Wprowadzenie do technologii Web Services: SOAP, WSDL i UDDI

Maciej Zakrzewicz, PLOUG
mzakrz@cs.put.poznan.pl

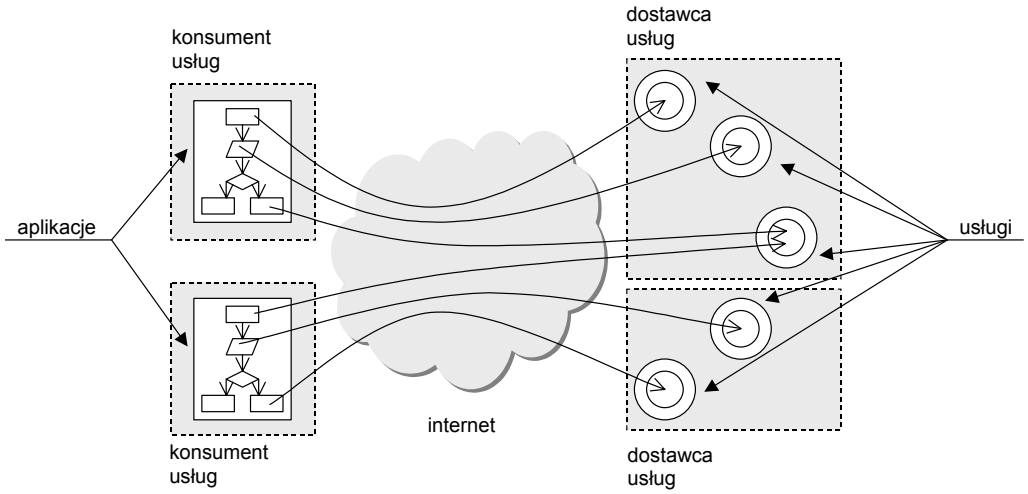
Streszczenie

Web Services to technologia implementacji rozproszonych komponentów programowych udostępnianych za pośrednictwem protokołu SOAP. Komponenty usługowe Web Services mogą być implementowane z użyciem rozmaitych języków programowania, platform sprzętowych i operacyjnych. W celu ułatwienia implementacji aplikacji klienckich, komponenty usługowe Web Services opisuje się w języku WSDL, dzięki czemu programiści aplikacji klienckich mogą korzystać z automatycznych generatorów kodu komunikacyjnego. Dalszym rozwinięciem tego rozwiązania jest specyfikacja baz danych UDDI umożliwiających gromadzenie informacji o dostępnych w sieci usługach Web Services.

Celem referatu jest przedstawienie podstawowych pojęć związanych z implementacją komponentów i klientów Web Services w kontekście architektur zorientowanych na usługi realizowanych w technologii Java.

1. Wprowadzenie: architektury zorientowane na usługi

Koncepcja architektury zorientowanej na usługi (Service-Oriented Architecture) opiera się na założeniu, że logika biznesowa nie stanowi monolitycznego programu, lecz jest rozbita pomiędzy wiele rozproszonych komponentów usługowych, koordynowanych przez centralną aplikację sterującą. Komponenty usługowe są luźno związane z aplikacją sterującą, nazywaną też konsumentem usług (Service Consumer), a ponadto mogą być współdzielone przez wiele aplikacji. Zwykle komponenty usługowe są implementowane i udostępniane przez niezależne podmioty, nazywane dostawcami usług (Service Providers). Łączność pomiędzy aplikacją sterującą a komponentami usługowymi odbywa się za pośrednictwem sieci Internet. Najczęściej do implementacji rozproszonych komponentów usługowych wykorzystuje się technologie CORBA, DCOM, EJB i Web Services, przy czym właśnie ta ostatnia cieszy się gwałtownie rosnąca popularnością. Najważniejsze elementy architektury zorientowanej na usługi przedstawiono na rys. 1. Rysunek przedstawia aplikacje biznesowe będące konsumentami zdalnych usług, tzn. realizujące kroki swoich procesów biznesowych za pomocą zdalnych komponentów usługowych. Komponenty usługowe są oferowane przez dostawców usług. Powiązania pomiędzy aplikacjami i komponentami mają charakter związków „wiele do wielu” – każda aplikacja może korzystać z wielu komponentów usługowych, a każdy komponent usługowy może być wykorzystywany przez wiele niezależnych aplikacji.



Rys. 1. Architektura zorientowana na usługi

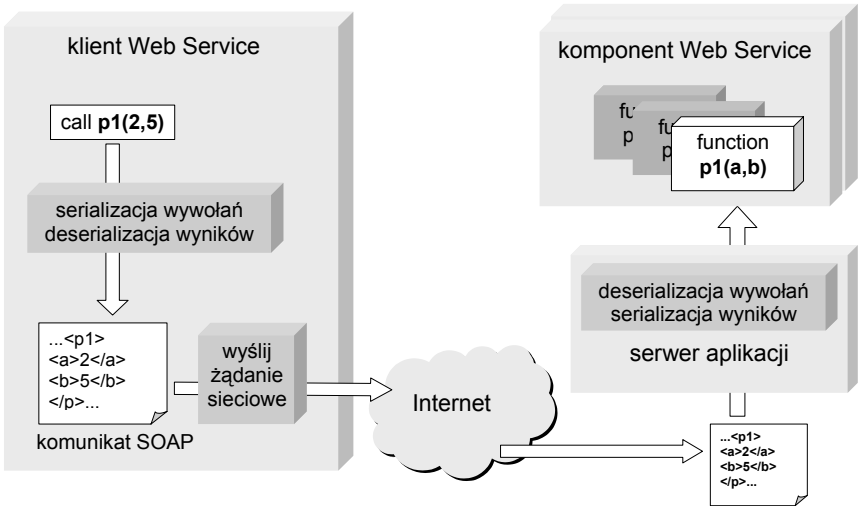
Powszechna akceptacja omawianej architektury systemów informatycznych spowodowała narodziny nowego modelu biznesu IT, bazującego na odpłatnym oferowaniu dostępu do biznesowych komponentów usługowych, z których z kolei korzystają twórcy aplikacji do obsługi przedsiębiorstw. Pomimo wciąż nieustabilizowanej sytuacji na rynku technologii billingowych wymaganych przez tego typu zastosowania, coraz więcej przedsiębiorstw docenia atrakcyjność takiej metody pozyskiwania przychodów, podkreślając jej zalety w kontekście ochrony własności intelektualnej (komponenty usługowe nie są fizycznie wręczane konsumentom) i ciągłości przychodów (konsument płaci za każde wywołanie usługi).

2. Technologia Web Services

Web Services to technologia konstrukcji rozproszonych komponentów usługowych, stanowiących podstawę dla realizacji aplikacji biznesowych w architekturze zorientowanej na usługi. Zgodnie z powszechnie akceptowaną definicją, usługa Web Service to zwarty, samodokumentujący się komponent programowy, który może być przez swojego twórcę zarejestrowany w sieci komputerowej, a następnie przez twórcę aplikacji-konsumenta odkryty i wywołany w trybie zdalnego wykonania. Technologia Web Services opiera się na szeregu skorelowanych rozwiązań informatycznych, spośród których najważniejsze to: protokół komunikacyjny SOAP – służący do przekazywania zdalnych wywołań, język opisu interfejsu usługi WSDL – służący do dystrybucji parametrów połączeń sieciowych, specyfikacja bazy danych UDDI – służącej do rejestracji udostępnianych komponentów usługowych.

Podstawowa architektura Web Services została przedstawiona na rys. 2. Rysunek przedstawia: (1) aplikację zainteresowaną wywołaniem kodu zdalnego modułu programowego – klient Web Service, (2) zdalny moduł programowy oferujący implementację funkcji logiki programowej – komponent Web Service, (3) serwer aplikacji odpowiedzialny za odbieranie i obsługę sieciowych żądań wywołania kodu zdalnego modułu programowego. Klient Web Service zapisuje treść wywołania komponentu w postaci komunikatu SOAP, stanowiącego rodzaj serializacji wywołania w formacie XML. Komunikat SOAP jest przekazywany za pośrednictwem wybranego protokołu sieciowego (najczęściej HTTP) do zdalnego serwera aplikacji. Serwer aplikacji dokonuje deserializacji komunikatu SOAP w

celu ekstrakcji pierwotnego zapisu wywołania komponentu Web Service, a następnie lokalnie przekazuje sterowanie do właściwej jednostki programowej komponentu. Ewentualne wyniki pracy komponentu mogą być w analogiczny sposób przekazane zwrotnie do klienta – następuje wówczas serializacja wyników do komunikatu SOAP, przesłanie sieciowe, a w ostatnim kroku deserializacja wyników po stronie klienta. Komponenty usługowe Web Service mogą pracować w trybie bezstanowym (ulotne wartości zmiennych pomiędzy wywołaniami) lub stanowym (nieulotne wartości zmiennych pomiędzy wywołaniami). Mogą również korzystać z dostępnych mechanizmów sesji, np. HTTPSession, oraz uwierzytelniania, np. HTTP Basic Authentication.



Rys. 2. Architektura wywołania komponentu Web Service

SOAP (Simple Object Access Protocol) to prosty protokół komunikacyjny oparty na języku XML, umożliwiający przekazywanie wywołań zdalnych komponentów Web Services. SOAP może współdziałać z dowolnym niskopoziomowym sieciowym mechanizmem transportowym, np. HTTP, HTTPS, SMTP, JMS, RMI. Podstawowymi znacznikami wykorzystywanymi do budowy komunikatów SOAP są: <Envelope> – otacza cały komunikat, <Header> – zawiera informacje nagłówkowe, <Body> – zawiera informacje o żądaniu i odpowiedzi, <Fault> – opisuje błędy, jakie wystąpiły podczas przetwarzania wywołania. Przykładowe komunikaty SOAP zostały przedstawione na rys. 3a i rys. 3b. Dotyczą one wywołania komponentu o nazwie „demo", zawierającego funkcję „multiply(int val1, int val2)". Przykład z rys. 3a przedstawia komunikat zawierający sparametryzowane wywołanie funkcji „multiply(3,2)". Natomiast przykład z rys. 3b obrazuje komunikat wynikowy, przekazujący klientowi wynik wywołania funkcji „multiply" – liczbę 6.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
   <soap:Body xmlns:m="demo">
      <m:multiply>
         <m:val1>3</m:val1>
         <m:val2>2</m:val2>
       </m:multiply>
```

```
    </soap:Body>
</soap:Envelope>
```

Rys. 3a. Komunikat SOAP zawierający wywołanie komponentu Web Service

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
   <soap:Body xmlns:m="demo">
      <m:multiplyResponse>
        <m:result>6</m:result>
      </m:multiplyResponse>
   </soap:Body>
</soap:Envelope>
```

Rys. 3b. Komunikat SOAP zawierający wynik wywołania komponentu Web Service

Protokół SOAP umożliwia wywoływanie komponentów Web Service w dwóch trybach: (1) Remote Procedure Call (RPC) i (2) dokumentowym (document-oriented). Tryby te różnią się formą przekazywania parametrów. W trybie RPC wywołanie ma charakter tradycyjny – komponentowi przekazywana jest lista parametrów formalnych wraz z ich bieżącymi wartościami. W trybie dokumentowym usługa otrzymuje tylko jeden parametr wywołania, którym jest dokument XML.

Mimo, iż implementacja środowiska aplikacyjnego w technologii Web Services jest możliwa w dowolnym z języków programowania, to jednak największą uwagę projektantów przyciąga obecnie język Java. W celu ułatwienia implementacji obsługi protokołu SOAP, programiści Java mogą korzystać z biblioteki Java API for XML-based RPC (JAX-RPC), która wyręcza ich z konieczności konstrukcji, przesyłania i parsowania XML-owych komunikatów SOAP. Warto nadmienić, że analogiczne biblioteki są dostępne dla innych popularnych języków programowania (np. SOAP::Lite dla Perla, gSOAP dla C/C++, ZSI dla Pythona, .NET). Przykład kodu aplikacji-klienta, dokonującej zdalnego wywołania funkcji „multiply(2,3)” komponentu Web Service o nazwie „demo”, przedstawiono na rys. 4.

```
Call call = new Call();
call.setTargetObjectURI("demo");
call.setMethodName("multiply");
call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
Vector params = new Vector();
params.addElement(new Parameter("val1", Integer.class, 2, null));
params.addElement(new Parameter("val2", Integer.class, 3, null));
call.setParams(params);
Response resp = call.invoke(new URL("http://miner/demo"), "");
Parameter ret = resp.getReturnValue();
Object value = ret.getValue();
System.out.println(value);
```

Rys. 4. Kod klienta Web Service w języku Java

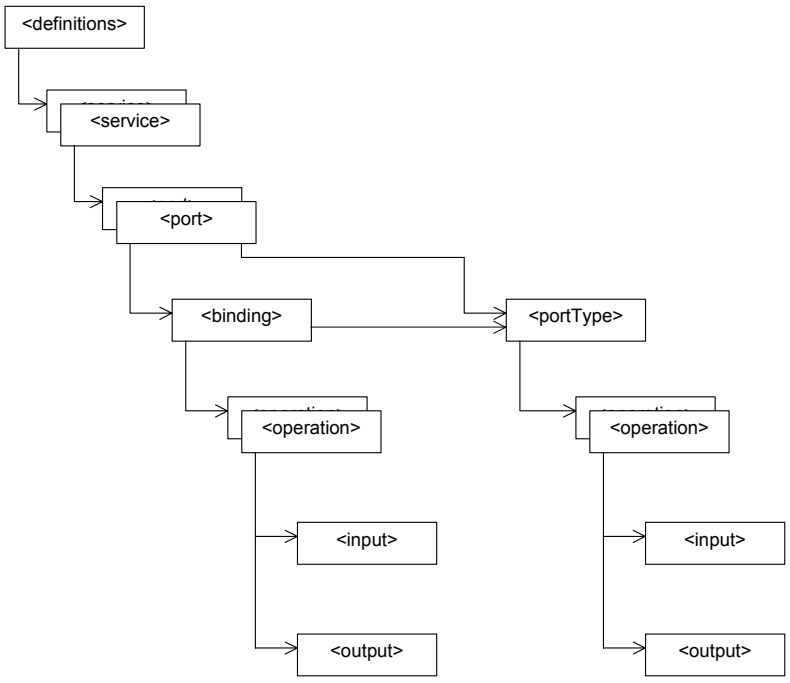
Pomimo dostępności bibliotek programistycznych wspomagających implementację komunikacji w oparciu o protokół SOAP, konstrukcja i pielęgnacja aplikacji-klienta wymaga od programisty istotnego wysiłku. W celu uproszczenia tych zadań postanowiono wykorzystać koncepcję automatycznego generowania kodu komunikacyjnego w oparciu o zestaw parametrów sieciowych opisujących komponent usługowy. Przyjęto, że twórca

komponentu Web Service opisuje jego interfejs w języku WSDL (Web Service Description Language), a twórca klienta Web Service dokonuje zautomatyzowanej transformacji tego opisu do kodu źródłowego obsługującego komunikację sieciową z komponentem (tzw. Web Service Proxy) w wybranym języku programowania. Transformacja ta może mieć charakter: (1) statyczny, gdy realizowana jest na etapie budowy/kompilacji aplikacji klienta, (2) dynamiczny, gdy realizowana jest na etapie wykonania aplikacji klienta. Przykład kodu źródłowego dokonującego dynamicznej transformacji dokumentu WSDL, a następnie wywołującego metodę „multiply(2,3)" przedstawiono na rys. 5.

```
WebServiceProxyFactory factory = new WebServiceProxyFactory();
WebServiceProxy proxy = factory.createWebServiceProxy("http:...demo.wsdl");
WebServiceMethod method = proxy.getMethod("multiply");
String paramNames[] = { "val1", "val2"};
Object paramValues[] = new Integer[2];
paramValues[0] = new Integer(2);
paramValues[1] = new Integer(3);
Object response = method.invoke(paramNames,paramValues);
```

Rys. 5. Dynamiczne wywołanie komponentu usługowego Web Service

WSDL to język znaczników XML służący do opisu technicznych parametrów połączenia sieciowego aplikacji-klienta z komponentem Web Service. Strukturę znaczników dokumentu WSDL przedstawiono na rys. 6. Role wymienionych znaczników są następujące. Znacznik <definitions> otacza całą zawartość dokumentu. Znaczniki <service> wraz ze znacznikami <port> definiują adresy punktów dostępowych dla usługi. Znaczniki <portType> służą do deklaracji funkcji biznesowych oferowanych przez usługę. Znaczniki <binding> określają metody kodowania parametrów wywołania i parametrów zwrotnych usługi. Przykładowy dokument WSDL przedstawiono w uproszczeniu na rys. 7. Jest to dokument opisujący usługę o nazwie „demo" zawierającą jedną metodę „multiply(int val1, int val2)".



Rys. 6. Struktura znaczników dokumentu WSDL

```
<definitions name="demo" ...>
   <message name="multiply0Request">
      <part name="val1" type="xsd:int"/>
      <part name="val2" type="xsd:int"/>
   </message>
   <message name="multiply0Response">
      <part name="return" type="xsd:int"/>
   </message>
   <portType name="DemoPortType">
      <operation name="multiply">
         <input name="multiply0Request"
                message="tns:multiply0Request"/>
         <output name="multiply0Response"
                 message="tns:multiply0Response"/>
      </operation>
   </portType>
   <binding name="DemoBinding" type="tns:DemoPortType">
      <soap:binding style="rpc" ...>
      <operation name="multiply">
         <input name="multiply0Request"> ... </input>
         <output name="multiply0Response"> ... </output>
      </operation>
   </binding>
   <service name="http://miner/Demo">
      <port name="DemoPort" binding="tns:DemoBinding">
         <soap:address location="http://miner/Demo"/>
      </port>
   </service>
</definitions>
```

Rys. 7. Przykładowy dokument WSDL

Wywołania komponentów usługowych Web Services mogą mieć charakter synchroniczny lub asynchroniczny. W trybie wywołania synchronicznego aplikacja klienta wysyła żądanie uruchomienia zdalnej funkcji biznesowej i wstrzymuje pracę aż do chwili otrzymania wyników jej realizacji. Tryb ten może opierać się na komunikacji HTTP, HTTPS, RMI/IIOP, SMTP, itp. W trybie wywołania asynchronicznego aplikacja klienta wysyła żądanie uruchomienia zdalnej funkcji biznesowej lecz nie oczekuje na jej wynik, kontynuując działanie. Tryb ten może opierać się na komunikacji HTTPR, JMS, IBM MQSeries Messaging, MS Messaging, itp. Zwykle tryb synchroniczny jest wykorzystywany przez komponenty RPC, natomiast tryb asynchroniczny – przez komponenty dokumentowe.

3. Tworzenie aplikacji Web Services z użyciem narzędzia JDeveloper

Oracle JDeveloper to wielozadaniowe, zintegrowane środowisko projektowania i programowania aplikacji w języku Java. Część funkcjonalności narzędzia JDeveloper umożliwia łatwą konstrukcję komponentów usługowych oraz klientów Web Services wykorzystujących tryb RPC i mechanizm transportowy HTTP lub tryb dokumentowy i mechanizm transportowy JMS. Kroki implementacji komponentu Web Service typu RPC są następujące: (1) utworzenie klasy Java zawierającej funkcje (metody) logiki biznesowej, (2) wygenerowanie definicji komponentu usługowego opartego na utworzonej klasie Java

(kreator Java Web Service), (3) instalacja komponentu na serwerze aplikacji J2EE (operacja Deploy). Wynikiem kroku (2) jest m.in. dokument deskryptora WSDL. JDeveloper umożliwia również graficzne projektowanie komponentów Web Services (kreator Web Service Diagram).

Tworzenie statycznego klienta Web Service sprowadza się do pozyskania pliku deskryptora WSDL usługi, której funkcje planuje się wywoływać, a następnie wygenerowania na jego podstawie klasy Java typu proxy (Web Service Stub), stanowiącej lokalne, lustrzane odbicie publicznej części klasy zdalnego komponentu usługowego. Użycie wygenerowanego kodu w aplikacji biznesowej polega na utworzeniu obiektu klasy Stub i wołaniu jej metod, nazwanych identycznie jak metody klasy zdalnego komponentu. Na rys. 8 przedstawiono: kod źródłowy przykładowej klasy Java zawierającej metody logiki biznesowej (klasa Demo), kod źródłowy klasy Stub statycznie wygenerowanej przez narzędzie JDeveloper (klasa DemoStub), kod aplikacji-klienta dokonującej wywołania metody zdalnego komponentu Web Service (klasa DemoClient).

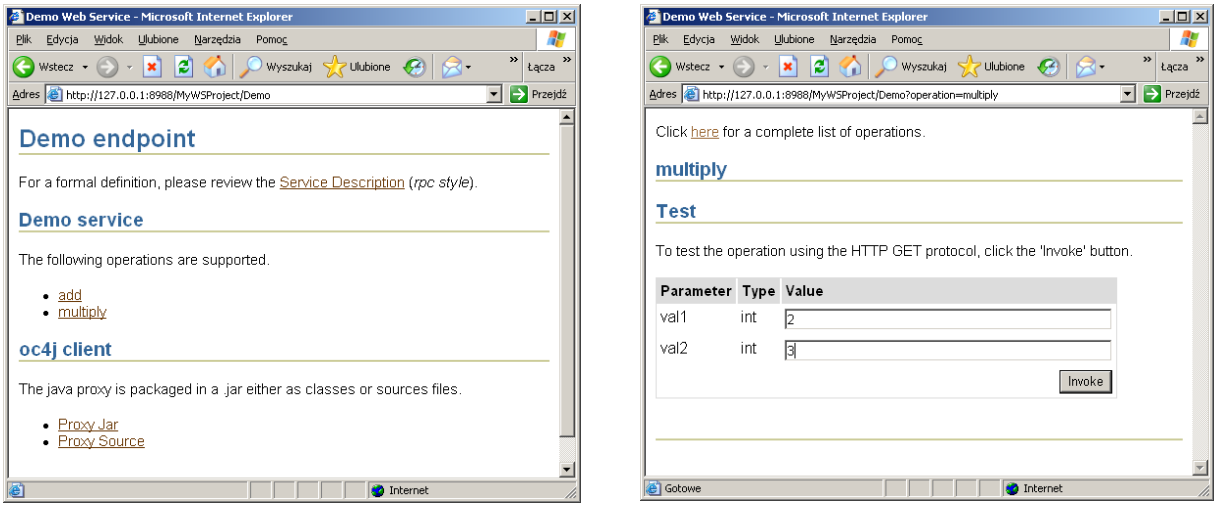
```
public class Demo
{
  public int multiply(int val1, int val2) {return val1 * val2;}
  public int add(int val1, int val2) {return val1 + val2;}
}

public class DemoStub
{
 public DemoStub() {...}
 public String getEndpoint() {...}
 public void setEndpoint(String endpoint) {...}
 public Integer add(Integer val1, Integer val2) throws Exception {...}
 public Integer multiply(Integer val1, Integer val2) throws Exception {...}
 public void setMaintainSession(boolean maintainSession) {...}
 public boolean getMaintainSession() {...}
 public void setTransportProperties(Properties props) {...}
 public Properties getTransportProperties() {...}
}

public class DemoClient
{
  public static void main(String[] args) throws Exception
  {
    DemoStub ds = new DemoStub();
    System.out.println(ds.multiply(new Integer(2),new Integer(3)));
  }
}
```

Rys. 8. Przykładowa implementacja środowiska Web Services

Inną interesującą funkcją narzędzia JDeveloper jest automatyczne generowanie warstw komponentów usługowych ponad istniejącymi w bazie danych Oracle pakietami PL/SQL (kreator PL/SQL Web Service). Dzięki temu możliwe jest szybkie przekształcenie posiadanych zasobów logiki biznesowej implementowanych w przeszłości w języku PL/SQL do technologii Web Services. Automatycznie wygenerowane komponenty zawierają kod SQLJ/JDBC odpowiedzialny za nawiązanie połączenia z bazą danych i za wywołanie funkcji lub procedury w pakiecie PL/SQL. Zadanie programisty sprowadza się zatem wyłącznie do implementacji aplikacji-klienta.

Komponenty Web Services tworzone za pomocą narzędzia JDeveloper mogą być instalowane na dowolnym serwerze aplikacji J2EE pod warunkiem dostępności biblioteki

JAX-RPC, zawierającej serwlety Java odpowiedzialne za odbieranie przybywających żądań sieciowych. Pomimo, iż dostęp do komponentu Web Service powinien odbywać się wyłącznie za pomocą protokołu SOAP, to jednak serwlety JAX-RPC oferują także interfejs HTML pozwalający programiście na wgląd w strukturę zainstalowanego komponentu. Na rys. 9 zamieszczono kopię okna przeglądarki WWW wyświetlającej wybrane strony interfejsu HTML. Strona główna interfejsu oferuje m.in. dokument WSDL (łącznik Service Description), gotową klasę Stub ułatwiającą implementację aplikacji-klienta (kod źródłowy: Proxy Source, kod skompilowany: Proxy Jar), a także formularze do testowania funkcji biznesowych usługi.



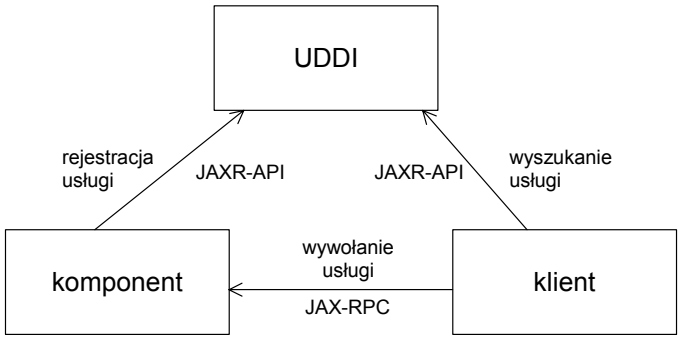
Rys. 9. Interfejs HTML serwletów JAX-RPC

4. Rejestry UDDI

W celu uproszczenia dystrybucji dokumentów WSDL i semantyki komponentów usługowych Web Services zaproponowano, aby dostępne komponenty były rejestrowane w specjalizowanej bazie danych, która może być następnie przeszukiwana przez twórców aplikacji-klientów lub przez same aplikacje-klientów z zamiarem odkrycia potrzebnych komponentów. Opracowaną specyfikację takiej bazy danych nazwano UDDI (Universal Description, Discovery, and Integration). Bazy danych UDDI deklarują dostępne usługi posługując się trzema poziomami opisu:

- White Pages: podstawowe dane kontaktowe o dostawcy usługi, obejmujące nazwę firmy, adres, identyfikator (np. numer identyfikacji podatkowej),
- Yellow Pages: lokalizacja usługi w taksonomiach kategoryzacyjnych,
- Green Pages: techniczny opis usługi i jej semantyki, często też wskaźnik do pliku WSDL.

Dostęp do rejestru UDDI jest zwykle możliwy zarówno poprzez interfejs HTML, jak i poprzez interfejs programistyczny (biblioteka JAXR-API). Na rys. 10 przedstawiono cykl życia usługi Web Service z punktu widzenia jej deklaracji w rejestrze UDDI: (1) twórca komponentu implementuje komponent usługowy Web Service i rejestruje go w UDDI, (2) twórca klienta wyszukuje opisu komponentu usługowego Web Service w UDDI i implementuje aplikację-klienta, (3) uruchomiona aplikacja-klient dokonuje zdalnego wywołania komponentu usługowego Web Service.



Rys. 10. Cykl życia usługi Web Service

Podczas trwania prac nad opracowaniem specyfikacji UDDI wiele uczestniczących w projekcie firm, m.in. Microsoft, IBM i SAP, uruchomiło publiczne rejestry UDDI, do których zgłaszane mogły być dowolne usługi. Stanowiło to interesującą pomoc dla programistów testujących własne rozwiązania. Jednak po zakończeniu prac projektowych, zarówno Microsoft, jak i IBM wstrzymały obsługę publicznych rejestrów, w zamian oferując komercyjne produkty umożliwiające uruchomienie własnych lokalnych rejestrów. W chwili przygotowywania tego opracowania dostęp do publicznego rejestru UDDI był możliwy wyłącznie w SAP (uddi.sap.com).

5. Zestaw programisty Java WSDP

Dla programistów Java zainteresowanych przede wszystkim konstrukcją aplikacji Web Services opracowano specjalny pakiet narzędziowy o nazwie Java WSDP. Pakiet ten zawiera zarówno podstawowe biblioteki programisty, jak i narzędzia uruchomieniowe. Poniżej wymieniono najważniejsze składniki pakietu:

- Java API for XML Processing (JAXP): podstawowe biblioteki przetwarzania danych XML, zgodne z rekomendacją W3C,
- Java API for XML Messaging (JAXM): biblioteka sieciowej komunikacji opartej na języku XML,
- SOAP with Attachments API for Java (SAAJ): biblioteka sieciowej komunikacji w oparciu o protokół SOAP,
- Java API for XML-based RPC (JAX-RPC): biblioteka służąca do realizacji zdalnych wywołań komponentów usługowych w trybie RPC,
- Java API for XML Registries (JAXR): biblioteka służąca do dostępu do XML-owych rejestrów usług Web Servicesm np. UDDI,
- Tomcat – zredukowany serwer J2EE umożliwiający instalowanie i uruchamianie komponentów usługowych Web Services,
- JavaServer Pages Standard Tag Library (JSTL): standardowa biblioteka znaczników JSP,
- Registry Server: rejestr UDDI.

6. Podsumowanie

Technologia Web Services umożliwia efektywną konstrukcję aplikacji biznesowych zgodnych z architekturą zorientowaną na usługi, według której logika biznesowa jest rozproszona pomiędzy wiele rozproszonych komponentów, często zarządzanych przez niezależne od siebie instytucje. Podstawowe kanony implementacji środowisk Web Services obejmują: implementację komponentu usługowego w dowolnym języku programowania, sporządzenie pliku opisu interfejsu komponentu (WSDL), zgłoszenie komponentu do centralnego rejestru (UDDI), wyszukanie komponentu w centralnym rejestrze (UDDI), wygenerowanie kodu komunikacyjnego dla klienta (Web Service Proxy), implementację klienta Web Service. Dzięki rozległemu wsparciu ze strony narzędzi programistycznych, tworzenie środowisk Web Services wymaga od programisty skupienia się prawie wyłącznie na implementacji kodu logiki biznesowej. Opisane w artykule rozwiązania nie stanowią jeszcze pełnej platformy rozwojowo-wdrożeniowej. Na dalszą uwagę zasługiwałyby m.in. rozwiązania z zakresu bezpieczeństwa (np. specyfikacja WS-Security) i opisu semantyki usług (np. koncepcja Semantic Web Services).

7. Literatura

[1] http://www.soaplite.com
[2] http://xml.apache.org/soap/
[3] http://uddi.microsoft.com
[4] http://www.microsoft.com/mind/0100/soap/soap.asp
[5] http://msdn.microsoft.com/soap/
[6] http://www.w3.org/TR/SOAP/
[7] http://www.w3.org/TR/wsdl