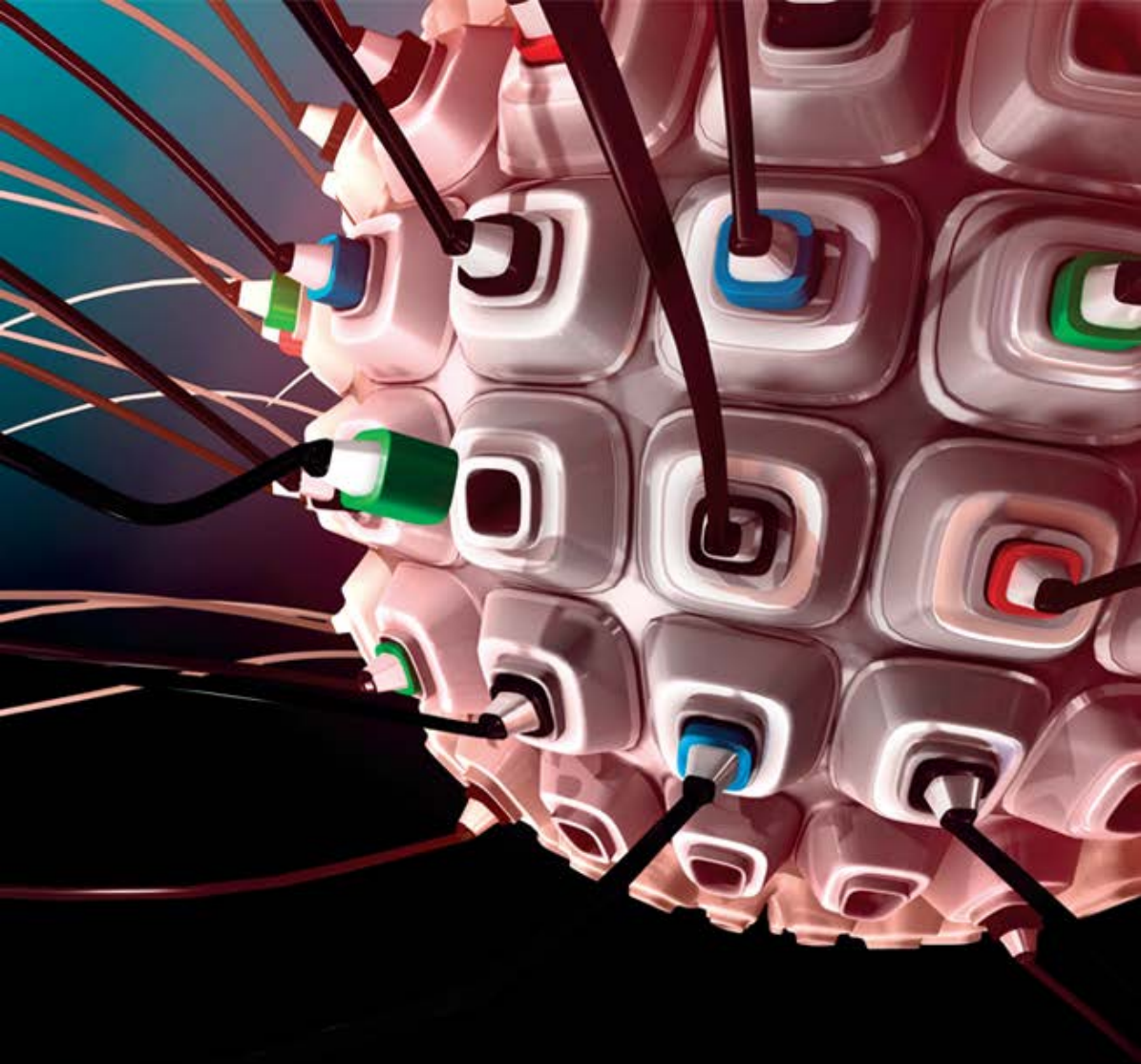


Maciej Grabek

WCF OD PODSTAW

Komunikacja sieciowa nowej generacji

Poznaj i wykorzystaj przyszłość komunikacji internetowej!
Wprowadzenie do Windows Communication Foundation
Konfiguracja usług sieciowych i nowości WCF 4.0
Narzędzia oraz wykorzystanie WCF

Helion

Redaktor prowadzący: Ewelina Burska

Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: http://helion.pl (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie?wfcodp
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Kody źródłowe wykorzystane w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/wfcodp.zip

ISBN: 978-83-246-3094-3

Printed in Poland.

Spis treści

Rozdział 4.

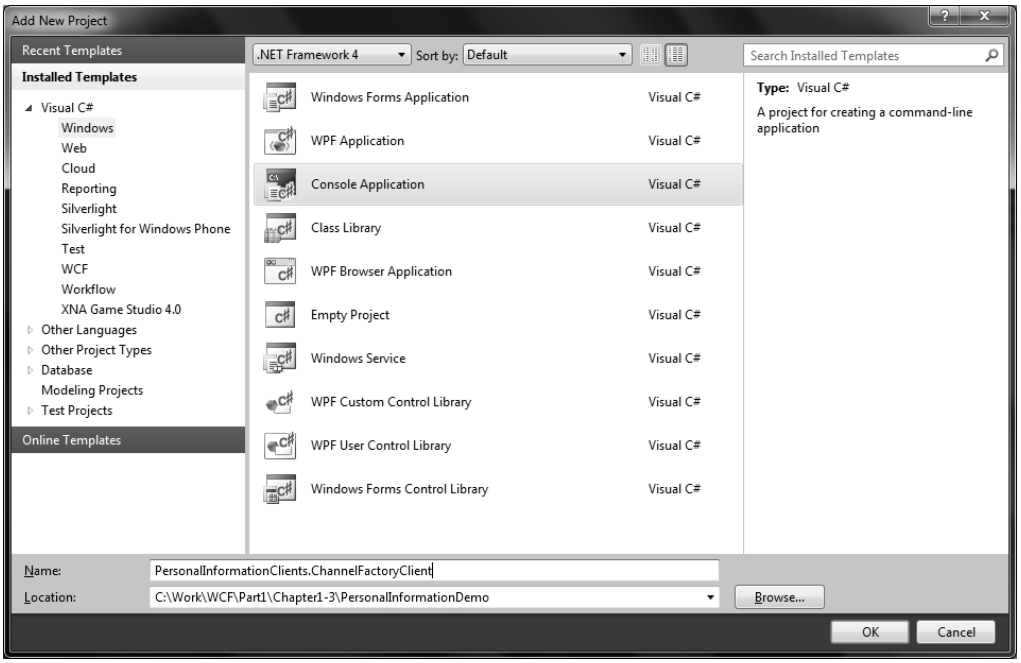
Tworzenie klienta

Trzecim elementem, którym zajmiemy się w tej części książki, jest tworzenie klienta dla naszych usług. Rolę tę pełniło do tej pory narzędzie WCF Test Client, jednakże oczywiste jest, że nie może ono realizować zadań biznesowych stawianych przed systemem, ponieważ służy tylko do testów. Działanie usług bez elementu, który by się do nich odwoływał, mija się z celem, dlatego można pokusić się o stwierdzenie, że aplikacje klienckie są równoprawną częścią wzorca projektowego Service Oriented Architecture. Czym właściwie jest klient usługi? Oczywiście, najczęściej jest to aplikacja, która prezentuje dane użytkownikowi końcowemu. Równie popularnym klientem może być proces działający w naszym systemie, a także odpowiedzialny za cykliczne wywoływanie odpowiednich procesów i zadań biznesowych udostępnianych przez nasze usługi. Klientem może być również inna usługa — mamy wówczas do czynienia z wielowarstwową architekturą, w której usługodawca jest też usługobiorcą. W tym rozdziale prześledzimy proces tworzenia klienta oraz zastanowimy się nad różnymi sposobami podejścia do tego zagadnienia.

Channel Factory

Pierwszym sposobem tworzenia klienta, jakim się zajmiemy, jest wykorzystanie `ChannelFactory<T>`. Klasa ta jest w stanie wygenerować odpowiednie proxy do połączenia z serwisem, jednakże ma pewne ograniczenie. Jest nim konieczność posiadania przez klienta definicji klas opisujących kontrakt usługi. W tworzonym przez nas projekcie taki dostęp jest zapewniony, zatem możemy pokusić się o przetestowanie tego sposobu. W związku z tym do solucji dodamy nowy projekt o nazwie *PersonalInformationClients.ChannelFactoryClient*; skorzystamy z szablonu aplikacji konsolowej, co widać na rysunku 4.1.

Klasa `ChannelFactory` znajduje się w przestrzeni nazw `System.ServiceModel`, zatem do projektu dołączamy taką właśnie referencję. Jednocześnie musimy również umożliwić aplikacji klienckiej dostęp do definicji kontraktu usługi, co wymaga dodania referencji do projektu *PersonalInformationService*. Hostem, do którego będziemy się podłączać,



Rysunek 4.1. *Nowy projekt klienta oparty na aplikacji konsolowej*

będzie *PersonalInformationHosts.WebHost*. Można też skorzystać z pozostałych projektów hostujących usługę, jednakże ten jest uruchamiany domyślnie przy starcie procesu debugowania tworzonego przez nas klienta, zatem nie będzie trzeba pamiętać o wystartowaniu hosta. Dlaczego tak się dzieje? Visual Studio, w momencie gdy rozpoczynane jest debugowanie, przegląda solucję w poszukiwaniu projektów, które bazują na serwerze WWW, a następnie uruchamia je za pomocą wbudowanego serwera WWW, co widać w pasku zadań obok zegara. W naszym przypadku dotyczyć to będzie projektu *PersonalInformationHosts.WebHost*.

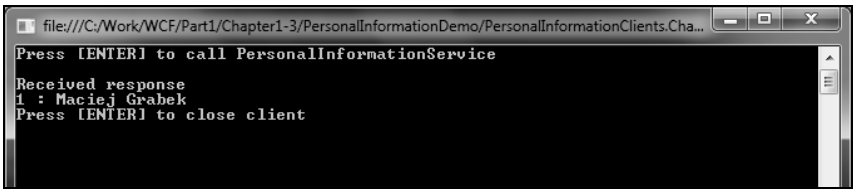
Korzystanie z `ChannelFactory` jest stosunkowo proste. Jednym z konstruktorów, których możemy użyć, jest konstruktor przyjmujący dwa parametry: typ wiązania oraz adres punktu końcowego. Aby zainicjować te wartości w poprawny sposób, musimy odwołać się do pliku konfiguracyjnego usługi. Widoczny na listingu 3.18 plik konfiguracyjny zawiera informacje o tym, że usługa jest dostępna poprzez wiązanie `WsHttp`
`↪Binding`. Adres, na jakim jest dostępna, to *http://localhost:9000/PersonalInformation Service.svc*. W związku z tym w celu uruchomienia klienta możemy użyć kodu widocznego na listingu 4.1. Kod ten na podstawie znanego nam interfejsu (w tym przypadku jest to `IPersonalInformationService`) tworzy obiekt fabryki (`ChannelFactory`), który inicjuje kanał komunikacji z serwisem. Tak utworzone proxy daje następnie możliwość wywoływania potrzebnych metod udostępnianych przez serwis. Oczywiście, jest to model w pełni obiektowy, dzięki czemu korzystanie z serwisów jest intuicyjne i wygodne.

Listing 4.1. *Wykorzystanie ChannelFactory*

```
Uri baseAddr = new Uri("http://localhost:9000/PersonalInformationService.svc/");
ChannelFactory<IPersonalInformationService> factory =
```

```
    new ChannelFactory<IPersonalInformationService>(new WSHttpBinding(),
    ↪new EndpointAddress(baseAddr));
IPersonalInformationService proxy = factory.CreateChannel();
var response = proxy.GetPersonalInformation(
    new PersonalInformationService.Messages.PersonalInformationRequest()
    {
        PersonId = 1
    });
```

Po uruchomieniu tak przygotowanego kodu otrzymujemy wynik widoczny na rysunku 4.2.



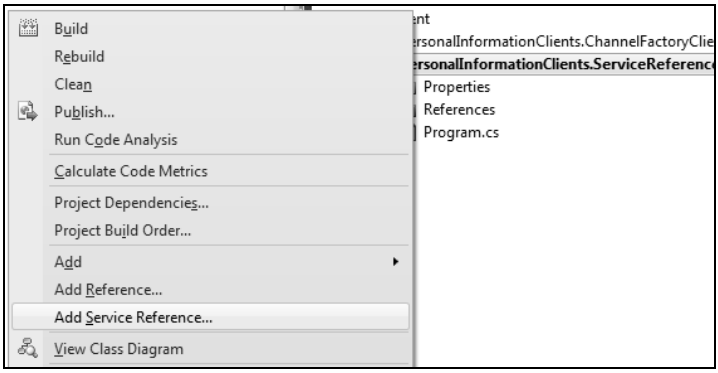
Rysunek 4.2. *Wynik wywołania klienta opartego na ChannelFactory*

Service Reference

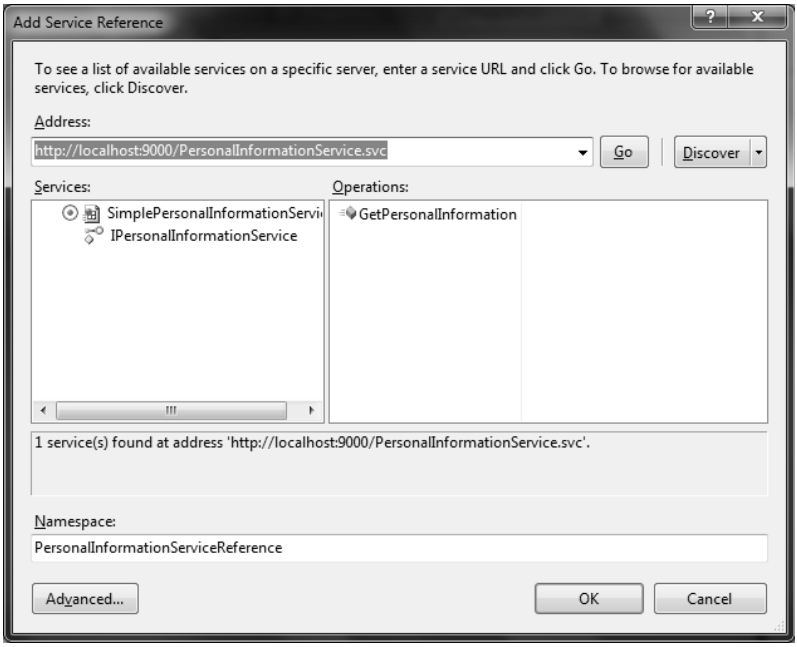
Wykorzystanie `ChannelFactory` do generowania klienta ma jedną wadę: podczas tworzenia projektu niezbędne jest posiadanie biblioteki zawierającej kontrakt. Taka sytuacja z przyczyn oczywistych nie zawsze jest możliwa. Wyobraźmy sobie, że rozważany przez nas serwis jest udostępniany przez firmę trzecią na jej serwerach. W takim przypadku nie ma szansy na uzyskanie dostępu do bibliotek opisujących kontrakty usług, a nawet jeżeli taki dostęp jest możliwy, to z dużymi trudnościami. W związku z tym istnieje druga możliwość utworzenia klienta. Polega ona na skorzystaniu z wbudowanych w Visual Studio narzędzi, które generują klasy niezbędne do komunikacji z usługą na podstawie metadanych. Podobnie jak poprzednio, do solucji dodamy nowy projekt będący aplikacją konsolową, który posłuży do utworzenia klienta w bardziej elastyczny sposób. Nowy projekt nazwiemy *PersonalInformationClients.ServiceReferenceClient*. Aby można było odwołać się do usługi, dodajemy do niej referencję, wybierając w menu kontekstowym projektu opcję *Add Service Reference*..., tak jak widać to na rysunku 4.3.

Po jej wybraniu pojawi się okno widoczne na rysunku 4.4.

W tym momencie mamy dwie możliwości. Pierwsza z nich polega na użyciu przycisku *Discover*, który powoduje przeanalizowanie solucji w poszukiwaniu zdefiniowanych serwisów. Podobnie jak w przypadku `ChannelFactory`, potrzebny jest kod źródłowego usługi, a takiej sytuacji nie rozważamy. W związku z tym posłużymy się drugą opcją, która polega na podaniu znanego nam adresu usługi (*http://localhost:9000/PersonalInformationService.svc*), a następnie wciśnięciu przycisku *Go*. W tym momencie zostaje nawiązane połączenie z usługą oraz pobranie informacji na jej temat poprzez zdefiniowany po stronie serwisu punkt końcowy MEX. Po przetworzeniu tych danych zostają wyświetlone struktura usługi oraz oferowane przez nią kontrakty i metody. Warto



Rysunek 4.3. *Opcja dodawania referencji do usługi*

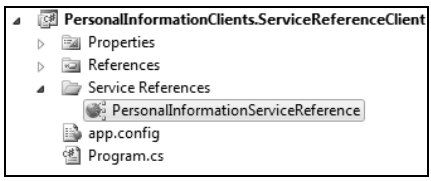


Rysunek 4.4. *Okno dodawania referencji*

jeszcze wspomnieć, że bez dostępnego punktu końcowego odpowiedzialnego za metadane metoda ta nie może zostać zastosowana. W wielu sytuacjach stosuje się wręcz polityki bezpieczeństwa nakazujące usuwanie na serwerach produkcyjnych punktów końcowych tego typu, co w pewnym stopniu umożliwia dodatkowe ich zabezpieczenie przed utworzeniem „niechcianych" aplikacji klienckich. Przed zatwierdzeniem dodawania referencji warto również zmienić domyślną przestrzeń nazw, którą jest `Service➥Reference1`, na bardziej znaczącą, np. `PersonalInformationServiceReference`. W wyniku tej operacji do projektu jest dodawany nowy element widoczny na rysunku 4.5.

W pliku konfiguracyjnym *app.config* pojawiły się również wpisy potrzebne do nawiązania komunikacji z usługą. Istotną w tym momencie częścią pliku jest sekcja `client`, która zawiera informacje i konfiguracje punktów końcowych, które są zdefiniowane w serwisie. Przykładowa treść takiej sekcji widoczna jest na listingu 4.2.

Rysunek 4.5. *Referencja do serwisu dodana do projektu*



Listing 4.2. *Sekcja client w konfiguracji usługi po stronie klienta*

```
<client>
   <endpoint address="http://localhost:9000/PersonalInformationService.svc"
      binding="wsHttpBinding" bindingConfiguration="WSHttpBinding_
      ↪IPersonalInformationService"
      contract="PersonalInformationServiceReference.IPersonalInformationService"
      name="WSHttpBinding_IPersonalInformationService">
   </endpoint>
</client>
```

Teraz nic nie stoi na przeszkodzie, aby odwołać się do usługi. W tym celu korzystamy z kodu widocznego na listingu 4.3.

Listing 4.3. *Korzystanie z referencji do serwisu*

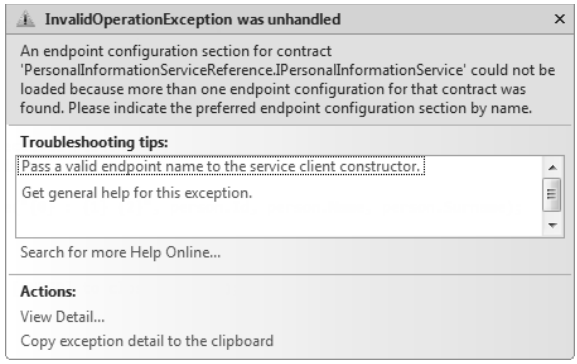
```
using (PersonalInformationServiceClient client = new
PersonalInformationServiceClient())
{
   var response = client.GetPersonalInformation(
      new PersonalInformationRequest()
      {
         PersonId = 1
      });

   Console.WriteLine("Received response");
   foreach (var person in response.Persons)
   {
      Console.WriteLine("{0} : {1} {2}", person.Id, person.Name, person.Surname);
   }
}
```

Klasą, która w tym przypadku służy do komunikacji, jest `PersonalInformationService ↪Client`. Wykorzystanie jej wewnątrz klauzuli `using` daje pewność, że na koniec pracy z usługą nawiązane połączenie zostanie poprawnie zamknięte. Wynik działania aplikacji klienckiej jest w tym momencie dokładnie taki sam jak na rysunku 4.2, zmienia się tylko sposób działania, a nie sposób prezentacji wyników.

Gdy określono więcej niż jeden punkt końcowy usługi, z której będziemy korzystać, należy podać jego nazwę jako parametr konstruktora klasy klienta. Inaczej próba uruchomienia tworzonego projektu skończy się komunikatem o błędzie działania widocznym na rysunku 4.6.

Aby wygenerować taki błąd, wystarczy dodać do serwisu nowy punkt końcowy widoczny na listingu 4.4, a do konfiguracji klienta dołączyć zawartość listingu 4.5.



Rysunek 4.6. *Błąd niezdefiniowanej nazwy konfiguracji punktu końcowego*

Listing 4.4. *Dodatkowy punkt końcowy po stronie serwera usługi*

```
<endpoint address="basic" binding="basicHttpBinding" contract=
➥"PersonalInformationService.ServiceContracts.IPersonalInformationService"/>
```

Listing 4.5. *Dodatkowy punkt końcowy po stronie klienta usługi*

```
<endpoint address="http://localhost:9000/PersonalInformationService.svc/basic"
➥binding="basicHttpBinding" contract="PersonalInformationServiceReference.
➥IPersonalInformationService" name="BasicHttpBinding_IPersonalInformationService">
➥</endpoint>
```

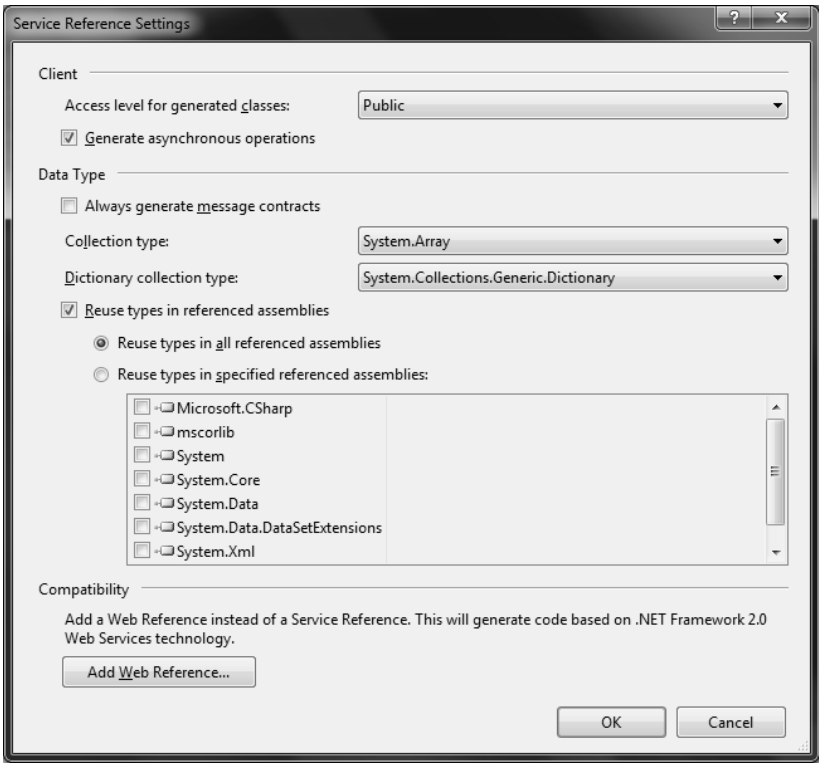
Aby po takich modyfikacjach poprawnie uruchomić klienta usługi, należy w konstruktorze wygenerowanej klasy `PersonalInformationServiceClient` podać nazwę konfiguracji, z której będziemy korzystać. W tym przypadku mamy dwie możliwości: `WSHttp➥Binding_IPersonalInformationService` lub `BasicHttpBinding_IPersonalInformation➥Service`. Po zastąpieniu konstruktora bezparametrowego wywołaniem widocznym na listingu 4.6 uzyskujemy poprawne działanie aplikacji klienckiej.

Listing 4.6. *Konstruktor proxy wykorzystujący parametr nazwy konfiguracji punktu końcowego*

```
new PersonalInformationServiceClient("BasicHttpBinding_
➥IPersonalInformationService")
```

Wywołania asynchroniczne

Kolejną zaletą zaprezentowanego podejścia do tworzenia klienta jest fakt, że w prosty sposób generowane proxy może również udostępnić asynchroniczne wywołania metod usługi. Dzięki nim nie będzie konieczności blokowania interfejsu użytkownika, czy też głównego wątku aplikacji, do czasu otrzymania odpowiedzi z serwisu. W celu zaprezentowania wywołań asynchronicznych utworzymy nowy projekt aplikacji konsolowej o nazwie *PersonalInformationClients.AsyncClient*. Podobnie jak poprzednio, dodajemy referencję do serwisu, jednakże w oknie widocznym na rysunku 4.4 wybieramy opcję *Advanced*. W rezultacie otwiera się kolejne okno dialogowe widoczne na rysunku 4.7.



Rysunek 4.7. *Okno zaawansowanych ustawień referencji do serwisu*

Istotna jest tu opcja *Generate asynchronous operations*. Dzięki niej tworzone automatycznie klasy będą zawierać również logikę niezbędną do asynchronicznego wywoływania metod usługi. Po jej zaznaczeniu i zatwierdzeniu wyboru w projekcie, podobnie jak poprzednio, pojawiają się nowe elementy odpowiedzialne za komunikację z usługą.

Po takim utworzeniu i skonfigurowaniu proxy każda z metod oferowanych przez serwis posiada również swój odpowiednik z postfiksem `Async` oraz zdarzenie `*Completed`, gdzie gwiazdka oznacza nazwę metody, która należy do kontraktu. Zdarzenie to pozwala na wpięcie się w moment, gdy aplikacja kliencka otrzyma odpowiedź od serwisu w celu jej dalszego przetworzenia. W związku z tym przed wywołaniem odpowiedniej metody musimy dodać również obsługę takiego zdarzenia. Pełny kod asynchronicznego odwołania do serwisu prezentuję na listingu 4.7.

Listing 4.7. *Asynchroniczne wywołanie metody udostępnionej przez serwis*

```
Console.WriteLine("Press [ENTER] to call PersonalInformationService");
Console.ReadLine();
using (PersonalInformationServiceClient client = new PersonalInformationService
↪Client("BasicHttpBinding_IPersonalInformationService"))
{
   client.GetPersonalInformationCompleted += new EventHandler
   ↪<GetPersonalInformationCompletedEventArgs>
   ↪(client_GetPersonalInformationCompleted);
```

```
    client.GetPersonalInformationAsync(
       new PersonalInformationRequest()
       {
          PersonId = 1
       });
}

Console.WriteLine("Press [ENTER] to close client");
Console.ReadLine();
```

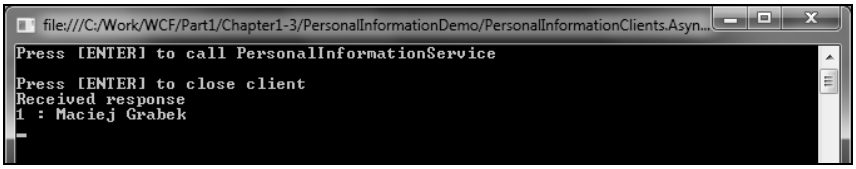
W przypadku komunikacji dwukierunkowej (pytanie — odpowiedź, ang. *request — reply*) niezbędne może okazać się też przetworzenie odpowiedzi otrzymanej od serwisu. W tym celu należy odpowiednio obsłużyć dane dostępne w metodzie podpiętej pod zdarzenie `Completed`. Niezbędny do tego kod widoczny jest na listingu 4.8.

Listing 4.8. *Obsługa odebranej asynchronicznie odpowiedzi*

```
static void client_GetPersonalInformationCompleted(object sender,
➥GetPersonalInformationCompletedEventArgs e)
{
   Console.WriteLine("Received response");
   foreach (var person in e.Result.Persons)
   {
      Console.WriteLine("{0} : {1} {2}", person.Id, person.Name, person.Surname);
   }
}
```

Aby uzyskać dostęp do danych zwracanych przez serwis, należy odwołać się do argumentu `e` oraz jego pola *Result*, które przechowuje wartość odpowiedniego typu; w tym przypadku będzie to typ `PersonalInformationResponse`.

Uruchomienie tak przygotowanego kodu generuje wynik widoczny na rysunku 4.8.



Rysunek 4.8. *Wynik działania asynchronicznego klienta*

Zgodnie z oczekiwaniem główny wątek programu nie został zatrzymany na czas komunikacji z serwisem i dotarł do komunikatu końcowego. Odpowiedź, którą otrzymał klient, została przetworzona i wyświetlona dopiero na samym końcu. Z jednej strony, zapewnia to dużo większą responsywność interfejsu użytkownika, z drugiej jednak, wymaga kilku dodatkowych zabiegów w celu poprawnego obsłużenia i dobrej prezentacji wyników komunikacji z usługą.

Istnieje również możliwość dodania referencji w sposób znany z wcześniejszych odsłon .NET Framework, czyli jako referencji Web, jednakże daje ona dużo mniejsze pole do manewru w porównaniu z WCF. Jest to możliwe po wybraniu opcji *Add Web Reference* widocznej na rysunku 4.7.

Wady i zalety

Podstawową wadą `ChannelFactory` jest konieczność posiadania dostępu do skompilowanej biblioteki zawierającej kontrakt usługi. Powoduje to zmniejszenie elastyczności projektu w systemach opartych na wielu dostawcach. Problem ten znika w przypadku wykorzystania referencji do serwisu. Dodatkową przewagą drugiego rozwiązania jest łatwość dodania metod asynchronicznych do komunikacji z usługą.

Warto zapamiętać

Gdy istnieje kilka punktów końcowych danej usługi, w kodzie aplikacji klienckiej należy użyć konstruktora z parametrem będącym nazwą punktu końcowego wybranego do komunikacji.

Korzystanie z referencji do serwisu daje możliwość prostego wygenerowania metod asynchronicznych odwołujących się do usługi.

Podsumowanie

W rozdziale zademonstrowałem dwa sposoby korzystania z serwisów utworzonych przy użyciu technologii WCF. W zależności od konkretnego zastosowania mamy do wyboru odmienne metody, które dają różne możliwości. Oczywiście, nie są to jedyne metody, bo istnieje możliwość odwołania do usług bez konieczności dodawania jakichkolwiek referencji od nich. Możemy odwołać się do usługi, korzystając z JavaScriptu lub odpowiedniej konstrukcji adresu URL, jak ma to miejsce m.in. w przypadku standardu OData i WCF Data Services. Temat ten podejmę jeszcze w części piątej dotyczącej wykorzystania WCF.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Windows Communication Foundation to technologia umożliwiająca komunikację pomiędzy praktycznie wszystkimi systemami wymiany danych, przeznaczona przede wszystkim dla tych użytkowników, których główną potrzebą jest zachowanie dużej elastyczności usług sieciowych. Technologia ta jest nadrzędna wobec wszystkich wcześniejszych rozwiązań Microsoftu w tej dziedzinie, zapewnia zgodność z innymi standardami, pozwala też używać różnych protokołów (m.in. HTTP, TCP) bez konieczności ingerowania w sam kod aplikacji. Jeśli chcesz uniknąć problemów związanych z koniecznością ciągłego dostosowywania się do wymagań klientów bądź współpracowników używających różnych kanałów komunikacji sieciowej, pora zapoznać się z możliwościami oferowanymi przez technologię WCF.

Książka *WCF od podstaw. Komunikacja sieciowa nowej generacji* składa się z pięciu części zawierających najważniejsze zagadnienia związane z technologią WCF. Znajdziesz w niej informacje na temat samej technologii i jej elementów oraz poczytasz o rozmaitych aspektach i wariantach konfiguracji. Dowiesz się, jakie nowości oferuje WCF 4.0, a także poznasz narzędzia używane podczas pracy z tą technologią, w tym Visual Studio, WCF Service Host i WCF Test Client. Ostatnim, choć wcale nie najmniej ważnym omówionym obszarem będzie kwestia wykorzystania serwisów WCF w kontekście standardu OData, biblioteki jQuery i aplikacji Silverlight. Lektura tej pozycji uświadomi Ci, jak wiele tracisz, nie używając na co dzień technologii WCF, a przede wszystkim, jak wiele możesz zyskać!

- Czym jest WCF i dlaczego warto stosować SOA?
- Definiowanie kontraktu usługi i hostowanie usługi
- Tworzenie klienta
- Podstawowa i zaawansowana konfiguracja
- Routing Service, Service Discovery, Workflow w WCF
- Visual Studio, WCF Service Host, WCF Test Client
- SVC Util, Microsoft Service Configuration Editor
- OData i WCF, WCF RIA Services
- Wydajność

Twórz przejrzysty kod, wykorzystując nowoczesne, elastyczne i wygodne w użyciu rozwiązanie WCF!

Helion

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

ISBN: 978-83-246-3094-3
9 788324 630943
Cena: 49,00 zł

Informatyka w najlepszym wydaniu