Ajax. Zaawansowane programowanie

Autorzy: Nicholas C. Zakas, Jeremy McPeak, Joe Fawcett
Tłumaczenie: Jaromir Senczyk
ISBN: 83-246-0567-3
Tytuł oryginału: Professional Ajax
Format: B5, stron: 432

Napisz ergonomiczne i wydajne aplikacje internetowe

- Poznaj metody komunikacji w technologii Ajax
- Wykorzystaj wzorce projektowe
- Stwórz komponenty i stosuj je w swoich projektach

Dynamiczny rozwój internetu, języka HTML, technologii serwerowych i multimedialnych sprawił, że witryny WWW stały się dziełami sztuki, wypełnionymi animacjami, grafiką i dźwiękiem. Nadal jednak po kliknięciu łącza lub przycisku nawigacyjnego musimy czekać na załadowanie się nowej treści z serwera. Bazując na języku JavaScript i jego możliwości stosowania asynchronicznych żądań HTTP służących do pobierania danych z serwera bez konieczności przeładowania strony WWW, opracowano technologię, która pozwala na wyeliminowanie tej niedogodności. Nosi nazwę Ajax, a po jej zastosowaniu witryny i aplikacje WWW pod względem obsługi coraz bardziej przypominają tradycyjne programy.

Książka „Ajax. Zaawansowane programowanie" opisuje możliwości technologii i sposoby tworzenia aplikacji internetowych z jej zastosowaniem. Czytając ją, dowiesz się, jak powstał Ajax i gdzie jest wykorzystywany. Zrozumiesz, na czym polega technika „ukrytej ramki" i pobierania danych w tle, a także poznasz wzorce projektowe dla aplikacji budowanych w oparciu o Ajax. Nauczysz się przetwarzać pliki XML, pobierać kanały RSS i tworzyć usługi sieciowe wykorzystujące protokół SOAP. Napiszesz przeglądarkę poczty i komponenty noszące nazwę widgetów, które będziesz mógł zastosować na innych witrynach WWW. Znajdziesz tu również informacje o najpopularniejszych frameworkach wspomagających pracę twórców aplikacji Ajax.

- Struktura aplikacji Ajax
- Komunikacja aplikacji Ajax z serwerem
- Wzorce projektowe
- Przetwarzanie plików XML
- Tworzenie usług WWW
- Korzystanie z JSON
- Tworzenie widgetów
- Frameworki dla Ajaksa

Zdobądź praktyczną wiedzę niezbędną do tworzenia aplikacji Ajax

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

O autorach 9

Wprowadzenie 11

Rozdział 1. Czym jest Ajax? 17

Narodziny Ajax 18
Ewolucja sieci WWW 18
JavaScript 19
Ramki 19
Technika ukrytej ramki 19
Dynamiczny HTML i model DOM 20
Ramki iframe 20
XMLHttp 21
Prawdziwy Ajax 21
Zasady tworzenia aplikacji Ajax 22
Technologie używane przez Ajax 23
Kto używa technologii Ajax? 24
Google Suggest 24
Gmail 25
Google Maps 26
A9 27
Yahoo! News 28
Bitflux Blog 29
Kontrowersje i nieporozumienia 29
Podsumowanie 31

Rozdział 2. Podstawy Ajax 33

Elementarz HTTP 33
Żądania HTTP 34
Odpowiedzi HTTP 36
Techniki komunikacyjne Ajax 37
Technika ukrytej ramki 37
Żądania wysyłane przez obiekt XMLHttp 51

Dalsze rozważania ... 62
Polityka tego samego pochodzenia ... 62
Sterowanie buforowaniem ... 63
Podsumowanie ... 64

Rozdział 3. Wzorce Ajax ... 65

Wprowadzenie ... 65
Wzorce sterowania komunikacją ... 66
Pobieranie predykcyjne ... 66
Przykład wstępnego ładowania strony ... 67
Dławienie wysyłania ... 74
Przykład stopniowej kontroli zawartości formularza ... 75
Przykład stopniowej weryfikacji pola ... 83
Okresowe odświeżanie ... 86
Przykład powiadamiania o nowym komentarzu ... 87
Ładowanie wieloetapowe ... 92
Przykład łączy prowadzących do dodatkowych informacji ... 92
Wzorce zachowania w przypadku błędu ... 95
Odwoływanie oczekujących żądań ... 95
Wzorzec ponownej próby ... 97
Podsumowanie ... 98

Rozdział 4. XML, XPath i XSLT ... 101

Obsługa XML w przeglądarkach ... 101
XML DOM w przeglądarce IE ... 101
XML DOM w przeglądarce Firefox ... 111
XML w różnych przeglądarkach ... 115
Przykład wykorzystania XML ... 116
Obsługa XPath w przeglądarkach ... 123
Wprowadzenie do XPath ... 123
XPath w przeglądarce IE ... 124
Posługiwanie się przestrzeniami nazw ... 125
XPath w przeglądarce Firefox ... 128
Funkcja rozwiązująca przedrostki przestrzeni nazw ... 129
XPath w różnych przeglądarkach ... 130
Obsługa przekształceń XSL w przeglądarkach ... 132
Wprowadzenie do XSLT ... 132
XSLT w przeglądarce IE ... 135
XSLT w przeglądarce Firefox ... 139
XSLT w różnych przeglądarkach ... 141
Przykład z bestsellerami raz jeszcze ... 141
Podsumowanie ... 144

Rozdział 5. Syndykacja treści — RSS/Atom ... 147

RSS ... 147
RSS 0.91 ... 148
RSS 1.0 ... 149
RSS 2.0 ... 150
Atom ... 150
FOOReader.NET ... 151
Komponenty po stronie klienta ... 152
Komponenty serwera ... 162
Wiązanie klienta z serwerem ... 169

Konfiguracja ... 176
Testowanie ... 177
Podsumowanie ... 179

Rozdział 6. Usługi WWW ... 181

Technologie ... 181
SOAP ... 181
WSDL ... 184
REST ... 188
Platforma .NET i SOAP ... 191
Decyzje podczas projektowania ... 191
Tworzenie usługi WWW w Windows ... 192
Wymagania systemowe ... 192
Konfiguracja serwera IIS ... 193
Tworzenie kodu ... 194
Tworzenie kodu wykonywalnego ... 196
Usługi WWW i Ajax ... 199
Tworzenie środowiska testowego ... 200
Rozwiązanie dla przeglądarki Internet Explorer ... 201
Rozwiązanie dla przeglądarki Mozilla ... 203
Rozwiązanie uniwersalne ... 206
Usługi WWW pomiędzy domenami ... 208
Interfejs usług Google ... 208
Konfiguracja proxy ... 209
Podsumowanie ... 213

Rozdział 7. JSON ... 215

Czym jest JSON? ... 215
Literały tablic ... 215
Literały obiektów ... 216
Literały mieszane ... 217
Składnia JSON ... 218
Kodowanie i dekodowanie danych JSON ... 219
JSON kontra XML ... 219
Narzędzia JSON działające na serwerze ... 221
JSON-PHP ... 221
Inne narzędzia ... 223
Pole tekstowe z automatycznymi podpowiedziami ... 223
Przegląd funkcjonalności ... 224
HTML ... 224
Tabela bazy danych ... 226
Architektura ... 227
Klasy ... 228
Kontrolka AutoSuggestControl ... 228
Dostawca podpowiedzi ... 245
Komponent serwera ... 247
Komponent klienta ... 248
Podsumowanie ... 250

Rozdział 8. Widgety WWW 251
W telegraficznym skrócie 251
Komponent serwera 252
Komponent klienta 253
Styl wiadomości 261
Implementacja widgetu paska wiadomości 263
Widget informacji pogodowych 264
Weather.com SDK 264
Komponent serwera 265
Komponent klienta 273
Pobieranie danych z serwera 273
Indywidualizacja widgetu pogody 274
Wdrożenie widgetu informacji pogodowych 278
Widget wyszukiwania 279
Komponent serwera 280
Komponent klienta 281
Indywidualizacja widgetu wyszukiwania 286
Wdrożenie widgetu wyszukiwania w sieci 288
Widget przeszukiwania witryny 289
Komponent serwera 290
Komponent klienta 296
Indywidualizacja widgetu przeszukiwania witryny 301
Wdrożenie widgetu przeszukiwania 303
Podsumowanie 304

Rozdział 9. AjaxMail 305
Wymagania 305
Architektura 306
Wykorzystane zasoby 306
Tabele bazy danych 307
Plik konfiguracyjny 308
Klasa AjaxMailbox 309
Wykonywanie akcji 331
Interfejs użytkownika 337
Widok katalogu 340
Widok odczytu 342
Widok kompozycji 344
Układ 346
Kompletowanie aplikacji 346
Funkcje pomocnicze 348
Mailbox 349
Funkcje wywoływane zwrotnie 367
Procedury obsługi zdarzeń 369
Ostatni etap 370
Podsumowanie 370

Rozdział 10. Szkielety Ajax 371
JPSpan 372
Jak to działa? 372
Instalacja JPSpan 372
Tworzenie strony na serwerze 373

Tworzenie strony na kliencie ... 379
Obsługa błędów ... 382
Translacja typów ... 384
Podsumowanie JPSpan ... 385
DWR ... 386
Jak to działa? ... 386
Instalacja DWR ... 386
Tworzenie strony na kliencie ... 390
Korzystanie z własnych klas ... 391
Obsługa błędów ... 395
Jeszcze o dwr.xml ... 395
Konwertery ... 397
Podsumowanie DWR ... 398
Ajax.NET ... 398
Jak to działa? ... 398
Instalacja Ajax.NET ... 399
Tworzenie strony WWW ... 400
Typy proste i złożone ... 407
Stan sesji ... 409
Podsumowanie Ajax.NET ... 410
Podsumowanie ... 410

Skorowidz ... 413

6

Usługi WWW

Gdy XML rozpowszechnił się po roku 2000, biznes, programiści i wszyscy zainteresowani poszukiwali nowych sposobów jego użycia. XML spełnił obietnicę oddzielenia treści od prezentacji, ale w jaki sposób można było wykorzystać tę zaletę? Odpowiedzią okazały się usługi WWW.

Usługi WWW umożliwiają wymianę danych pomiędzy aplikacjami i serwerami. Komunikacja ta odbywa się w ten sposób, że usługi WWW wymieniają Internetem komunikaty złożone z danych XML pomiędzy **konsumentem** (aplikacją, która używa tych danych) a **dostawcą** (serwerem udostępniającym dane). Ten sposób komunikacji różni się od tradycyjnych modeli rozproszonego przetwarzania, takich jak CORBA, DCOM i RMI, które polegają na zdalnym wywoływaniu metod za pośrednictwem połączeń sieciowych. W przypadku usług WWW główna różnica polega na tym, że w sieci przesyłane są dane XML zamiast danych binarnych.

Usługi WWW mają być programowymi komponentami dostępnymi dla dowolnej aplikacji. Niezależnie od tego, czy będzie to aplikacja WWW czy aplikacja tradycyjna, będzie mogła użyć tej samej usługi do wykonania tego samego zadania.

Technologie

Usługi WWW nie są pojedynczą technologią ani osobną platformą. Stanowią w rzeczywistości kombinację wielu protokołów, języków i formatów. Obecnie są oferowane jako komponent kilku różnych platform, ale podstawowy sposób działania jest zawsze taki sam.

SOAP

SOAP stanowi połączenie języka opartego na XML i dowolnej liczby protokołów przesyłania danych. Specyfikacja SOAP przedstawia skomplikowany język składający się z wielu elementów i atrybutów pomyślany jako sposób opisu większości typów danych. Informacje

zapisane w tym języku mogą być przesyłane za pomocą wielu różnych protokołów, ale najczęściej są wysyłane przy użyciu protokołu HTTP wraz z innymi danymi WWW.

> SOAP jest skrótem od *Simple Object Access Protocol*.

Istnieją dwa podstawowe sposoby korzystania z SOAP, **styl RPC** (remote procedure call) i **styl dokumentowy**.

Styl RPC

W tym stylu usługa WWW jest traktowana jak obiekt zawierający jedną lub więcej metod (w podobny sposób korzysta się z lokalnej klasy do nawiązania połączenia z bazą danych). Żądanie wykonuje się, podając nazwę metody i jej ewentualne parametry. Metoda zostaje wykonana przez serwer, który wysyła odpowiedź XML zawierającą wynik lub komunikat o błędzie. Wyobraźmy sobie usługę WWW wykonującą proste działania arytmetyczne: dodawanie, odejmowanie, mnożenie i dzielenie. Parametrami każdej z metod są dwie wartości liczbowe. Używając stylu RPC, żądanie wykonania dodawania będzie mieć następującą postać:

```
<?xml version="1.0" encoding=" utf-8" ?>
<soap:Envelope xmlns:soap=" http://schemas.xmlsoap.org/soap/envelope/"
               soap:encodingStyle=" http://schemas.xmlsoap.org/soap/encoding/">
    <soap:Body>
        <w:add xmlns:w=" http://www.wrox.com/services/math">
            <w:op1>4.5</w:op1>
            <w:op2>5.4</w:op2>
        </w:add>
    </soap:Body>
</soap:Envelope>
```

Gdy tworzy i rozpowszechniamy złożone dokumenty XML, powinniśmy pomyśleć o zastosowaniu przestrzeni nazw. Przestrzenie nazw są szczególnie ważne w SOAP, ponieważ dokumenty są w tym przypadku tworzone i wykorzystywane przez różne systemy. Przestrzenią nazw SOAP użytą w powyższym przykładzie jest *http://schemas.xmlsoap.org/soap/envelope*. Przestrzeń ta obowiązuje dla wersji SOAP 1.1, natomiast w wersji 1.2 używamy przestrzeni *http://www.w3.org/2003/05/soap-envelope*.

Element `<w:add>` określa nazwę wywoływanej metody (`add`) i zawiera kolejną przestrzeń nazw użytą w tym przykładzie, *http://www.wrox.com/services/math*. Ta przestrzeń jest specyficzna dla wywoływanej usługi i może być dowolnie definiowana przez programistę. Atrybut `soap:encodingStyle` wskazuje identyfikator URI informujący o sposobie kodowania żądania. Istnieje wiele innych stylów kodowania, takich jak na przykład system typów używany w schematach XML.

> Opcjonalny element `<soap:Header>` można wykorzystać dla przekazania dodatkowych informacji, na przykład związanych z uwierzytelnianiem. Element ten występuje wtedy bezpośrednio przed elementem `<soap:Body/>`.

Jeśli żądanie dodania dwóch liczb zostało poprawnie wykonane, to komunikat odpowiedzi może mieć następującą postać:

```
<?xml version="1.0" encoding=" utf-8" ?>
<soap:Envelope xmlns:soap=" http://schemas.xmlsoap.org/soap/envelope/"
               soap:encodingStyle=" http://schemas.xmlsoap.org/soap/encoding/">

    <soap:Body>
        <w:addResponse xmlns:w=" http://www.wrox.com/services/math">
            <w:addResult>9.9</w:addResult>
        </w:addResponse>
    </soap:Body>
</soap:Envelope>
```

Jak łatwo zauważyć, format odpowiedzi przypomina format żądania. Standardowy sposób dostarczania wyniku polega na tworzeniu elementu, którego nazwa jest nazwą metody z przyrostkiem `Response`. W tym przykładzie elementem tym jest `<w:addResponse/>` należący do tej samej przestrzeni nazw co element `<w:add/>` żądania. Natomiast sam wynik umieszczony został w elemencie `<w:addResult/>`. Zwróćmy uwagę, że nazwy wszystkich wymienionych elementów są definiowane przez programistę.

Jeśli podczas przetwarzania żądania SOAP przez serwer wystąpił błąd, to odpowiedź będzie zawierać element `<soap:Fault>`. Na przykład, jeśli pierwszy parametr operacji dodawania okazał się literą zamiast liczbą, to odpowiedź serwera może wyglądać następująco:

```
<?xml version="1.0" encoding=" utf-8" ?>
<soap:Envelope xmlns:soap=" http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <soap:Fault>
            <faultcode>soap:Client</faultcode>
            <faultstring>Serwer nie mógł odczytać żądania.
                         Łańcuch wejściowy jest niepoprawny.
                         Błąd w dokumencie XML (4, 13).
            </faultstring>
            <detail/>
        </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

Odpowiedź może zawierać tylko jeden element `<soap:Fault>`, który przekazuje informacje o zaistniałym problemie. Najwięcej mówi nam informacja umieszczona w elemencie `<faultcode/>`. Może on przyjmować kilka wartości, z których najczęściej spotykane to `soap:Server` i `soap:Client`. Kod `soap:Server` może wskazywać na przykład brak możliwości połączenia serwera z bazą danych. W tym przypadku ponowne wysłanie żądanie może przynieść oczekiwany rezultat. Natomiast błąd `soap:Client` oznacza najczęściej, że żądanie zostało niepoprawnie sformatowane i nie zostanie wykonane bez wprowadzenia odpowiednich modyfikacji.

Element `<faultstring/>` jest komunikatem o błędzie zawierającym bardziej szczegółowe informacje, specyficzne dla danej aplikacji. Jeśli przyczyną błędu jest inny system, na przykład baza danych, to informacja o będzie może zostać zwrócona w opcjonalnym elemencie `<faultactor/>` (element ten nie został użyty w naszym przykładzie).

Styl dokumentowy

Ten styl wykorzystuje schematy XML do określenia formatu żądania i odpowiedzi. Zdobywa on coraz większą popularność i niektórzy uważają, że w końcu wyeliminuje styl RPC. Wyjaśnienie powodów, dla których programiści rezygnują z użycia stylu RPC, można znaleźć na stronie *http://msdn.microsoft.com/library/en-us/dnsoap/html/argsoape.asp*.

Żądanie wykorzystujące omawiany styl nie musi różnić się od żądań stylu RPC. Na przykład żądanie RPC przedstawione w poprzednim podrozdziale mogłoby zostać poprawnym żądaniem stylu dokumentowego, gdyby usunąć atrybut `soap:encodingStyle`. Podstawowa różnica pomiędzy oboma stylami polega na tym, że żądania RPC muszą zawsze mieć element nazwy metody zawierający jej parametry, podczas gdy styl dokumentowy nie wprowadza takich ograniczeń. Oto przykład żądania w tym stylu różniący się całkowicie od żądań w stylu RPC:

```
<?xml version="1.0" encoding=" utf-8" ?>
<soap:Envelope xmlns:soap=" http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <w:add xmlns:w=" http://www.wrox.com/services/math" op1="4.5" op2="5.4" />
    </soap:Body>
</soap:Envelope>
```

Zauważmy, że pogrubiony wiersz zawiera nazwę metody (`add`) i jej dwa parametry (`op1` i `op2`), wszystko w jednym elemencie. Taka konstrukcja nie jest możliwa w stylu RPC. Styl dokumentowy jest bardziej elastyczny dzięki zastosowaniu schematu XML. Usługa WWW może używać schematu XML do kontroli poprawności żądania. Podobna sytuacja ma miejsce w przypadku odpowiedzi — mogą one przypominać odpowiedzi w stylu RPC bądź różnić się od nich zupełnie. Odpowiedzi wykorzystują również schematy XML.

> Usługi WWW tworzone przy wykorzystaniu Visual Studio .NET używają domyślnie stylu dokumentowego (można to zmienić, stosując odpowiednie atrybuty w uzyskanym kodzie).

Po przeczytaniu tych informacji z pewnością zadajesz sobie pytanie, gdzie przechowywany jest schemat XML i w jaki sposób jest on dostępny zarówno dla klienta, jak i usługi. Odpowiedzią na to pytanie jest kolejny skrót: WSDL.

WSDL

WSDL (*Web Services Description Language*) jest kolejnym językiem opartym na XML. Opisuje on sposób użycia usługi WWW. Jego specyfikacja opisuje złożony język, niezwykle elastyczny i zaprojektowany tak, by jak najwięcej kodu można było ponownie wykorzystać. Dokumenty w tym języku rzadko są tworzone ręcznie. Zwykle początkowa wersja dokumentu XSDL jest tworzona za pomocą odpowiedniego narzędzia, a dopiero potem ręcznie modyfikowana, jeśli jest to jeszcze konieczne.

Poniżej przedstawiamy plik XSDL opisujący prostą usługę matematyczną z jedną metodą dodawania (którą stworzymy w dalszej części rozdziału):

```
<?xml version="1.0" encoding=" utf-8"?>
<wsdl:definitions
```

```
xmlns:http=" http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap=" http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s=" http://www.w3.org/2001/XMLSchema"
xmlns:tns=" http://www.wrox.com/services/math"
xmlns:wsdl=" http://schemas.xmlsoap.org/wsdl/"
targetNamespace=" http://www.wrox.com/services/math">
<wsdl:types>
    <s:schema elementFormDefault="qualified"
              targetNamespace=" http://www.wrox.com/services/math">
        <s:element name=" add">
            <s:complexType>
                <s:sequence>
                   <s:element minOccurs="1" maxOccurs="1" name=" op1" type=" s:float" />
                   <s:element minOccurs="1" maxOccurs="1" name=" op2" type=" s:float" />
                </s:sequence>
            </s:complexType>
        </s:element>
        <s:element name=" addResponse">
            <s:complexType>
                <s:sequence>
                    <s:element minOccurs="1" maxOccurs="1" name=" addResult"
                    type=" s:float" />
                </s:sequence>
            </s:complexType>
        </s:element>
    </s:schema>
</wsdl:types>
<wsdl:message name=" addSoapIn">
    <wsdl:part name=" parameters" element=" tns:add" />
</wsdl:message>
<wsdl:message name=" addSoapOut">
    <wsdl:part name=" parameters" element=" tns:addResponse" />
</wsdl:message> <wsdl:portType name=" MathSoap">
    <wsdl:operation name=" add">
        <wsdl:documentation>
        Zwraca sumę dwóch liczb zmiennoprzecinkowych typu float
        </wsdl:documentation>
        <wsdl:input message=" tns:addSoapIn" />
        <wsdl:output message=" tns:addSoapOut" />
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name=" MathSoap" type=" tns:MathSoap">
    <soap:binding transport=" http://schemas.xmlsoap.org/soap/http" style=" document" />
    <wsdl:operation name=" add">
        <soap:operation soapAction=" http://www.wrox.com/services/math/add" style="
        document" />
        <wsdl:input>
            <soap:body use=" literal" />
        </wsdl:input>
        <wsdl:output>
            <soap:body use=" literal" />
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:service name=" Math">
    <wsdl:documentation>
```

```
            Zawiera kilka prostych funkcji arytmetycznych
            </wsdl:documentation>
            <wsdl:port name=" MathSoap" binding=" tns:MathSoap">
                <soap:address location=" http://localhost/Math/Math.asmx" />
            </wsdl:port>
        </wsdl:service>
    </wsdl:definitions>
```

Analizując zwartość tego pliku, pamiętajmy, że opisuje on tylko prostą usługę dodającą dwie liczby (dla uproszczenia pominęliśmy pozostałe trzy metody, które będziemy implementować). Chociaż przedstawiony plik WSDL jest długi i skomplikowany, to powinieneś rozumieć, co oznaczają jego poszczególne sekcje.

Element dokumentu `<wsdl:definitions>` obejmuje treść dokumentu i umożliwia deklarację różnych przestrzeni nazw. Kolejny element, `<wsdl:types/>`, zawiera schemat XML wykorzystywany przez usługę. Wewnątrz tego elementu znajduje się element `<s:schema>`, który opisuje format wszystkich elementów, jakie mogą pojawić się wewnątrz elementu `<soap:Body/>` — zarówno żądania, jak i odpowiedzi.

Pierwszym elementem opisanym przez schemat jest `<add/>`. Ponieważ atrybut `elementFormDefault` elementu `<s:schema>` ma wartość `qualified`, to przyjmuje się, że element `<add/>` należy do przestrzeni nazw określonej przez atrybut `targetNamespace`, `http://www.wrox.com/services/math`. Następnie zadeklarowane zostało, że element `<add/>` zawiera sekwencję dwóch innych elementów, `<op1/>` i `<op2/>`. Dla obu tych elementów skonfigurowano `minOccurs` i `maxOccurs` równe 1, co oznacza, że mogą pojawić się dokładnie jeden raz. Oba mają również atrybut typu `s:float`, który jest jednym z wbudowanych typów schematów XML.

> *Kompletną listę typów danych dostępnych w schematach XML można znaleźć na stronie www.w3.org/TR/xmlschema-0/#CreateDt. Jeśli usługa wymaga bardziej złożonych typów, to można je określić agregując i ograniczając typy podstawowe.*

Następnie w schemacie pojawia się kolejny element `<s:element/>`, tym razem opisujący `<addResponse/>`. Element ten będzie mieć jeden element podrzędny, `<addResult/>`, zawierający wynik działania (również typu `s:float`). Na tym kończy się nasz schemat XML.

W ciele pliku WSDL znajduje się również krótka sekcja opisująca dwa elementy `<wsdl:message/>`: `addSoapIn` i `addSoapOut`. Każdy z tych elementów ma element `<wsdl:part/>` określający element schematu XML, którego należy użyć. W tym przypadku elementy te odwołują się, odpowiednio, do `add` i `addResponse`. Sekcja ta określa format każdego z komunikatów.

Kolejna sekcja, `<wsdl:portType/>`, jest używana do pogrupowania elementów `<wsdl:message>` w operacje. Operacja będąca jednostkowe działanie usługi zawiera elementy `<wsdl:input>`, `<wsdl:output>` oraz, opcjonalnie, `<wsdl:fault>`. W naszym przykładzie występuje jedna sekcja `<wsdl:portType/>` opisująca `<wsdl:operation/>` o nazwie `add`. Atrybuty `message` jej elementów podrzędnych `<wsdl:input>` i `<wsdl:output>` odwołują się do zdefiniowanych wcześniej elementów `<wsdl:message>`. Istnieje również element `<wsdl:documentation>` zawierający słowny opis metody. (Skąd pochodzi ta informacja, dowiemy się w dalszej części rozdziału.)

Kolejnym blokiem jest <wsdl:binding>. Wiąże on operację z protokołem używanym do komunikacji z usługą. Specyfikacja WSDL opisuje trzy rodzaje wiązań: SOAP, HTTP GET/POST i MIME.

> W tym rozdziale skoncentrujemy się na wiązaniu SOAP. Wiązanie HTTP GET/POST odnosi się do sposobu tworzenia adresów URL (w przypadku żądań GET) lub kodowania zawartości formularza (w przypadku żądań POST). Wiązanie MIME umożliwia wyrażenie części komunikatu za pomocą różnych typów MIME. Oznacza to na przykład, że jedna część odpowiedzi może być w XML, a druga w HTML.
>
> Więcej informacji o tych sposobach wiązania można znaleźć na stronie *www.w3.org/TR/2003/WD-wsld112-bindings-20020709/*.

Najpierw atrybut name wiązania otrzymuje wartość MathSoap, a atrybut type wskazuje typ MathSoap zdefiniowany w sekcji <wsdl:portType>. Następnie element <soap:binding/> używa atrybutu transport do określenia, że usługa używa protokołu HTTP. Element <wsdl:operation/> definiuje nazwę metody, add. element <soap:operation/> zawiera atrybut soapAction, który musi zostać włączony do nagłówka żądania HTTP, a także atrybut style, który informuje, że komunikat SOAP będzie używać stylu dokumentowego, a nie RPC.

> *Główna różnica pomiędzy komunikatem SOAP używającym stylu dokumentowego a komunikatem w stylu RPC polega na tym, że komunikat w stylu dokumentowym wysyłany jest w postaci elementów umieszczonych wewnątrz elementu <soap:body>, które mogą mieć dowolną strukturę uzgodnioną przez nadawcę i odbiorcę za pomocą dołączonego schematu. Z kolei komunikat w stylu RPC ma element o nazwie odpowiadającej wywoływanej metodzie, który zas ma elementy odpowiadające parametrom metody.*

Element <soap:operation> ma dwa elementy podrzędne, <wsdl:input> i <wsdl:output>, używane do opisu formatu żądania i odpowiedzi. W naszym przykładzie atrybut use elementu <soap:body> otrzymał wartość literal. W praktyce jest to jedyna możliwość w przypadku usług stosujących styl dokumentowy. W przypadku stylu RPC atrybut ten może otrzymać wartość encoded i wtedy element <soap:body> musi dokładniej określić sposób kodowania typu parametrów.

Ostatnia część dokumentu, <wsdl:service/>, jest związana ze sposobem wywołania usługi przez klienta. Zawiera również tekst opisu usługi oraz element <wsdl:port/>, który odwołuje się do wiązania MathSoap. Prawdopodobnie najważniejszym elementem w tej części jest <soap:address/>, który zawiera atrybut location określający adres URL dostępu do usługi.

```
<wsdl:service name=" Math">
    <wsdl:documentation>
        Zawiera kilka prostych funkcji arytmetycznych
    </wsdl:documentation>
    <wsdl:port name=" MathSoap" binding=" tns:MathSoap">
        <soap:address location=" http://localhost/Math/Math.asmx" />
    </wsdl:port>
</wsdl:service>
```

Chociaż pierwszy kontakt z plikiem WSDL może wzbudzać lęk, to na szczęście prawie nigdy nie musimy ręcznie tworzyć takiego pliku. Przykład zamieszczony w tym podrozdziale został automatycznie stworzony przez usługę WWW na platformie .NET, która analizuje kod i generuje niezbędny XML.

Schematy XML stanowią obszerne zagadnienie, którego pełne omówienie wykracza poza zakres tej książki. Jeśli chcesz dowiedzieć się więcej na ten temat, powinieneś rozważyć lekturę książki Beginning XML *(Wiley Publishing) lub zapoznać się z informacjami zamieszczonymi na stronie www.w3schools.com/schema/default.asp.*

REST

REST (*Representational State Transfer*) opisuje sposób wykorzystania istniejącego protokołu HTTP do transmisji danych. Chociaż REST używany jest głównie przez usługi WWW, to można zastosować go w dowolnym systemie pracującym w trybie żądanie-odpowiedź z protokołem HTTP. W odniesieniu do usług WWW REST pozwala nam wywołać dany URL w określonym formacie, aby uzyskać dane (również w określonym formacie). Otrzymane dane mogą zawierać dalsze informacje o tym, w jaki sposób można pobrać więcej danych. W przypadku usług WWW dane te są zwracane jako XML.

Załóżmy na przykład, że Wrox chciałby udostępnić możliwość pobierania listy wszystkich autorów. Usługi WWW stosujące REST używają prostych URL dostępu do danych. Załóżmy zatem, że listę autorów można będzie pobrać, używając następującego URL:

```
http://www.wrox.com/services/authors/
```

Usługa ta może zwrócić reprezentację autorów w postaci danych XML wraz z informacją o sposobie dostępu do szczegółów o każdym autorze, na przykład:

```
<?xml version="1.0" encoding=" utf-8" ?>
<authors xmlns:xlink="http://www.w3.org/1999/xlink"
         xmlns="http://www.wrox.com/services/authors-books"
         xlink:href="http://www.wrox.com/services/authors/">
    <author forenames="Michael" surname="Kay"
        xlink:href="http://www.wrox.com/services/authors/kaym"
        id=" kaym"/>
    <author forenames="Joe" surname=" Fawcett"
        xlink:href="http://www.wrox.com/services/authors/fawcettj"
        id=" fawcettj"/>
    <author forenames="Jeremy" surname=" McPeak"
        xlink:href="http://www.wrox.com/services/authors/mcpeakj"
        id=" mcpeakj"/>
    <author forename=" Nicholas" surname=" Zakas"
        xlink:href="http://www.wrox.com/services/authors/zakasn"
        id=" zakasn"/>
    <!--
        More authors
    -->
</authors>
```

Warto zwrócić uwagę na parę rzeczy w powyższych danych XML. Po pierwsze, deklarują one domyślną przestrzeń nazw `http://www.wrox.com/services/authors-books` i wobec tego każdy element nieposiadający przedrostka, na przykład `<authors/>`, należy do tej przestrzeni. Oznacza to, że element `<authors/>` można odróżnić od innego elementu o tej samej nazwie, ale należącego do innej przestrzeni nazw. Identyfikator URI tej przestrzeni, `http://www.wrox.com/services/authors-books`, został użyty jedynie jako unikalny łańcuch znaków i nie gwarantuje, że w określonej przez niego lokalizacji będą dostępne odpowiednie dane. Należy zwrócić uwagę

na to, że został użyty jako URI (Uniform Resource Identifier), który jest tylko zwykłym identyfikatorem, a nie URL (*Uniform Resource Locator*), który określa dostępność zasobu w danej lokalizacji.

Po drugie, zwróćmy uwagę na użycie atrybutu `href` z przestrzeni nazw `http://www.w3.org/1999/xlink`. Wiele usług używających REST stosuje taki sposób zapisu, podczas gdy w HTML użylibyśmy zamiast niego zwykłego hiperłącza.

> *XLink jest sposobem łączenia dokumentów wykraczającym poza możliwości hiperłączy HTML. Umożliwia on specyfikację zależności dwukierunkowych, na skutek czego dokumenty stają się dostępne dla siebie nawzajem, a także określenie sposobu aktywacji łącza — na przykład ręcznie, automatycznie lub po pewnym czasie. Pokrewne rozwiązanie, XPointer, umożliwia specyfikację sekcji dokumentu i ma większe możliwości od prostych łączy HTML.*
>
> *Chociaż oba rozwiązania uzyskały rekomendację W3C, to nie są zbyt rozpowszechnione. Więcej informacji na ich temat można znaleźć na stronie www.w3.org/XML/Linking.*

Dane XML zwrócone przez usługę REST zostają przekształcone, przez klienta lub przez serwer, na bardziej czytelny format (zwykle HTML), na przykład do takiej postaci:

```
<html>
    <head>
        <title>Autorzy Wrox</title>
    </head>
    <body>
        <a href="http://www.wrox.com/services/authors/kaym">Michael Kay</a>
        <a href="http://www.wrox.com/services/authors/fawcettj">Joe Fawcett</a>
        <a href="http://www.wrox.com/services/authors/mcpeakj">Jeremy McPeak</a>
        <a href="http://www.wrox.com/services/authors/zakasn">Nicholas Zakas</a>
    </body>
</html>
```

Następnie użytkownik może pobrać dane o wybranym autorze, wybierając odpowiednie łącze. W rezultacie usługa WWW zwróci dane XML podobne do poniższych:

```
<?xml version="1.0" encoding=" utf-8" ?>
<author xmlns:xlink="http://www.w3.org/1999/xlink"
        xmlns="http://www.wrox.com/services/authors-books"
        xlink:href="http://www.wrox.com/services/authors/fawcettj"
        id=" fawcettj" forenames=" Joe" surname=" Fawcett">
    <books>
        <book
            xlink:href="http://www.wrox.com/services/books/0764570773"
            isbn="0764570773" title=" Beginning XML"/>
        <book
            xlink:href="http://www.wrox.com/services/books/0471777781"
            isbn="0471777781" title=" Professional Ajax"/>
    </books>
</author>
```

Po raz kolejny dane XML zawierają elementy należące do przestrzeni nazw `http://www.wrox.com/services/authors-books`, a atrybut `xlink:href` pozwala dotrzeć do dalszych informacji. Reprezentacja tych danych w HTML może wyglądać następująco:

```
<html>
    <head>
        <title>Informacje o autorze</title>
    </head>
    <body>
        <p>Szczegółowe informacje o
        <a href="http://www.wrox.com/services/authors/fawcettj">Joe Fawcett</a></p>
        <p>Books</p>
        <a href="http://www.wrox.com/services/books/0764570773">Beginning XML</a>
        <a href="http://www.wrox.com/services/books/0471777781">Professional Ajax</a>
    </body>
</html>
```

Jeśli użytkownik wybierze ostatnie z łączy na powyższej stronie, to w odpowiedzi uzyska następujące dane XML:

```
<?xml version="1.0" encoding=" utf-8" ?>
<book xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns="http://www.wrox.com/services/authors-books"
     xlink:href="http://www.wrox.com/services/books/0471777781"
     isbn="0471777781">
    <genre>Web Programming</genre>
    <title>Professional AJAX</title>
    <description>Jak wykorzystać JavaScript i XML do tworzenia stron WWW
    charakteryzujących się zaawansowanym interfejsem użytkownika.</description>
    <authors>
        <author forenames=" Nicholas" surname=" Zakas"
            xlink:href="http://www.wrox.com/services/authors/zakasn"
            id=" zakasn" />
        <author forenames=" Jeremy" surname=" McPeak"
            xlink:href="http://www.wrox.com/services/authors/mcpeakj"
            id=" mcpeakj" />
        <author forenames=" Joe" surname=" Fawcett"
            xlink:href="http://www.wrox.com/services/authors/fawcettj"
            id=" fawcettj" />
    </authors>
</book>
```

Łatwo zauważyć, że usługi REST stosują powtarzający się wzorzec. Na przykład, jeśli kompletną listę autorów otrzymujemy, używając *http://www.wrox.com/services/authors/*, to wprowadzenie prostej modyfikacji polegającej na dołączeniu identyfikatora autora pozwala uzyskać informacje o tym autorze (*http://www.wrox.com/services/authors/fawcettj*).

Przedstawioną usługę można zaimplementować na wiele sposobów. Na przykład za pomocą statycznych stron WWW lub, co bardziej prawdopodobne, kodu działającego na serwerze, na przykład ASP, JSP lub PHP, który pobierze odpowiednie dane z bazy i zwróci je w formacie XML. W tym przypadku adres URL zostanie odwzorowany przez serwer na określony sposób pobrania danych, na przykład wywołanie procedury bazy danych.

> *Więcej informacji na temat usług REST (zwanych również usługami RESTful) można znaleźć na stronie www.network.world.com/ee/2003/eerest.html.*

Platforma .NET i SOAP

Wprowadzenie SOAP pozwoliło firmie Microsoft objąć przywództwo ruchu związanego z usługami WWW. Po przedstawieniu przez Microsoft firmie IBM rozwiązania SOAP jako sposobu transportu danych również ta druga firma przyłączyła się do tego ruchu, pomagając w stworzeniu specyfikacji WSDL. Do Microsoftu i IBM dołączyło następnie wiele dużych firm takich jak Oracle, Sun czy HP. Powstały odpowiednie standardy i na horyzoncie pojawiła się nowa era usług WWW. Obraz ten mącił tylko jeden fakt: nie było narzędzi pozwalających tworzyć usługi WWW. Wkrótce lukę tę wypełniła platforma .NET.

Microsoft wprowadził platformę .NET na początku 2000 roku z myślą o rywalizacji z platformą Java w zakresie tworzenia przenośnych aplikacji. Ponieważ platforma .NET powstała praktycznie od zera, to możliwe było wprowadzenie odpowiedniej obsługi XML, a także tworzenia i konsumpcji usług WWW używających SOAP i WSDL. Platforma .NET oferuje proste sposoby obudowywania istniejących aplikacji za pomocą usług WWW, jak również udostępniania większości klas .NET jako usług WWW.

Tworząc usługę WWW, musimy zdecydować, w jakim stopniu konieczne będą interakcje z SOAP i WSDL. Istnieją narzędzia pozwalające odizolować programistę od wykorzystywanej struktury, ale jeśli to konieczne, możliwa jest również modyfikacja jej szczegółów. Wersja platformy .NET wypuszczona w 2005 roku w jeszcze większym stopniu bazuje na XML i usługach WWW.

Decyzje podczas projektowania

Chociaż platforma .NET ułatwia tworzenie usług WWW, to nie jest w żadnym razie jedynym sposobem ich tworzenia. Podobnie jak podczas tworzenia innego oprogramowania także i w tym przypadku musimy podjąć kilka ważnych decyzji na etapie projektowania i implementacji. Przypomnijmy, że usługi WWW stanowią niezależny od platformy sposób żądania i otrzymywania danych i w związku z tym użytkownik usługi nie potrzebuje informacji o sposobie implementacji usługi. Niestety, w praktyce należy jednak wziąć pod uwagę następujące kwestie związane z różnymi implementacjami:

- **Nie wszystkie platformy obsługują te same typy danych.** Na przykład wiele usług zwraca zbiór danych ADO.NET. System, który nie implementuje platformy .NET, nie będzie mógł poprawnie zinterpretować tych danych. Podobnie problematyczne może okazać się zastosowanie tablic, ponieważ mogą one być reprezentowane na wiele sposobów.
- **Niektóre usługi są bardziej tolerancyjne dla dodatkowych lub brakujących nagłówków w żądaniach.** Problem ten związany jest z konsumentami usług, którzy nie wysyłają poprawnie wszystkich nagłówków i mogą stanowić zagrożenie dla bezpieczeństwa usługi.

Aby rozwiązać te i inne problemy, stworzono Web Services Interoperability Organization. Z jej zadaniami, osiągnięciami i rekomendacjami można zapoznać się pod adresem *www.ws-i.org/*.

Pierwszą decyzją, którą musimy podjąć, tworząc usługę WWW, jest wybór platformy. Jeśli wybierzemy Windows, to prawie na pewno zdecydujemy się również użyć IIS jako serwera WWW. Usługę będziemy tworzyć przy użyciu ASP.NET lub ASP w przypadku starszych wersji serwera IIS (chociaż ten ostatni sposób jest nieco trudniejszy). W przykładach przedstawionych w tym rozdziale wykorzystamy ASP.NET.

Jeśli wybierzemy UNIX lub Linux, to prawdopodobnie będziemy używać JSP lub PHP. Dla obu systemów istnieją implementacje serwerów WWW należące do kategorii „open source".

Projekt Axis (http://ws.apache.org/axis/) oferuje narzędzia zarówno dla języka Java, jak i C++.

Również w przypadku PHP mamy spory wybór, między innymi PhpXMLRPC (http://phpxmlrpc.sourceforge.net/) i Pear SOAP (http://pear.php.net/package/SOAP).

Po wyborze języka programowania musimy się zastanowić, kto będzie mieć dostęp do naszej usługi. Czy tylko nasza aplikacja czy też usługa będzie dostępna publicznie? W tym drugim przypadku musimy wziąć pod uwagę wspomniane wcześniej problemy współpracy. Jeśli zdecydujemy się na pierwszy przypadek, to mamy swobodę korzystania z zalet różnych rozszerzeń przez klienta lub serwer.

Po stworzeniu usługi WWW należy ją skonsumować. Przez konsumenta usługi rozumiemy dowolną aplikację, która wywołuje naszą usługę. Zwykle konsument działa według następującego schematu: tworzy żądanie, wysyła je i podejmuje pewne działania na podstawie otrzymanej odpowiedzi. Dokładna metoda implementacji tego schematu zależy od funkcjonalności oferowanej użytkownikowi przez konsumenta.

Tworzenie usługi WWW w Windows

Przejdźmy teraz od specyfikacji i teorii do praktyki związanej z implementacją prostej usługi WWW. Będzie ona używać stylu dokumentowego SOAP do implementacji usługi Math opisanej za pomocą pliku WSDL przedstawionego wcześniej w tym rozdziale. Zwróćmy uwagę, że w procesie jej tworzenia zastosujemy darmowe narzędzia udostępniane przez Microsoft, co będzie wymagać nieco większego nakładu pracy niż w przypadku zastosowania na przykład Visual Studio .NET. Jednak tym razem warto ponieść ten dodatkowy wysiłek, ponieważ zaowocuje on lepszym zrozumieniem usług WWW i zwróci się z nawiązką w przyszłości, jeśli trafimy na problemy związane z automatycznie wygenerowaną usługą WWW.

Wymagania systemowe

Stworzenie tej usługi wymagać będzie co najmniej:

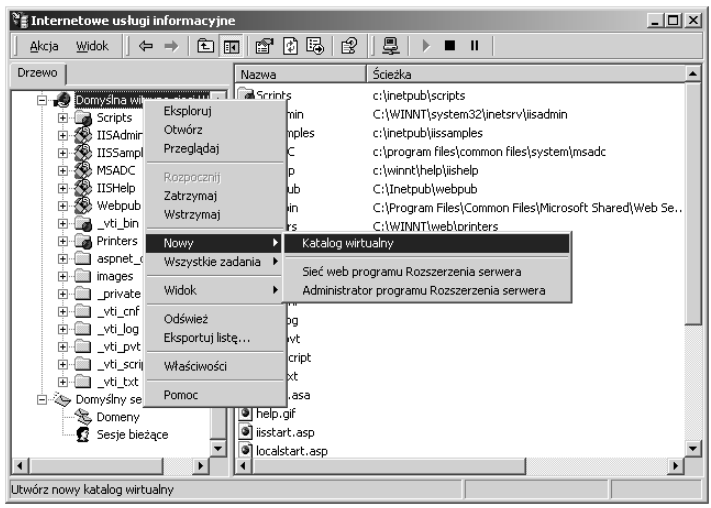
- Komputera z systemem Windows i działającym na nim serwerem IIS 5 lub nowszym. Jest on dostępny na wszystkich komputerach wyposażonych w system XP Professional i wszystkich serwerach począwszy od Windows 2000.

- .NET Framework zainstalowanej na komputerze, na którym działa IIS, a także .NET SDK (*Software Development Kit*) na komputerze, na którym będziemy tworzyć kod. Na potrzeby tego przykładu założymy, że będzie to ten sam komputer, na którym działa IIS. (.NET Framework i SDK można załadować ze strony *http://msdn.microsoft.com/netframework/downloads/updates/default.aspx*.)
- Edytor tekstu do pisania kodu. Może być nim nawet Notatnik zainstalowany standardowo na wszystkich komputerach z systemem Windows. Dla potrzeb tego przykładu będzie on więcej niż wystarczający (chociaż w przypadku tworzenia poważniejszych aplikacji z pewnością pomocny będzie edytor o większych możliwościach).

Konfiguracja serwera IIS

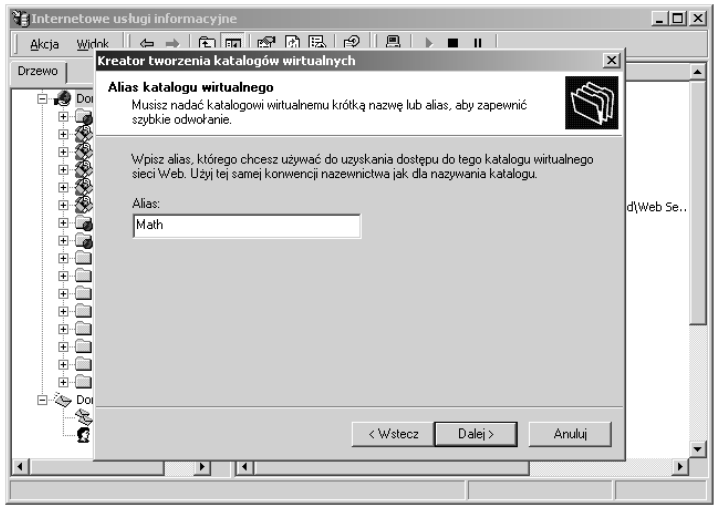
Najpierw musimy utworzyć katalog wirtualny dla naszej usługi. W tym celu wybieramy *Start/Ustawienia/Panel sterowania/Narzędzia administracyjne* i uruchamiamy *Menedżer usług internetowych* (alternatywnie możemy wprowadzić *%SystemRoot%\System32\inetsrv\iis.smc* w oknie *Start/Uruchom* i wybrać przycisk *OK*). Po uruchomieniu menedżera rozwijamy drzewo w lewej części okna, aby odnaleźć węzeł *Domyślna witryna sieci Web*. Prawym przyciskiem myszy rozwijamy menu tego węzła i wybieramy *Nowy/Katalog wirtualny*, tak jak pokazano na rysunku 6.1. W efekcie uruchomiony zostanie *Kreator tworzenia katalogów wirtualnych*, w którym tworzymy alias katalogu usługi WWW używany przez klienta (patrz rysunek 6.2).

Rysunek 6.1.

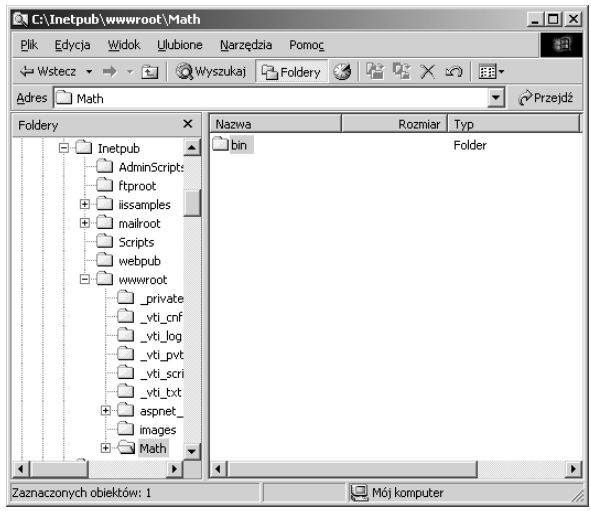


Wpisujemy nazwę *Math* i wybieramy przycisk *Dalej*. Na kolejnym ekranie przechodzimy do katalogu serwera IIS, czyli *C:\InetPub\wwwroot*. Tworzymy w nim nowy katalog o nazwie *Math*. Na pozostałych ekranach kreatora akceptujemy domyślnie wybrane opcje. Następnie, za pomocą *Eksploratora Windows*, odnajdujemy utworzony katalog *Math* i tworzymy w nim podkatalog *bin*, w którym po skompilowaniu usługi WWW zostanie utworzona biblioteka DLL. Uzyskana hierarchia powinna wyglądać tak jak na rysunku 6.3.

Rysunek 6.2.



Rysunek 6.3.



Tworzenie kodu

Nasza usługa WWW jest bardzo prosta. Ma nazwę *Math* i implementuje cztery podstawowe działania arytmetyczne: dodawanie, odejmowanie, mnożenie i dzielenie. Każda z tych operacji ma dwa parametry typu `float` i zwraca wynik takiego samego typu. Samą klasę zaimplementujemy w C#, a usługa WWW zostanie opublikowana w ASP.NET.

Tworzymy nowy plik w wybranym edytorze i umieszczamy w nim trzy wiersze:

```
using System;
using System.Web;
using System.Web.Services;
```

Powyższy kod nie wprowadza jeszcze żadnej funkcjonalności, ale oszczędza nam pisania pełnych nazw klas należących do wymienionych przestrzeni nazw.

Następnie stworzymy własną przestrzeń nazw `Wrox.Services` i klasę `Math` dziedziczącą po `System.Web.Services.WebService`:

```
namespace Wrox.Services
{
    [WebService (Description = "Zawiera kilka prostych funkcji arytmetycznych",
Namespace = "http://www.wrox.com/services/math")]

    public class Math : System.Web.Services.WebService
    {
        //tutaj kod klasy
    }
}
```

Słowo kluczowe `namespace` stosuje się w podobny sposób jak przestrzenie nazw w XML. Oznacza to, że pełną nazwą klasy `Math` jest `Wrox.Services.Math`. Wewnątrz definicji przestrzeni nazw umieszczony został atrybut o nazwie `WebService`. Oznacza on, że klasa w następnym wierszu jest usługą WWW. W ten sposób włączona zostaje dla tej klasy dodatkowa funkcjonalność polegająca między innymi na generowaniu pliku WSDL. Zwróćmy również uwagę na parametr `Description` (który pojawi się także w pliku WSDL).

Następnie mamy nazwę klasy, `Math`, która dziedziczy po klasie bazowej `System.Web.Services.WebService`. Dziedziczenie po tej klasie oznacza, że nie musimy przejmować się tworzeniem kodu specyficznego dla usługi WWW, ponieważ zrobi to za nas klasa bazowa. W ten sposób możemy skoncentrować się na tworzeniu metod, które zostaną udostępnione jako część usługi.

Definiowanie metody używanej jako część usługi WWW odbywa się tak jak w przypadku zwykłej metody, z tą różnicą, że musimy oznaczyć ją atrybutem `WebMethod`:

```
[WebMethod(Description = "Zwraca sumę dwóch liczb zmiennoprzecinkowych typu float")]
public float add(float op1, float op2)
{
    return op1 + op2;
}
```

Po raz kolejny kod okazuje się bardzo prosty (czy można sobie wyobrazić prostszą operację niż dodawanie?). Każda metoda poprzedzona atrybutem `WebMethod` jest uważana za część usługi WWW. Parametr `Description` zostanie umieszczony w wygenerowanym pliku WSDL. Chociaż możemy stworzyć dowolną liczbę takich metod, to poniższy przykład zawiera cztery odpowiadające podstawowym działaniom arytmetycznym:

```
using System;
using System.Web;
using System.Web.Services;

namespace Wrox.Services
{
     [WebService (Description = "Zawiera kilka prostych funkcji arytmetycznych",
                  Namespace = "http://www.wrox.com/services/math")]
        public class Math : System.Web.Services.WebService
        {
```

```
        [WebMethod(Description = "Zwraca sumę dwóch liczb zmiennoprzecinkowych typu
        float")]
        public float add(float op1, float op2)
        {
            return op1 + op2;
        }

        [WebMethod(Description = "Zwraca różnicę dwóch liczb zmiennoprzecinkowych
        typu float")]
        public float subtract(float op1, float op2)
        {
            return op1 - op2;
        }

        [WebMethod(Description = "Zwraca iloczyn dwóch liczb zmiennoprzecinkowych
        typu float")]
        public float multiply(float op1, float op2)
        {
            return op1 * op2;
        }

        [WebMethod(Description = "Zwraca iloraz dwóch liczb zmiennoprzecinkowych typu
        float")]
        public float divide(float op1, float op2)
        {
            return op1 / op2;
        }
    }
}
```

Plik ten zapiszemy w katalogu *Math* jako *Math.asmx.cs*.

Następnie utworzymy kolejny plik tekstowy i wprowadzimy w nim poniższy wiersz:

```
<%@WebService Language=" c#" Codebehind=" Math.asmx.cs" Class=" Wrox.Services.Math" %>
```

Jest to plik ASP.NET, który używa stworzonej przed chwilą klasy `Math`. Dyrektywa `@WebService` nakazuje stronie działać jako usługa WWW. Znaczenie pozostałych atrybutów jest dość oczywiste — `Language` określa język kodu, `Codebehind` określa nazwę pliku zawierającego kod, `Class` specyfikuje pełną nazwę klasy, którą należy użyć. Również ten plik zapisujemy w katalogu *Math* pod nazwą *Math.asmx*.

Tworzenie kodu wykonywalnego

Po utworzeniu tych dwóch plików możemy przejść do następnego etapu — kompilacji kodu źródłowego do kodu wykonywalnego, który zostanie umieszczony w bibliotece DLL. W tym celu możemy użyć kompilatora C# dostarczanego z .NET SDK. Kompilator ten znajduje się w katalogu *Microsoft.Net\Framework\ <numer wersji>* utworzonym w katalogu systemu Windows (na przykład *C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322*).

Najłatwiej można kompilować i uruchamiać kod za pomocą pliku wsadowego. Stwórzmy zatem kolejny plik tekstowy i wprowadźmy w nim następujące polecenie (całość w jednym wierszu, niezależnie od sposobu jego sformatowania w tej książce):

```
C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322\csc.exe /r:System.dll /r:System.Web.dll
/r:System.Web.Services.dll /t:library /out:bin\Math.dll Math.asmx.cs
```

Musimy jeszcze pamiętać o dodaniu ścieżki prowadzącej do *csc.exe* (jeśli to konieczne) i zapisaniu pliku wsadowego w katalogu *Math* pod nazwą *MakeService.bat*.

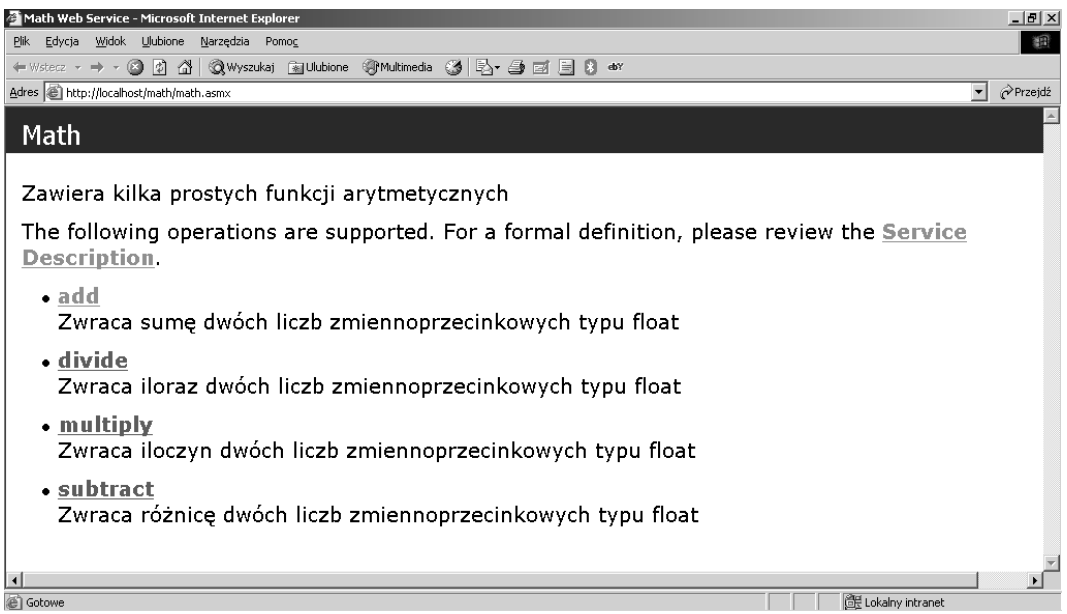
> Jeśli używasz edytora Notatnik, to powinieneś zwrócić szczególną uwagę na okno dialogowe *Zapisz jako*. Upewnij się, że wybrałeś *Wszystkie pliki* na liście rozwijalnej *Zapisz jako typ* lub ujmij nazwę pliku w cudzysłowy. W przeciwnym razie Notatnik automatycznie doda do nazwy pliku rozszerzenie *.txt*.

Następnie skompilujemy DLL. Otwieramy okno poleceń, wybierając *Start/Uruchom* i wpisując *cmd*. Przechodzimy do katalogu *Math*, wpisując *cd \inetpub\wwwroot\Math*. Uruchamiamy plik wsadowy:

```
C:\inetpub\wwwroot\Math\MakeService.bat
```

Jeśli wszystko przebiegnie po naszej myśli, to zobaczymy tekst powitania kompilatora, a po nim zostanie wyświetlony pusty wiersz. Oznacza to, że kompilacja się udała. (Jeśli wystąpią błędy, to informacja o nich zostanie wyświetlona na konsoli. Wtedy należy sprawdzić wiersze kodu wskazane przez kompilator i poprawić błędy składniowe lub inne pomyłki.)

Po skompilowaniu biblioteki DLL możemy przetestować usługę. Jedną z zalet usług WWW tworzonych na platformie .NET jest automatyczne tworzenie środowiska testowego. Po uruchomieniu przeglądarki i wpisaniu adresu *http://localhost/Math/math.asmx* zobaczymy stronę podobną do przedstawionej na rysunku 6.4.

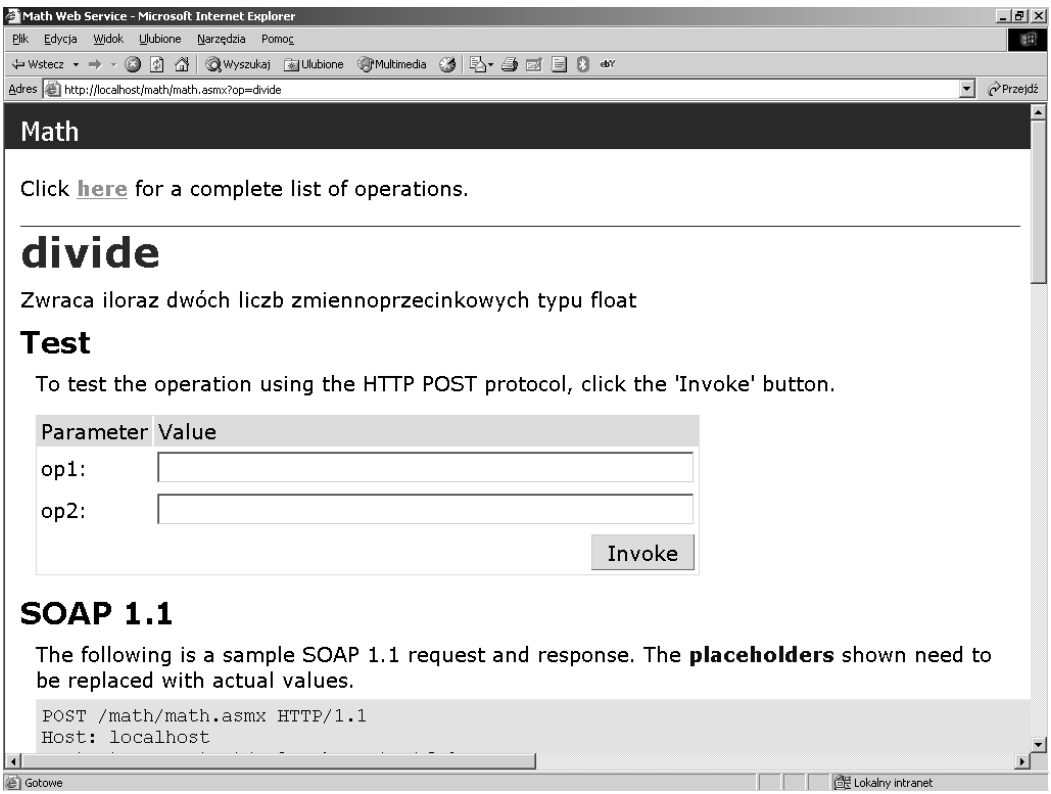


Rysunek 6.4.

Na stronie tej możemy wypróbować dowolną z czterech metod usługi lub obejrzeć wygenerowany plik WSDL, klikając łącze *Service Description*. Plik ten jest podobny do przedstawionego wcześniej w tym rozdziale, ale opisuje wszystkie cztery metody.

> Inny sposób obejrzenia pliku WSDL polega na dodaniu łańcucha *?WSDL* do URL usługi, na przykład *http://localhost/Math/math.asmx?WSDL*.

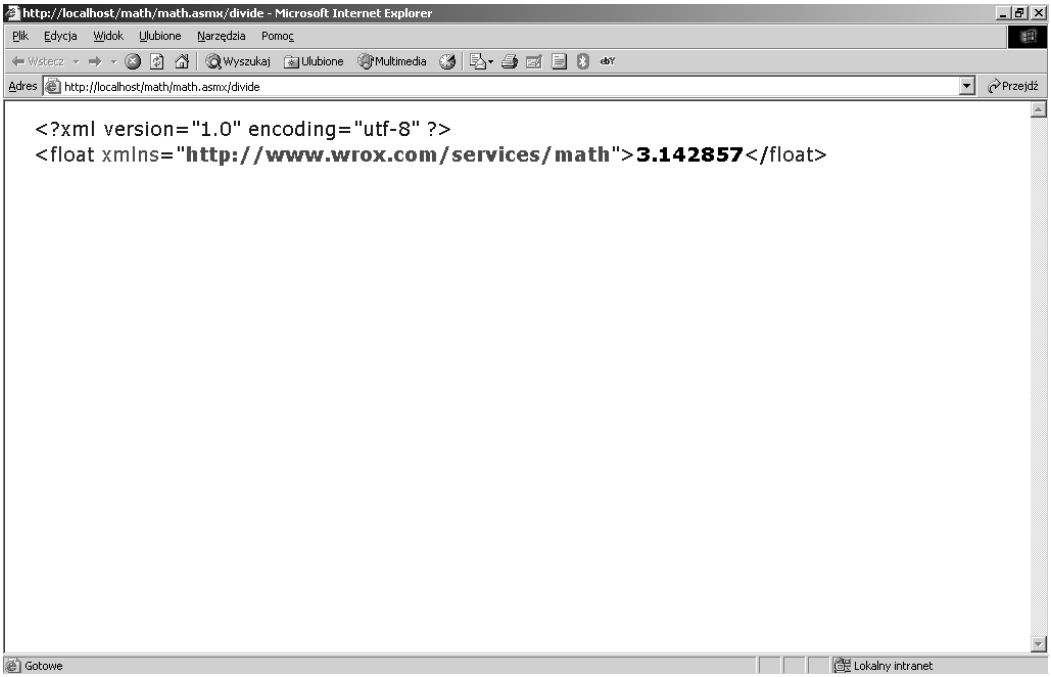
Metodą `add` zajmowaliśmy się już wystarczająco, sprawdźmy zatem działanie metody `divide`. Po kliknięciu łącza przedstawionego na poprzednim rysunku otrzymamy stronę przedstawioną na rysunku 6.5.



Rysunek 6.5.

Poniżej nagłówka wyświetlającego nazwę metody `divide` umieszczony zostaje opis, który skonfigurowaliśmy za pomocą atrybutu `WebMethod`, a poniżej znajduje się prosty formularz umożliwiający testowanie metody. Jeśli wprowadzimy w nim wartości 22 i 7, to uzyskamy odpowiedź przedstawioną na rysunku 6.6.

Po wdrożeniu aplikacji możemy usunąć plik *Math.asmx.cs*, który nie jest już potrzebny. *Math.asmx* przekazuje bowiem wszystkie żądania bezpośrednio do biblioteki DLL.



Rysunek 6.6.

Środowisko testowe nie używa SOAP dla żądań i odpowiedzi. Zamiast tego przekazuje oba argumenty za pomocą żądania POST. Szczegóły tej operacji są widoczne w dolnej części strony przedstawionej na rysunku 6.5. Po zdefiniowaniu usługi WWW pora, aby wywołał ją Ajax.

Usługi WWW i Ajax

Teraz, gdy posiadamy już podstawową wiedzę na temat usług WWW i stworzyliśmy pierwszą własną usługę, możemy wyjaśnić, co mają one wspólnego z rozwiązaniami Ajax. Usługi WWW stanowią kolejny sposób dostępu do informacji używany przez aplikacje Ajax. W poprzednim rozdziale przedstawiliśmy sposób dostępu do źródeł RSS i Atom, który przypomina mocno korzystanie z usług WWW. Główna różnica polega na tym, że używając usług WWW, możemy przekazać serwerowi informację, która zostanie przez niego przetworzona i odesłana z powrotem; usługi WWW wykraczają zatem możliwościami poza proste pobieranie danych.

Strona WWW może konsumować usługę WWW za pomocą JavaScript pod warunkiem, że użytkownicy mają przeglądarki i odpowiednich możliwościach. Może to być Internet Explorer w wersji 5.0 lub wyższej albo przeglądarka rodziny Mozilla (na przykład Firefox).

Tworzenie środowiska testowego

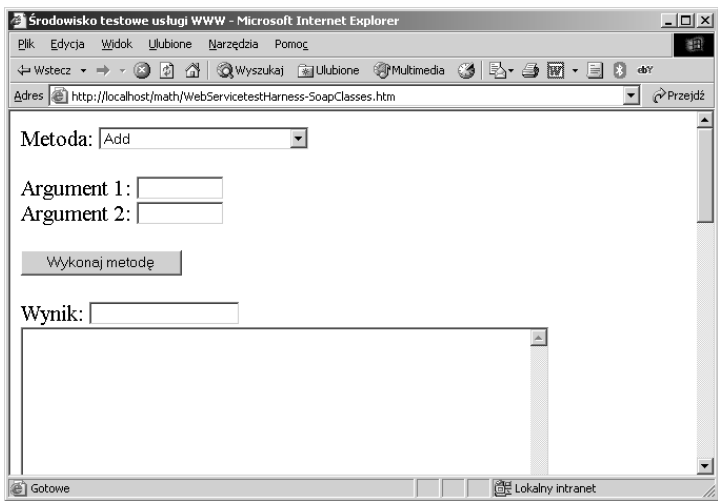
Aby przetestować różne sposoby wywoływania usług WWW przez przeglądarkę, musimy stworzyć odpowiednie środowisko testowe. Będzie się ono składać z: listy pozwalającej wybrać jedno z czterech działań arytmetycznych, pola tekstowego dla każdego argumentu oraz przycisku umożliwiającego wywołanie usługi. Wszystkie te elementy kontrolne pozostaną nieaktywne do momentu całkowitego załadowania strony. Poniżej nich znajdzie się kolejne pole tekstowe wyświetlające wynik działania usługi oraz dwa obszary tekstowe prezentujące dane żądania i odpowiedzi:

```
<html>
    <head>
        <title>Środowisko testowe usługi WWW</title>
        <script type=" text/javascript">

            var SERVICE_URL = "http://localhost/Math/Math.asmx";
            var SOAP_ACTION_BASE = "http://www.wrox.com/services/math";

            function setUIEnabled(bEnabled)
            {
                var oButton = document.getElementById("cmdRequest");
                oButton.disabled = !bEnabled;
                var oList = document.getElementById("lstMethods");
                oList.disabled = !bEnabled
            }

            function performOperation()
            {
                var oList = document.getElementById("lstMethods");
                var sMethod = oList.options[oList.selectedIndex].value;
                var sOp1 = document.getElementById("txtOp1").value;
                var sOp2 = document.getElementById("txtOp2").value;

                //Opróżnia panele komunikatów
                document.getElementById("txtRequest").value = "";
                document.getElementById("txtResponse").value = "";
                document.getElementById("txtResult").value = "";
                performSpecificOperation(sMethod, sOp1, sOp2);
            }
        </script>
    </head>
    <body onload=" setUIEnabled(true)">
        Metoda: <select id=" lstMethods" style=" width: 200px" disabled=" disabled">
            <option value=" add" selected=" selected">Add</option>
            <option value=" subtract">Subtract</option>
            <option value=" multiply">Multiply</option>
            <option value=" divide">Divide</option>
        </select>
        <br/><br/>
        Argument 1: <input type=" text" id=" txtOp1" size="10"/><br/>
        Argument 2: <input type=" text" id=" txtOp2" size="10"/><br/><br/>
        <input type=" button" id=" cmdRequest"
               value="Wykonaj metodę"
               onclick=" performOperation();" disabled=" disabled"/>
        <br/><br/>
        Wynik: <input type=" text" size="20" id=" txtResult">
        <br/>
```

```
        <textarea rows="30" cols="60" id=" txtRequest"></textarea>
        <textarea rows="30" cols="60" id=" txtResponse"></textarea>
    </body>
</html>
```

Funkcja `setUIEnabled()` jest używana do włączania i wyłączania interfejsu użytkownika środowiska testowego. Gwarantuje to pojedyncze wysyłanie żądań. Zdefiniowane zostały również dwie stałe, `SERVICE_URL` i `SOAP_ACTION_BASE`, zawierające, odpowiednio, URL usługi WWW i nagłówek akcji SOAP wymagany do jej wywołania. Wybranie przycisku powołuje wywołanie funkcji `performOperation()`, która zbiera odpowiednie dane i opróżnia pola tekstowe przed wywołaniem `performSpecificOperation()`. Metoda ta musi zostać zdefiniowana przez test używany do wywołania usług WWW (dołączony za pomocą pliku JavaScript). W zależności od indywidualnych preferencji przeglądarki uzyskana strona będzie przypominać pokazaną na rysunku 6.7.

Rysunek 6.7.



Rozwiązanie dla przeglądarki Internet Explorer

Aby początkowo zainteresować programistów usługami WWW, Microsoft wprowadził w przeglądarce IE tak zwane **zachowania** usług WWW. Pozwalają one na nowo definiować funkcjonalność, właściwości i metody istniejącego elementu HTML lub tworzyć zupełnie nowy element. Zaletą stosowania zachowań jest możliwość umieszczenia funkcjonalności w pojedynczym pliku z rozszerzeniem *.htc*. Chociaż rozwiązanie to się nie rozpowszechniło, to jest bardzo przydatnym komponentem dla programistów tworzących usługi WWW. Więcej informacji na temat zachowań znajdziesz pod adresem *http://msdn.microsoft.com/library/default.asp?url=/workshop/author/webservice/webservice.asp*.

Zachowanie można dodawać do elementów na wiele sposobów. Najbardziej oczywisty polega na użyciu właściwości `style` określającej klasę stylu CSS. Na przykład kod dodający zachowanie do elementu `<div/>` może wyglądać tak:

```
<div id=" divServiceHandler" style=" behavior: url(webservice.htc);"></div>
```

Kod ten zakłada, że zachowanie zostało umieszczone w tym samym katalogu serwera co strona WWW.

Aby pokazać sposób użycia tego zachowania, stworzymy nową wersję środowiska testowego i wstawimy wiersz wyróżniony pogrubieniem bezpośrednio za elementem <body/>, tak jak poniżej:

```
<body onload=" setUIEnabled(true);">
<div id=" divServiceHandler" style=" behavior: url(webservice.htc);"></div>
Operation: <select id=" lstMethods" style=" width: 200px" name=" lstMethods"
disabled=" disabled">
```

Następny krok polega na zdefiniowaniu metody performSpecificOperation() związanej z użyciem zachowania usługi WWW. Metoda ta ma trzy argumenty: nazwę metody i dwa argumenty. Jej kod jest następujący:

```
var iCallId = 0;
function performSpecificOperation(sMethod, sOp1, sOp2)
{
    var oServiceHandler = document.getElementById("divServiceHandler");
    if (!oServiceHandler.Math)
    {
        oServiceHandler.useService(SERVICE_URL + "?WSDL", "Math");
    }
    iCallId = oServiceHandler.Math.callService(handleResponseFromBehavior, sMethod,
    sOp1, sOp2);
}
```

Zmienna iCallId zostaje zainicjowana wartością 0. Chociaż nie bierze ona udziału w teście, to pozwala orientować się w sytuacji polegającej na wielu równoczesnych wywołaniach. Następnie referencja elementu , do którego dołączono zachowanie, zostaje umieszczona w zmiennej oServiceHandler. Instrukcja warunkowa if sprawdza, czy zachowanie zostało już użyte. W tym celu bada istnienie właściwości Math. Jeśli właściwość ta nie istnieje, to musimy skonfigurować zachowanie, przekazując funkcji useService() URL pliku WSDL i nazwę identyfikującą usługę. Powodem zastosowania tego identyfikatora jest możliwość jednoczesnego użycia więcej niż jednej usługi WWW przez zachowanie. Następnie wykonana zostaje metoda callService(), której przekazujemy funkcję wywoływaną zwrotnie (handleResponseFromBehavior()), nazwę metody i dwa argumenty.

Funkcja handleResponseFromBehavior() zostanie wywołana automatycznie po odebraniu odpowiedzi:

```
function handleResponseFromBehavior(oResult)
{
    var oResponseOutput = document.getElementById("txtResponse");
    if (oResult.error)
    {
        var sErrorMessage = oResult.errorDetail.code + "\n"
                          + oResult.errorDetail.string;
        alert("Wystąpił błąd:\n"
              + sErrorMessage
              + "Patrz informacje o błędzie SOAP w panelu komunikatów.");
        oResponseOutput.value = oResult.errorDetail.raw.xml;
    }
    else
```

```
    {
        var oResultOutput = document.getElementById("txtResult");
        oResultOutput.value = oResult.value;
        oResponseOutput.value = oResult.raw.xml;
    }
}
```

Funkcji wywoływanej zwrotnie zostaje przekazany obiekt `oResult` zawierający szczegóły wywołania. Jeśli właściwość `error` jest różna od zera, to wyświetlone zostają szczegóły błędu SOAP. W przeciwnym razie na stronie zostaje wyświetlona wartość zwrócona przez funkcję, `oResult.value`.

Funkcje `performSpecificOperation()` i `handleResponseFromBehavior()` możemy umieścić w zewnętrznym pliku JavaScript i dołączyć je do strony środowiska testowego za pomocą elementu `<script/>`:

```
<script type=" text/javascript" src=" WebServiceExampleBehavior.js"></script>
```

Posługiwanie się zachowaniem usługi WWW nie jest skomplikowane. Całe działanie wykonywane jest przez zachowanie w tle i chociaż plik *webservice.htc* jest sporych rozmiarów jak na skrypt (51 kB), to dostarcza bardzo przydatnej funkcjonalności.

> *Jeśli chcesz zobaczyć, w jaki sposób działa zachowanie, to otwórz plik webservice.htc w edytorze tekstu. Nie jest to jednak lektura dla początkujących i zawiera prawie 2300 wierszy kodu JavaScript.*

Rozwiązanie dla przeglądarki Mozilla

Nowoczesne przeglądarki Mozilla, takie jak Firefox czy Netscape, mają wbudowane klasy SOAP w implementację JavaScript. Zadaniem tych klas jest ułatwienie korzystania z podstawowej strategii wykonywania wywołań SOAP. Podobnie jak w poprzednim przykładzie, najpierw musimy zdefiniować funkcję `performSpecificOperation()`:

```
function performSpecificOperation(sMethod, sOp1, sOp2)
{
    var oSoapCall = new SOAPCall();
    oSoapCall.transportURI = SERVICE_URL;
    oSoapCall.actionURI = SOAP_ACTION_BASE + "/" + sMethod;
    var aParams = [];
    var oParam = new SOAPParameter(sOp1, "op1");
    oParam.namespaceURI = SOAP_ACTION_BASE;
    aParams.push(oParam); oParam = new SOAPParameter(sOp2, "op2");
    oParam.namespaceURI = SOAP_ACTION_BASE;
    aParams.push(oParam);
    oSoapCall.encode(0, sMethod, SOAP_ACTION_BASE, 0, null, aParams.length, aParams);
    var oSerializer = new XMLSerializer();
    document.getElementById("txtRequest").value =
                        oSerializer.serializeToString(oSoapCall.envelope);
    setUIEnabled(false);

    //tutaj reszta kodu
}
```

Skrypt ten wykorzystuje kilka wbudowanych klas, z których pierwszą jest `SOAPCall`. Klasa ta obudowuje funkcjonalność usługi WWW w podobny sposób jak zachowania udostępniane przez przeglądarkę Internet Explorer. Po utworzeniu instancji klasy `SOAPCall` konfigurujemy jej dwie właściwości: `transportURI`, która wskazuje lokalizację usługi WWW, i `actionURI`, która specyfikuje akcję SOAP i nazwę metody.

Następnie za pomocą konstruktora klasy `SOAPParameter` tworzone są dwa parametry. Konstruktorowi zostają przekazane nazwa i wartość parametru. Dla każdego parametru zostaje skonfigurowany identyfikator URI przestrzeni nazw za pomocą wartości `targetNamespace` pochodzącej z sekcji schematu w pliku WSDL. Teoretycznie nie jest to konieczne, ale klasy SOAP zaimplementowano w Mozilla raczej z myślą o stylu RPC, a nasza usługa używa stylu dokumentowego. Oba parametry zostają umieszczone w tablicy `aParams`. Metoda `encode()` przygotowuje wszystkie dane wywołania. Wywołanie to ma siedem parametrów. Pierwszym jest używana wersja SOAP. Parametrowi temu możemy nadać wartość zero, chyba że zależy nam na użyciu określonej wersji. Drugim parametrem jest nazwa metody, którą należy użyć, a trzecim wartość `targetNamespace` z sekcji schematu w pliku WSDL. Kolejny parametr określa liczbę dodatkowych nagłówków potrzebnych w wywołaniu (w naszym przykładzie nie jest potrzebny żaden), a następny jest tablicą tych nagłówków (`null` w tym przypadku). Ostatnie dwa parametry zawierają liczbę wysyłanych obiektów `SOAPParameter` oraz liczbę rzeczywistych parametrów.

Wreszcie możemy wysłać żądanie, wywołując metodę `asyncInvoke()`:

```
function performSpecificOperation(sMethod, sOp1, sOp2)
{
    var oSoapCall = new SOAPCall();
    oSoapCall.transportURI = SERVICE_URL;
    oSoapCall.actionURI = SOAP_ACTION_BASE + "/" + sMethod;
    var aParams = [];
    var oParam = new SOAPParameter(sOp1, "op1");
    oParam.namespaceURI = SOAP_ACTION_BASE;
    aParams.push(oParam);
    oParam = new SOAPParameter(sOp2, "op2");
    oParam.namespaceURI = SOAP_ACTION_BASE;
    aParams.push(oParam); oSoapCall.encode(0, sMethod, SOAP_ACTION_BASE, 0, null,
aParams.length, aParams);
    document.getElementById("txtRequest").value =
    oSerializer.serializeToString(oSoapCall.envelope);
    setUIEnabled(false);
    oSoapCall.asyncInvoke(
        function (oResponse, oCall, iError)
        {
            var oResult = handleResponse(oResponse, oCall, iError);
            showSoapResults(oResult);
        }
        );
}
```

Metoda `asyncInvoke()` ma tylko jeden argument — funkcję wywoływaną zwrotnie (`handleResponse()`). Wywołanie SOAP przekaże tej funkcji trzy argumenty: obiekt `SOAPResponse`, wskaźnik oryginalnego wywołania SOAP (potrzebny w przypadku wielu wywołań) oraz kod błędu.

```
function handleResponse(oResponse, oCall, iError)
{
    setUIEnabled(true);
    if (iError != 0)
    {
        alert("Nieznany błąd.");
        return false;
    }
    else
    {
        var oSerializer = new XMLSerializer();
        document.getElementById("txtResponse").value =
        oSerializer.serializeToString(oResponse.envelope);
        var oFault = oResponse.fault;
        if (oFault != null)
        {
            var sName = oFault.faultCode;
            var sSummary = oFault.faultString;
            alert("Wystąpił błąd:\n" + sSummary + "\n" + sName + "\nPatrz informacje
            o błędzie SOAP w panelu komunikatów");
            return false;
        }
        else
        {
            return oResponse;
        }
    }
}
```

Jeśli kod `IError` jest różny od zera, to wystąpił nieoczekiwany błąd, którego nie można wyjaśnić. Sytuacja taka ma miejsce rzadko; w większości przypadków błąd zostaje zwrócony za pomocą właściwości `fault` obiektu odpowiedzi.

Kolejną użytą wbudowaną klasą jest `XMLSerializer`. Przekształca ona węzeł XML na łańcuch znaków lub strumień. W naszym przykładzie pobieramy łańcuch znaków i wyświetlamy go w obszarze tekstowym po prawej stronie.

Jeśli `oResponse.fault` jest różne od `null`, oznacza to, że wystąpił błąd SOAP. W tej sytuacji tworzymy komunikat o błędzie, wyświetlamy go użytkownikowi i nie podejmujemy żadnej dalszej akcji. Jeśli wywołanie zakończyło się pomyślnie, to funkcja zwraca obiekt odpowiedzi, który zostaje przetworzony przez funkcję `showSoapResults()`:

```
function showSoapResults(oResult)
{
    if (!oResult) return;
    document.getElementById("txtResult").value =
oResult.body.firstChild.firstChild.firstChild.data;
}
```

Po sprawdzeniu, że obiekt `oResult` istnieje, funkcja pobiera wartość elementu `<methodResult>`, używając DOM.

> *Istnieje metoda `getParameters` klasy `SOAPResponse`, która teoretycznie pozwala pobrać parametry w bardziej elegancki sposób. Jednak nie działa ona poprawnie w przypadku wywołań stylu dokumentowego i dlatego konieczne jest sprawdzanie struktury `soap:Body` bardziej prymitywnymi sposobami.*

Rozwiązanie uniwersalne

Jedynym sposobem konsumpcji usług WWW działającym dla prawie wszystkich przeglądarek jest wykorzystanie obiektu `XMLHttp`. Ponieważ Internet Explorer, Firefox, Opera i Safari obsługują w podstawowym stopniu ten obiekt, to możliwe jest stworzenie jednego rozwiązania działającego ze wszystkimi przeglądarkami. Koszt takiego rozwiązania jest jednak wyższy, ponieważ sami musimy tworzyć żądania SOAP, wysyłać je do serwera, parsować wynik i obsługiwać błędy.

Jak pamiętamy, w scenariuszu tym pojawiają się dwaj konkurenci: klasa `XmlHttp` zaimplementowana przez Microsoft jako kontrolka ActiveX oraz klasa `XmlHttpRequest` udostępniana przez najnowsze wersje przeglądarek Mozilla. Obie klasy mają podobne metody i właściwości. Microsoft zaimplementował swoją klasę, zanim uzgodniono standard, i dlatego późniejsza wersja Mozilli jest bardziej zgodna ze standardem W3C. Podstawowym zadaniem obu klas jest wysyłanie żądania HTTP do dowolnego adresu w sieci WWW. Żądanie nie musi zwracać danych XML, dozwolona jest dowolna treść. Możliwe jest również wysyłanie danych za pomocą żądań POST. W przypadku wywołań SOAP wysyła się żądanie POST z ładunkiem w postaci elementu `<soap:Envelope>`.

W tym przykładzie po raz kolejny będziemy wywoływać środowisko testowe. Tym razem jednak użyjemy biblioteki zXML do tworzenia obiektów `XMLHttp` i będziemy samodzielnie tworzyć kompletne wywołanie SOAP. Wspomniana biblioteka używa różnych technik w celu ustalenia, które klasy XML są obsługiwane przez daną przeglądarkę. Biblioteka obudowuje te klasy i wyposaża je w dodatkowe metody i właściwości, aby mogły być używane w identyczny sposób niezależnie od typu przeglądarki. Łańcuch żądania SOAP tworzony będzie przez funkcję `getRequest()`:

```
function getRequest(sMethod, sOp1, sOp2)
{
    var sRequest = "<soap:Envelope xmlns:xsi=\""
                 + "http://www.w3.org/2001/XMLSchema-instance\" "
                 + "xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" "
                 + "xmlns:soap=\" http://schemas.xmlsoap.org/soap/envelope/\">\n"
                 + "<soap:Body>\n" + "<" + sMethod + "xmlns=\"" + SOAP_ACTION_BASE +
                 "\">\n"
                 + "<op1>" + sOp1 + "</op1>\n"
                 + "<op2>" + sOp2 + "</op2>\n"
                 + "</" + sMethod + ">\n"
                 + "</soap:Body>\n"
                 + "</soap:Envelope>\n";
    return sRequest;
}
```

Działanie funkcji `getRequest()` jest dość oczywiste; tworzy ona łańcuch SOAP we właściwym formacie (format ten można podejrzeć używając środowiska testowego .NET przedstawionego przy okazji tworzenia usługi Math). Kompletny łańcuch SOAP zwrócony przez funkcję `getRequest()` zostaje użyty przez funkcję `performSpecificOperation()` do stworzenia żądania SOAP:

```
function performSpecificOperation(sMethod, sOp1, sOp2)
{
    oXmlHttp = zXmlHttp.createRequest();
```

```
    setUIEnabled(false);
    var sRequest = getRequest(sMethod, sOp1, sOp2);
    var sSoapActionHeader = SOAP_ACTION_BASE + "/" + sMethod;
    oXmlHttp.open("POST", SERVICE_URL, true);
    oXmlHttp.onreadystatechange = handleResponse;
    oXmlHttp.setRequestHeader("Content-Type", "text/xml");
    oXmlHttp.setRequestHeader("SOAPAction", sSoapActionHeader);
    oXmlHttp.send(sRequest);
    document.getElementById("txtRequest").value = sRequest;
}
```

Funkcja powyższa tworzy najpierw żądanie XMLHttp, wywołując metodę klasy `zXMLHttp`. Jak wyjaśniliśmy wcześniej, w rezultacie powstanie obiekt klasy ActiveX, jeśli przeglądarką jest Internet Explorer, lub obiekt klasy `XmlHttpRequest`, jeśli przeglądarka używa technologii Mozilla. Następnie obiekt ten zostaje zainicjowany za pomocą metody `open()`. Pierwszy jej parametr określa typ żądania jako POST, drugi reprezentuje URL usługi, a ostatni określa tryb wykonania żądania (wartość `true` oznacza tryb asynchroniczny).

Właściwość `onreadystatechange` określa funkcję, która zostanie wywołana, gdy stan wykonania żądania się zmieni.

Następnie funkcja `performSpecificOperation()` dodaje dwa nagłówki do żądania HTML. Pierwszy z nich określa typ treści jako `text/xml`, a drugi dodaje nagłówek `SOAPAction`. Wartość tę możemy odczytać, korzystając ze strony środowiska testowego .NET lub podejrzeć w pliku WSDL jako atrybut `soapAction` odpowiedniego elementu `<soap:operation/>`. Po wysłaniu żądania jego dane XML zostają wyświetlone w obszarze tekstowym po lewej stronie. Gdy stan wykonania żądania się zmieni, zostaje wywołana funkcja `handleResponse()`:

```
function handleResponse()
{
    if (oXmlHttp.readyState == 4)
    {
        setUIEnabled(true);
        var oResponseOutput = document.getElementById("txtResponse");
        var oResultOutput = document.getElementById("txtResult");
        var oXmlResponse = oXmlHttp.responseXML;
        var sHeaders = oXmlHttp.getAllResponseHeaders();
        if (oXmlHttp.status != 200 || !oXmlResponse.xml)
        {
            alert("Błąd dostępu do usługi WWW.\n"
                  + oXmlHttp.statusText
                  + "\nSzczegóły w panelu odpowiedzi.");
            var sResponse = (oXmlResponse.xml ? oXmlResponse.xml : oXmlResponseText);
            oResponseOutput.value = sHeaders + sResponse;
            return;
        }
        oResponseOutput.value = sHeaders + oXmlResponse.xml;
        var sResult = oXmlResponse.documentElement.firstChild.firstChild.firstChild.
        firstChild.data;
        oResultOutput.value = sResult;
    }
}
```

Funkcja `handleResponse()` reaguje na każdą zmianę stanu żądania. Gdy właściwość `readyState` jest równa 4, co oznacza zakończenie żądania, jej rola jest skończona i można sprawdzić wynik żądania.

Jeśli `oXmlHttp.status` jest różny od 200 lub właściwość `responseXML` jest pusta, oznacza to, że wystąpił błąd i wyświetlony zostaje odpowiedni komunikat. Jeśli jest to błąd SOAP, to informacja ta zostaje również wyświetlona w panelu komunikatów. W przeciwnym razie wyświetlona zostaje `responseText`. Jeśli wywołanie zakończyło się pomyślnie, to nieprzetworzone dane XML zostają wyświetlone w obszarze tekstowym po prawej stronie.

Przy założeniu, że odpowiedź XML jest dostępna, istnieje wiele sposobów wyodrębnienia właściwego wyniku działania usługi. Można użyć w tym celu XSLT, DOM lub parsowania tekstu, jednak rzadko która z tych metod może działać niezależnie od typu przeglądarki. Wykorzystanie DOM do stopniowego przeglądania drzewa nie jest szczególnie eleganckim sposobem, ale może zostać skutecznie użyte dla dowolnego dokumentu XML.

Usługi WWW pomiędzy domenami

Jak dotąd ograniczyliśmy się do wywoływania usługi WWW dostępnej w tej samej domenie, w której znajduje się strona korzystająca z tej usługi. W ten sposób uniknęliśmy problemu używania skryptów działających w różnych domenach. Jak wyjaśniliśmy we wcześniejszej części książki, problem ten wynika z ograniczeń wprowadzonych w przeglądarkach w związku z ryzykiem wywołań zewnętrznych witryn. Jeśli usługa znajduje się w tej samej domenie co wywołująca ją strona, to przeglądarka zezwoli na wykonanie żądania SOAP, ale co w przypadku, gdy chcemy użyć jednej z usług udostępnianych przez Google lub Amazon.com?

W takiej sytuacji potrzebny będzie proxy działający na naszym serwerze WWW i wykonujący wywołania w imieniu klienta. Następnie proxy zwróci klientowi informację otrzymaną od usługi zdalnej. Konfiguracja ta przypomina serwer proxy stosowany w sieciach korporacyjnych do dostępu do zewnętrznych witryn. W tym modelu wszystkie żądania są przekazywane do centralnego serwera, który pobiera odpowiednie strony WWW i zwraca je klientom.

Interfejs usług Google

Google udostępnia wiele metod, które mogą być wywoływane za pomocą usługi WWW, na przykład służących pobieraniu buforowanych kopii stron oraz wyników wyszukiwań określonych fraz. Zastosowanie serwera proxy zademonstrujemy na przykładzie metody `doSpellingSuggestion`, która na podstawie przekazanej jej frazy zwraca podpowiedzi sugerowane przez Google. Jeśli fraza zawiera błędy lub jest na tyle trudna do rozpoznania, że nie można zaproponować żadnej sensownej podpowiedzi, metoda zwraca wtedy pusty łańcuch.

Na stronie *www.google.com/apis/index.html* można znaleźć zestaw przeznaczony dla programistów pragnących wykorzystać usługi Google. Zawiera on standardy SOAP i WSDL, jak również przykłady w językach C#, Visual Basic .NET i Java.

Aby uzyskać dostęp do usług Google, należy się zarejestrować i otrzymać specjalny klucz. Sposób wykorzystania usługi Google (tak w celach testowych, jak i komercyjnych) regulują określone zasady.

Konfiguracja proxy

Po załadowaniu dokumentacji Google i uzyskaniu klucza musimy skonfigurować usługę na własnym serwerze, która będzie akceptować wywołania otrzymywane od klientów i przekazywać je usłudze Google. Usługę tę stworzymy w taki sam sposób jak omówioną wcześniej usługę Math.

Na początek uruchamiamy program umożliwiający administrowanie serwerem IIS (*Start/Ustawienia/Panel sterowania/Narzędzia administracyjne/Menedżer usług internetowych*). Po uruchomieniu menedżera rozwijamy drzewo w lewej części okna, aby odnaleźć węzeł *Domyślna witryna sieci Web*. Prawym przyciskiem myszy otwieramy menu tego węzła i wybieramy *Nowy/Katalog wirtualny*. Uruchomiony zostaje *Kreator tworzenia katalogów wirtualnych*. W polu *Alias* wpisujemy nazwę katalogu, *GoogleProxy*, i wybieramy przycisk *Dalej*. Na kolejnym ekranie kreatora odnajdujemy standardowy katalog serwera IIS, *C:\InetPub\wwwroot*, i tworzymy w nim podkatalog, również o nazwie *GoogleProxy*. Na pozostałych ekranach kreatora akceptujemy domyślne opcje, a następnie uruchamiamy *Eksploratora Windows* i tworzymy w katalogu *GoogleProxy* podkatalog o nazwie *bin*.

Następnie otwieramy w edytorze tekstu nowy plik, w którym wprowadzamy następujący tekst (wszystko w jednym wierszu):

```
<%@ WebService Language=" c#" Codebehind=" GoogleProxy.asmx.cs" Class="
Wrox.Services.GoogleProxyService" %>
```

Zapisujemy ten plik jako *GoogleProxy.asmx* w katalogu *GoogleProxy*.

Teraz stworzymy główny plik usługi, *GoogleProxy.asmx.cs*:

```
using System;
using System.Web;
using System.Web.Services;
using GoogleService;

namespace Wrox.Services
{
     [WebService (Description = "Umożliwia wywołania Google API",
                  Namespace = "http://www.wrox.com/services/googleProxy")]
     public class GoogleProxyService : System.Web.Services.WebService
     {
        readonly string GoogleKey = "EwVqJPJQFHL4inHoIQMEP9jExTpcf/KG";

        [WebMethod( Description = "Zwraca podpowiedź Google dla podanej frazy.")]
        public string doSpellingSuggestion(string Phrase)
        {
```

```
            GoogleSearchService s = new GoogleSearchService();
            s.Url = "http://api.google.com/search/beta2";
            string suggestion = "";
            try
            {
                suggestion = s.doSpellingSuggestion(GoogleKey, Phrase);
            }

            catch(Exception Ex)
            {
                throw Ex;
            }
            if (suggestion == null) suggestion = "Brak podpowiedzi.";
            return suggestion;
        }
    }
}
```

Pamiętaj, aby zmiennej `GoogleKey` przypisać klucz, który otrzymałeś od Google, i zapisać plik w tym samym katalogu co poprzednio.

Sam kod jest dość prosty; właściwe zadanie i tak zostaje wykonane przez `GoogleSearchService`. Metoda `doSpellingSuggestion` tworzy instancję klasy `GoogleSearchService`. Następnie konfiguruje URL usługi. Ten krok nie jest zawsze konieczny, ale praktyka pokazuje, że możliwość łatwej zmiany URL bywa często przydatna. Prawdziwa aplikacja pobierałaby ten URL z pliku konfiguracyjnego, umożliwiając łatwą zmianę serwera.

Następnie wywołana zostaje metoda `doSpellingSuggestion`, której przekazujemy klucz Google i argument `Phrase`. Jest to kolejna zaleta stosowania serwera proxy: poufne informacje takie jak na przykład klucz są przechowywane w bezpiecznym miejscu, z dala od środowiska klienta i przeglądarki.

Jeśli wywołanie `doSpellingSuggestion` wygeneruje wyjątek, to zostanie on ponownie zgłoszony przez funkcję wywołującą i w efekcie zostanie zwrócony jako błąd SOAP. Jeśli `suggestion` okaże się równa `null`, to zwrócony zostanie tekst komunikatu; w przeciwnym razie podpowiedź uzyskana od Google zostanie przekazana bezpośrednio klientowi.

Stworzymy teraz klasę umożliwiającą interakcję z Google. Rozpoczniemy od skopiowania pliku *GoogleSearch.wsdl* do katalogu *GoogleProxy*. Następnie otworzymy okno wiersza poleceń, przejdziemy do katalogu *GoogleProxy* i wywołamy (ignorując ostrzeżenia o brakującym schemacie):

```
WSDL /namespace:GoogleService /out:GoogleSearchService.cs GoogleSearch.wsdl
```

Program WSDL wczyta plik *GoogleSearch.wsdl* i stworzy kod źródłowy klasy umożliwiającej komunikację z usługą. Klasa ta będzie należeć do przestrzeni nazw `GoogleService`, co określiliśmy za pomocą pierwszego parametru wywołania programu WSDL. Otrzymany kod źródłowy musimy przekształcić w plik DLL, używając tak jak poprzednio kompilatora C#. W tym celu należy użyć pliku wsadowego *MakeGoogleServiceDLL.bat* załadowanego wraz z kodem przykładów lub wpisać w wierszu poleceń.

> Program *WSDL.exe* może znajdować się w wielu różnych miejscach w zależności od tego, jakie inne komponenty Microsoft zostały zainstalowane na komputerze. Na maszynach, na których zainstalowano Visual Studio .NET, znajdziemy go w katalogu *C:\Program Files\Microsoft Visual Studio.NET 2003\SDK\v1.1\Bin*. W innych przypadkach należy go po prostu poszukać.

```
C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322\csc.exe /r:System.dll /r:System.Web.dll
/r:System.Web.Services.dll /t:library /out:bin\GoogleSearchService.dll
GoogleSearchService.cs
```

Tak jak poprzednio parametry `/r:` informują kompilator o tym, które pliki DLL zawierają klasy potrzebne do działania docelowego pliku DLL, w którym znajdzie się klasa `GoogleSearchService`.

Ostatni etap polega na kompilacji klasy `GoogleProxy`:

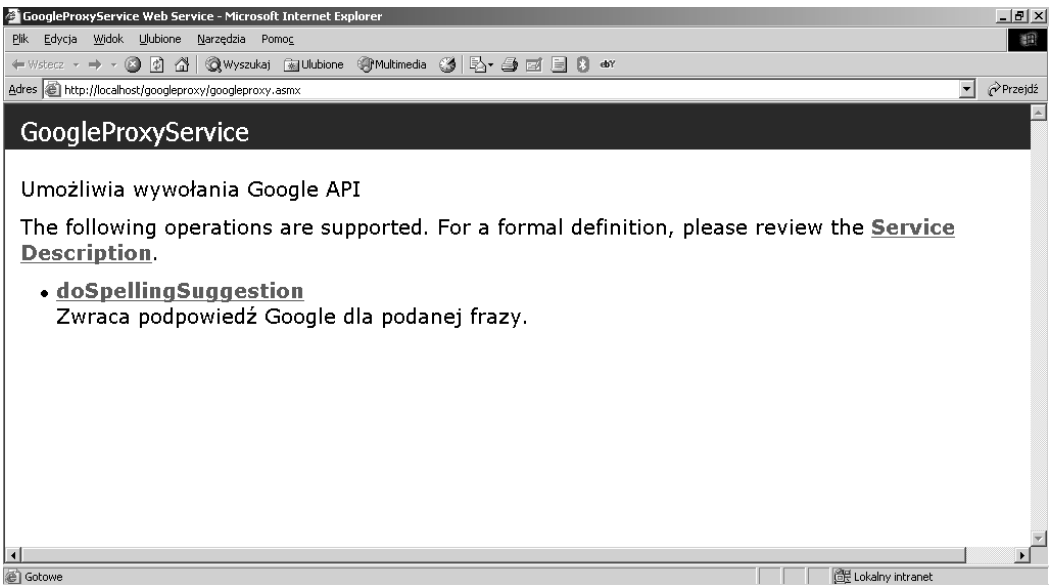
```
C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322\csc.exe /r:System.dll /r:System.Web.dll
/r:System.Web.Services.dll /r:bin\GoogleSearchService.dll /t:library
/out:bin\GoogleProxy.dll GoogleProxy.asmx.cs
```

Zwróćmy uwagę na odwołanie do stworzonej przed chwilą biblioteki *GoogleSearchService.dll*.

Możemy już przetestować usługę, wprowadzając w przeglądarce poniższy URL:

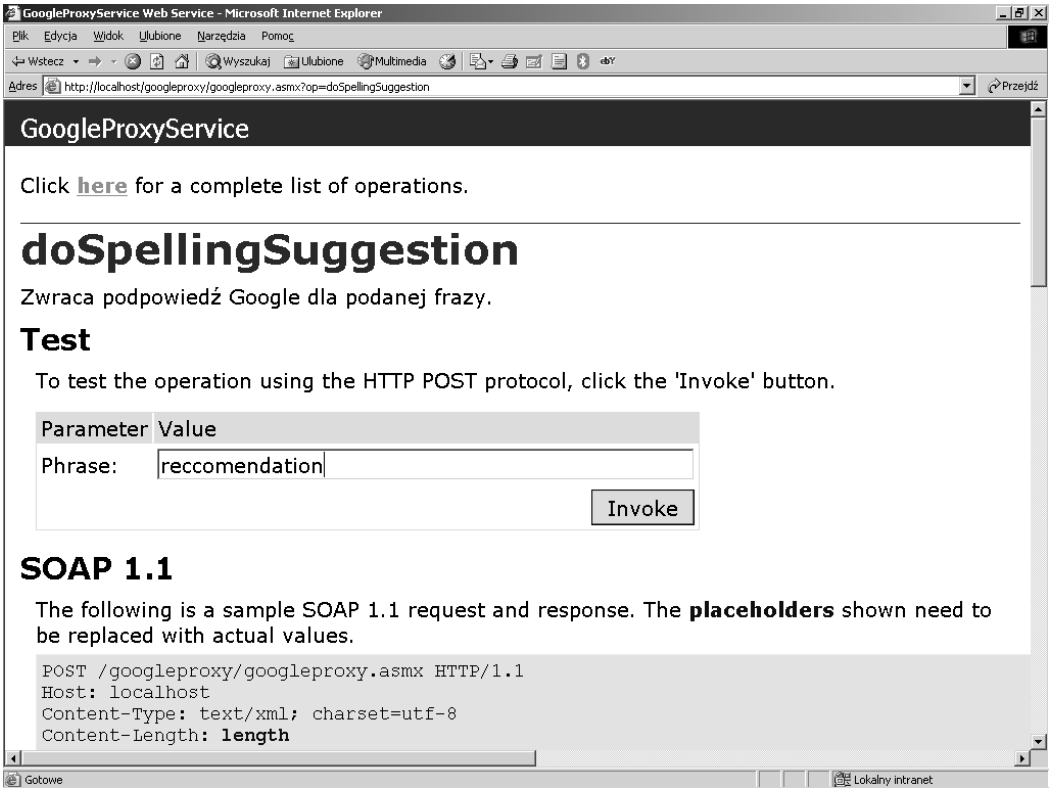
```
http://localhost/GoogleProxy/GoogleProxy.asmx
```

Powinniśmy otrzymać standardowy ekran powitalny usługi przedstawiony na rysunku 6.8.



Rysunek 6.8.

Jeśli klikniemy łącze *doSpellingSuggestion*, możemy wypróbować metodę, używając wbudowanego środowiska testowego pokazanego na rysunku 6.9.



Rysunek 6.9.

Po wprowadzeniu frazy i kliknięciu przycisku *Invoke* otrzymamy odpowiedź XML przedstawioną na rysunku 6.10.

Rysunek 6.10.

Podsumowanie

W rozdziale przedstawiliśmy koncepcję usług WWW jako architektury umożliwiającej przesyłanie danych pomiędzy różnymi lokalizacjami w Internecie. Omówiliśmy ewolucję tych usług oraz technologie związane z nimi, takie jak SOAP, WSDL i REST. Poruszyliśmy również kwestię różnic i podobieństw pomiędzy SOAP i REST.

Następnie pokazaliśmy, w jaki sposób można stworzyć własną usługę WWW za pomocą ASP.NET i języka C#. Wymaga to ściągnięcia pakietu .NET SDK i wykorzystania odpowiednich narzędzi tworzenia usług WWW. Przedstawiliśmy również sposób testowania usługi WWW za pomocą środowiska testowego generowanego automatycznie na platformie .NET.

Następnie przeszliśmy do zagadnienia tworzenia środowiska testowego najpierw dla przeglądarki Internet Explorer, a później Mozilla, pokazując przy tym różne sposoby wywoływania usługi WWW. Przedstawiliśmy zachowania usług WWW dostępne w przeglądarce Internet Explorer oraz klasy SOAP dostępne w przeglądarkach Mozilla. Na koniec stworzyliśmy uniwersalne środowisko testowe, używając obiektu XMLHttp do wysyłania i odbierania komunikatów SOAP.

Ostatnim poruszonym zagadnieniem był problem związany z dostępem do usług WWW znajdujących się w różnych domenach i sposób jego uniknięcia poprzez zastosowanie serwera proxy.

W rozdziale używaliśmy specyfikacji SOAP do przekazywania argumentów pomiędzy klientem i serwerem. Chociaż jest to sposób uniwersalny, to wymaga sporo przetwarzania oraz obsługi XML przez klienta. W następnym rozdziale pokażemy mniej formalny, ale prostszy sposób przekazywania danych pomiędzy komputerami, nazwany JSON (*JavaScript Object Notation*).