünk a szolgáltatásokról, akkor arra utalunk, hogy az adott WCF-szolgáltatás (amennyire ez lehetséges) magányos sziget. Az autonóm szolgáltatás legyen független a verziótól és telepítési kérdésektől. Az alapelv támogatásának érdekében megint az interfészalapú programozás kulcskérdéséhez térünk vissza. Ha egyszer az interfész működésbe lép, már nem szabad rajta változtatni (különben fennáll a kockázat, hogy létező kliensek munkáját tesszük lehetetlenné). Ha szeretnénk kibővíteni a WCF-szolgáltatás funkcionalitását, egyszerűen a kívánt funkciót modellező új interfészeket kell készíteni.

3. alapelv: A szolgáltatások szerződésen keresztül és nem implementáción keresztül kommunikálnak

A harmadik alapelv is az interfészalapú programozás egyik melléktermékе, mert a WCF-szolgáltatás implementációs részletei (milyen nyelven készült, hogyan valósítja meg a feladatait) nem érdeklik a külső hívót. A WCF-ügyfelek kizárólag a kívülről elérhető nyilvános interfészeiken keresztül kommunikálnak a szolgáltatásokkal. Továbbá, ha egy szolgáltatás interfésztagjai egyedi összetett típusokat tesznek elérhetővé, ezeket egy *adatszerződésben* részletesen le kell írni, hogy a hívók biztosan leképezhessék a tartalmat egy meghatározott adatszerkezetbe.

4. alapelv: A szolgáltatás kompatibilitása házirenden alapul

Mivel a CLR-interfészek erősen típusos szerződéseket tartalmaznak minden WCF-ügyfélhez (és ezekkel kapcsolódó, a választott kötésen alapuló WSDL-dokumentumot lehet generálni), fontos azt hangsúlyozni, hogy az interfészek vagy a WSDL egyedül nem elég kifejezőek ahhoz, hogy részletesen kifejtsük, mire képes a szolgáltatás.

Ezek alapján a SOA segítségével „házirendeket" definiálhatunk, amelyek tovább minősítik a szolgáltatás szemantikáját (pl. az elvárt biztonsági elvárásokat, amelyekkel a szolgáltatással kommunikálunk). Ezekkel az házirendekkel tulajdonképpen elválaszthatjuk a szolgáltatás (az elérhető interfészek) alacsony szintű szintaktikai leírását a szolgáltatás hívásának módját leíró szemantikai részletektől.

WCF: A lényeg

Ez a kis összefoglalás bemutatta, hogy a WCF a legjobb módszer az elosztott alkalmazások készítésére a .NET 3.0 és újabb verzióiban. Akár házon belüli alkalmazást hozunk létre TCP protokollal, vagy adatokat mozgatunk programok között ugyanazon a gépen named pipe-okkal, vagy adatokat teszünk elérhetővé a külvilág számára webszolgáltatás-protokollokkal, a WCF az ajánlott API ennek a megvalósítására.

Ez nem azt jelenti, hogy új fejlesztéseknél nem használhatjuk az eredeti .NET elosztott rendszerekhez tartozó névtereket (`System.Runtime.Remoting`, `System.Messaging`, `System.EnterpriseServices`, `System.Web.Services` stb.). Tulajdonképpen, néha (különösen, ha COM+-objektumokat kell létrehozni), ez még kötelező is. Mindenesetre, ha már korábbi projektekben használtuk ezeket az API-kat, a WCF megismerése nem fog gondot okozni. Csakúgy, mint a korábbi technológiák, a WCF nagy hasznát veszi az XML-alapú konfigurációs fájloknak, a .NET-attribútumoknak és a proxygeneráló eszközöknek.

A következőkben rátérünk a konkrét WCF-alkalmazások készítésére. A WCF teljes ismertetése egy egész könyvet igényelne, mivel minden támogatott szolgáltatásnak (MSMQ, COM+, P2P, named pipes stb.) önmagában egy teljes fejezetet szentelhetnénk. Megismerjük a WCF-programok készítésének teljes folyamatát TCP- és HTTP-alapú (pl. webszolgáltatás) protokollok használatával is. Ennek segítségével már egyszerűen továbbléphetünk a jövőben.

Az alapvető WCF-szerelvények

A WCF programozási alapja a globális szerelvénytárba (GAC) telepített .NET-szerelvények halmaza. A 25.1. táblázat leírja azon alapvető WCF-szerelvények szerepét, amelyekre nagyjából minden WCF-alkalmazásban szükség lesz.

Szerelvény	Jelentés
`System.Runtime.Serialization.dll`	Definiálja azokat a névtereket és típusokat, amelyekkel a WCF keretrendszerben objektumokat sorosíthatunk és állíthatunk vissza.
`System.ServiceModel.dll`	Alapvető szerelvény, amely bármilyen WCF-alkalmazás készítéséhez szükséges típusokat tartalmazza.

25.1. táblázat: *Alapvető WCF-szerelvények*

Ez a két szerelvény sok új névteret és típust definiál. Míg a részletes referenciát a .NET Framework 3.5 SDK dokumentációjában találjuk, a 25.2. táblázat leírja a legfontosabb névterek szerepét.

Névtér	Jelentés
`System.Runtime.Serialization`	Több olyan típust definiál, amelyek szabályozzák az adatok sorosítását és a visszaállítását a WCF keretrendszeren belül.
`System.ServiceModel`	Az elsődleges WCF-névtér, amely definiálja a kötési és hosztolási típusokat, valamint az alapvető biztonsági és tranzakciós típusokat.
`System.ServiceModel.Configuration`	Számos olyan típust definiál, amely programozott elérést biztosít a WCF konfigurációs fájlokhoz.
`System.ServiceModel.Description`	Olyan típusokat definiál, amelyek a WCF konfigurációs fájlokban definiált címekhez, kötésekhez és szerződésekhez biztosítanak objektummodellt.
`System.ServiceModel.MsmqIntegration`	Az MSMQ-szolgáltatásokkal integrálható típusokat tartalmazza.
`System.ServiceModel.Security`	Olyan típusokat definiál, amelyek a WCF biztonsági rétegeinek beállításait szabályozzák.

25.2. táblázat: *Alapvető WCF-névterek*

NÉHÁNY SZÓ A CARDSPACE-RŐL

A `System.ServiceModel.dll`-en és a `System.Runtime.Serialization.dll`-en kívül a WCF egy harmadik WCF-szerelvényt is biztosít: a `System.IdentityModel.dll`-t. Itt számos olyan egyéb névteret és típust találunk, amelyek a WCF CardSpace API-t támogatják. Ez a technológia lehetővé teszi digitális azonosítók létrehozását és kezelését a WCF-alkalmazáson belül. Lényegében a CardSpace API egységes programozási modellt biztosít a különböző biztonsági részletek kezelésére a WCF-alkalmazásokban, mint például a hívó azonosítója, felhasználó azonosító/hitelesítő szolgáltatások és egyebek.

Ebben a kiadásban nem lesz szó a CardSpace API-ról, így ha további részletekre van szükségünk, forduljunk a .NET Framework 3.5 SDK dokumentációjához.

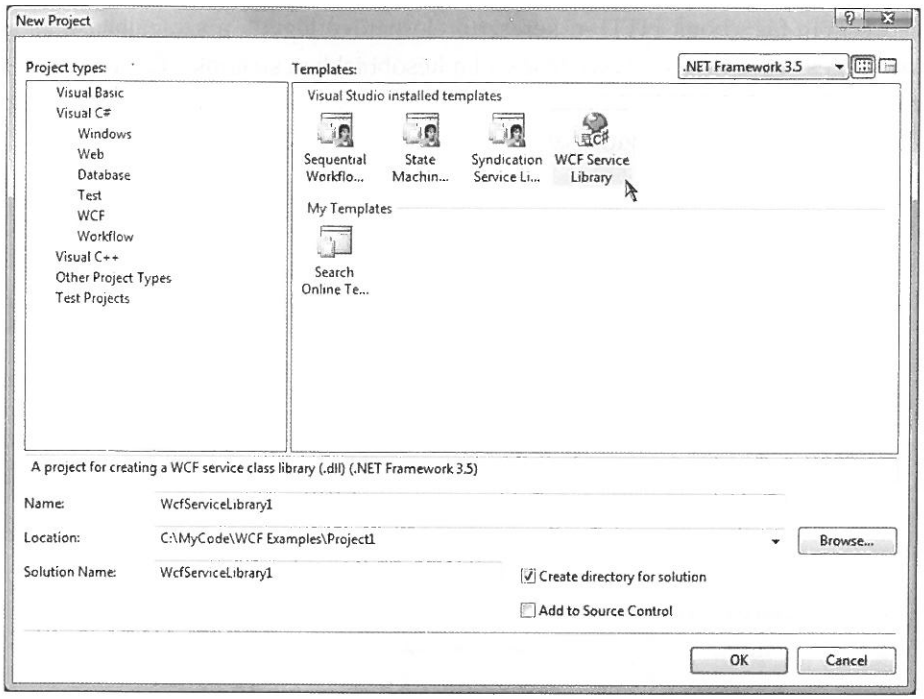
A Visual Studio WCF projektsablonok

A WCF-alkalmazást általában három kapcsolódó szerelvény képviseli, amelyek egyike egy `*.dll`, s ez azokat a típusokat tartalmazza, amelyekkel a külső hívók kommunikálhatnak (más szóval ez maga a WCF-szolgáltatás). Ha WCF-szolgáltatást akarunk készíteni, tökéletesen elfogadható kiindulási pont egy szabványos osztálykönyvtárprojekt-sablon kiválasztása (lásd az első kötet 15. fejezetét), utána pedig manuálisan hivatkozhatunk a WCF-szerelvényekre.

De készíthetünk új WCF-szolgáltatást a Visual Studio 2008 WCF Service Library projektsablon kiválasztásával is (lásd a 25.2. ábrát). Ez a projekttípus automatikusan beállítja a referenciákat a szükséges WCF-szerelvényekre, viszont elég nagy mennyiségű „kezdeti kódot" generál, amelyet nagy valószínűséggel később egyszerűen ki fogunk törölni.

A WCF Service Library projektsablon egyike előnye, hogy tartalmazza az `App.config` fájlt is, ez pedig furcsának tűnhet, hiszen .NET `*.dll` fájlt készítünk, és nem .NET `*.exe` fájlt. Ám ez a fájl nagyon hasznos hibakereséskor vagy a WCF Service Library projekt futtatásakor, mert a Visual Studio IDE automatikusan elindítja a WCF Test Client alkalmazást. A program (`WcfTestClient.exe`) kiolvassa a beállításokat az `App.config` fájlból, hogy tesztelés céljából hosztolhassa a szolgáltatásunkat. A WCF tesztelőkliensével a későbbiekben még foglalkozunk.

Megjegyzés A WCF Service Library projekt `App.config` fájlja hasznos abból a szempontból is, hogy megmutatja a WCF-hoszt-alkalmazás konfigurálásához szükséges alapbeállításokat. Igazából ennek a kódnak nagy részét bemásolhatjuk a saját hosztunk konfigurációs fájljába.



25.2. ábra: *A Visual Studio 2008 WCF Service Library projektsablonja*

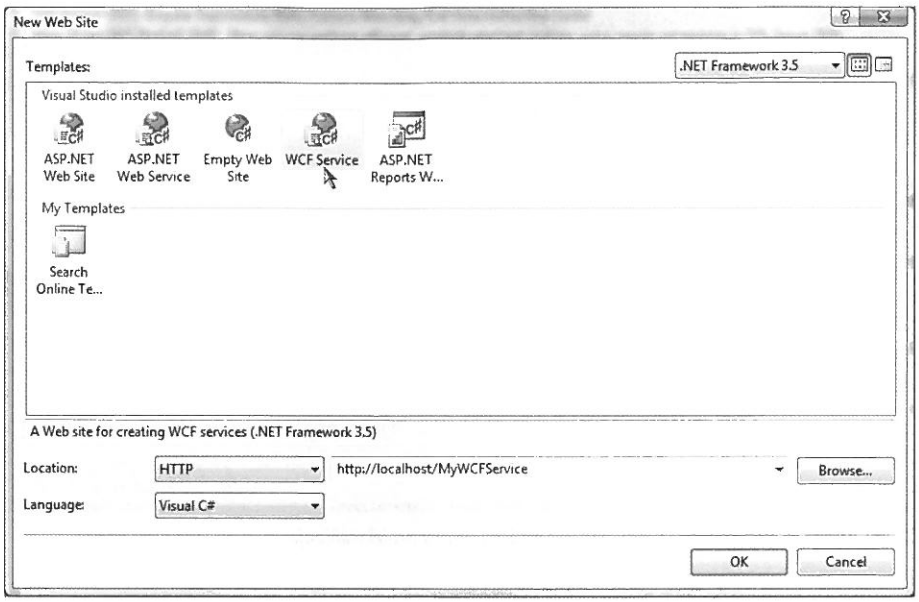
Az alap WCF Service Library sablonon kívül a New Project párbeszédablak WCF projekt kategóriája még két WCF-könyvtárprojektet definiál, amelyek a WCF-szolgáltatásba integrálják a Windows Workflow Foundation funkcionalitását, valamint egy sablont az RSS-könyvtárak készítéséhez (amelyek szintén láthatóak a 25.2. ábrán). A következő fejezetben szó lesz a Windows Workflow Foundation technológiáról, így most a WCF-projektsablonokat figyelmen kívül hagyjuk.

A WCF Service Website projektsablon

Még egy Visual Studio 2008 WCF-központú projektsablon található a New Web Site párbeszédablakban a File > New > Web Site menüpont alatt (lásd a 25.3. ábrát).

Ez a WCF Service (szolgáltatás) projektsablon akkor hasznos, ha már a kezdetektől tudjuk, hogy a WCF-szolgáltatásunk inkább webszolgáltatás-alapú protokollokat használ, mint named pipe-okat. Ez az opció automatikusan elkészít egy új IIS-beli virtuális könyvtárat a WCF-programfájlok számára, egy megfelelő

`web.config` fájlt, hogy HTTP-n keresztül elérhetővé tegyük a szolgáltatást, és a szükséges `*.svc` fájlt (az `*.svc` fájlokról a későbbiekben szólunk). Erről az oldalról nézve a webalapú WCF Service projekt egyszerűen időmegtakarítás, mivel az IDE automatikusan elkészíti a szükséges IIS-infrastruktúrát.



25.3. ábra: *A Visual Studio 2008 webalapú WCF Service projektsablonja*

Ezzel ellentétben, ha a WCF Service Library opcióval készítünk új WCF-szolgáltatást, akkor azt többféleképpen is hosztolhatjuk (egyedi hoszt, Windows-szolgáltatás, manuálisan készített IIS-beli virtuális könyvtár stb.). Ez az opció akkor célszerűbb, ha egyedi hosztot szeretnénk készíteni a WCF-szolgáltatáshoz.

A WCF-alkalmazás alapösszeállítása

A WCF elosztott rendszerhez általában három kapcsolódó szerelvényt készítünk:

- *A WCF Service (WCF szolgáltatás) szerelvény.* Ez a `*.dll` tartalmazza a külső hívók számára elérhető funkcionalitást képviselő osztályokat és interfészeket.

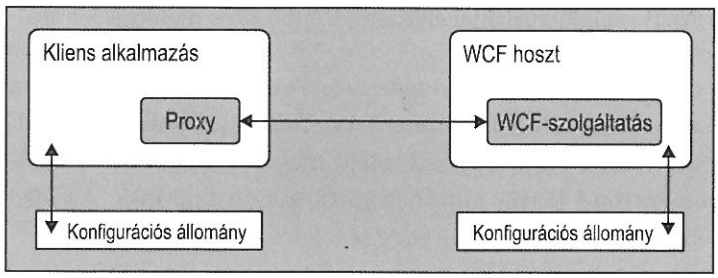
- *A WCF Service hoszt:* Ez a programmodul hosztolja a WCF-szolgáltatás szerelvényét.
- *A WCF-ügyfél:* Ez az alkalmazás hozzáfér a szolgáltatás funkcionalitásához egy közbeépülő proxyn keresztül.

A WCF-szolgáltatás szerelvénye olyan .NET-osztálykönyvtár, amely WCF-szerződéseket és az implementációjukat foglalja magában. Az egyik fő különbség az, hogy az interfészszerződéseket felvérteztük különböző olyan attribútumokkal, amelyek adattípusok reprezentációját vezérlik, például azt, hogy a WCF-futtatókörnyezet hogyan működik együtt az elérhető típusokkal.

A második szerelvény, a WCF-hoszt, bármilyen .NET végrehajtható fájl lehet. A WCF segítségével a szolgáltatások bármilyen típusú alkalmazásból elérhetővé tehetők (Windows Forms-alkalmazások, Windows-szolgáltatások, WCF-alkalmazások stb.). Ha egyedi hosztot készítünk, használjuk a `ServiceHost` típust és a hozzá tartozó `*.config` fájlt, amely a használni kívánt kiszolgálóoldali összeköttetések részleteit tartalmazza. Ha azonban IIS-t használunk WCF-szolgáltatásunk hosztjaként, nem kell programozottan egyedi hosztot készítenünk, mivel az IIS a háttérben a `ServiceHost` típust használja.

Megjegyzés A Vista-specifikus Windows Activation Service (WAS – Windows aktivációs szerver) segítségével is lehet a WCF-szolgáltatást hosztolni. További részletekért forduljunk a .NET Framework 3.5 SDK dokumentációjához.

Az utolsó szerelvény azt a klienst jelképezi, amely a WCF-szolgáltatást hívja. Ez a kliens bármilyen típusú .NET-alkalmazás lehet. A hoszthoz hasonlóan a kliensalkalmazások általában a kliensoldali `*.config` fájlt használják, amely a kliensoldali összeköttetéseket definiálja.



25.4. ábra: *Egy tipikus WCF-alkalmazás szerkezetének áttekintése*

A 25.4. ábra (nagy vonalakban) mutatja a három összekapcsolódó WCF-szerelvény kapcsolatát. Feltételezésünknek megfelelően a háttérben több alacsony szintű részlet jelképezi a szükséges összeköttetéseket (factoryk, csatornák, figyelők stb.). Ezek az alacsony szintű részletek általában rejtve vannak, ha azonban szükség van rá, igény szerint bővíteni vagy módosítani is lehet őket. Szerencsére a legtöbb esetben az alapértelmezett működés megfelelő.

A kiszolgáló- vagy kliensoldali `*.config` fájl használata opcionális. Ha szeretnénk, a szükséges összeköttetéseket (végpontokat, kötéseket, címeket stb.) specifikálhatjuk a hoszt (vagy a kliens) kódolásával. Ezzel a megközelítéssel nyilvánvalóan az a probléma, hogy ha meg kell változtatnunk az összeköttetések részleteit, sok szerelvényt újra kell kódolni, fordítani és telepíteni. A `*.config` fájl sokkal rugalmasabban kezeli a kódot, mivel az összeköttetések megváltoztatása csak annyit jelent, hogy frissítjük a fájl tartalmát, és újraindítjuk az alkalmazást.

A WCF ABC-je

A hosztok és a kliensek úgy kommunikálnak, hogy megegyeznek az ABC-ben. Ez egy egyszerű rövidítés a WCF-alkalmazás alapegységeinek jelölésére, úgymint *address* (cím), *binding* (kötés) és *contract* (szerződés).

- *Cím:* A szolgáltatás helye. A kódban ezt egy `System.Uri` típus jelképezi, az értéket azonban általában `*.config` fájlok tárolják.
- *Kötés:* A WCF-hez több különböző kötés jár, amelyek hálózati protokollokat, kódolási mechanizmusokat és a szállítási réteget specifikálják.
- *Szerződés:* A WCF-szolgáltatás által elérhetővé tett összes metódus leírása.

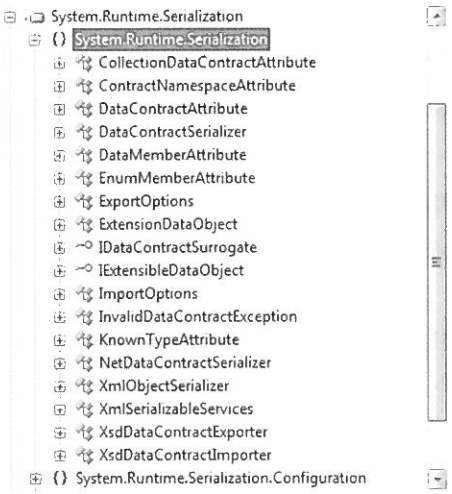
Az ABC rövidítés nem azt jelenti, hogy a fejlesztőnek először a címet, majd a kötést és végül a szerződést kell definiálnia. A WCF-fejlesztő sokszor a szolgáltatás szerződésének definiálásával kezd, majd megadja a címet és a kötéseket (de bármilyen sorrend jó, ha mindháromról gondoskodunk). Az első WCF-alkalmazás elkészítése előtt nézzük meg részletesebben az ABC-ket.

A WCF-szerződések

A *szerződés* fogalma kulcsfontosságú a WCF-szolgáltatások készítésekor. Bár nem kötelező, de a WCF-alkalmazások nagy többsége .NET-interfésztípusok definiálásával kezdődik, amelyek olyan tagok gyűjteményét jelképezik, amelyeket az adott WCF-típus támogat. Az olyan interfészeket, amelyek kifejezetten WCF-szerződéseket jelképeznek, *szolgáltatásszerződéseknek* nevezzük. Az osztályokat (vagy struktúrákat), amelyek ezeket implementálják, *szolgáltatástípusoknak* hívjuk.

A WCF-szolgáltatásszerződések több attribútummal rendelkeznek, amelyek közül a leggyakoribbakat a `System.ServiceModel` névtér definiálja. Amikor a szolgáltatásszerződés tagjai csak egyszerű adattípusokat tartalmaznak (mint pl. numerikus, logikai és sztringadatokat), teljes WCF-alkalmazást készíthetünk kizárólag a `[ServiceContract]` és az `[OperationContract]` attribútumok használatával.

Ha azonban a tagunk egyedi típusokat tesz elérhetővé, használnunk kell a `System.Runtime.Serialization.dll` szerelvény `System.Runtime.Serialization` névterében található típusokat (lásd a 25.5. ábrát). Itt további attribútumokat (mint pl. a `[DataMember]` és a `[DataContract]`) találunk az interfésztípusok finomításához.



25.5. ábra: *A System.Runtime.Serialization számos attribútumot definiál WCF-adatszerződések készítése során*

Valójában nem kötelező a CLR-interfészek használata WCF-szerződés definiálásához. Ezek közül az attribútumok közül sokat alkalmazhatunk nyilvános osztályok (vagy struktúrák) nyilvános tagjaira is. De az interfészalapú programozás (polimorfizmus, elegáns verziókövetés stb.) sok előnye ellenére is a CLR-interfészek használata a legjobb a WCF-szerződések leírására.

A WCF-kötések

Miután a szerződést (vagy szerződéseket) a szolgáltatáskönyvtárban definiáltuk és implementáltuk, a következő logikai lépés maga a WCF-szolgáltatás hosztolóprogramjának elkészítése. Több különböző hoszt közül választhatunk, ezek közül mindegyiknek meg kell határoznia azokat a *kötéseket*, amelyeket a távoli hívók használnak a szolgáltatástípus funkcionalitásának elérésére.

A kötések kiválasztása az egyik olyan terület, ahol a WCF-fejlesztés meglehetősen eltér a .NET-remoting és/vagy XML-webszolgáltatás fejlesztésétől abban, hogy a WCF-hez járnak azok a kötési választási lehetőségek, amelyek mindegyike speciális igények kielégítésére szolgál. Ha a szállított kötések egyike sem felel meg, akkor lehetőség van saját kötés létrehozására a `CustomBinding` típus kibővítésével (ezt ebben a fejezetben nem tesszük meg). A WCF-kötés egyszerűen a következő jellemzőket adhatja meg:

- A szolgáltatás által implementált szerződések.
- Az adatok mozgatására használt szállítási réteg (HTTP, MSMQ, named pipe-ok, TCP).
- A szállításra használt csatornák (egyirányú, kérés-válasz, duplex).
- Magukkal az adatokkal foglalkozó kódolási mechanizmus (XML, bináris stb.).
- Bármilyen támogatott webszolgáltatás-protokoll (ha a kötés engedélyezi), mint például WS-Security, WSTransactions, WS-Reliability és így tovább.

Nézzük meg a választási lehetőségeinket.

HTTP-alapú kötések

A `BasicHttpBinding`, a `WSHttpBinding`, a `WSDualHttpBinding` és a `WSFederationHttpBinding` opciók arra szolgálnak, hogy a szerződéstípusokat XML-webszolgáltatás-protokollokon keresztül elérhetővé tegyék. Ha a szolgáltatás számára a lehető legtávolabbi elérésre van szükségünk (több operációs rendszer és többféle programozási architektúra között), ezekre a kötésekre kell összpontosítanunk, mert minden ilyen kötéstípus XML-megjelenítés alapján kódolja az adatokat, és HTTP-t használ.

A WCF-kötést megjeleníthetjük a kódban (a System.`ServiceModel` névtérben levő osztálytípusok révén), vagy definiálhatjuk XML attribútumként a `*.config` fájlokban (lásd 25.3. táblázat).

Kötési osztály	Kötéselem	Jelentés
`BasicHttpBinding`	`<basicHttpBinding>`	WS-Basic Profile-kompatibilis (WS-I Basic Profile 1.1) WCF-szolgáltatás készítésére alkalmas. Ez a kötés szállításra HTTP-t, az üzenetkódolásra az alapértelmezés szerint szöveg/XML-t használ.
`WSHttpBinding`	`<wsHttpBinding>`	A `BasicHttpBinding`hoz hasonló, de több webszolgáltatási funkciót kínál. Ez a kötés támogatást nyújt tranzakciókhoz, megbízható üzenetküldéshez és WS-címzéshez.
`WSDualHttpBinding`	`<wsDualHttpBinding>`	A `WSHttpBinding` elemhez hasonló, de duplex szerződésekhez használható (pl. a szolgáltatás és a kliens üzeneteket tud küldeni oda-vissza). Ez a kötés csak a SOAP biztonsági beállításait támogatja, és megbízható üzenetküldést követel meg.

Kötési osztály	Kötéselem	Jelentés
`WSFederationHttp-Binding`	`<wsFederationHttp-Binding>`	Biztonságos és együttműködő kötés, amely támogatja a WS-Fedaration protokollt, lehetővé téve felhasználók hatékony azonosítását és engedélyezését szövetséget alkotó szervezetek számára.

25.3. táblázat: *HTTP-központú WCF-kötések*

A `BasicHttpBinding` a legegyszerűbb webszolgáltatás-központú protokoll. Ez a kötés biztosítja, hogy a WCF-szolgáltatásunk megfeleljen a WS-I-ben definiált WS-I Basic Profile 1.1 specifikációnak. Ezt a kötést főleg a visszamenőleges kompatibilitás megtartása miatt használjuk olyan alkalmazásokkal, amelyek korábban készültek ASP.NET-webszolgáltatásokkal folytatott kommunikációra (amelyek az 1.0 verzió óta a .NET-könyvtárak részei).

A `WSHttpBinding` protokoll nemcsak a WS-* specifikációk részhalmazát támogatja (tranzakciók, biztonság és megbízható munkamenetek), hanem a bináris adatkódolás kezelésének lehetőségét MTOM-technológiával (Message Transmission Optimization Mechanism - üzenetátvitel optimalizációs mechanizmusa).

A `WSDualHttpBinding` legfőbb előnye az, hogy a hívónak és a küldőnek lehetővé teszi a kommunikációt *duplex üzenetküldéssel*, amely azt jelenti, hogy két irányban tudnak egymással társalogni. Ha a `WSDualHttpBindingot` választjuk, bekapcsolódhatunk a WCF publish/subscribe (közzétesz és előfizet) eseménymodelljébe.

Végül a `WSFederationHttpBinding` az a webszolgáltatás-alapú protokoll, amelyet akkor érdemes használni, ha a biztonság a legfontosabb szempont. A kötés támogatja a WS-Trust, WS-Security és a WS-SecureConversation specifikációkat, amelyeket a WCF CardSpace API-k valósítanak meg.

TCP-alapú kötések

.NET 3.0/3.5 könyvtárakkal konfigurált számítógépekkel (más szóval minden Windows XP, Windows Server 2003 vagy Windows Vista operációs rendszert futtató gépet) üzemelő elosztott alkalmazás készítése során javulhat a teljesítmény, ha megkerüljük a webszolgáltatási kötéseket, és inkább TCP-kötést választunk, amely biztosítja, hogy minden adat XML helyett kompakt bináris formátumban legyen kódolva. A 25.4. táblázatban szereplő kötések használatakor a kliensnek és a hosztnak .NET-alkalmazásnak kell lennie.

Kötési osztály	Kötéselem	Jelentés
`NetNamedPipeBinding`	`<netNamedPipe-Binding>`	Biztonságos, megbízható, optimalizált kötés egy számítógépen levő .NET-alkalmazások közötti kommunikációhoz.
`NetPeerTcpBinding`	`<netPeerTcpBinding>`	Biztonságos kötést biztosít P2P hálózati alkalmazásokhoz.
`NetTcpBinding`	`<netTcpBinding>`	Biztonságos és optimalizált kötés .NET-alkalmazások közötti kommunikációhoz több számítógép között.

25.4. táblázat: *TCP-központú WCF-kötések*

A `NetTcpBinding` osztály TCP-t használ bináris adatok átvitelére a kliens és a WCF-szolgáltatás között. Ez jobb teljesítményt eredményez, mint a webszolgáltatás-protokollok, de így a lehetőségeink házon belüli Windows-megoldásokra korlátozódnak. Előnye viszont, hogy a `NetTcpBinding` támogatja a tranzakciókat, a megbízható munkameneteket és a biztonságos kommunikációt.

A `NetTcpBindinghoz` hasonlóan a `NetNamedPipeBinding` ugyancsak támogatja a tranzakciókat, a megbízható munkameneteket és a biztonságos kommunikációt, de nem képes gépek közötti hívások lebonyolítására. Ha az egy gépen futó WCF-alkalmazások közötti leggyorsabb adatküldési módot keressük (pl. alkalmazástartományok közötti kommunikáció), akkor a `NetNamedPipeBinding` a legjobb választása a kötésre. Ami a `NetPeerTcpBindingot` illeti, a P2P további részleteiért forduljunk a .NET Framework 3.5 dokumentációjához.

MSMQ-alapú kötések

Végül, ha Microsoft MSMQ szervert próbálunk használni, a `NetMsmqBinding` és az `MsmqIntegrationBinding` kötés kerül a középpontba. Ebben a fejezetben nem vizsgáljuk meg részletesen az MSMQ-kötések használatát, de a 25.5. táblázat bemutatja az alapvető szerepüket.

Kötési osztály	Kötéselem	Jelentés
`MsmqIntegration-Binding`	`<msmqIntegration-Binding>`	Ez a kötés lehetővé teszi, hogy WCF-alkalmazások olyan létező MSMQ-alkalmazásoknak küldjenek, és alkalmazásoktól fogadjanak üzeneteket, amelyek COM-ot, natív C++-t vagy a `System.Messaging` névtérben definiált típusokat használnak.
`NetMsmqBinding`	`<netMsmqBinding>`	Ez a kötés alkalmas .NET-alkalmazások gépek közötti kommunikációjára.

25.5. táblázat: *Az MSMQ-központú WCF-kötések*

NÉHÁNY SZÓ A COM+-OBJEKTUMOKKAL FOLYTATOTT KOMMUNIKÁCIÓRÓL

Nincs olyan kötés, amely kifejezetten a COM+-objektumokkal folytatott kommunikációt szolgálja. A WCF révén teljes mértékben tudunk COM+-komponensekkel kommunikálni, ez azonban azt jelenti, hogy a COM+-típusokat elérhetővé kell tennünk XML-webszolgáltatási kötésen keresztül a `ComSvcConfig.exe` parancssori eszköz segítségével – amely a .NET Framework 3.5 SDK része – vagy az `SvcConfigEditor.exe` eszközzel (amelyet a későbbiekben megvizsgálunk).

A WCF-címek

Ha már készen vannak a szerződések és a kötések, végül meg kell adnunk a WCF-szolgáltatás *címét*. Ez nyilvánvalóan meglehetősen fontos, hiszen távoli hívók nem tudnak kommunikálni távoli típusokkal, ha nem tudják, hol vannak. A WCF egyéb funkcióihoz hasonlóan a címet beírhatjuk egy szerelvény kódjába (a `System.Uri` típus segítségével) vagy betölthetjük a `*.config` fájlból. A WCF-cím pontos formátuma mindkét esetben a választott kötéstől (HTTP-alapú, named pipe-ok, TCP- vagy MSMQ-alapú) függően változik. A WCF-címek a következő infromációkat adhatják meg:

- `Scheme`: A szállítási protokoll (HTTP stb.).
- `MachineName`: A számítógép teljesen meghatározott tartománya.
- `Port`: Ez sok esetben opcionális. A HTTP-kötések alapértelmezett portja például a 80-as.
- `Path`: A WCF-szolgáltatás elérési útvonala.

Ezeket az információkat a következő általánosított sablon mutatja be (a `Port` érték opcionális, mivel néhány kötés nem használja):

```
scheme://<MachineName>[:Port]/Path
```

Webszolgáltatás-alapú kötés (`basicHttpBinding`, `wsHttpBinding`, `wsDualHttpBinding` vagy `wsFederationHttpBinding`) használatakor a cím így áll össze (ha nem adunk meg portszámot, a HTTP-alapú protokollok portja alapértelmezés szerint a 80-as):

```
http://localhost:8080/MyWCFService
```

Megjegyzés Ha SSL-t (Secure Sockets Layer) használunk, csak írjuk át a http-t https-re.

Ha TCP-központú kötéseket használunk (mint pl. NetTcpBinding vagy `NetPeerTcpBinding` kötéseket), az URI-formátuma a következő lesz:

```
net.tcp://localhost:8080/MyWCFService
```

Az MSMQ-központú kötések (`NetMsmqBinding` és `MsmqIntegrationBinding`) URI-formátuma kicsit egyedi, mivel az MSMQ használhat nyilvános vagy privát sorokat (amelyek csak a helyi gépen állnak rendelkezésre), valamint a portszámoknak nincs jelentése az MSMQ-központú URI-ben. Nézzük meg a következő URI-t, amely a `MyPrivateQ` nevű privát sort jellemzi:

```
net.msmq://localhost/private$/MyPrivateQ
```

Végül, de nem utolsósorban a named-pipe kötéssel (`NetNamedPipeBinding`) használt címformátum így épül fel (a named pipe-ok teszik lehetővé a folyamatok közötti kommunikációt ugyanazon a gépen üzemelő alkalmazások számára):

```
net.pipe://localhost/MyWCFService
```

Bár lehet, hogy egyetlen WCF-szolgáltatás csak egy címet tesz elérhetővé (egyetlen kötésre alapozva), lehetőség van egyedi címek gyűjteményének a definiálására is (különböző kötésekkel). Ezt több `<endpoint>` elem definiálásával tehetjük meg a `*.config` fájlban. Itt tetszőleges számú ABC-t megadhatunk ugyanahhoz a szolgáltatáshoz. Ez a megközelítés hasznos lehet, ha meg akarjuk engedni a hívóknak, hogy kiválaszthassák a használni kívánt protokollt, amikor a szolgáltatással kommunikálnak.

WCF-szolgáltatás készítése

Hozzuk létre első példaprogramunkat, hogy megnézzük, az ABC-k hogyan jelennek meg a kódban. Első lépésként definiáljuk a WCF-szolgáltatáskönyvtárat, amely tartalmazza a szerződéseket és implementációjukat.

Az első példában nem használjuk a Visual Studio WCF projektsablonjait, hogy a WCF-szolgáltatás készítésének egyes lépéseire figyeljünk. Kiindulásként hozzunk létre egy C#-osztálykönyvtárt MagicEightBallServiceLib névvel. Nevezzük át az eredeti fájlt Class1.cs-ről MagicEightBallService.cs-re, és adjunk hozzá referenciát a `System.ServiceModel.dll` szerelvényre. A kiindulási kódfájlban adjuk meg, hogy a `System.ServiceModel` névteret használjuk. Ekkor a C#-fájlnak így kell kinéznie:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

// A fő WCF-névtér.
using System.ServiceModel;

namespace MagicEightBallServiceLib
{
  public class MagicEightBallService
  {
  }
}
```

Az osztálytípusunk egyetlen WCF szolgáltatási szerződést valósít meg az erősen típusos `IEightBall` nevű CLR-interfész révén. Talán ismerjük a Magic 8-Ball nevű játékot, ebben egy feltett kérdésre előre megadott válaszok közül kell kiválasztani egyet. Az interfészünk egyetlen metódust definiál, amellyel a hívó feltehet egy kérdést a Magic 8-Ball játéknak, a program pedig véletlenszerű választ ad.

A WCF szolgáltatási interfészek rendelkeznek a `[ServiceConract]` attribútummal, az interfész minden tagjához pedig hozzárendeljük a `[OperationContract]` attribútumot (ezek részleteit lásd később). Az `IEightBall` interfész definíciója a következő:

```
[ServiceContract]
public interface IEightBall
{
  // Kérdezz, hogy választ kapj!
  [OperationContract]
  string ObtainAnswerToQuestion(string userQuestion);
}
```

Megjegyzés Olyan szolgáltatásszerződési interfészeket lehet definiálni, amelyek metódusai nem tartalmazzák az `[OperationContract]` attribútumot, de az ilyen tagok nem lesznek elérhetőek a WCF-futtatórendszerben.

Az interfészek addig használhatatlanok, amíg egy osztály vagy struktúra nem implementálja őket, hogy kihasználja funkcionalitásukat (lásd az előző kötet 9. fejezetét). Az igazi Magic 8-Ball játékhoz hasonlóan az általunk implementált szolgáltatástípus (`MagicEightBallService`) véletlenszerűen ad vissza egy választ az előre megadott sztringtömb elemei közül. Az alapértelmezett konstruktor üzenetet küld, amely (végül) a hoszt konzolablakában jelenik meg (az ellenőrzés miatt):

```
public class MagicEightBallService : IEightBall
{
  // Csak megjelenítési célból a hoszton.
  public MagicEightBallService()
  {
    Console.WriteLine("The 8-Ball awaits your question...");
  }

  public string ObtainAnswerToQuestion(string userQuestion)
  {
    string[] answers = { "Future Uncertain", "Yes", "No",
                         "Hazy", "Ask again later", "Definitely" };

    // Véletlenszerűen visszaad egy választ.
    Random r = new Random();
    return string.Format("{0}? {1}.",
    userQuestion, answers[r.Next(answers.Length)]);
  }
}
```

Készen van a WCF-szolgáltatáskönyvtárunk. Mielőtt elkészítenénk a hosztot a szolgáltatáshoz, nézzük meg bővebben a `[ServiceContract]` és az `[OperationContract]` attribútumokat.

A [ServiceContract] attribútum

A CLR-interfésznek tartalmaznia kell a `[ServiceContract]` attribútumot, hogy részt tudjon venni a WCF szolgáltatásokban. Több más .NET-attribútumhoz hasonlóan a `ServiceContractAttribute` típus számos tulajdonságot támogat funkciójának jobb kiaknázására. A `Name` és a `Namespace` tulajdonságot beállíthatjuk, hogy megadja a szolgáltatástípus és a szolgáltatástípust definiáló XML-névtér nevét. Ha webszolgáltatás-specifikus kötést használunk, ezek az értékek megadják a `<portType>` elemet a kapcsolódó WSDL-dokumentumban.

A `Name` tulajdonságnak nem adtunk értéket, mivel a szolgáltatástípus alapértelmezett neve közvetlenül a C#-osztály nevén alapul. Az alapul szolgáló XML-névtér alapértelmezett neve azonban egyszerűen http://tempuri.org (amelyet illik megváltoztatni minden WCF-szolgáltatásnál).

Egyedi adattípusokat küldő és fogadó WCF-szolgálatás készítése során fontos, hogy értelmes értéket adjunk az alapul szolgáló XML-névtérnek, mert ez biztosítja, hogy az egyedi típusaink valóban egyediek legyenek. Ha már készítettünk XML-webszolgáltatást, akkor tudjuk, hogy az XML-névtér módot ad arra, hogy az egyedi típusokat egyedi tárolóba helyezzük, hogy ezek ne ütközzenek más szervezetek típusaival.

Ezért az interfészdefiníciónkat frissíthetjük egy jobb definícióval, amely csakúgy, mint a .NET-webszolgáltatás-projektben az XML-névtér definiálásakor, igazából a szolgáltatás származási helyének URI-je, például:

```
[ServiceContract(Namespace = "http://Intertech.com")]
public interface IEightBall
{
  ...
}
```

A `Namespace` és a `Name` tulajdonságokon kívül a `[ServiceContract]` attribútumot konfigurálhatjuk egyéb, a 25.6. táblázatban látható tulajdonságokkal. A választott kötéstől függően néhány beállítást a rendszer figyelmen kívül hagy.

Tulajdonság	Jelentés
`CallbackContract`	Megadja, hogy a szolgáltatásszerződésnek szüksége van visszahívási funkcionalitásra a kétirányú üzenetváltáshoz.
`ConfigurationName`	Az alkalmazás konfigurációs fájljában a szolgáltatáselem megkereséséhez használt nevet foglalja magában. Az alapértelmezés szerint a szolgáltatás implementációs osztályának neve.

Tulajdonság	Jelentés
`ProtectionLevel`	Megadja a szerződés kötése által megkövetelt titkosítás, a digitális aláírások vagy mindkettő szükségességének szintjét a szerződést elérhetővé tevő végpontok számára.
`SessionMode`	Megadja, hogy a munkamenetek engedélyezettek, nem engedélyezettek vagy szükségesek-e a szolgáltatási szerződésben.

25.6. táblázat: *A [ServiceContract] attribútum megnevezett tulajdonságai*

Az [OperationContract] attribútum

A WCF keretrendszerben használni kívánt metódusokhoz hozzá kell rendelni az `[OperationContract]` attribútumot, amelyet szintén több nevesített tulajdonsággal konfigurálhatunk. A 25.7. táblázatban található tulajdonságok használatával megadhatjuk, hogy az adott metódus természete egyirányú legyen, támogassa az aszinkron hívást, kódolt üzenetadatokat követeljen meg, és így tovább (az értékek közül a rendszer többet figyelmen kívül hagyhat a választott kötéstől függően).

Tulajdonság	Jelentés
`Action`	A kérési üzenet WS-Addressing tevékenységét állítja be vagy adja vissza.
`AsyncPattern`	A szolgáltatásban a Begin/End metóduspár segítségével jelzi, ha a művelet aszinkron módon van implementálva. Ez teszi lehetővé a szolgáltatás számára, hogy átadja a feldolgozást egy másik szerveroldali szálnak. Ennek semmi köze a kliens aszinkron metódushívásához.
`IsInitiating`	Megadja, hogy a művelet lehet egy munkamenet kezdőművelete.
`IsOneWay`	Jelzi, hogy a művelet csak egyetlen bemenő üzenetből áll (és nincs hozzátartozó kimenet).
`IsTerminating`	Megadja, hogy a művelet befejezése után a WCF-futtatókörnyezet megpróbálja leállítani a jelenlegi munkamenetet.

25.7. táblázat: *Az [OperationContract] attribútum megnevezett tulajdonságai*

Ebben a kezdeti példában nem kell az `ObtainAnswerToQuestion()` metódust egyéb jellemzőkkel konfigurálni, az `[OperationContract]` attribútum használható az eredetileg definiált módon.

Szolgáltatástípusok mint működési szerződések

WCF-szolgáltatástípus készítése során nem kell interfészeket használni. Tulajdonképpen a `[ServiceContract]` és az `[OperationContract]` attribútumot közvetlenül a szolgáltatástípusra is alkalmazhatjuk:

```
// Csak illusztráció,
//az aktuális példában nem szerepel.
[ServiceContract(Namespace = "http://Intertech.com")]
public class ServiceTypeAsContract
{
  [OperationContract]
  void SomeMethod() { }
  [OperationContract]
  void AnotherMethod() { }
}
```

Jóllehet ez is egy lehetséges megközelítés, sok előnye van az interfésztípus explicit definiálásának a szolgáltatásszerződés elkészítésében. A legnyilvánvalóbb előny az, hogy az adott interfész többféle szolgáltatástípushoz (különböző nyelveken és architektúrákon készült) használható a nagyfokú polimorfizmus biztosítása érdekében. Másik előny az, hogy a szolgáltatásszerződési interfész használható új szerződések alapjaként (interfészszármaztatás révén), az implementáció terhe nélkül.

Jelen pillanatban az első WCF-szolgáltatáskönyvtárunk készen van. Fordítsuk le a projektet, hogy meggyőződhessünk róla, nincs-e benne gépelési hiba.

Forráskód A MagicEightBallServiceClient kódfájlokat a forráskódkönyvtár 25. fejezetének alkönyvtára tartalmazza. A forráskódkönyvtárról lásd a Bevezetés xlv. oldalát.

A WCF-szolgáltatás hosztolása

Bár a termékszintű szolgáltatásokat rendszerint Windows-szolgáltatás vagy IIS-beli virtuális könyvtár hosztolja, az első hosztunk egyszerű parancssor lesz `MagicEightBallServiceHost` névvel.

Ha elkészült az új konzolalkalmazás-projekt, adjunk hozzá referenciát a `System.ServiceModel.dll` és a `MagicEightBallServiceLib.dll` szerelvényre, majd frissítsük az eredeti kódfájlt a `System.ServiceModel` és a `MagicEightBallServiceLib` névterek importálásával:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.ServiceModel;
using MagicEightBallServiceLib;

namespace MagicEightBallServiceHost
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("***** Console Based WCF Host *****");
      Console.ReadLine();
    }
  }
}
```

WCF-szolgáltatástípus készítésekor első lépésként el kell dönteni, hogy a szükséges hosztolási logikát teljes mértékben a kódban szeretnénk-e definiálni, vagy néhány alacsony szintű részletet az alkalmazás konfigurációs fájljában szeretnénk megadni. A `*.config` fájlok előnye az, hogy a hoszt megváltoztathatja az alapul szolgáló beállításokat anélkül, hogy újra kellene fordítanunk és telepítenünk a végrehajtható fájlt. Ez szigorúan opcionális, mivel a hosztolási logikát a kódban a `System.ServiceModel.dll` szerelvény típusaival is megadhatjuk.

Ez a parancssori hoszt ténylegesen az alkalmazás konfigurációs fájlját használja, ezért adjunk hozzá az aktuális projekthez új alkalmazáskonfigurációs fájlt a Project > Add New Item menüponttal.

ABC-k létrehozása az App.config fájlban

A WCF-szolgáltatástípushoz a hoszt készítése megjósolható lépések sorozatából áll - a konfigurációs fájlban és a forráskódban:

1. Definiáljuk a hosztolni kívánt WCF-szolgáltatáshoz a *végpontot* a hoszt konfigurációs fájljában.

2. Programozottan használjuk a `ServiceHost` típust az ebből a végpontból rendelkezésre álló szolgáltatástípusok elérhetővé tételéhez.
3. Győződjünk meg róla, hogy a hoszt még fut azért, hogy kiszolgálja a kliensektől bejövő kéréseket. Ez a lépés nyilván nem szükséges, ha Windows-szolgáltatással vagy IIS-sel hosztoljuk a szolgáltatástípusokat.

A WCF világában a „végpont" kifejezés egyszerűen a címet, a kötést és a szerződést jelenti, csinos kis csomagban összekötve. XML-ben a végpontot az `<endpoint>` elem, valamint az `address`, a `binding` és a `contract` elem fejezi ki. Frissítsük a `*.config` fájlt, és adjunk meg egy végpontot (a 8080-as porton keresztül érhető el), amelyet ezen a hoszton teszünk elérhetővé:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
    <service name="MagicEightBallServiceLib.MagicEightBallService">
    <endpoint address ="http://localhost:8080/MagicEightBallService"
              binding="basicHttpBinding"
              contract="MagicEightBallServiceLib.IEightBall"/>
    </service>
    </services>
  </system.serviceModel>
</configuration>
```

A `<system.serviceModel>` a gyökérelem a hoszt összes WCF-beállításához. A hoszton minden elérhető szolgáltatást a `<service>` elem képvisel, amelyeket a `<services>` alapelembe ágyazunk. Itt az egyetlen `<service>` elem az (opcionális) `name` attribútummal adja meg a szolgáltatástípus barátságos nevét.

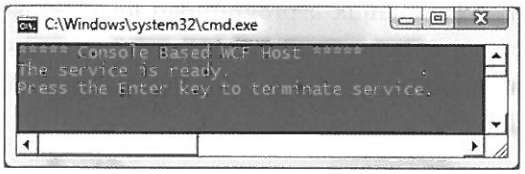
A beágyazott `<endpoint>` elem definiálja a címet, a kötési modellt (a példában `basicHttpBinding`) és a WCF-szolgáltatásszerződést definiáló interfésztípus teljesen meghatározott nevét (`IEightBall`). Mivel HTTP-alapú kötést használunk, a http:// sémát alkalmazzuk, és tetszés szerinti portazonosítót határozunk meg.

A ServiceHost típus használata

Az aktuális konfigurációs fájl birtokában a hoszt befejezéséhez a programozási logika elkészítése nagyon egyszerű. Amikor a végrehajtható fájl elindul, létrehozzuk a `ServiceHost` típus egy példányát. Futásidőben az objektum automatikusan kiolvassa a `<system.serviceModel>` elem hatókörében található adatokat a hoszt `*.config` fájljából, hogy meghatározza a helyes címet, kötést és szerződést, és elkészíti a szükséges összeköttetéseket:

```
static void Main(string[] args)
{
  Console.WriteLine("***** Console Based WCF Host *****");
  using (ServiceHost serviceHost =
                    new ServiceHost(typeof(MagicEightBallService)))
  {
    // Megnyitja a hosztot, és figyelni kezdi a bejövő üzeneteket.
    serviceHost.Open();

    // Az Enter billentyű leütéséig futtatja a szolgáltatást.
    Console.WriteLine("The service is ready.");
    Console.WriteLine("Press the Enter key to terminate service.");
    Console.ReadLine();
  }
}
```

Ha futtatjuk az alkalmazást, kiderül, hogy a hoszt él a memóriában, készen a távoli kliensektől érkező kérések fogadására (lásd a 25.6. ábrát).



25.6. ábra: *Hosztunk készen áll a külső hívásokra HTTP-kötésen keresztül*

Hosztfejlesztési lehetőségek

Jelenleg olyan konstruktorral hozzuk létre a `ServiceHostot`, amely csak a szolgáltatás típusáról igényel információt. Ezt azonban átadhatjuk konstruktorparamétereként `System.Uri` típusok tömbjében, amely olyan címek gyűjteményét képviseli, ahonnan a szolgáltatás elérhető. A cím jelenleg a `*.config` fájl révén érhető el, de ha a hatókört frissítenénk, akkor:

```
using (ServiceHost serviceHost =
       new ServiceHost(typeof(MagicEightBallService),
       new Uri[]{new
           Uri("http://localhost:8080/MagicEightBallService")}))
{
  ...
}
```

így végpontot is tudnánk definiálni:

```
<endpoint address =""
          binding="basicHttpBinding"
          contract="MagicEightBallServiceLib.IEightBall"/>
```

A túl sok kódolás a hoszt programkódjában csökkenti a rugalmasságot, így a jelenlegi hosztpéldánkhoz a szolgáltatási hosztot egyszerűen a típusinformáció megadásával hozzuk létre, ahogy eddig is tettük:

```
using (ServiceHost serviceHost =
      new ServiceHost(typeof(MagicEightBallService)))
{
  ...
}
```

A hoszt `*.config` fájlok létrehozásának egyik (kissé zavaró) vonása az, hogy az XML-leírókat többféleképpen összeállíthatjuk a programban levő forráskód mennyiségétől függően (ahogyan éppen az előbb láttuk az opcionális URI-tömb esetében). Hogy megmutassuk a `*.config` fájlok létrehozásának egy másik módját, gondoljuk át a következő újrafeldolgozást:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
    <service name="MagicEightBallServiceLib.MagicEightBallService">

      <!-- A címet a <baseAddresses> adja meg.-->
      <endpoint address =""
                binding="basicHttpBinding"
                contract="MagicEightBallServiceLib.IEightBall"/>

      <!-- Megjelölt részben felsorolja az összes alapcímet.-->
      <host>
        <baseAddresses>
          <add baseAddress =
               "http://localhost:8080/MagicEightBallService"/>
        </baseAddresses>
      </host>
    </service>
    </services>
  </system.serviceModel>
</configuration>
```

Ebben az esetben az <endpoint> elem address attribútuma még üres, és függetlenül attól, hogy a ServiceHost létrehozásakor a kódban nem adjuk meg az URI-objektumok tömbjét, az alkalmazás ugyanúgy fut, mint az előbb, mert az értéket a baseAddresses hatóköréből veszi. Az alapcím tárolásának előnye a <host> <baseAddresses> területén az, hogy a *.config fájlnak (mint a MEX, lásd később) ismernie kell a szolgáltatás végpontjának címét is. Így ahelyett, hogy egyetlen *.config fájlba kellene másolni és abban küldeni a címértékeket, a bemutatott módon elkülöníthetjük az egyetlen értéket.

Megjegyzés A fejezet későbbi részében bemutatunk egy grafikus konfigurációs eszközt a konfigurációs fájlok kevésbé bonyolult elkészítéséhez.

Mielőtt a szerverünkkel kommunikáló kliensalkalmazást készítenénk, vizsgáljuk meg a ServiceHost osztálytípus, a <service.serviceModel> elem és a MEX (metadata exchange – metaadatcsere) szerepét.

A ServiceHost típus

A ServiceHost osztálytípus segítségével tudjuk beállítani és elérhetővé tenni a WCF-szolgáltatást a hosztoló végrehajtható fájlból. Ezt a típust csak akkor használjuk közvetlenül, ha egyedi *.exe hosztolja szolgáltatásainkat. Ha a szolgáltatás közzétételéhez IIS-t (vagy a Vista-specifikus WAS-t) használunk, a ServiceHost objektum automatikusan létrejön.

Ez a típus teljes szolgáltatásleírást igényel, amelyet dinamikusan kap meg a hoszt *.config fájl konfigurációs beállításaiból. Amíg ez az objektum létrehozása során automatikusan megtörténik, a ServiceHost objektum állapota manuálisan állítható be néhány tag segítségével. A 25.8. táblázat mutatja az Open() és Close() tagok (amelyek aszinkron módon kommunikálnak a szolgáltatásunkkal) mellett szót érdemlő tagokat.

Tagok	Jelentés
Authorization	Ez a tulajdonság visszaadja a hosztolt szolgáltatás hitelesítési szintjét.
AddServiceEndpoint()	Ezzel a metódussal programozottan regisztrálhatunk végpontot a hoszthoz.

Tagok	Jelentés
`BaseAddresses`	Ez a tulajdonság megszerzi az adott szolgáltatás regisztrált alapcímeinek a listáját.
`BeginOpen()` `BeginClose()`	Ezek a metódusok teszik lehetővé a `ServiceHost` objektum aszinkron megnyitását és bezárását a szabványos .NET-es metódusreferencia-szintaxis használatával.
`CloseTimeout`	Ezzel a tulajdonsággal beállíthatjuk vagy visszakapjuk a szolgáltatás leállítására engedélyezett időt.
`Credentials`	Ez a tulajdonság megszerzi az adott szolgáltatás biztonsági igazolásait.
`EndOpen()` `EndClose()`	Ezek a `BeginOpen()` és a `BeginClose()` metódusok aszinkron megfelelői.
`OpenTimeout`	Ezzel a tulajdonsággal beállíthatjuk vagy kiolvashatjuk a szolgáltatás elindítására engedélyezett időt.
`State`	Ez a tulajdonság visszaad egy értéket, amely a kommunikációs objektum – ezt a `CommunicationState` felsorolt típus (`opened`, `closed`, `created` stb.) képviseli – aktuális állapotát jelzi.

25.8. táblázat: *A ServiceHost típus néhány tagja*

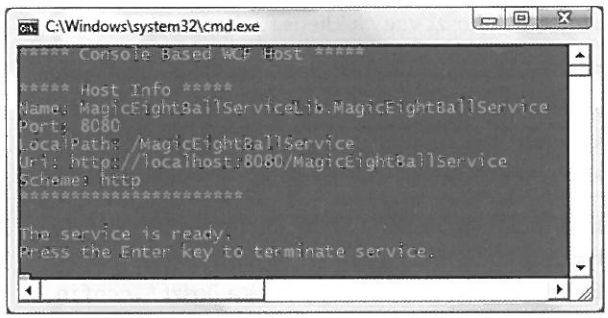
A `ServiceHost` néhány további funkciójának bemutatásához módosítsuk a Program osztályt új statikus metódussal, amely kiírja az aktuális hoszt különböző jellemzőit:

```
static void DisplayHostInfo(ServiceHost host)
{
  Console.WriteLine();
  Console.WriteLine("***** Host Info *****");

  Console.WriteLine("Name: {0}",
              host.Description.ConfigurationName);
  Console.WriteLine("Port: {0}",
              host.BaseAddresses[0].Port);
  Console.WriteLine("LocalPath: {0}",
              host.BaseAddresses[0].LocalPath);
  Console.WriteLine("Uri: {0}",
              host.BaseAddresses[0].AbsoluteUri);
  Console.WriteLine("Scheme: {0}",
              host.BaseAddresses[0].Scheme);
  Console.WriteLine("**********************");
  Console.WriteLine();
}
```

Tételezzük fel, hogy a hoszt megnyitása után ezt az új metódust hívjuk a `Main()` metódusból:

```
using (ServiceHost serviceHost =
                    new ServiceHost(typeof(MagicEightBallService)))
{
  // Megnyitja a hosztot, és figyelni kezdi a bejövő üzeneteket.
  serviceHost.Open();
  DisplayHostInfo(serviceHost);
  ...
}
```

így a 25.7. ábrán látható statisztikákat kapjuk.



25.7. ábra: *A hoszt tulajdonságai*

A <system.serviceModel> elem jellemzői

Más XML-elemekhez hasonlóan a `<system.serviceModel>` elemben is definiálhatunk részelemeket, amelyek mindegyikét több attribútum minősítheti. Míg a lehetséges attribútumokat illetően részleteket a .NET Framework 3.5 SDK dokumentációja tartalmaz, a következő vázlat felsorolja az érvényes részelemeket:

```
<system.serviceModel>
  <behaviors>
  </behaviors>
  <client>
  </client>
  <commonBehaviors>
  </commonBehaviors>
  <diagnostics>
  </diagnostics>
  <serviceHostingEnvironment>
  </serviceHostingEnvironment>
```

```
    <comContracts>
    </comContracts>
    <services>
    </services>
    <bindings>
    </bindings>
  </system.serviceModel>
```

Az egyes részelemek lényege megtalálható a 25.9. táblázatban.

Részelem	Jelentés
`behaviors`	A WCF különböző végpont- és szolgáltatási viselkedést támogat. Röviden: a `behavior` segítségével tovább bővíthetjük a hoszt vagy a kliens funkcionalitását.
`bindings`	Ezzel az elemmel beállíthatjuk a WCF által biztosított kötéseket (`basicHttpBinding`, `netMsmqBinding` stb.), valamint megadhatjuk a hoszt által használt egyedi kötéseket.
`client`	Ez az elem tartalmazza azon végpontok listáját, amelyeket a kliens a szolgáltatáshoz kapcsolódva használ. Ez természetesen nem túl hasznos a hoszt `*.config` fájljában.
`comContracts`	Ez az elem definiálja a WCF és COM együttműködéséhez engedélyezett COM-szerződéseket.
`commonBehaviors`	Ezt az elemet csak a `machine.config` fájlban lehet alkalmazni. Segítségével lehet beállítani minden olyan viselkedést, amelyet az egyes WCF-szolgáltatások egy adott gépen használnak.
`diagnostics`	Ez az elem tartalmazza a WCF diagnosztikai jellemzőinek beállításait. A felhasználó beállíthatja a nyomkövetés, a teljesítményszámlálók és a WMI-szolgáltató engedélyezését/letiltását, valamint egyedi üzenetszűrőket adhat meg.
`serviceHosting-Environment`	Ez az elem határozza meg, hogy a művelet lehet egy munkamenet kezdőművelete.
`services`	Ez az elem tartalmazza a hoszton elérhető WCF-szolgáltatások gyűjteményét.

25.9. táblázat: *A <service.serviceModel> részelemei*

Metaadatcsere (Metadata Exchange) engedélyezése

A WCF-ügyfélalkalmazások egy közbeépülő proxytípuson keresztül kommunikálnak a WCF-szolgáltatással. Bár a proxy kódja teljes egészében elkészíthető manuálisan, ez fárasztó és hibákat magában rejtő folyamat. Ideális esetben rendelkezésre állna egy eszköz a szükséges kód generálására (a kliensoldali `*.config` fájlt is beleértve). Szerencsére a .NET Framework 3.5 SDK pontosan erre a célra biztosít egy parancssori eszközt (`svcutil.exe`), valamint a Visual Studio 2008 Project >Add Service Reference menüpontja is hasonló funkcionalitást nyújt.

A szükséges proxykód/`*.config` fájl generálásához ezeknek az eszközöknek fel kell térképezniük a WCF-szolgáltatás interfészeinek és az összes definiált adatszerződésnek a formátumát (a metódusneveket, a paraméterek típusát stb.).

A MEX (metadata exchange - metaadatcsere) olyan WCF *szolgáltatási viselkedés*, amelynek megadásával beállíthatjuk, hogy a WCF-futtatókörnyezet hogyan kezelje a szolgáltatást. Minden `<behavior>` elem megadhat egy olyan tevékenységkészletet, amelyre egy adott szolgáltatás előfizethet. A WCF több viselkedést készen kínál, de a sajátunkat is elkészíthetjük.

A MEX-viselkedés (amely az alapértelmezés szerint nem engedélyezett) minden HTTP GET utasításon keresztül küldött metaadatkérést észlel. Ha megengedjük az `svcutil.exe`-nek vagy a Visual Studio 2008-nak, hogy automatikusan hozza létre a kívánt ügyféloldali proxy `*.config` fájlját, engedélyezni kell a MEX-et.

A MEX engedélyezése a hoszt `*.config` fájljában a megfelelő beállítások módosításával (vagy a megfelelő C#-kód létrehozásával) történik. Először hozzá kell adni egy új `<endpoint>` elemet kizárólag a MEX miatt. Másodszor WCF-viselkedést kell definiálni, hogy megengedjük a HTTP GET hozzáférést. Harmadszor ezt a viselkedést hozzá kell rendelni név alapján a szolgáltatáshoz a nyitó `<service>` elemen a `behaviorConfiguration` attribútum segítségével. Végül hozzá kell adni egy `<host>` elemet a szolgáltatás alapcímének megadásához (a MEX itt fogja keresni a leírni kívánt típusok helyét).

Megjegyzés Ez a végső lépés megkerülhető, ha paraméterként egy alapcímet képviselő `System.Uri` objektumot adunk át a `ServiceHost` konstruktornak.

Nézzük meg a következő módosított hoszt `*.config` fájlt, amely egyedi `<behavior>` elemet definiál (a neve `EightBallMEXBehavior`), amely a `<service>` definícióban szereplő `behaviorConfiguration` attribútum révén kapcsolódik a szolgáltatáshoz:

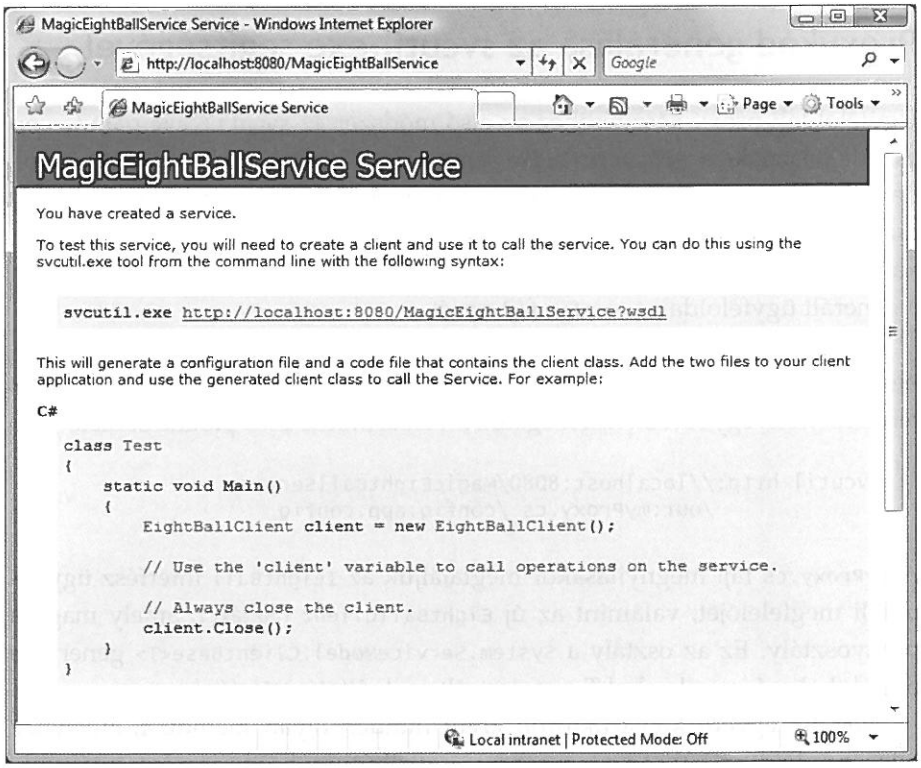
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="MagicEightBallServiceLib.MagicEightBallService"
            behaviorConfiguration = "EightBallServiceMEXBehavior">
        <endpoint address =""
                  binding="basicHttpBinding"
                  contract="MagicEightBallServiceLib.IEightBall"/>

        <!-- A MEX végpont engedélyezése. -->
        <endpoint address="mex"
                  binding="mexHttpBinding"
                  contract="IMetadataExchange" />

        <!-- Ezt hozzá kell adni, hogy a MEX tudja a
             szolgáltatásunk címét -->
        <host>
          <baseAddresses>
            <add baseAddress =
                    "http://localhost:8080/MagicEightBallService"/>
          </baseAddresses>
        </host>
      </service>
    </services>

    <!-- Viselkedésdefiníció a MEX-nek -->
    <behaviors>
      <serviceBehaviors>
        <behavior name="EightBallServiceMEXBehavior" >
          <serviceMetadata httpGetEnabled="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Most már újraindíthatjuk a szolgáltatást, majd bármelyik böngészőben megtekinthetjük a metaadatleírást. Ehhez egyszerűen írjuk be URL-ként a címet, amíg a hoszt fut:

```
http://localhost:8080/MagicEightBallService
```

A WCF-szolgáltatásunk kezdőoldalán (lásd a 25.7. ábrát) megkapjuk az alapvető részleteket arról, hogyan kommunikálhatunk ezzel a szolgáltatással programozottan, valamint megtekinthetjük a WSDL-szerződést, ha a lap tetején levő linkre kattintunk. A WSDL (Web Service Description Language – webszolgáltatás-leíró nyelv) olyan nyelvtan, amely leírja egy adott végponton elhelyezkedő webszolgáltatások szerkezetét.

Forráskód A MagicEightBallServiceHost kódfájlokat a forráskódkönyvtár 25. fejezetének alkönyvtára tartalmazza. A forráskódkönyvtárról lásd a Bevezetés xlv. oldalát.



25.8. ábra: *MEX-en keresztül megtekinthető metaadatok*

WCF-ügyfélalkalmazás készítése

Most, hogy a hoszt készen áll, már csak a szoftver egy részét kell elkészítenünk, hogy a WCF-szolgáltatástípussal kommunikáljon. Bár választhatnánk a szükséges infrastruktúra manuális felépítését (megvalósítható, de munkaigényes feladat), a .NET Framework 3.5 SDK több módszert kínál ügyféloldali proxy gyors generálására. Először hozzunk létre új parancssori alkalmazást MagicEightBallServiceClient névvel.

Proxykód generálása az svcutil.exe segítségével

Ügyféloldali proxy készítéséhez az első módszer az `svcutil.exe` parancssori eszköz használata. Az `svcutil.exe` segítségével a proxykódot és az ügyféloldali konfigurációs fájlt képviselő új C# nyelvű fájlt generálhatunk. Ehhez csak első paraméterként kell megadnunk a szolgáltatás végpontját. Az `/out:` kapcsoló adja meg a proxyt tartalmazó `*.cs` fájl nevét, a `/config:` opció pedig a generált ügyféloldali `*.config` fájl nevét.

Tételezzük fel, hogy fut a szolgáltatásunk. Ekkor a következő, `svcutil.exe`-nek átadott utasításkészlet két új fájlt generál a munkakönyvtárban (amelyet természetesen egy sorban kell megadni a Visual Studio 2008 parancssorában):

```
svcutil http://localhost:8080/MagicEightBallService
            /out:myProxy.cs /config:app.config
```

A `myProxy.cs` fájl megnyitásakor megtaláljuk az `IEightBall` interfész ügyféloldali megfelelőjét, valamint az új `EightBallClient` osztályt, amely maga a proxyosztály. Ez az osztály a `System.ServiceModel.ClientBase<T>` generikus osztályból származik, ahol T a regisztrált szolgáltatásinterfész.

Néhány egyedi konstruktoron kívül minden olyan metódust, amelyhez hozzárendeltük az `[OperationContract]` attribútumot, úgy implementáltunk, hogy az delegálja a szülőosztály `Channels` tulajdonságát a megfelelő külső metódus meghívásához. Nézzünk meg egy kódrészletet a proxytípusból:

```
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.CodeDom.Compiler.GeneratedCodeAttribute(
                          "System.ServiceModel", "3.0.0.0")]
public partial class EightBallClient :
       System.ServiceModel.ClientBase<IEightBall>, IEightBall
{
  ...
```

```
    public string ObtainAnswerToQuestion(string userQuestion)
    {
      return base.Channel.ObtainAnswerToQuestion(userQuestion);
    }
}
```

A proxytípus példányának létrehozásakor az ősosztály kapcsolatot létesít a végponttal az ügyféloldali alkalmazás konfigurációs fájljában található beállítások alapján. A kiszolgálóoldali konfigurációs fájlhoz hasonlóan a generált ügyféloldali `App.config` fájl tartalmaz egy `<endpoint>` elemet, valamint részleteket a szolgáltatással való kommunikációra használt `basicHttpBindingot` illetően.

Ezenkívül itt találjuk a következő `<client>` elemet, amely (újra) létrehozza az ABC-ket a kliens oldaláról:

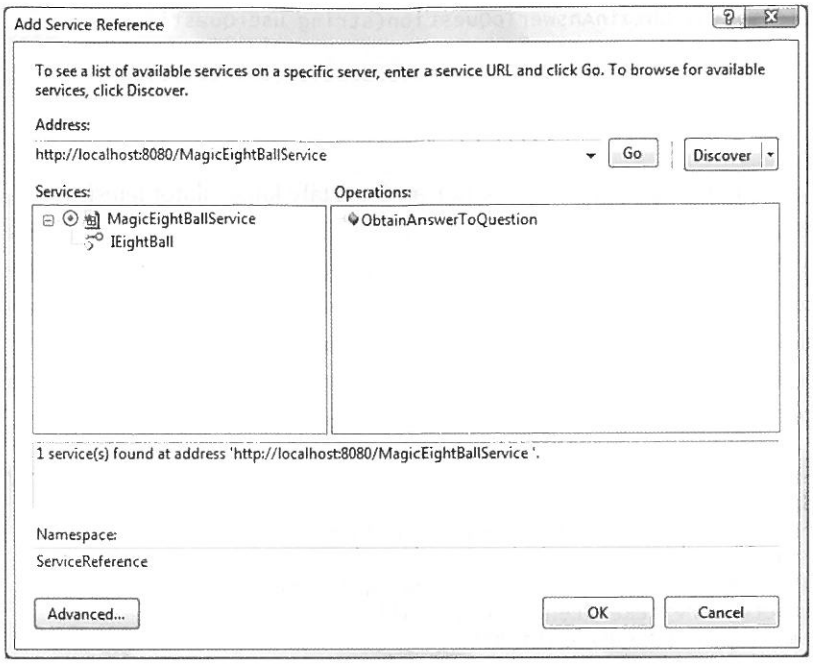
```
<client>
  <endpoint
  address="http://localhost:8080/MagicEightBallService"
  binding="basicHttpBinding"
  bindingConfiguration="BasicHttpBinding_IEightBall"
  contract="ServiceReference.IEightBall"
  name="BasicHttpBinding_IEightBall" />
</client>
```

Ezt a két fájlt hozzáadhatnánk a kliensprojekthez (referenciával a `System.ServiceModel.dll`-re), és kommunikálhatnánk a proxytípus segítségével a távoli WCF-szolgáltatással. Ehelyett nézzük meg, hogyan lehet a Visual Studio még inkább a segítségünkre ügyféloldali proxyfájlok automatikus létrehozásában.

Proxykód generálása Visual Studio 2008-ban

Megfelelő parancssori eszközként az `svcutil.exe` számos lehetőséget kínál ügyféloldali proxy generálásának meghatározására. Ha nincs szükségünk ezekre a speciális beállításokra, akkor a Visual Studio 2008 IDE segítségével generálhatjuk ugyanezt a két fájlt. Válasszuk ki az Add Service Reference pontot a Project menüből.

Ha kiválasztottuk a menüpontot, be kell írnunk a szolgáltatás URI-jét. Ekkor kattintsunk a Go gombra, és megtekinthetjük a szolgáltatás leírását (lásd a 25.9. ábrát).



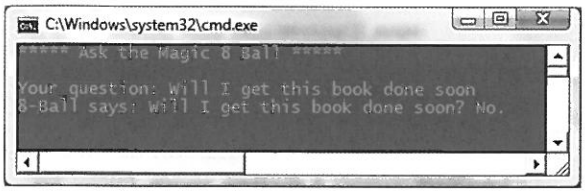
25.9. ábra: *Proxyfájlok generálása Visual Studio 2008 segítségével*

A proxyfájlok létrehozásán és projektbe illesztésén kívül az eszköz automatikusan létrehozza helyettünk a WCF-szerelvények referenciáit. Az elnevezési konvenciók alapján a proxyosztályt a `ServiceReference` névtéren belül definiáljuk, amelyet beágyazunk a kliens névterébe (hogy elkerüljük a lehetséges névütközéseket). A teljes kliensforráskód a következő:

```
// A proxy helye.
using MagicEightBallServiceClient.ServiceReference;

namespace MagicEightBallServiceClient
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("***** Ask the Magic 8 Ball *****\n");
      using (EightBallClient ball = new EightBallClient())
      {
        Console.Write("Your question: ");
        string question = Console.ReadLine();
        string answer =
               ball.ObtainAnswerToQuestion(question);
```

```
            Console.WriteLine("8-Ball says: {0}", answer);
        }
        Console.ReadLine();
    }
  }
}
```

Feltételezve, hogy a WCF-hoszt fut, futtathatjuk a klienst. A 25.10. ábrán egy lehetséges választ látunk.



25.10. ábra: *A kész WCF-ügyfélhoszt*

Forráskód A MagicEightBallServiceClient kódfájlokat a forráskódkönyvtár 25. fejezetének alkönyvtára tartalmazza. A forráskódkönyvtárról lásd a Bevezetés xlv. oldalát.

TCP-alapú kötés konfigurálása

A hoszt- és a kliensalkalmazást úgy állítottuk be, hogy a `basicHttpBindingot`, a legegyszerűbb HTTP-alapú kötést használja. A beállítások konfigurációs fájlokba történő áthelyezésének előnye, hogy deklaratív módon megváltoztathatjuk az alapul szolgáló beállításokat.

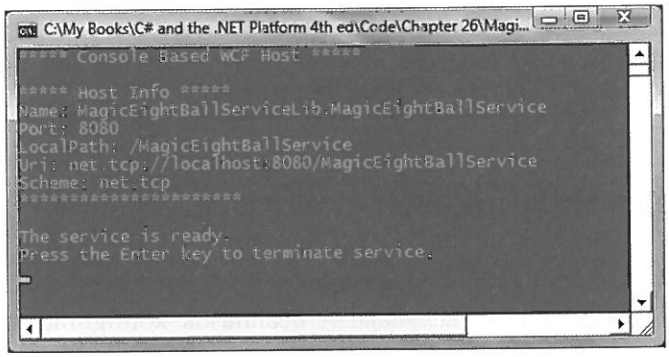
Ennek bemutatására nézzünk meg egy rövid kísérletet. Hozzunk létre egy EightBallTCP könyvtárt a C meghajtón (vagy ahová a kódot mentjük), ebben a mappában pedig hozzunk létre két alkönyvtárt Host és Client néven.

Utána Windows Intézőben keressük meg a hosztprojekt \bin\Debug könyvtárát, és másoljuk a `MagicEightBallServiceHost.exe`, `MagicEightBallServiceHost.exe.config` és a `MagicEightBallServiceLib.dll` fájlokat a C:\EightBallTCP\Host könyvtárba. Nyissuk meg a `*.config` fájlt egyszerű szövegszerkesztőben, és a következőképpen módosítsuk a tartalmát:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
    <service name="MagicEightBallServiceLib.MagicEightBallService">
      <endpoint address =""
```

```
                binding="netTcpBinding"
                contract="MagicEightBallServiceLib.IEightBall"/>
      <host>
        <baseAddresses>
          <add baseAddress =
              "net.tcp://localhost:8080/MagicEightBallService"/>
        </baseAddresses>
      </host>
    </service>
    </services>
  </system.serviceModel>
</configuration>
```

A hoszt `*.config` fájlja lényegében eltakarította az összes MEX-beállítást (mivel már elkészítettük a proxyt), és meghatározta, hogy a `netTcpBinding` kötéstípust használja. Kattintsunk kétszer a `*.exe` fájlra, és futtassuk az alkalmazást. Ha minden jól sikerült, a 25.11. ábrán látható hosztkimenetet látjuk.



25.11. ábra: *TCP-kötéseket alkalmazó WCF-szolgáltatás hosztolása*

A teszt befejezéséhez másoljuk a `MagicEightBallServiceClient.exe` és a `MagicEightBallServiceClient.exe.config` fájlokat a kliensalkalmazás \bin\Debug mappájából a C:\EightBallTCP\Client mappába. Módosítsuk az ügyfélkonfigurációs fájlt a következőképpen:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <client>
      <endpoint
            address="net.tcp://localhost:8080/MagicEightBallService"
            binding="netTcpBinding"
            contract="ServiceReference.IEightBall"
            name="netTcpBinding_IEightBall" />
    </client>
  </system.serviceModel>
</configuration>
```

Ez az ügyféloldali konfigurációs fájl hatalmas egyszerűsítés ahhoz képest, amelyet a Visual Studio proxygenerátora készített. Teljesen eltávolítottuk a létező `<bindings>` elemet. Eredetileg a `*.config` fájl tartalmazta a `<bindings>` elemet a `<basicHttpBinding>` részelemmel, amely a kliens kötési beállításaival kapcsolatos részleteket határozza meg (időtúllépések stb.).

Valójában a példánkban soha nem volt szükség ezekre a beállításokra, mivel automatikusan megkaptuk az alapul szolgáló `BasicHttpBinding` objektum alapértelmezett értékeit. Ha szükség lenne rá, módosíthatnánk a létező `<bindings>` elemet, hogy megadjuk a `<netTcpBinding>` alelem részleteit, de ez nem szükséges, ha elégedettek vagyunk a `NetTcpBinding` objektum alapértelmezett értékeivel.

Ekkor a kliensalkalmazást már futtathatjuk, és ha a hoszt még mindig fut a háttérben, már képesek leszünk a TCP segítségével adatokat mozgatni a szerelvények között.

Forráskód A MagicEightBallTCP kódfájlokat a forráskódkönyvtár 25. fejezetének alkönyvtára tartalmazza. A forráskódkönyvtárról lásd a Bevezetés xlv. oldalát.

A WCF Service Library projektsablon használata

A 22. fejezetben készített AutoLot adatbázisunkkal kommunikáló még különlegesebb WCF-szolgáltatások készítése előtt a következő példa számos fontos témát mutat be, például a WCF Service Library projektsablont , a WCF Test Client alkalmazást, a WCF-konfigurációszerkesztőt, WCF-szolgáltatások Windows-szolgáltatásban történő hosztolásának és az aszinkron ügyfélhívásoknak az előnyeit. Ez a WCF-szolgáltatás szándékosan egyszerű, hogy az új fogalmakra összpontosítsunk.

Egyszerű Matek-szolgáltatás készítése

Először készítsünk egy teljesen új WCF Service Library projektet MathServiceLibrary névvel, de figyeljünk arra, hogy a New Project párbeszédablak WCF-csomópontjánál a megfelelő opciót válasszuk (segítségül lásd a 25.2. ábrát). Változtassuk meg az `IService1.cs` fájl eredeti nevét `IBasicMath.cs`-re.

Ha készen vagyunk, *töröljük* az összes példakódot a `MathServiceLibrary` névtérből, és helyette illesszük be a következő kódot:

```
namespace MathServiceLibrary
{
  [ServiceContract(Namespace="www.Intertech.com")]
  public interface IBasicMath
  {
    [OperationContract]
    int Add(int x, int y);
  }
}
```

Módosítsuk a Service1.cs fájl nevét `MathService.cs`-re, majd (újra) töröljük ki az összes példakódot a `MathServiceLibrary` névtérből, és a következőképpen implementáljuk a szolgáltatássszerződésünket:

```
namespace MathServiceLibrary
{
  public class MathService : IBasicMath
  {
    public int Add(int x, int y)
    {
      // Hogy hosszú kérést szimuláljunk.
      System.Threading.Thread.Sleep(5000);
      return x + y;
    }
  }
}
```

Végül nyissuk meg az `App.config` fájlt, és cseréljük ki az `IService1` összes előfordulását `IBasicMathre`, és a `Service1` összes előfordulását `MathService`-re. Ez a `*.config` fájl már engedélyezi a MEX-támogatását, és az alapértelmezés szerint a `wsHttpBinding` protokollt használja.

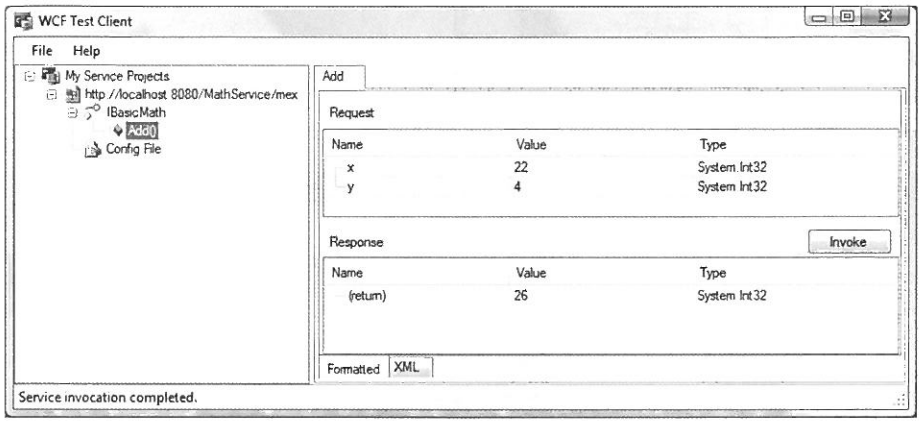
A WCF-szolgáltatás tesztelése a WcfTestClient.exe-vel

A WCF Service Library projekt használatának egyik előnye, hogy hibakeresés-kor vagy futtatáskor kiolvassa a beállításokat a `*.config` fájlból, és ezekkel tölti be a WCF Test Client alkalmazást (`WcfTestClient.exe`). Ezzel a GUI-alapú alkalmazással a WCF-szolgáltatás készítése közben tesztelhetjük a szolgáltatásinterfész minden tagját, ahelyett, hogy manuálisan kellene hosztot klienst készíteni csak a tesztelés kedvéért.

A 25.12. ábra mutatja a `MathService` tesztkörnyezetét. Ha duplán kattintunk egy interfészmetódusra, megadhatunk bemenő paramétereket, és meghívhatjuk a tagot.

Ez az eszköz működésre készen áll, amikor elkészül a WCF Service Library projekt, de tudnunk kell, hogy tesztelhetünk vele bármilyen WCF-szolgáltatást, ha parancssorból indítjuk MEX-végpont megadásával. Ha például szeretnénk elindítani a `MagicEightBallServiceHost.exe` alkalmazást, akkor a Visual Studio 2008 parancssorából a következő parancsot adhatnánk ki:

```
wcftestclient http://localhost:8080/MagicEightBallService
```

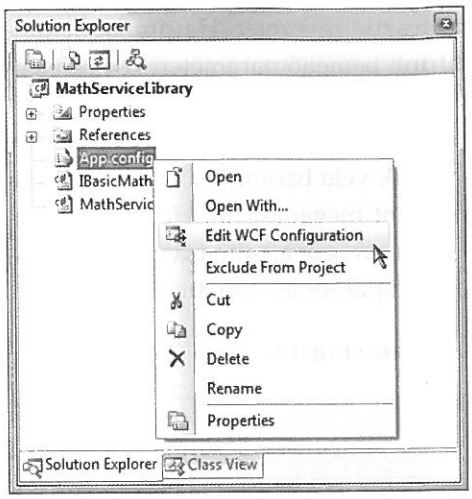


25.12. ábra: *A WCF-szolgáltatás tesztelése WcfTestClient.exe alkalmazással*

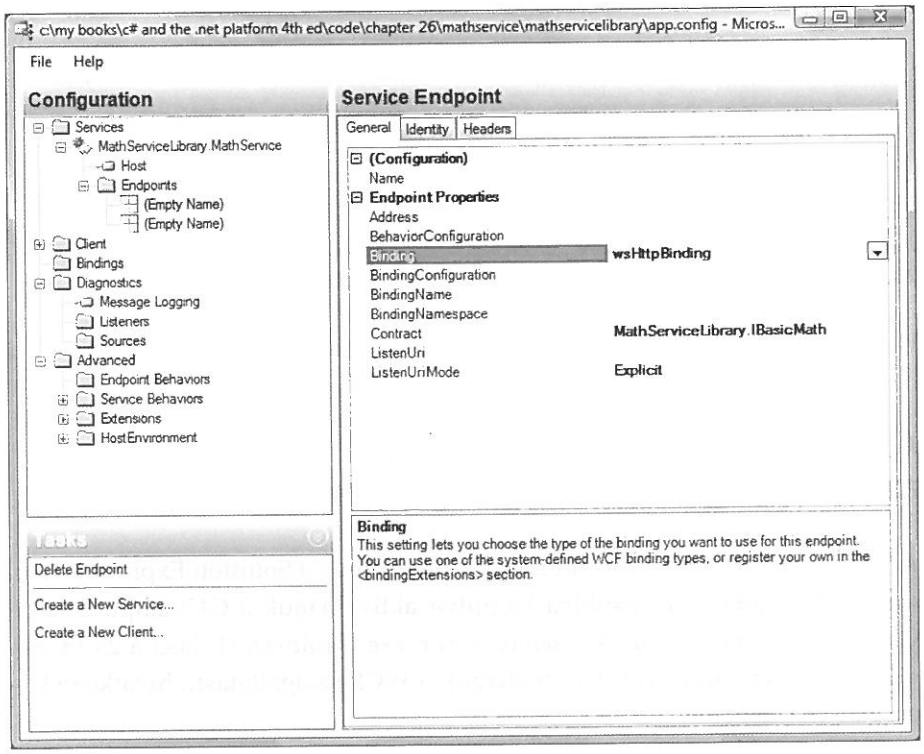
Hasonló módon hívhatjuk meg az `ObtainAnswerToQuestion()` metódust.

A konfigurációs fájl módosítása az SvcConfigEditor.exe programmal

A WCF Service Library projekt másik előnye, hogy a Solution Explorerben az `App.config` fájlra jobb gombbal kattintva aktiválhatjuk a GUI-alapú Service Configuration Editort, az `SvcConfigEditor.exe` alkalmazást (lásd a 25.13. ábrát). Ugyanezt a technikát használhatjuk a WCF-szolgáltatásra hivatkozó kliensalkalmazásból is.



25.13. ábra: *A GUI-alapú *.config fájl szerkesztése itt kezdődik*



25.14. ábra: *A WCF Service Configuration Editor használata*

Az eszköz elindítása után felhasználóbarát környezetben megváltoztathatjuk az XML-alapú adatokat. A `*.config` fájlok karbantartására nyilvánvalóan több szempontból előnyös az ilyen eszköz használata. Először is, biztosak lehetünk abban, hogy a generált címkézés megfelel az elvárt formátumnak, és gépelési hibáktól mentes. Másodszor, kiváló módja annak, hogy adott attribútumokhoz rendelhető érvényes értékeket *lássunk*. Végül, de nem utolsósorban, nincs szükség XML-adatok manuális megírására.

A 25.14. ábra mutatja a Service Configuration Editor általános megjelenését. Valójában hosszasan lehetne az `SvcConfigEditor.exe` által támogatott érdekes lehetőségeket vizsgálni (COM+-integrálás, új `*.config` fájlok létrehozása stb.). Érdemes időt szánni az eszköz tanulmányozására. Az F1 lenyomásával részletes súgórendszer áll a rendelkezésünkre.

Megjegyzés Az `SvcConfigEditor.exe` eszközzel akkor is szerkeszthetünk (vagy létrehozhatunk) konfigurációs fájlokat, ha nem választunk kezdeti WCF Service Library projektet. A Visual Studio 2008 parancssorából indítsuk el az eszközt, és a File>Open menüponttal töltsünk be szerkesztésre egy létező `*.config` fájlt.

Mivel a WCF `MathService`-t nem kell tovább konfigurálni, továbbléphetünk az egyedi hoszt építésére.

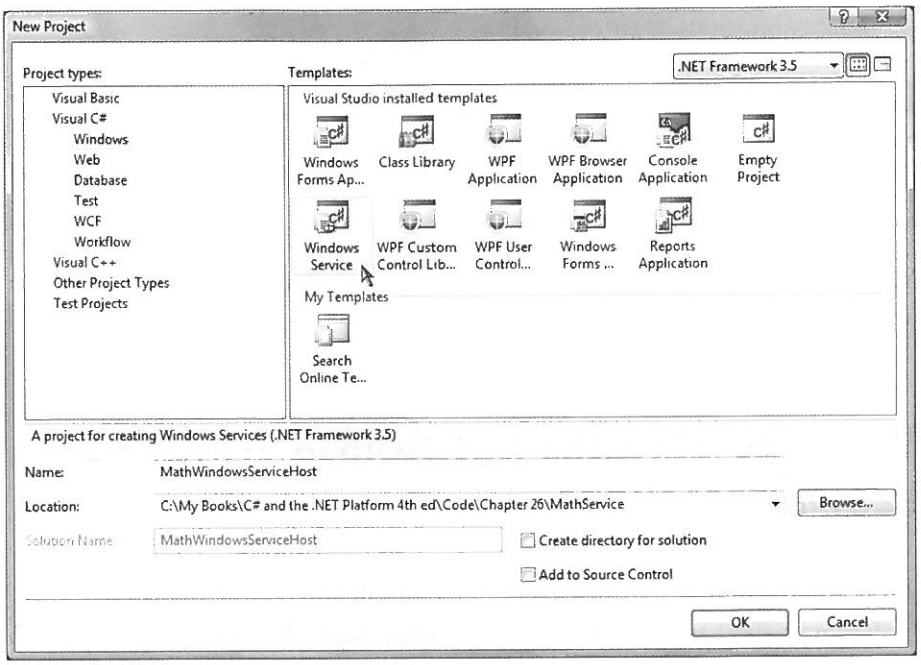
WCF-szolgáltatás hosztolása Windows-szolgáltatásként

A WCF-szolgáltatás hosztolása a parancssori alkalmazásból (vagy GUI asztali alkalmazásból) termékszintű kiszolgálóhoz nem ideális megoldás, mivel a hosztnak láthatóan futnia kell a háttérben, hogy kiszolgálja a kilenseket. Ha letesszük a hosztoló alkalmazást a tálcára, így is túl egyszerű lenne véletlenül leállítani a hosztot és megszakítani a kapcsolatot a kliensalkalmazásokkal.

Megjegyzés Bár igaz, hogy az asztali Windows-alkalmazásnak nem *kell* feltétlenül mutatnia a főablakot, egy tipikus `*.exe`-nek szüksége van felhasználói beavatkozásra a végrehajtható fájl betöltéséhez. Egy Windows-szolgáltatás (lásd a következő részben) úgy is konfigurálható, hogy akkor is fusson, ha pillanatnyilag egy felhasználó sem jelentkezett be a munkaállomásra.

Házon belüli WCF-alkalmazás készítése során másik alternatívaként a WCF-szolgáltatáskönyvtárt a kijelölt Windows-szolgáltatásból is hosztolhatjuk. Ennek egyik előnye az, hogy a Windows-szolgáltatás úgy is konfigurálható, hogy a célzott számítógép indulásakor automatikusan induljon el. Másik előny az, hogy a Windows-szolgáltatás láthatatlanul fut a háttérben (a parancssori alkalmazással ellentétben), és nincs szüksége felhasználói beavatkozásra.

Hogy bemutassuk, hogyan kell ilyen hosztot készíteni, először létrehozunk egy új Windows-szolgáltatásprojektet MathWindowsServiceHost névvel (lásd a 25.15. ábrát). Ha kész van, a Solution Explorerben nevezzük át a kezdeti `Service1.cs` fájlt `MathWinService.cs`-re.



25.15. ábra: *Windows-szolgáltatás készítése a WCF-szolgáltatás hosztolására*

Az ABC-k megadása a forráskódban

Ha beállítottunk referenciát a `MathServiceLibrary.dll` és a `System.ServiceModel.dll` szerelvényre, elég csak a `ServiceHost` típust használni a Windows-szolgáltatástípus `OnStart()` és `OnStop()` metódusaiban. Nyissuk meg a szolgáltatás hosztosztályának kódfájlját (jobb gombbal kattintsunk a tervezőre, és válasszuk a View Code opciót), és adjuk hozzá a következő kódot:

```
// Mindenképpen importáljuk a következő névtereket:
using MathServiceLibrary;
using System.ServiceModel;

public partial class MathWinService: ServiceBase
{
  // ServiceHost típusú tagváltozó.
  private ServiceHost myHost;

  public MathWinService()
  {
    InitializeComponent();
  }

  protected override void OnStart(string[] args)
  {
    // A biztonság kedvéért.
    if (myHost != null)
    {
      myHost.Close();
      myHost = null;
    }

    // Hoszt létrehozása.
    myHost = new ServiceHost(typeof(MathService));

    // ABC-k a kódban!
    Uri address =
        new Uri("http://localhost:8080/MathServiceLibrary");
    WSHttpBinding binding = new WSHttpBinding();
    Type contract = typeof(IBasicMath);

    // Végpont hozzáadása.
    myHost.AddServiceEndpoint(contract, binding, address);

    // Hoszt megnyitása.
    myHost.Open();
  }

  protected override void OnStop()
  {
    // Hoszt leállítása.
    if(myHost != null)
      myHost.Close();
  }
}
```

Bár semmi nem gátol meg abban, hogy konfigurációs fájlt használjunk Windows szolgáltatási hoszt készítésekor a WCF-szolgáltatáshoz, a `*.config` fájl helyett programozottan hozzuk létre a végpontot az `Uri`, `WSHttpBinding` és `Type` osztályok segítségével. Ha készen van az ABC minden összetevője, a hosztot programozottan tájékoztatjuk az `AddServiceEndpoint()` hívásával.

A MEX engedélyezése

Bár a MEX-et programkódból is engedélyezhetnénk, inkább a konfigurációs fájlt választjuk. Adjunk új `App.config` fájlt a a Windows-szolgáltatásprojekthez, amely a következő MEX-beállításokat tartalmazza:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="MathServiceLibrary.MathService"
               behaviorConfiguration = "MathServiceMEXBehavior">

        <!-- A MEX végpont engedélyezése. -->
        <endpoint address="mex"
                  binding="mexHttpBinding"
                  contract="IMetadataExchange" />

        <!-- Ezt hozzá kell adni, hogy a MEX tudja a
             szolgáltatásunk címét -->
        <host>
          <baseAddresses>
            <add baseAddress ="http://localhost:8080/MathService"/>
          </baseAddresses>
        </host>
      </service>
    </services>

    <!-- Viselkedésdefiníció a MEX-nek -->
    <behaviors>
      <serviceBehaviors>
        <behavior name="MathServiceMEXBehavior" >
          <serviceMetadata httpGetEnabled="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Windows-szolgáltatástelepítő létrehozása

Ahhoz, hogy a Windows-szolgáltatást regisztráljuk az operációs rendszerben, a projekthez kell adnunk egy telepítőt, amely tartalmazni fogja a szükséges forráskódot a szolgáltatás regisztrálásához. Ehhez csak kattintsunk jobb gombbal a Windows szolgáltatástervező felületén, és válasszuk az Add Installer lehetőséget (lásd a 25.16. ábrát).