

Implementacja usług w środowisku bazy danych Oracle 10g

Bartłomiej Jabłoński

Uniwersytet Łódzki
e-mail: bartek@math.uni.lodz.pl

Abstrakt. Oracle 10g jako baza danych jest zazwyczaj tylko jednym z komponentów składających się na stosowane w IT rozwiązania. Do bazy danych dochodzą różnorodne aplikacje często pracujące w różnych środowiskach na różnych komputerach. O ile podstawy komunikacji aplikacja-baza danych zostały już dawno zaprojektowane i sprawdzone w praktyce, to integracja aplikacji między sobą, i to zarówno aplikacji pracujących w ramach jednego przedsiębiorstwa, jak i aplikacji udostępniających usługi różnych dostawców (B2B), wciąż pozostaje problemem otwartym.

Zaproponowany przez W3C i OASIS standard SOAP doczekał się już wielu rozwiązań po stronie producentów narzędzi deweloperskich. Twórcy oprogramowania mogą teraz w swoich produktach umieszczać uniwersalny interfejs, dzięki któremu wymiana danych pomiędzy aplikacjami jest łatwiejsza. Zwiększająca się popularność tego standardu sprawiła, że zmienia się także podejście do projektowania systemów – mówi się o architekturze komponentowo-usługowej (SOA – service oriented architecture).

Serwer bazy danych Oracle 10g zawiera metody i narzędzia, przy pomocy których programista może zarówno udostępniać swoje usługi jak i czerpać z szerokiego wachlarza usług dostępnych w Internecie. Niniejszy artykuł opisuje te możliwości skupiając się głównie na rozwiązaniach niskopoziomowych nie wymagających dużego udziału programisty.

Informacja o autorze. Bartłomiej Jabłoński od 1999 roku pracuje na stanowisku asystenta na Wydziale Matematyki Uniwersytetu Łódzkiego, gdzie zajmuje się głównie bazami danych Oracle oraz standardem XML ze szczególnym uwzględnieniem jego zastosowań w bazach danych. Od 1994 roku wielokrotnie uczestniczył w projektach informatycznych związanych z bazami danych Oracle jako konsultant. Od 1995 prowadzi szkolenia w Centrum Edukacyjnym Oracle Polska. Jest autorem trzech i tłumaczem czterech książek z zakresu informatyki.

1. Wstęp

Baza danych jako taka, mimo iż posiada wbudowany serwer WWW, to jednak nie może sama być serwerem usług SOAP. Nie znaczy to wcale, że nie może takich usług świadczyć. Rolę pośrednika pomiędzy klientem SOAP a bazą danych musi pełnić zewnętrzny serwer WWW. Programista jednak nie jest skazany na własne rozwiązania, firma Oracle bazę danych uzupełniła o szereg bibliotek i narzędzi, dzięki którym udział wspomnianego programisty sprowadza się do minimum.

Należy jednak podkreślić, że serwer WWW jest tylko biernym pośrednikiem, nie wymaga się od niego ingerowania w komunikację na poziomie SOAP (choć jest to możliwe). Za całość odpowiada serwer bazy danych. Takie rozwiązanie sprawia, że problemy związane z buforowaniem komunikatów, przetwarzaniem nagłówek i samą realizacją usług spoczywają na serwerze bazy danych, w której istnieją doskonałe, od wielu lat sprawdzone, mechanizmy monitorowania, i zabezpieczania na wypadek katastrof.

Choć baza danych Oracle nie może być serwerem usług SOAP, to może być ich klientem. Cała komunikacja SOAP odbywa się przecież w oparciu o protokół HTTP. W standardowych pakietach Oracle znajduje się taki, który umożliwia przesyłanie takich zleceń. Należy jednak podkreślić, że formowanie komunikatu SOAP niestety trzeba oprogramować już ręcznie.

W dalszej części artykułu zostaną przedstawione następujące rozwiązania:

- Baza danych Oracle jako "serwer" SOAP z wykorzystaniem serwletu AQ oraz protokołu IDAP,
- Baza danych Oracle jako "serwer" SOAP z realizowaniem usług przy pomocy PL/SQL-a,
- Baza danych Oracle jako klient SOAP z wykorzystaniem pakietu UTL_HTTP.

Słowo "serwer" zostało wzięte w cudzysłów, gdyż jak wspomniano wcześniej rolę faktycznego serwera będzie pełnił zewnętrzny serwer WWW. W pierwszym przypadku będzie to Tomcat, w drugim – OC4J.

2. Protokół IDAP i zaawansowane kolejki (AQ)

Zaawansowane kolejki

Advanced Queue (AQ) to mechanizm kolejkowania komunikatów zaimplementowany w bazie danych Oracle. Użytkownicy mogą wkładać komunikaty i pobierać je według ustalonych reguł, co zapewnia asynchroniczny sposób porozumiewania się użytkowników/aplikacji. Do ważniejszych cech kolejek można zaliczyć:

- ustalanie priorytetów,
- łączenie komunikatów w grupy,
- rozgłaszanie do wielu odbiorców,
- wyszukiwanie w oparciu o zadane kryteria,
- automatyczne przekierowywania pomiędzy kolejkami,
- automatyczne transformacje,
- kwalifikacja komunikatów do specjalnych kolejek (śmieć, błędy, przeterminowania),
- określanie czasu dotarcia komunikatu do adresata lub moment przedawnienia się komunikatu.

W przypadku korzystania z Web Services, wielkim atutem kolejek AQ jest funkcja buforowania nadchodzących komunikatów, wykorzystanie mechanizmów serwera w zakresie odporności na awarie, wykonywanie kopii zapasowych, śledzenia drogi komunikatu, ustalania uprawnień, wykorzystywania standardowych protokołów i narzędzi do obsługi komunikatów jako danych relacyjnych. Najważniejszą chyba jednak cechą jest przechowywanie treści komunikatu SOAP w postaci danych typu XMLType.

IDAP

Kolejki mogą być obsługiwane (w sensie kolejkowania i dekokolejkowania komunikatów) z poziomu PL/SQL, języka Java (technologia Java Messaging Service) oraz zdalnie przy pomocy SOAP. Odpowiednie polecenie wraz z parametrami opisane jest przy pomocy XMLa i stanowi treść właściwą (body) komunikatu SOAP. Zawartość ta musi być zgodna z IDAP (Internet Data Access Presentation), który jest niczym więcej jak opisem schematu, na którym musi być oparty wspomniany dokument XML¹.

Protokół IDAP pozwala na komunikację klientom SOAP z kolejkami Advanced Queue. Oracle udostępnia komponent J2EE (serwlet), który prowadzi nasłuch komunikatów SOAP. Komunikaty te w swojej treści (<Body>) muszą zawierać odpowiednio skomponowany dokument XML, który zawiera rozkazy wstawiania lub pobierania danych z kolejek.

Protokół IDAP charakteryzuje się przede wszystkim:

- Transakcyjnością – kilka oddzielnych komunikatów SOAP (czyli poleceń kolejkowych IDAP) może być potraktowane jako jedna transakcja biznesowa.
- Bezpieczeństwem – wszystkie polecenia IDAP mogą być wykonywane wyłącznie przez autoryzowanych i uwierzytelnionych użytkowników.

Rozwiązanie

Przygotowanie środowiska składa się z kilku etapów:

1. Przygotowanie i skonfigurowanie kolejek oraz nadanie uprawnień do korzystania z nich odpowiednim agentom.
2. Przygotowanie i zainstalowanie serwletu w środowisku serwera J2EE (a właściwie kontenera serwletów).
3. Przygotowanie klienta, który będzie wysyłał komunikaty SOAP/IDAP z odpowiednią zawartością. Rolą klienta obarczona zostanie strona WWW.

Rozwiązanie zostanie zilustrowane na przykładzie Fabryki Samochodów, która przyjmuje zamówienia na dostawę samochodów przy pomocy SOAPa. Zakłada się, że z tej możliwości będą mogli korzystać również "zwykli" użytkownicy. Specjalnie dla nich zostanie przygotowany klient SOAP w postaci strony WWW. W celu uniknięcia "fałszywych zleceń" każde zlecenie musi być potwierdzone mailem.

A. Konfiguracja kolejek AQ

W pierwszej kolejności tworzymy tabelę, w której fizycznie będą przechowywane przysyłane komunikaty SOAP oraz właściwe kolejki (obie kolejki w jednej tabeli):

```
begin
  DBMS_AQADM.CREATE_QUEUE_TABLE('ZAMOWIENIA_TAB', 'SYS.XMLTYPE',
                                multiple_consumers=>true);

  DBMS_AQADM.CREATE_QUEUE('ZAMOWIENIA_AQ', 'ZAMOWIENIA_TAB');
  DBMS_AQADM.CREATE_QUEUE('ZAMOWIENIA_MAIL_AQ', 'ZAMOWIENIA_TAB');

  DBMS_AQADM.START_QUEUE('ZAMOWIENIA_AQ');
  DBMS_AQADM.START_QUEUE('ZAMOWIENIA_MAIL_AQ');
end;
/
```

Należy pamiętać o uruchomieniu kolejek każdorazowo po restarcie serwera. W przeciwnym wypadku nie jest możliwe umieszczanie lub odczytywanie komunikatów. Kolejny etap, to zdefiniowanie transformacji. Zakładamy, że po przysłaniu zamówienia (umieszczeniu go w kolejce ZAMOWIENIA_AQ, należy wysłać do klienta mail z prośbą o potwierdzenie. Wysyłką maila będzie się zajmować wyspecjalizowany proces nasłuchujący komunikatów w kolejce ZAMOWIE-

¹ IDAP może być również zaimplementowany w środowisku nieSOAPowym.

NIA_MAIL_AQ. Ponieważ do wysłania takiego maila potrzebny jest wyłącznie adres mailowy oraz identyfikator, więc z całego zamówienia wycinany jest znacznik EMAIL oraz dokładany identyfikator wzięty z sekwencji:

```
begin
  execute immediate 'create sequence aq_seq';
  DBMS_TRANSFORM.CREATE_TRANSFORMATION(
    'FABRYKA', 'ZAM_TO_MAIL', /* schemat i nazwa */
    'SYS', 'XMLTYPE',        /* typ źródłowy */
    'SYS', 'XMLTYPE',        /* typ docelowy */
    'XMLElement("EMAIL", xmlattributes(aq_seq.nextval as "id"),
      extractvalue(source.user_data, '/ZAMOWIENIE/EMAIL'))');
end;
/
```

W kolejnym etapie należy zdefiniować użytkowników kolejek:

```
begin
  DBMS_AQADM.ADD_SUBSCRIBER(
    'FABRYKA.ZAMOWIENIA_AQ',
    sys.aq$_agent(null, 'FABRYKA.ZAMOWIENIA_MAIL_AQ', null),
    transformation=>'FABRYKA.ZAM_TO_MAIL');
  DBMS_AQADM.ADD_SUBSCRIBER(
    'FABRYKA.ZAMOWIENIA_MAIL_AQ',
    sys.aq$_agent('Local2', null, null));
end;
/
```

Aby komunikaty automatycznie przepływały z jednej kolejki do drugiej (w między czasie przechodząc transformację) należy zdefiniować proces za to odpowiedzialny. W serwerze Oracle wystarczy uruchomić usługę, a w oparciu o wcześniej zdefiniowane reguły proces sam zorientuje się, które komunikaty z jakiej kolejki do jakiej przepisać.

```
begin
  DBMS_AQADM.SCHEDULE_PROPAGATION('FABRYKA.ZAMOWIENIA_AQ');
end;
/
```

Można teraz przetestować same kolejki wstawiając wiadomość do ZAMOWIENIA_AQ (symulacja napłynięcie komunikatu SOAP):

```
declare
  enq_opt dbms_aq.enqueue_options_t;
  msg_prop dbms_aq.message_properties_t;
  msgid raw(16);
  xml XMLType;
begin
  xml := XMLType('<?xml version="1.0"?>
<ZAMOWIENIE>
  <NAZWISKO>Kowalski</NAZWISKO>
  <EMAIL>testsoap@tai.math.uni.lodz.pl</EMAIL>
  <WYPOSAZENIE>
    <ELEMENT>klima</ELEMENT>
  </WYPOSAZENIE>
</ZAMOWIENIE>');
  DBMS_AQ.ENQUEUE('ZAMOWIENIA_AQ', enq_opt, msg_prop, xml, msgid);
  commit;
end;
/
```

Za wysyłanie maila do klienta będzie odpowiedzialna procedura, która co jakiś czas sprawdzi, czy są w kolejce wiadomości i przy pomocy pakietu UTL_MAIL² dokona wysyłki³:

```

create or replace procedure sendPotwZamowienie is
  tresc varchar2(1000);
  adres varchar2(1000);
  dequeue_options      DBMS_AQ.dequeue_options_t;
  message_properties   DBMS_AQ.message_properties_t;
  wiadomosc xmltype;
  mid raw(16);
  no_messages          exception;
  bad_mail_address     exception;
  PRAGMA EXCEPTION_INIT (no_messages, -25228);
  PRAGMA EXCEPTION_INIT (bad_mail_address, -29279);
begin
  loop //pobranie wiadomości (o ile jest coś)
    dequeue_options.navigation := DBMS_AQ.FIRST_MESSAGE;
    dequeue_options.wait       := DBMS_AQ.NO_WAIT;
    dequeue_options.consumer_name := 'Local2';
    dequeue_options.dequeue_mode := DBMS_AQ.REMOVE;
    //nie ma błędu, czyli jest komunikat, trzeba wysłać mailem
    DBMS_AQ.DEQUEUE('FABRYKA.ZAMOWIENIA_MAIL_AQ', dequeue_options,
      message_properties, wiadomosc, mid);
    select 'Prosimy o potwierdzenie zamówienia poprzez kliknięcie '
      ||'w poniższy link' ||Chr(13)||Chr(10)
      ||'   http://www.fabryka.com/potwierdzenie/potw?id='
      ||'                                     extractvalue(wiadomosc, '/EMAIL/@id')
      into tresc
      from dual;
    select extractvalue(wiadomosc, '/EMAIL')
      into adres
      from dual;

    begin
      UTL_MAIL.SEND ('Fabryka Samochodów <potwierdzenie@fabryka.com>', adres,
        subject=>'Potwierdzenie zamówienia', message=> tresc);
    exception
      when bad_mail_address then null;
    end;
  end loop;
exception
  when no_messages then null;
end;
/

exec dbms_scheduler.create_job('MAIL_POTW_ZAM', 'STORED_PROCEDURE', -
  'sendPotwZamowienie', enabled=>true, -
  start_date=>systimestamp, -
  repeat_interval=>'freq=MINUTELY');

```

² Nie należy zapominać o zdefiniowaniu serwera SMTP przy pomocy zmiennej inicjalizującej SMTP_OUT_SERVER.

³ Istnieje możliwość automatycznego powiadamiania o nadejściu komunikatu (notyfikacja), jednak wymaga ona specjalnej rejestracji użytkowników. Dla zewnętrznych "jednorazowych" klientów taka rejestracja i późniejsze wyrejestrowanie jest nieopłacalne.

B. Konfiguracja serwletu AQ i kontenera serwletów

Serwlet AQ otrzymujemy poprzez napisanie klasy dziedziczonej od klasy *oracle.AQ.xml.AQxmlServlet*. Dzięki temu mamy możliwość zdefiniowania połączenia JDBC oraz dodatkowych akcji związanych z obsługą nadchodzących komunikatów SOAP lub wychodzących odpowiedzi (np. zaprezentowana tu możliwość debugowania).

```
public class SOAPServer extends oracle.AQ.xml.AQxmlServlet {

    public AQxmlDataSource createAQDataSource()
        throws AQxmlException, SQLException { //definiowanie połączenia
        AQxmlDataSource aqXmlDataSource = new AQxmlDataSource(
            "uzyt", "pass", "sid", "host",
"port");
        return aqXmlDataSource;
    }

    public void init(ServletConfig config) throws ServletException {
        AQxmlDataSource aqXmlDataSource = null;
        try { //ustawianie poziomu śledzenia
            AQxmlDebug.setTraceLevel(5);
            AQxmlDebug.setDebug(true);
            AQxmlDebug.setLogStream(new FileOutputStream("logs\\aqlogfile.log"));
        } catch (Exception excep) {excep.printStackTrace(System.out);}
        try { //definiowanie źródła danych
            super.init(config);
            aqXmlDataSource = this.createAQDataSource();
            setAQDataSource(aqXmlDataSource);
        } catch (Exception excep) {excep.printStackTrace(System.out);}
    }
}
```

Serwlet umieszczamy w kontenerze serwletów (tutaj w Tomcat 5.0) rejestrując go w deskrypcorze rozmieszczenia. Należy pamiętać o tym, aby na zasobie (na serwlecie) zdefiniować zabezpieczenia (przykładowy plik *web.xml* zaprezentowano poniżej – rola soap i użytkownicy przypisani do tej roli zdefiniowani są w pliku *tomcat-users.xml*):

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<web-app>
    <!--rejestracja serwletu i mapowanie URLa-->
    <servlet>
        <servlet-name>SOAPServer</servlet-name>
        <servlet-class>SOAPServer</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>SOAPServer</servlet-name>
        <url-pattern>/SOAPServer</url-pattern>
    </servlet-mapping>

    <!--ustanowienie reguł bezpieczeństwa-->
    <security-role>
        <role-name>soap</role-name>
    </security-role>
    <login-config>
        <auth-method>BASIC</auth-method>
    </login-config>
    <security-constraint>
        <web-resource-collection>
```

```

    <web-resource-name>Wpisywanie zamowien</web-resource-name>
    <url-pattern>/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>soap</role-name>
  </auth-constraint>
</security-constraint>
</web-app>

```

Po zarejestrowaniu można przetestować prawidłowe działanie serwletu wysyłając dowolne żądanie GET:



Rysunek 1. Testowanie serwletu AQ.

C. Przygotowanie strony WWW

Dla klienta przygotowano formularz HTML, w którym może on zdefiniować swoje potrzeby. Do celów testowych na stronie umieszczono dodatkowo dwa pola prezentujące wysłany komunikat i odebraną odpowiedź (oczywiście w postaci SOAP). Sama komunikacja odbywa się z wykorzystaniem technologii AJAX, co pozwala na realizację zadania bez konieczności przeładowania strony.

```

<%@ page language = "java" %>
<html>
<head>
<title>Składanie zamówień</title>
<script language="javascript">
function sendSOAP(){
  //przygotowanie obiektu komunikacji AJAX
  if (window.XMLHttpRequest) { //np. Mozilla
    xmlhttp = new XMLHttpRequest();
  } else if(window.ActiveXObject) { //IE
    xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
  }
  var async = true;
  //przygotowanie komunikatu SOAP
  var xml = "<?xml version='1.0'?>"
+ " <Envelope xmlns= 'http://schemas.xmlsoap.org/soap/envelope/'>"
+ "   <Body>"
+ "     <AQXmlSend xmlns = 'http://ns.oracle.com/AQ/schemas/access'>"
+ "       <producer_options>"
+ "         <destination>FABRYKA.ZAMOWIENIA_AQ</destination>"
+ "       </producer_options>"
+ "       <message_set>"
+ "         <message_count>1</message_count>           "
+ "         <message>"
+ "           <message_number>1</message_number>"
+ "         <message_header>"

```

```

+ "          <correlation>ORDER1</correlation>"
+ "          <sender_id>"
+ "            <agent_name>klient</agent_name>"
+ "          </sender_id>"
+ "        </message_header>"
+ "        <message_payload> "
+ "          <ZAMOWIENIE xmlns=''>"
+ "            <NAZWISKO>"+document.zamowienie.nazwisko.value+"</NAZWISKO>"
+ "            <EMAIL>"+document.zamowienie.email.value+"</EMAIL>"
+ "            <WYPOSAZENIE>;
if (document.zamowienie.klima.checked) {
  xml = xml + "<ELEMENT>klima</ELEMENT>";
}
if (document.zamowienie.szyby.checked) {
  xml = xml + "<ELEMENT>szyby</ELEMENT>";
}
xml = xml + "          </WYPOSAZENIE>"
+ "        </ZAMOWIENIE>"
+ "      </message_payload>"
+ "    </message>"
+ "  </message_set>"
+ "  <AQXmlCommit/>"
+ " </AQXmlSend>"
+ " </Body>"
+ "</Envelope>";
      //wyświetlenie żądania SOAP
      document.getElementById("send").innerHTML= xml;
      //przygotowanie procedury odbioru odpowiedzi SOAP
      xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4) {
          document.getElementById("result").innerHTML= xmlhttp.responseText;
        }
      }
      //wysyłka komunikatu
      xmlhttp.open("POST", "http://localhost/soap2/SOAPServer", async);
      xmlhttp.setRequestHeader("SOAPMethodName",
"http://ns.oracle.com/AQ/schemas/access#AQXmlSend");
      xmlhttp.setRequestHeader("SOAPAction", "test");
      xmlhttp.setRequestHeader("Content-Type", "text/xml;charset=UTF-8");
      xmlhttp.setRequestHeader("Authorization", "Basic a2xpZW50Og==");

      xmlhttp.send(xml.toString());
    }
  </script>
</head>
<body>
<!--właściwy formularz-->
<form name="zamowienie" action="" method="post">
Nazwisko: <input name="nazwisko" type="text"/><br/>
E-mail: <input name="email" type="text"/><br/>
Klimatyzacja: <input name="klima" type="checkbox"/><br/>
Elektryczne szyby: <input name="szyby" type="checkbox"/><br/>
<input type="button" value="Zamów" onClick='sendSOAP()' />
</form>

```

```

<!--pola testowe -->
<textarea id="send" cols="80" rows="20"></textarea><br/>
<textarea id="result" cols="80" rows="20"></textarea>
</body>
</html>

```

W zaprezentowanym kodzie, którego efekt można zobaczyć na rysunku 2. można zwrócić uwagę na tworzony komunikat SOAP i jego zawartość w postaci IDAP. Wykorzystano tutaj możliwość wkładania komunikatów do kolejki (element `<AQXmlSend>`). Kolejka identyfikowana jest poprzez element `<destination>`. Na koniec transakcja jest zatwierdzana (element `<AQXmlCommit>`).



Rysunek 2. Formularz WWW zamówienia.

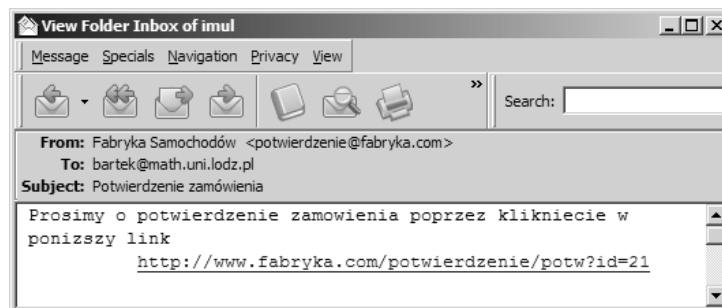
Ważną rolę stanowi tutaj również zdefiniowanie użytkownika uprawnionego do korzystania z kolejek poprzez WWW (element `<agent_name>` oraz deklaracja zaprezentowana poniżej).

```

begin
  dbms_aqadm.create_aq_agent(agent_name=>'klient', enable_http =>true);
  dbms_aqadm.enable_db_access('klient', 'FABRYKA');
end;
/

```

Dowodem działania tego scenariusza jest otrzymany mail z prośbą o potwierdzenie zamówienia. Obsługa tego potwierdzenia daje się również łatwo zaimplementować w postaci kolejek.



Rysunek 3. Mail z prośbą o potwierdzenie.

Uwagi końcowe

Może się wydawać, że dla tak prostego przykładu napisano dużo kodu i że można było zrobić to prościej. Należy jednak zwrócić uwagę, że większość kodu ma charakter deklaracyjny (deklaracja, że kolejka ma istnieć, że komunikaty mają się przepisywać z jednej do drugiej, że serwlet ma się łączyć z bazą, itd), jedyny kod, który "coś robi", to JavaScript w formularzu do zbudowania poprawnego komunikatu SOAP. W większych aplikacjach stosunek ilości kodu do liczby usług przedstawia się dużo korzystniej.

Kolejną obserwacją jest fakt, że serwlet AQ łączy się z bazą danych przy pomocy JDBC OCI8, co oznacza, że na komputerze z Tomcatem należy wcześniej zainstalować klienta Oracle.

Protokół IDAP posiada jeszcze następujące polecenia:

| Znacznik | Opis polecenia |
|----------------------|--|
| AQXmlSend | Umieszczenie komunikatu w kolejce AQ |
| AQXmlPublish | Umieszczenie komunikatu w kolejkach AQ |
| AQXmlReceive | Pobranie komunikatu z kolejki AQ |
| AQXmlRegister | Rejestracja powiadamiania – notyfikacji - (użytkownicy mogą być informowani nadejściu komunikatu w ich kolejce, np mailem, dzięki temu nie muszą sami nasłuchiwać) |
| AQXmlCommit | Zatwierdzenie transakcji |
| AQXmlRollback | Cofnięcie transakcji |

3. Usługi sieciowe w PL/SQL

Serwer Oracle może udostępniać swoje usługi zdefiniowane za pomocą procedur w języku PL/SQL. Jedyną trudność to, podobnie jak w poprzednim przypadku, konfiguracja serwera WWW, który będzie odbierał komunikaty SOAP i tłumaczył je na wywołania procedur bazodanowych oraz generował również SOAPową odpowiedź. Aby maksymalnie uprościć zadanie skorzystamy z narzędzi, które automatycznie wykonają tę pracę za programistę. Domyślny interfejs będzie może i naiwny, ale łatwiej jest zmodyfikować już coś działającego i przetestowanego, niż pisać setkę linii kodu od nowa.

Aby zilustrować mechanizm, weźmiemy pod uwagę dostawcę części samochodowych. Klienci, tacy jak użyta we wcześniejszym przykładzie Fabryka Samochodów, bardzo często zamawiają poszczególne komponenty. Jeśli takie zamówienia będzie mógł składać program zarządzający produkcją, to obrót materiałami będzie mógł odbywać się szybciej i taniej. Napijemy usługę realizującą przyjmowanie zamówień na komponenty.

Poszczególne etapy projektu to:

1. Utworzenie usługi w języku PL/SQL
2. Konfiguracja serwera SOAP
3. Utworzenie klienta SOAP

A. Tworzenie usługi

Tworzymy tabelę, w której będą rejestrowane poszczególne zlecenia:

```
create table oferty(
  fabryka varchar2(100),
  element varchar2(100),
  ilosc number(10)
);
```

Wpisanie zlecenia realizowane będzie poprzez procedurę napisaną w PL/SQL-u (z przyczyn formalnych musi być ona składową pakietu).

```
create or replace package zamowienia is
  function zamowienie_elementu(pFabryka varchar2, pElement varchar2,
```

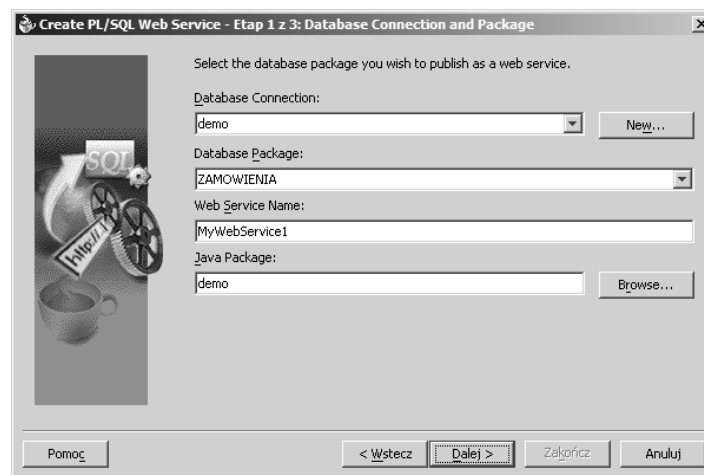
```
                pIlosc number) return varchar2;  
end;  
/  
  
create or replace package body zamowienia is  
    function zamowienie_elementu(pFabryka varchar2, pElement varchar2,  
                                pIlosc number) return varchar2 is  
begin  
    insert into oferty(fabryka, element, ilosc)  
    values(pFabryka, pElement, pIlosc);  
    commit;  
    return 'SUKCES';  
end;  
end;  
/
```

B. Konfiguracja serwera SOAP

Serwerem SOAP będzie w tym przypadku serwer OC4J. Odpowiednia konfiguracja w postaci pliku deskryptora rozmieszczenia (*web.xml*) oraz opis usługi (**.wsdl*) zostanie wykonana automatycznie w oparciu o definicję pakietu przez narzędzie JDeveloper.

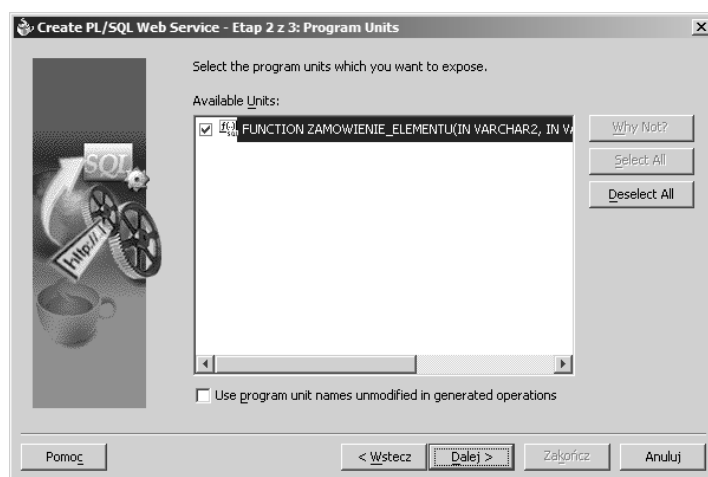
W pierwszej kolejności należy zdefiniować połączenie bazodanowe (wybieramy File ->New ->Database Tier -> Database Connection). Na kolejnych kilku stronach kreatora należy podać odpowiednie parametry do zdefiniowania wspomnianego połączenia (użytkownik, hasło, itp).

Generację serwera SOAP wykonamy również w oparciu o odpowiedni kreator. Po utworzeniu obszaru roboczego i projektu aplikacyjnego wybieramy z menu: File->New->Business Tier->PL/SQL WebService i na kolejnych stronach kreatora podajemy (a właściwie zatwierdzamy) właściwości naszego serwera.



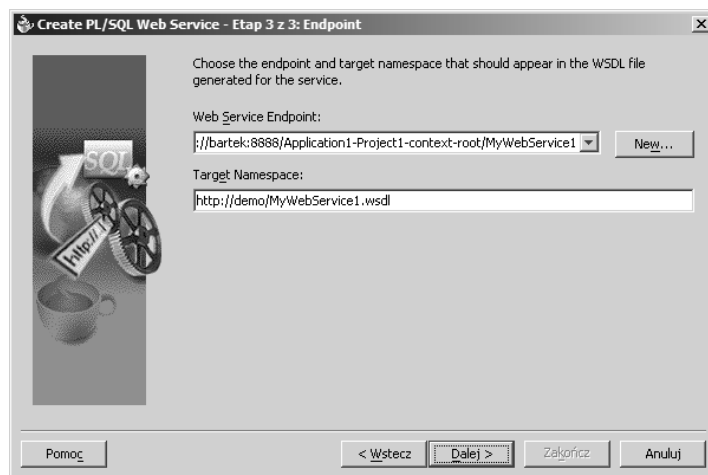
Rysunek 4. Wybór połączenia bazodanowego i określenie nazw pakietów.

Na kolejnej stronie należy zaznaczyć procedury do serwisowania:



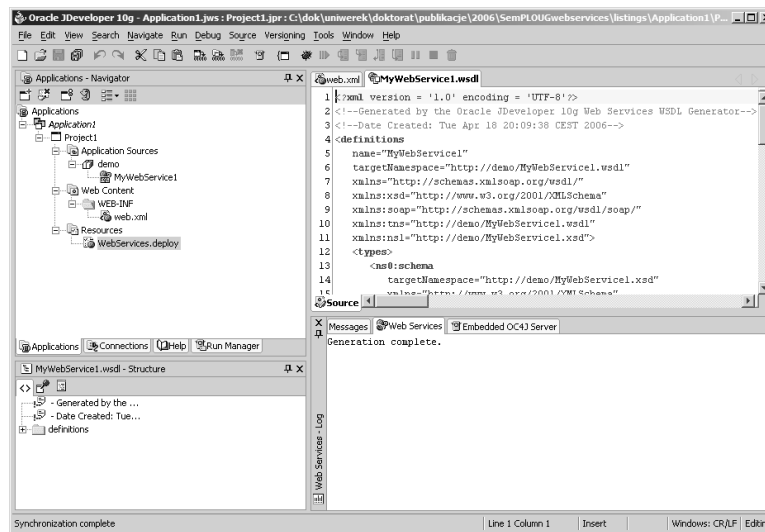
Rysunek 5. Wybór procedur PL/SQL do udostępnienia za pomocą protokołu SOAP.

Na koniec trzeba podać punkty wejściowe dla usług (generują się same – wystarczy zatwierdzić):



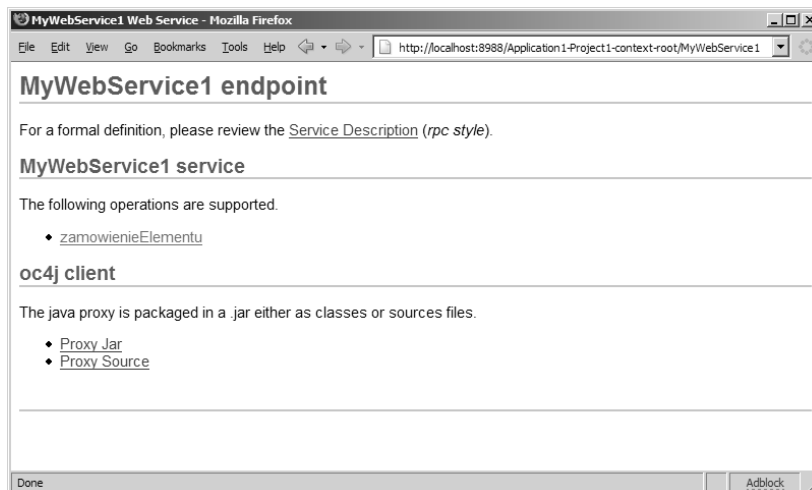
Rysunek 6. Wybór adresów dla zainstalowanej usługi.

Na zakończenie mamy już wszystkie elementy konfiguracji wraz z dodatkowo wygenerowaną aplikacją kliencką. Na rysunku 7. widoczny jest fragment dokumentu wygenerowanego przez JDeveloper.



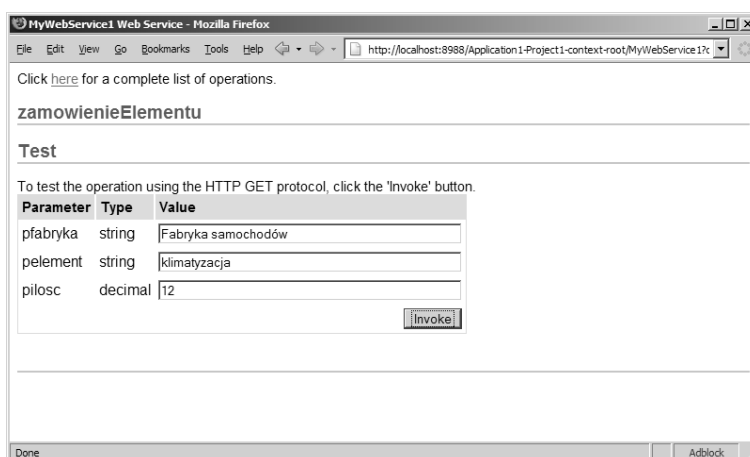
Rysunek 7. Widok wygenerowanej aplikacji SOAP

Po uruchomieniu aplikacji, można zorientować się, że oprócz możliwości przyjmowania komunikatów SOAP, wygenerowana została strona WWW dla potencjalnych użytkowników systemu. Ekran powitalny zawiera linki do formularzy obsługujących poszczególne usługi.

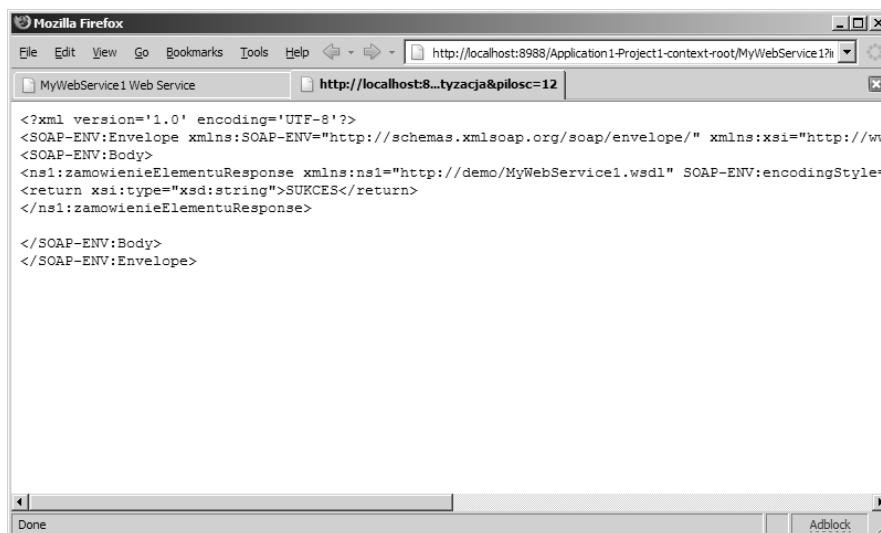


Rysunek 8. Aplikacja SOAP – klient WWW

Po kliknięciu w „zamówienieElementu” otwiera się formularz, w którym można zdefiniować zamówienie:



Rysunek 9. Wypełniony formularz zamówienia (przed ostateczną akceptacją).



Rysunek 10. Odpowiedź serwera SOAP.

Praktyczny skutek działania formularza można zaobserwować w bazie danych. Zgodnie z oczekiwaniami, zamówienie zostało zarejestrowane w tabeli:

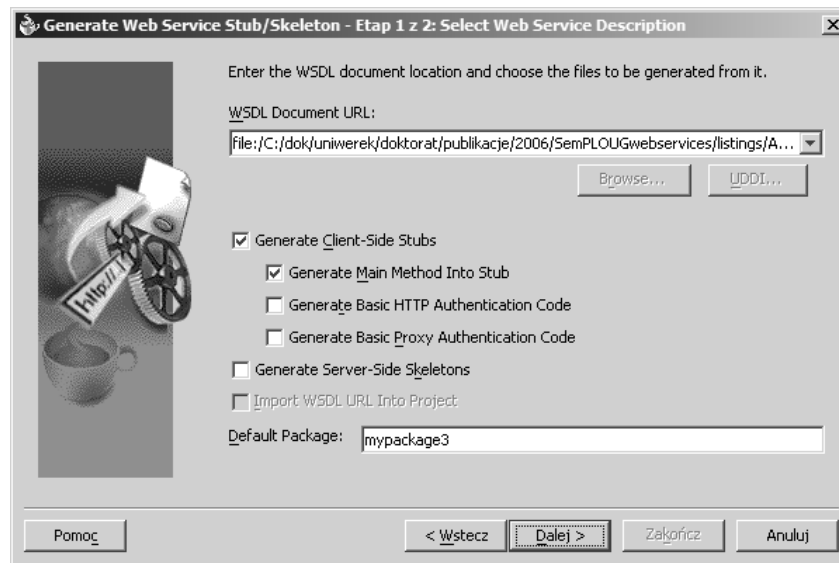
```
SQL> select * from oferty;
```

```
FABRYKA                ELEMENT                ILOSC
-----
Fabryka samochodów    klimatyzacja          12
```

C. Przygotowanie klienta

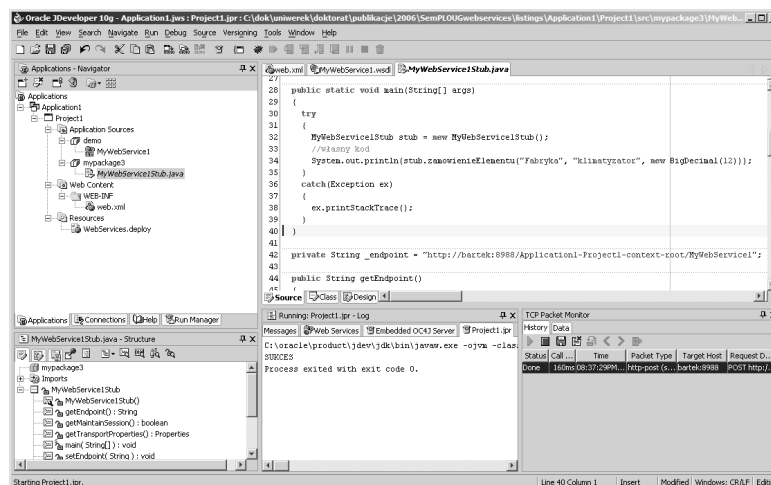
Jeżeli wczujemy się teraz w rolę twórców oprogramowania do zarządzającego produkcją, chcielibyśmy móc dopisać moduł, który w sposób programowy zamówi komponent u dostawcy części samochodowych. Innymi słowy chcemy napisać procedurę, która skorzysta z udostępnionej w poprzednim przykładzie usługi. W tym celu ponownie skorzystamy z możliwości jakie dają nam kreatory JDevelopera.

Klikając prawym klawiszem myszki na moduł *Zamowienia* i wybierając opcję *Generate Web Service Stub* otrzymamy szkielet klasy Javy, która pozwoli nam na komunikację SOAP z serwerem usługi.



Rysunek 11. Generator szkieletu klienta.

Na potrzeby tego konkretnego przykładu zdecydowano się na zaznaczenie opcji "Generate Main Method", aby można było łatwo przetestować klasę.



Rysunek 12. Wygenerowany szkielet aplikacji, z uzupełnionym kodem po przetestowaniu.

Wygenerowany szkielet należy teraz wypełnić własnym kodem. W naszym przypadku dodano jedną linijkę formułującą żądanie i wypisującą odpowiedź usługi (nie odpowiedź SOAP!) na ekranie (linijka ta jest widoczna na rysunku 12).

```
System.out.println(stub.zamowienieElementu("Fabryka", "klimatyzator",
    new BigDecimal(12)));
```

Na rysunku 12. widać również, że w ramach możliwości debugowania kodu, istnieje też narzędzie (TCP Packet Monitor) do podglądania komunikatów wysyłanych i otrzymywanych z serwera SOAP. Treść takiego komunikatu można nagrać do pliku (możliwość ta została praktycznie wykorzystana w kolejnym przykładzie).

4. Baza danych Oracle jako klient SOAP

Protokół SOAP jest przesyłany poprzez protokół HTTP⁴. Wykorzystując ten protokół można napisać procedurę PL/SQL, która sama zbuduje komunikat SOAP i sama przetworzy jego odpowiedź (tej ostatniej możliwości w artykule nie zaprezentowano).

```
create or replace function getSOAPZamowienie(pFabryka varchar2,
      pElement varchar2, pIlosc number) return XMLType is
begin
  return XMLType('<?xml version = "1.0" encoding = "UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:zamowienieElementu xmlns:ns1="MyWebService1"
      SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <pfabryka xsi:type="xsd:string">' || pFabryka || '</pfabryka>
      <pelement xsi:type="xsd:string">' || pElement || '</pelement>
      <pilosc xsi:type="xsd:decimal">' || pIlosc || '</pilosc>
    </ns1:zamowienieElementu>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>');
end;
/
```

Powyższy listing zakłada, że komunikat SOAP tworzony jest statycznie, bez oparcia o opis usługi. Jest to rzecz jasna najprostsze rozwiązanie, ale niestety najbardziej wrażliwe na błędy spowodowane zmianą interfejsu usługi, za którą odpowiedzialny jest inny, zewnętrzny podmiot. W procedurze wykorzystano postać komunikatu utworzonego w poprzednim przykładzie nagrany z monitora śledzenia TCP.

Przygotowany komunikat, wzbogacony o stosowne nagłówki, może być następnie wysłany do serwera SOAP realizującego usługę przyjmowania zamówienia na części samochodowe.

```
create or replace procedure sendSOAPZamowienie(soap XMLType) is
  http_req utl_HTTP.req;
  http_res utl_HTTP.resp;
begin
  http_req := utl_http.begin_request(
    'http://bartek:8988/Application1-Project1-context-root/MyWebService1',
    'POST', 'HTTP/1.0');
  utl_http.set_header(http_req, 'Content-Type', 'text/xml');
  utl_http.set_header(http_req, 'Content-Length',
length(soap.getStringval()));
  utl_http.set_header(http_req, 'SOAPAction', 'MyWebService1');
  utl_http.write_text(http_req, soap.getStringval());
  http_res := utl_http.get_response(http_req);
end;
/

exec sendSOAPZamowienie(getSOAPZamowienie('Fabryka Samochodów', -
      'klimatyzator', 12));
```

⁴ Można w tym celu wykorzystać również inne protokoły (SMTP, FTP), choć implementacje takie są rzadko stosowane w praktyce.

Bibliografia

8. [GrSi02] Graham S., Simeonov S., i in.: Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI. 2002 SAMS Publishing.
9. [GSB+03] Graham S., Simeonov S., Boubez T., Davis D., Daniels G., i in.: Java Usługi WWW Vademecum profesjonalisty, Helion 2003.
10. [Jab06] Jabłoński B.: Dane relacyjne a XML – metody konwersji, PLOUG XII Seminarium, Warszawa 2006
11. [JaWi05] Jabłoński B., Włodarski M.: Usługi sieciowe WebServices. Opis wdrożenia aplikacji Rezerwacja sal., mat. konf. BDAS Ustronie 2005
12. [OraAQ] Oracle Streams Advanced Queuing User's Guide and Reference, B14257-01
13. [OraDB] Oracle XML DB Developer's Guide, B14259-02
14. [OraXDK] Oracle XML Developer's Kit Programmer's Guide, B14252-01
15. [OraJava] Oracle XML Database XML Java API Reference B14293-01
16. [OraOtn] <http://otn.oracle.com>
17. [OraMet] <http://metalink.oracle.com>
18. [Pal02] Palarczyk E.: Advanced Queuing w biznesie. PLOUGtki nr 21
19. [Pri04] Prise J.: Build a PL/SQL Web Service, OTN
20. [SCW04] Scardina M., Chang B., Wang J.: Oracle Database 10g XML&SQL, Osborne/McGraw-Hill, 2004, ISBN 0-07-222952-7
21. [Woj06] Wojciechowski M.: AJAX – rewolucja w tworzeniu aplikacji internetowych, PLOUG XII Seminarium, Warszawa 2006