
Service Oriented Architecture, Web Services and REST Services

Maciej Malawski,
Tomasz Szepieniec,
Marek Kasztelnik

O czym w tym dziale?

- SOA
- XML
- SOAP
- WSDL
- UDDI

- REST

Architektury Strukturalne

- Dobre podejście
 - Obietnica: organizacja i powtórne użycie kodu
 - Procedury, funkcje i dane
 - Informatyka wyszła z ery lodowcowej
- W swoim czasie osiągnięto wiele:
 - Monolityczne, odizolowane aplikacje serwerowe
 - Asynchroniczne serwisy przetwarzania wsadowego (batch)
 - Synchroniczne serwisy terminalowe

Obiektowe-zorientowanie(OO)

- Bardzo dobre
 - Obietnica: organizacja i powtórne wykorzystanie kodu
 - Klasy (dane i funkcje razem ze sobą)
 - Zwiększenie produktywności developerów
- W swoim czasie osiągnięto wiele:
 - Monolityczne aplikacje klient-serwer
 - Bogate, interakcyjne interfejsy graficzne (GUI)
 - Stan-zachowujące (stateful) serwery o dużej wydajności lecz niskiej skalowalności

Zorientowanie na komponenty

- Bardzo, bardzo dobry pomysł
 - Obietnica: organizacja i powtórne wykorzystanie kodu
 - Komponenty (grupowanie interfejsów)
 - Niezależność od implementacji
 - Współpraca między-aplikacyjna stała się rzeczywistością
- Super!
 - Wielo-poziomowe, wielo-warstwowe aplikacje
 - Duże osiągi wydajności i wysoka skalowalność
 - OLE!

SP, OOP, COP

Programowanie Strukturalne, Programowanie OO, Programowanie Komponentowe

- Złe pomysły...
 - Paradygmaty „programowania”
 - One opisywały jak programować, a nie jak projektować architektury
 - Architektura była wymuszana przez styl programowania
 - Abstrakcja koncepcji lecz konkretne nie-elastyczne rzutowanie w języki
 - Reprezentacje w Pascalu, C++, CORBA, COM
 - Wtórne wykorzystanie kodu ograniczone wewnątrz domeny
 - Wtórne wykorzystywanie między domenami bardzo bolesne w planowaniu i realizacji
 - Kłopot przekraczania granic domen abstrakcji
 - Zbędne wymuszanie rozwiązań synchronicznych

Czego dziś potrzebujemy?

- Mało jest już projektów aplikacji typu „zielone działka” („od zera”), większość to „brązowa działka”
 - Nowe rozwiązania z reguły wykorzystują istniejące
 - Istniejące systemy potrzebują nowych
- Systemy heterogeniczne
 - Upgrade-y i ulepszenia nie są synchronizowane
 - Brak jednej rodziny systemów operacyjnych i architektur sprzętowych
- Efekt „Big Bang”
 - Wszystko umyka coraz to dalej od wszystkiego
 - Wszystko chce dostępu i prawa manipulacji każdą bazą danych...

Czego ponadto chcielibyśmy?

- Organizacji i powtórnego wykorzystanie kodu...
 - Wzdłuż warstw i aplikacji
 - Wszerz organizacji biznesowych i granic zaufania
- Niezależności od implementacji
 - Współpraca oparta o standardy
 - Niezależność od języka programowania i platformy
- Dynamicznego grupowania
 - Koncentracja lub rozproszenie klasterowe zależnie od potrzeb

Czego nie chcemy

- Obsługiwać infrastrukturę (transzacje, bezpieczeństwo itd.)
- Topić się w logice wyszukiwania, wywoływania i niezawodnej obsługi serwisów
- Tracić czas na pisanie nowych schematów pracy (workflow) dla serwisów infrastrukturalnych

Zorientowanie na serwisy (Service-Orientation)

- **SOA – Service Oriented Architecture**
- Czym jest „Serwis”?
 - Jakaś dostępna funkcjonalność
 - Lokalizacja, platforma i styl kodowania nie mają znaczenia
- Wnioski
 - Zakładaj: zawsze zdalny (remote) dostęp
 - Zakładaj: zawsze oczekuj dostępu z przeróżnych platform
 - Zignoruj wewnętrzną implementację serwisu

Unikaj tych założeń

- Nie myśl o serwisie jako o:
 - Transakcji, obiekcie czy funkcji
 - Synchronicznym czy asynchronicznym
 - Zachowującym-stan czy bezstanowym
 - Adresie, który wywołujesz
- Bo serwisy mogą być:
 - Odbiorcami wiadomości (aplikacje)
 - Obsługą i przetwarzaniem wiadomości (infrastruktura)
 - Nośnikami wiadomości (transport)

Pomocne adresy

- www.w3schools.org
- www.xml.pl
- www.w3c.org
- JSR-224 jcp.org/en/jsr/detail?id=224
(jax-ws 2.0)
- JSR-370 jcp.org/en/jsr/detail?id=370
(jax-rs 2.1)

Web Service po polsku

- „Webserwisy”
- Usługi sieci Web – Microsoft
- Usługi „webowe”
- Usługi WWW
- **Usługi sieciowe**

Web Service – po co to jest

- Pojęcie usługi sieciowej
- „WWW dla maszyny”
- Integracja systemów - „interoperability”
- Standardy internetowe W3C: HTTP, SOAP, WSDL, UDDI
- Udział gigantów: Microsoft, IBM

Przykłady usług

- Notowania giełdowe
- Serwisy pogodowe
- FedEx – serwis do śledzenia przesyłek
- www.kinkos.com - firma drukarska widziana jako drukarka Windows

RPC - problemy

- RMI, CORBA, DCOM – używają RPC (Remote Procedure Call) do komunikacji między obiektami
- Problemy z kompatybilnością
- Problemy z firewallami
- Duża złożoność tych systemów

Rozwiązanie: SOAP

- Przenoszony przez HTTP – protokół obsługiwany przez serwery, przeglądarki, serwery proxy, przechodzi przez firewalle
- Prosty, tekstowy format komunikatów zapisany w XML
- Obsługiwany przez różne języki programowania, systemy operacyjne, platformy sprzętowe

Uzupełnienie: XML – co to jest

- eXtensible Markup Language – rozszerzalny język znaczników
- Podobnie jak HTML
 - Znaczniki **<element> ... </element>**
 - Można definiować własne znaczniki
 - Idealny do opisu strukturalnych danych
 - Metajęzyk – język do tworzenia języków

Przykład dokumentu w XML

```
<?xml version="1.0"?>
<student>
  <imie>Adam</imie>
  <nazwisko>Kowalski</nazwisko>
</student>
<student>
  <imie>Barbara</imie>
  <nazwisko>Nowak</nazwisko>
</student>
```

SOAP

- Simple Object Access Protocol
- Protokół komunikacyjny
- Komunikacja między aplikacjami
- Format do przesyłania komunikatów przez Internet
- Niezależny od systemu i języka programowania
- Oparty na XML
- Standaryzowany przez W3C

Format komunikatu SOAP

1. **Envelope** (koperta, opakowanie) – mówi, że dokument XML jest komunikatem SOAP
2. **Header** (nagłówek) informacje pomocnicze – **opcjonalne**
3. **Body** (treść) – zawiera informacje o wywołaniu i wyniku
4. **Fault** – informacje o błędach - **opcjonalne**

Szkielet komunikatu SOAP

```
<?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:Header>
    ...
    ...
  </soap:Header>
  <soap:Body>
    ...
    ...
    <soap:Fault>
      ...
      ...
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Envelope

- **xmlns:soap=**
<http://www.w3.org/2001/12/soap-envelope>
– definiuje przestrzeń nazw dla elementów XML w SOAP
- **soap:encodingStyle=**
<http://www.w3.org/2001/12/soap-encoding>
- definiuje typy danych używane w dokumencie SOAP

Uzupełnienie: Przestrzenie nazw XML

- W jednym dokumencie może być wiele znaczników – np. w recenzji książki może być znacznik **<autor>** i nie wiadomo, czy jest to autor książki, czy autor recenzji
- Rozwiązanie:
`<książki:autor xmlns:książki=www.recenzje.pl/ksiazki>`
`</książki:autor>`
`<recenzje:autor xmlns:recenzje=www.recenzje.pl/recenzje>`
`</recenzje:autor>`
`<książki:autor> ... </książki:autor>`

Uzupełnienie - URI

- Uniform Resource Identifier – jednolity identyfikator zasobów
- np. <http://www.wszib.edu.pl/1K221/>
- Pod tym adresem sieciowym nie musi wcale znajdować się żaden dokument (ale może i zazwyczaj jest)

Nagłówek

```
<soap:Header>  
  <m:Trans  
    xmlns:m=http://www.test.com/transaction  
    soap:mustUnderstand="1">  
    234  
  </m:Trans>  
</soap:Header>
```

Dowolny element
z dowolnej
przestrzeni nazw

Czy element jest
obowiązkowy

Treść komunikatu - pytanie

```
<?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>
    <m:PodajCene xmlns:m=http://www.test.pl/ceny>
      <m:Towar>
        Jabłka
      </m:Towar>
    </m:PodajCene>
  </soap:Body>
</soap:Envelope>
```

Treść komunikatu - odpowiedź

```
<?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>
    <m:PodajCeneOdpowiedz xmlns:m=http://www.test.pl/ceny>
      <m:Cena>
        2.20
      </m:Cena>
    </m:PodajCeneOdpowiedz>
  </soap:Body>
</soap:Envelope>
```

SOAP nad HTTP

Zapytanie HTTP

```
POST /ceny HTTP/1.1
Host: www.test.pl
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

...Zapytanie (dokument SOAP) ...
```

Odpowiedź HTTP

```
HTTP/1.1 200 OK
Content-Type: application/soap; charset=utf-8
Content-Length: nnn

... Odpowiedź (dokument SOAP) ...
```

WSDL

- Problem:
 - Skąd wiemy, jaki komunikat SOAP wysłać?
 - Jakie są nazwy metod, jakie parametry, jaki format?
 - Jak automatycznie generować kod?
- Rozwiązanie:
 - Web Services Description Language

WSDL – co to jest

- Język opisu interfejsu usług sieciowych
- Oparty na XML
- Umożliwia generowanie kodu
- Zawiera wszystko, co potrzeba, by móc skorzystać z usługi

WSDL - elementy

- **Type** – definicja typów
- **Message** – definicja komunikatu zawierającego typy danych
- **Operation** – definicja operacji, czyli wymiany komunikatów
- **Port Type** – abstrakcyjny zestaw operacji
- **Binding** – wiązanie: konkretny protokół i format danych dla typu portu
- **Port** – punkt końcowy: kombinacja wiązania i adresu sieciowego
- **Service** – kolekcja połączonych portów

Type – definicja typów

- Kontener dla definicji typów danych
- Przykład: XSD – XML Schema Definition

```
<schema targetNamespace="http://www.test.pl/schematy"
xmlns="http://www.w3.org/2000/10/XMLSchema">
  <element name="CenaTowaru">
    <complexType>
      <all>
        <element name="Cena" type="float"/>
      </all>
    </complexType>
  </element>
</schema>
```

<Cena> 13.7 <Cena>

Message – definicja komunikatu

- Komunikat składa się z części (part)
- Każda z części jest określonego typu (zdefiniowanego wcześniej)
- Jest to definicja abstrakcyjna – faktyczny format komunikatu jest zależny od wiązania ← może być wiele wiązań

Przykład definicji komunikatu

```
<schema targetNamespace="http://www.test.pl/schematy"
xmlns="http://www.w3.org/2000/10/XMLSchema">
  <element name="CenaTowaru">
    <complexType>
      <all>
        <element name="Cena" type="float"/>
      </all>
    </complexType>
  </element>
</schema>
```

```
<message name="CenaOdpowiedź">
  <part name="ct" element="tns:CenaTowaru"/>
</message>
```

Typ Portu

- Zestaw operacji
- Rodzaje operacji:
 - Jednostronna (tylko odbiera komunikat)
 - Pytanie-odpowiedź (odbiera komunikat i wysyła odpowiedź)
 - Ogłoszenie-odpowiedź (wysyła komunikat i odbiera odpowiedź)
 - Powiadomienie (notyfikacja – tylko wysyła komunikat)
- 3 rodzaje komunikatów:
 - Input
 - Output
 - Fault

Przykład Typu Portu

```
<message name=„CenaPytanie">
    <part name=„ct" element="tns:NazwaTowaru"/>
</message>
<message name=„CenaOdpowiedź">
    <part name=„ct" element="tns:CenaTowaru"/>
</message>

<portType name=„CenaTowaruPortType">
    <operation name=„PodajCeneTowaru">
        <input message=„tns:CenaPytanie"/>
        <output message=„tns:CenaOdpowiedź"/>
    </operation>
</portType>
```

Wiązanie - binding

- Powiązanie typu port z konkretnym protokołem
- Jeden typ portu może mieć wiele wiązań: SOAP, HTTP GET/POST, MIME
- Każde wiązanie ma taką samą semantykę – klient analizując WSDL może wybrać odpowiadające mu wiązanie i dostanie ten sam wynik

Wiązanie SOAP

- 2 style – ważne przy używaniu w kodzie programu:
 - RPC – wymiana komunikatów jest traktowana jako wywołanie procedury (wychodzi z użycia)
 - Document – komunikat jest traktowany jako przekazywany dokument XML-owy
- Rodzaje kodowania
 - Encoded - Kodowanie SOAP - <http://schemas.xmlsoap.org/soap/encoding/> zawiera standard kodowania typów prostych i złożonych (struktury, tablice) – odwzorowanie na języki programowania
 - Literal („dosłowne”) – treść komunikatu jest traktowana jak dokument XML
 - Inne typy definiowane przez użytkownika

Przykład wiązania SOAP

```
<binding name=„CenaTowaruBinding" type="tns:CenaTowaruPortType">
  <soap:binding style=„rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name=„PodajCeneTowaru">
    <soap:operation
      soapAction="http://www.test.pl/PodajCeneTowaru"/>
    <input>
      <soap:body use="encoded"
        namespace="http://www.test.pl/towary"

        encodingStyle=
          "http://schemas.xmlsoap.org/soap/encoding"/>
    </input>
    <output>
      <soap:body use="encoded"
        namespace="http:// www.test.pl/towary "

        encodingStyle=
          "http://schemas.xmlsoap.org/soap/encoding"/>
    </output>
  </operation>
</binding>
```


Przykład Service

```
<service name="StockQuoteService">  
  <documentation>Moja pierwsza usługa</documentation>  
  <port name="CenaTowaruPort" binding="tns:CenaTowaruBinding">  
    <soap:address location="http://www.test.pl/towary"/>  
  </port>  
</service>
```

Historia: Jak to było w javie

- JAX-RPC – standard odwzorowujący typy danych w Javie na typy danych w SOAP
- Apache AXIS
 - implementacja środowiska do tworzenia usług sieciowych
 - Narzędzia Java2WSDL i WSDL2Java
 - **`http://www.test.pl/mojSerwis?WSDL`** zwraca dokument WSDL-owy z opisem serwisu

Historia: WSDL - Java

WSDL	Java
Type	Klasa
Port Type	Interfejs
Binding	Stub
Service	Interfejs serwisu, implementacja (locator) Skeleton (od strony serwera)

Historia: WSDL2Java - przykład

```
<message name="empty">
<message name="AddEntryRequest">
  <part name="name" type="xsd:string"/>
  <part name="address" type="typens:address"/>
</message>
<portType name="AddressBook">
  <operation name="addEntry">
    <input message="tns:AddEntryRequest"/>
    <output message="tns:empty"/>
  </operation>
</portType>
```

```
public interface AddressBook extends java.rmi.Remote {
    public void addEntry(String name, Address address)
        throws java.rmi.RemoteException;
}
```

Historia: WSDL2Java - klient

```
public class Tester {  
    public static void main(String [] args) throws Exception  
    {  
        // Make a service  
        AddressBookService service  
            = new AddressBookServiceLocator();  
        // Now use the service to get a stub which  
        //implements the SDI.  
        AddressBook port = service.getAddressBook();  
        // Make the actual call  
        Address address = new Address(...);  
        port.addEntry("Russell Butek", address);  
    }  
}
```

WSDL2Java - serwer

```
public class AddressBookSOAPBindingImpl implements
AddressBook {
    public void addEntry(String name, Address address)
throws java.rmi.RemoteException {    } }
```

JAX-WS 2.0: Teraz jest o wiele łatwiej

- Pełnoprawny serwer WS:

```
1 package example;
2 import javax.ws.WebMethod;
3 import javax.ws.WebService;
4 import javax.xml.ws.Endpoint;
5
6 @WebService
7 public class EchoWSServer {
8     @WebMethod
9     public String echo(String msg) {
10         System.out.println("Otrzymałem wiadomość: " + msg);
11         return msg;
12     }
13
14     public static void main(String[] args){
15         EchoWSServer calculator = new EchoWSServer();
16
17         Endpoint endpoint = Endpoint.publish(
18             "http://localhost:8080/echo", calculator);
19     }
20 }
```

Stworzenie klienta

- `wsimport -p client`
`http://localhost:8080/echo?wsdl`
- Tworzymy instancje wygenerowanego klienta
- Wywołujemy metody serwisu

JAX-WS 2.0: Klient (1)

```
13
14 /**
15  * This class was generated by the JAX-WS RI.
16  * JAX-WS RI 2.2.9-b130926.1035
17  * Generated source version: 2.2
18  *
19  */
20 @WebService(name = "EchoWSServer", targetNamespace = "http://example/")
21 @XmlSeeAlso({
22     ObjectFactory.class
23 })
24 public interface EchoWSServer {
25
26     @WebMethod
27     @WebResult(targetNamespace = "")
28     @RequestWrapper(localName = "echo",
29                     targetNamespace = "http://example/",
30                     className = "client.Echo")
31     @ResponseWrapper(localName = "echoResponse",
32                      targetNamespace = "http://example/",
33                      className = "client.EchoResponse")
34     @Action(input = "http://example/EchoWSServer/echoRequest",
35            output = "http://example/EchoWSServer/echoResponse")
36     public String echo(
37         @WebParam(name = "arg0", targetNamespace = "")
38         String arg0);
39
40 }
```

JAX-WS 2.0: Klient (2)

- Stwórz instancje serwisu
- Pobierz port
- Wywołaj metody

```
3 public class Client {  
4     public static void main(String[] args) {  
5         EchoWSServer server = new EchoWSServerService().getEchoWSServerPort();  
6         System.out.println(server.echo("testing123"));  
7     }  
8 }
```

Co to za cudak z @?

- @WebService
- @WebMethod
- @WebParam
- @WebResult
- @OneWay

- @Override

@WebService

- Marks a Java class as implementing a Web Service, or a Java interface as defining a Web Service interface
- Parametry:
 - endpointInterface
 - name
 - portName
 - serviceName
 - targetNamespace
 - wsdlLocation

@WebMethod

- Customizes a method that is exposed as a Web Service operation.
- Parametry
 - action
 - exclude
 - operationName

@WebParam

- Customizes the mapping of an individual parameter to a Web Service message part and XML element.
- Parametry:
 - header
 - mode
 - name
 - partName
 - targetNamespace

@WebResult

- Customizes the mapping of the return value to a WSDL part and XML element.
- Parametry:
 - header
 - name
 - partName
 - targetNamespace

@OneWay

- Indicated that the given @WebMethod has only an input message and no output.

SOAP w Perlu

```
use SOAP::Lite;  
print SOAP::Lite  
-> service  
( 'http://services.xmethods.net/soap/urn:xmethods-  
    delayed-quotes.wsdl' )  
-> getQuote( 'MSFT' );
```

Możliwość integracji

- WS może zostać stworzony w dowolnym języku
- WS może być wywołany w dowolnym języku
- Każdy ważniejszy język programowania posiada bibliotekę do WS

Inne platformy

- Perl: SOAP::Lite
<http://www.soaplite.com/>
- C/C++: gSOAP
<http://gsoap2.sourceforge.net/>
- Windows: MS .NET
- IBM WebSphere, WSDTK
- CXF
- Spring

Standardy, standardy

- SOAP
- WSDL
- WS-Notification
- WS-Addressing
- WS-Transfer
- WS-Eventing
- WS-Policy
- WS-Inspection
- WS-Security
- WS-Trust
- ...

Kolejny problem – jak znaleźć usługi sieciowe?

- Problem marketingowy i techniczny
- Przykład:
 - jak znaleźć wszystkie firmy sprzedające komputery przez Internet?
 - Skąd wziąć definicje interfejsów tych usług (WSDL)?
- Rozwiązania:
 - WWW: np. programmableweb.com, service-repository.com, Google
 - Specjalizowane rejestry: UDDI
 - Inne standardy: WS-Inspection

UDDI

- Universal Description, Discovery and Integration
- Katalog do przechowywania informacji o usługach sieciowych
- Przechowuje informacje o interfejsach opisane przy pomocy WSDL
- Jest dostępne przez interfejs SOAP

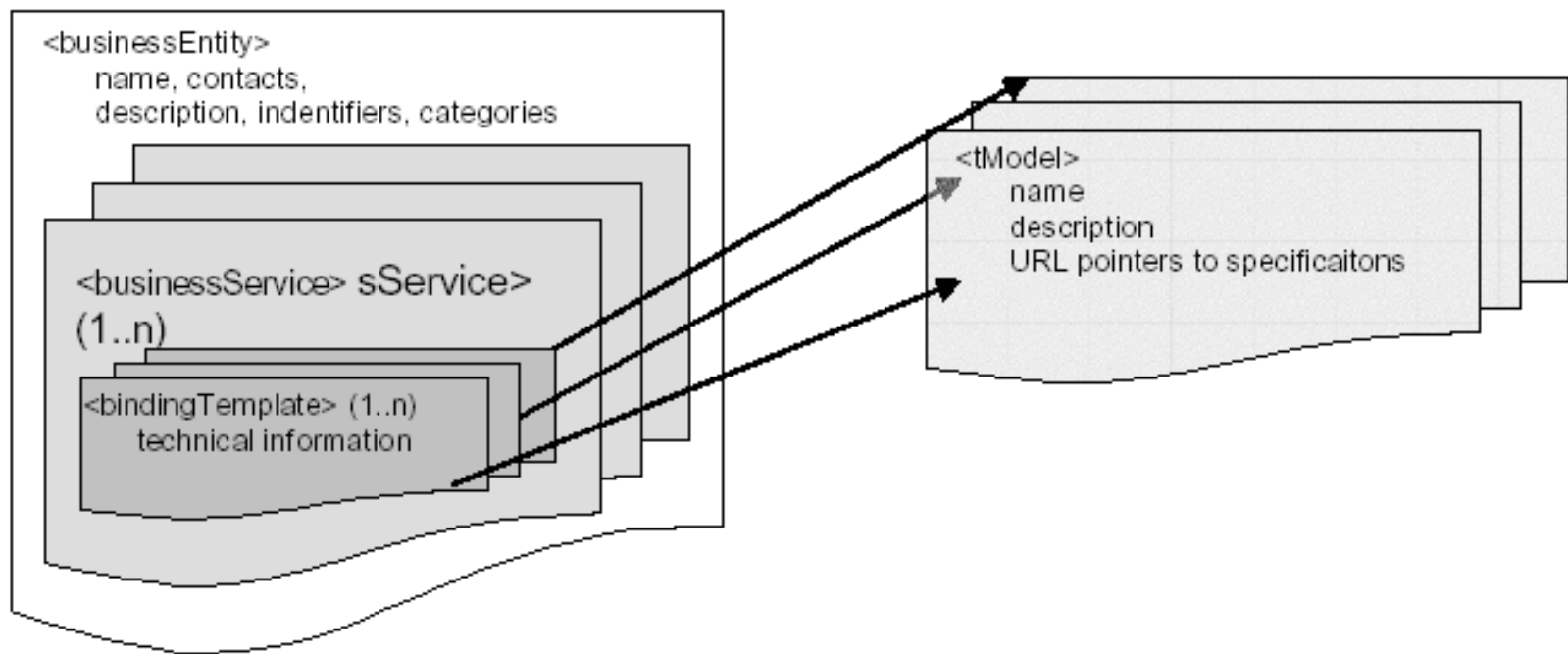
UDDI – podobne do wyszukiwarek internetowych, ale:

- Przechowuje nie tylko adresy stron WWW
- O treści informacji o usłudze decyduje sam dostawca usługi
- Zawartość rejestru można modyfikować przy pomocy przeglądarki lub specjalnego programu klienta
- Zapewnia standardowy format opisu usług

Jaka informacja o firmie znajduje się w UDDI

- „Białe strony” – „white pages”
informacje podstawowe (adresy, telefony itp.)
- „Żółte strony” – zakres działalności według określonych klasyfikacji
- „Zielone strony” – informacje techniczne, jak korzystać z usług sieciowych

Informacje biznesowe – struktury danych w UDDI



tModel

- Abstrakcyjny opis specyfikacji lub zachowania, z którymi usługa sieciowa jest zgodna
- Zawiera odsyłacz do właściwej specyfikacji
- Zawiera tModelKey – będący UUID
- Digital fingerprint – odcisk palca, klucz inentyfikacyjny

UUID

- Universally Unique Identifier
- Uniwersalnie jednoznaczny identyfikator
- Format:
 - Zdefiniowany w standardzie ISO/IEC 11578:1996
 - Zapisany w systemie szesnastkowym
 - Utworzony z kombinacji aktualnego czasu, adresu fizycznego, adresu IP i liczby losowej

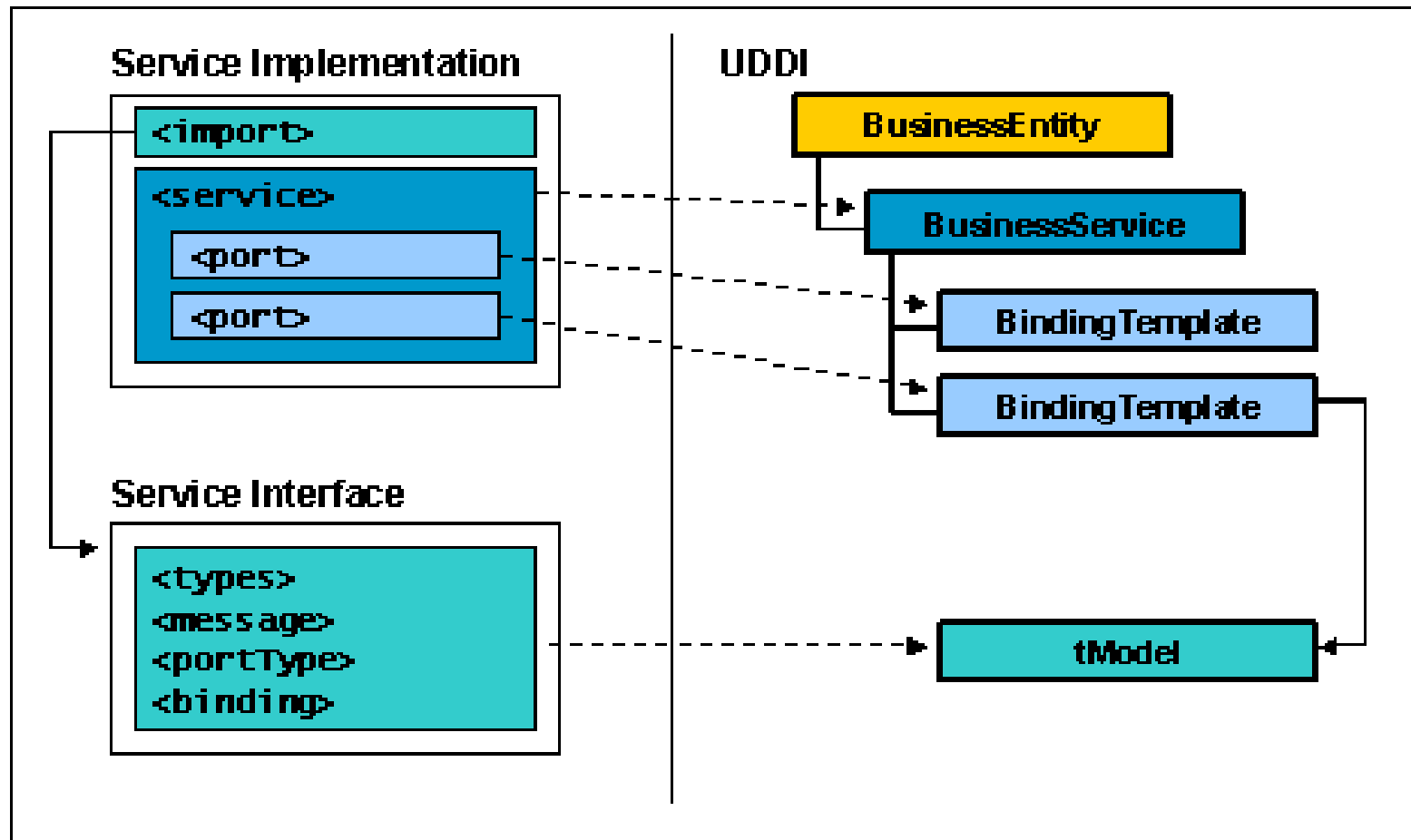
Klasyfikacje, taksonomie

- Ustanowione przez organizacje rządowe lub niezależne systemy klasyfikacji towarów, usług
- Przykłady:
 - North American Industry Classification System (NAICS)
www.census.gov/epcd/www/naics.html
(produkcja komputerów ma kod 3341)
 - Universal Standard Products and Services Classification(UNSPSC)- www.unspsc.org
 - International Organization for Standardization (ISO)
www.din.de/gremien/nas/nabd/iso3166ma
(regiony geograficzne, kody krajów, itd..)

Operacje na UDDI

- Zapytania
 - Szukanie
 - Informacje szczegółowe
- Wprowadzanie danych
- Replikacja
- Wszystkie posiadają interfejs SOAP
- Mogą być przeglądane przy pomocy przeglądarki WWW

WSDL a UDDI



Połączenie WSDL i UDDI umożliwia

- Rejestrację i wyszukiwanie usług według typów
- Korzystanie z usług różnych dostawców przy pomocy tego samego interfejsu programistycznego

Przykład wykorzystania

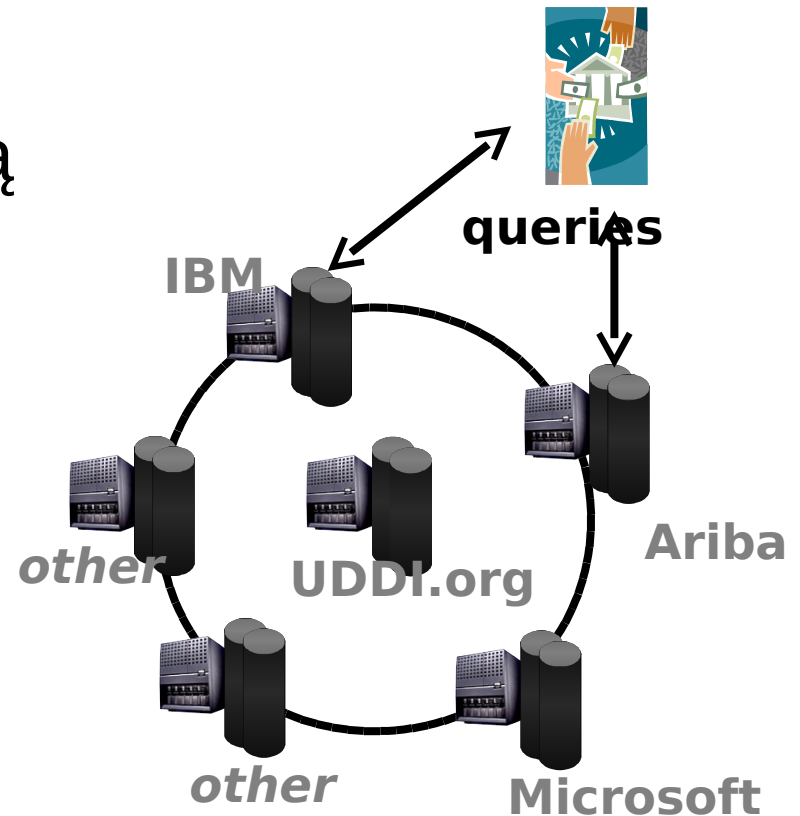
- Jeżeli mamy standardowy interfejs do systemów wyszukiwania i rezerwacji biletów lotniczych
- Linie lotnicze rejestrują w UDDI swoje serwisy
- Agenci wyszukują w UDDI serwisy różnych linii
- Mogą korzystać z nich ze zintegrowanego systemu, ponieważ interfejs jest uniwersalny

UDDI.xml.org

- Organizacja, która utrzymuje globalny system UDDI
- Implementacja standardu UDDI
- Wybrani członkowie: *Cisco Systems, Fujitsu, IBM, Intel, Microsoft Corporation, NEC Corporation, Oracle, SAP, Sun Microsystems*
 - To musi być ważne
- (Przyszły) model finansowania serwisu

Architektura

- Wiele węzłów
- Każdy węzeł ma swoją autoryzację rejestracji
- Replikacja danych (w nocy)
- Trochę na wzór DNS



Marzenia...

- Wywoływanie jednego Web-Serwisu poprzez drugi
- Kompozycja całych aplikacji z Web-Serwisów – Work-Flow
- Opis semantyczny
- Automatyczna budowa work-flow'ów
- Web-Serwisowe środowisko Gridowe

WS-Inspection (WSIL)

- XML-owy format do zbierania odsyłaczy (referencji) do istniejących opisów usług (WSDL, UDDI)
- Konwencje dotyczące umieszczenia dokumentów WS-Inspection
 - www.myservices.com/inspection.wsil



Przykład WSIL

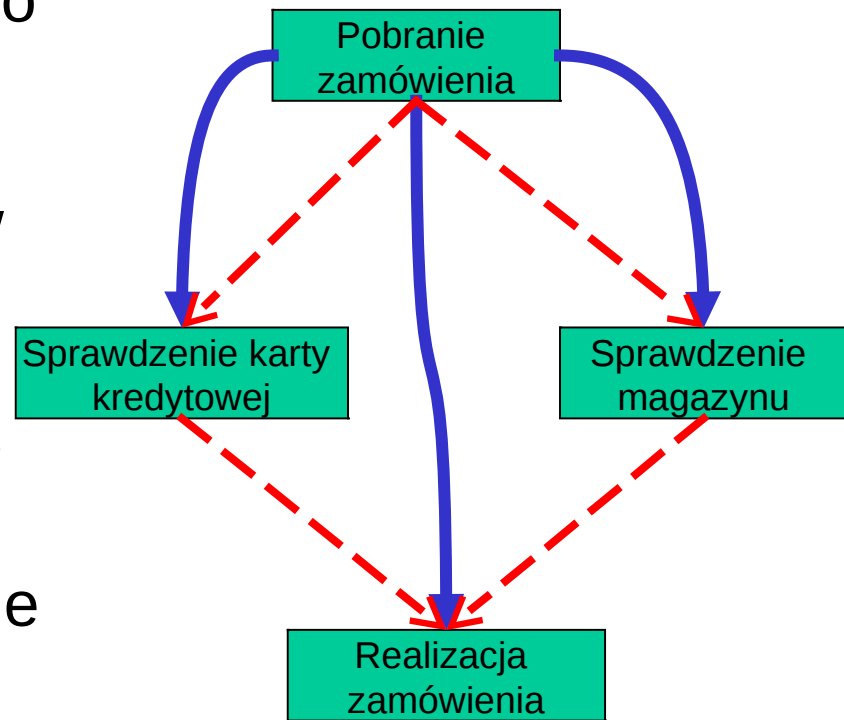
```
<?xml version="1.0"?>
<inspection xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/">
<service>
  <description referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
    location="http://example.com/exampleservice.wsdl" />
</service>
<service>
  <description referencedNamespace="urn:uddi-org:api">
    <wsiluddi:serviceDescription location= "http://example.com/uddi/inquiryapi">
      <wsiluddi:serviceKey> 52946BB0-BC28-11D5-A432-0004AC49CC1E</wsiluddi:serviceKey>
    </wsiluddi:serviceDescription>
  </description>
</service>
<link referencedNamespace= "http://schemas.xmlsoap.org/ws/2001/10/inspection/"
  location="http://example.com/tools/toolservices.wsil"/>
</inspection>
```

Modelowanie przepływu w usługach sieciowych

- Łączenie usług sieciowych w złożone procesy
- Języki
 - WSFL – Web Services Flow Language (IBM)
 - XLANG (Microsoft)
 - BPEL4WS - Business Process Execution Language for Web Services (IBM, Microsoft)

Web Services Flow Language

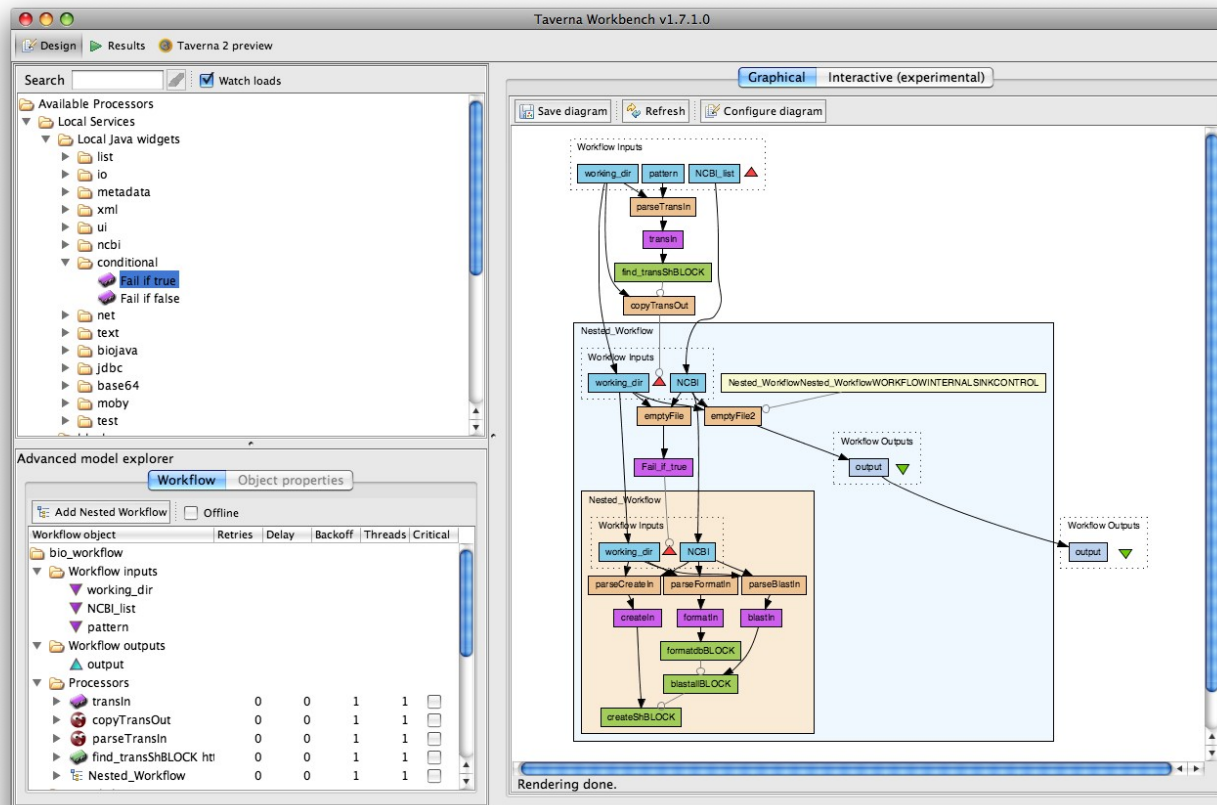
- Opis procesu złożonego z mniejszych elementów
- Łączenie wielu usług w jedną
- Przepływ
 - sterowania 
 - danych 
- Flow Engine – wykonuje operacje według opisu WSFL



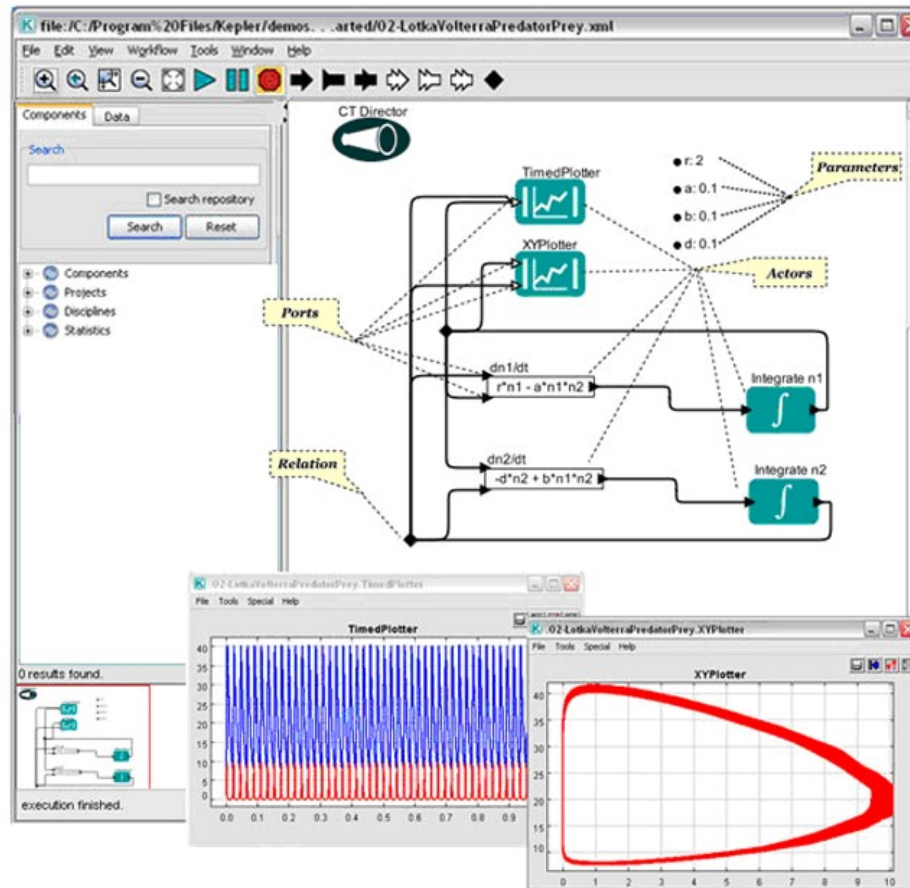
WSIF Web Services Invocation Framework

- API w Javie do korzystania z usług bezpośrednio na poziomie WSDL
- Wystarczy napisać program korzystając z abstrakcyjnej definicji PortType
- Środowisko zapewnia dynamiczne wykorzystanie wiązań (binding) dla różnych standardów (SOAP, EJB, Java, JMS)

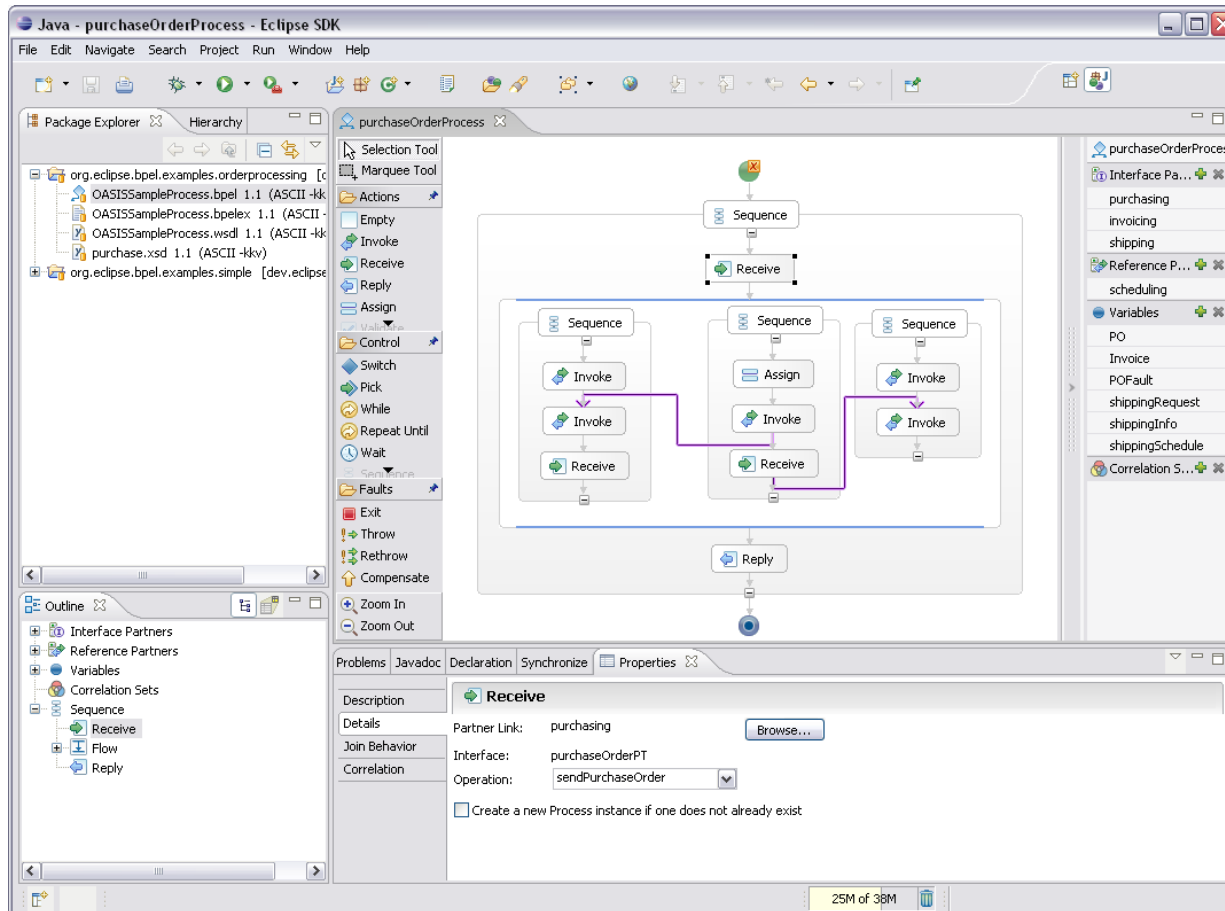
Kompozycja Web Serwisow - Taverna



Kompozycja WS - Kepler



Kompozycja WS - bpel



Podsumowanie

- SOA – Architektura serwisowa
- SOAP – Komunikacja wszystko z wszystkim
- WSDL – Wpis co serwis robi
- UDDI – Rejestr i organizacja wielkich