



Spis treści:

- 1 Usługi Web (*web services*) - kontekst
- 2 Tworzenie i korzystanie z usług Web
- 3 Protokół SOAP
- 4 Opis usług WSDL
- 5 Odkrywanie Web Services: UDDI
6. Bezpieczeństwo usług Web

4. Usługi Web

Maciej Piechówka
macpi@eti.pg.gda.pl
2009

W3C.org, WS-I.ORG

← - - - → **WCF** (Windows Communication Foundation)
REST (Representational State Transfer)

www.xmethods.com - zbiór usług WWW, testowanie
<http://terraservice.net/>
<http://www.google.com/apis/index.html> - API Google Web Services
<http://www.websvcex.net/WCF/default.aspx>
<http://www.cdyne.com/>
<http://aws.amazon.com/> - Amazon Web Services
...

Serwis turystyczny ABC:

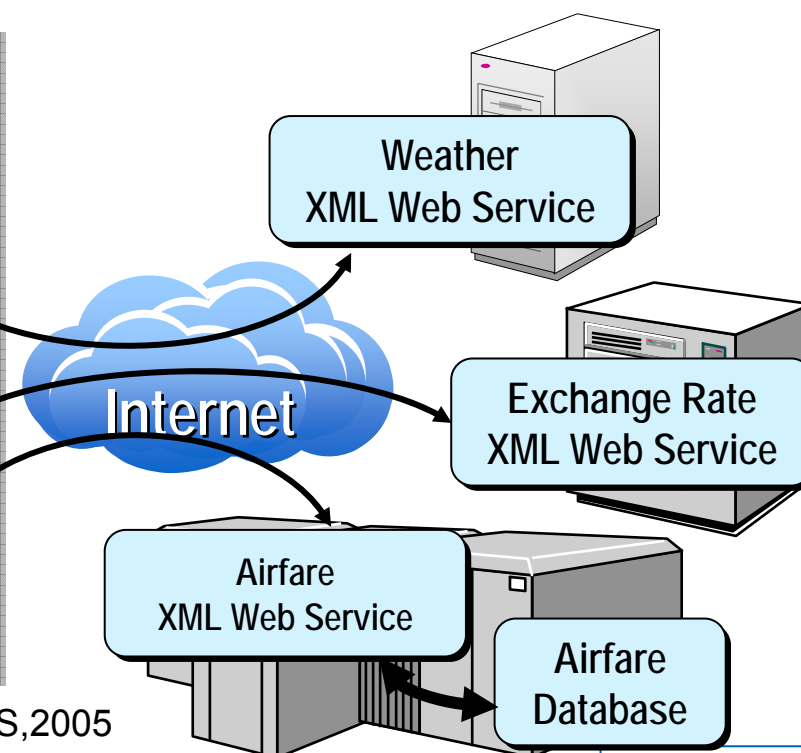
Wybierz miejscowość:

Prognoza pogody dla:



Kurs wymiany to:

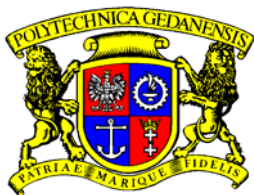
Możesz tam lecieć za jedyne:



L.F. Cabrera, Ch. Kurt *Architektura Usług WEB i jej specyfikacje*, MS, 2005

4.1 Fyżlewicz Z., Salamon, *Podstawy architektury i technologii usług XML sieci Web*, Mikom, 2008

08 M. Piechówka



SOA: Usługa

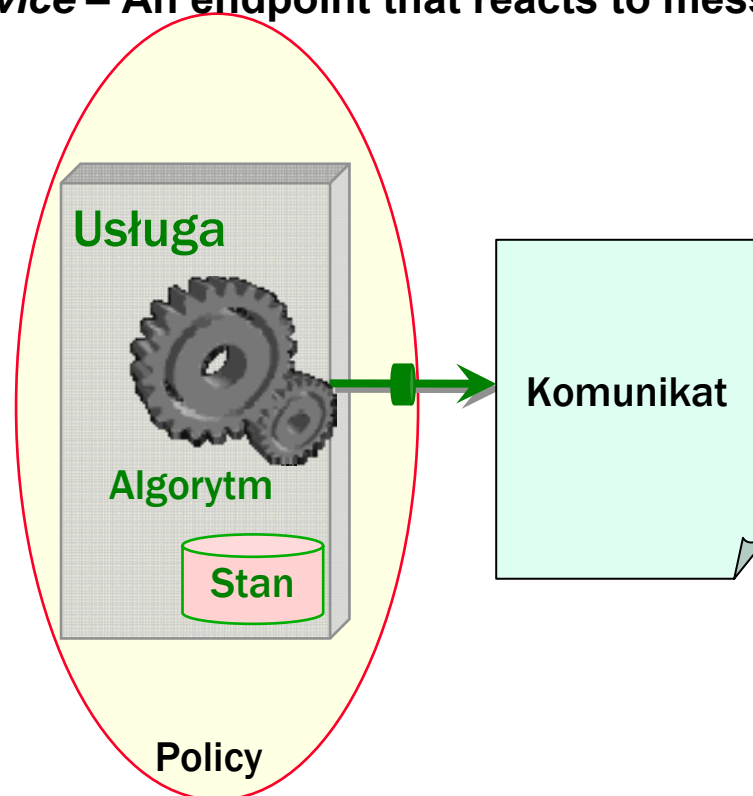
Service – An endpoint that reacts to messages

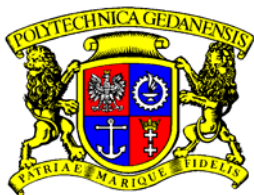
- **Jednostka funkcjonalności**

- Zdefiniowana poprzez **interfejs**, definiujący sposób dostępu do usługi
- Otrzymuje i wysyła **komunikaty** zgodnie z **kontraktem**

Główne zasady architektury usług Web:

- **Zorientowanie na komunikaty**
- **Modułowość protokołów**
 - stosowanie bloków składających się na protokoły infrastrukturalne, które następnie mogą być użyte w prawie dowolnej kombinacji.
- **Autonomiczne usługi**
 - Zezwalanie na niezależne konstruowanie, rozwijanie, zarządzanie, opracowywanie wersji i zabezpieczanie punktów końcowych.
- **Zarządzana przezroczystość**
 - Kontrolowanie, które aspekty **punktu końcowego** są (lub nie są) widziane przez usługi zewnętrzne.



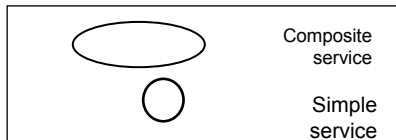
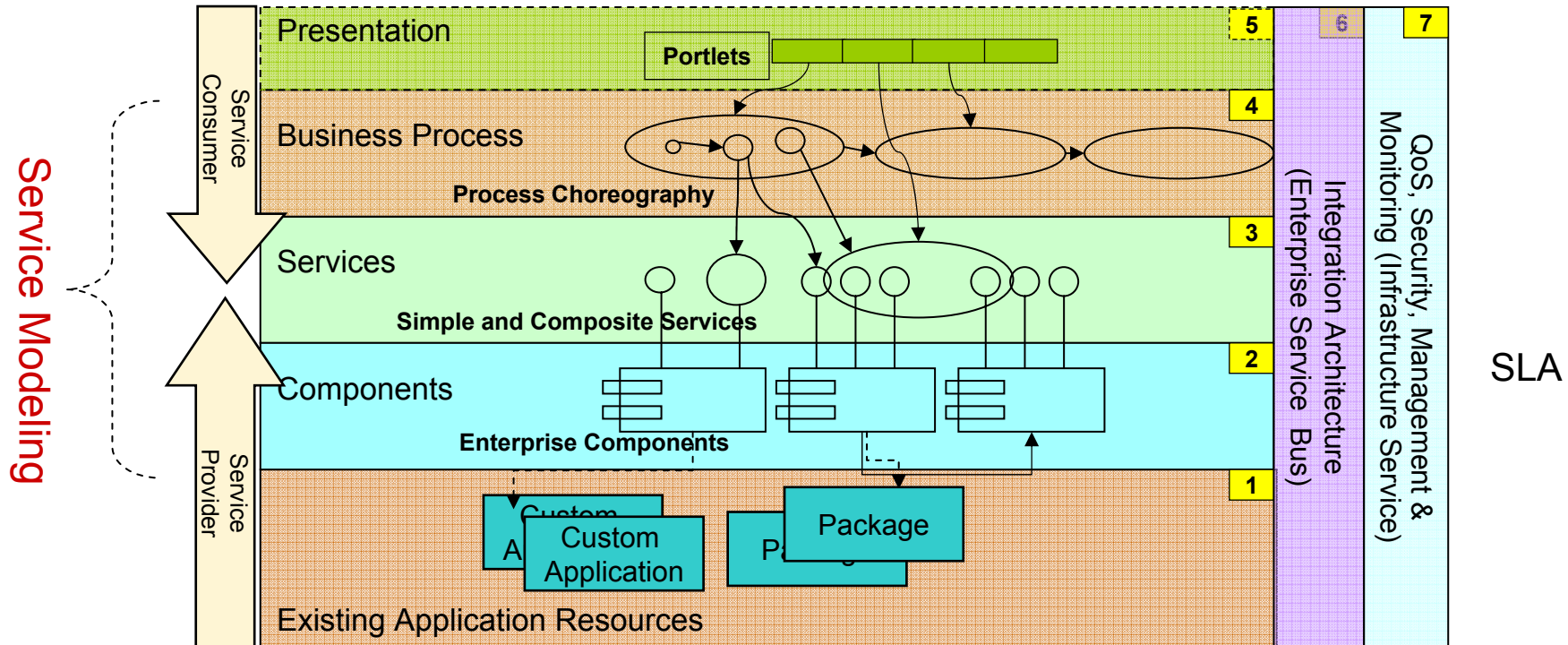


SOA – architektura zorientowana na usługi





Warstwy SOA



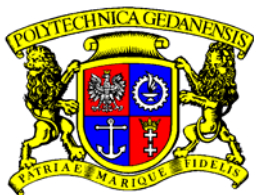
IBM SOA



4.1 Usługi WWW (*web services*) (1)

SOA - Service Oriented Architecture

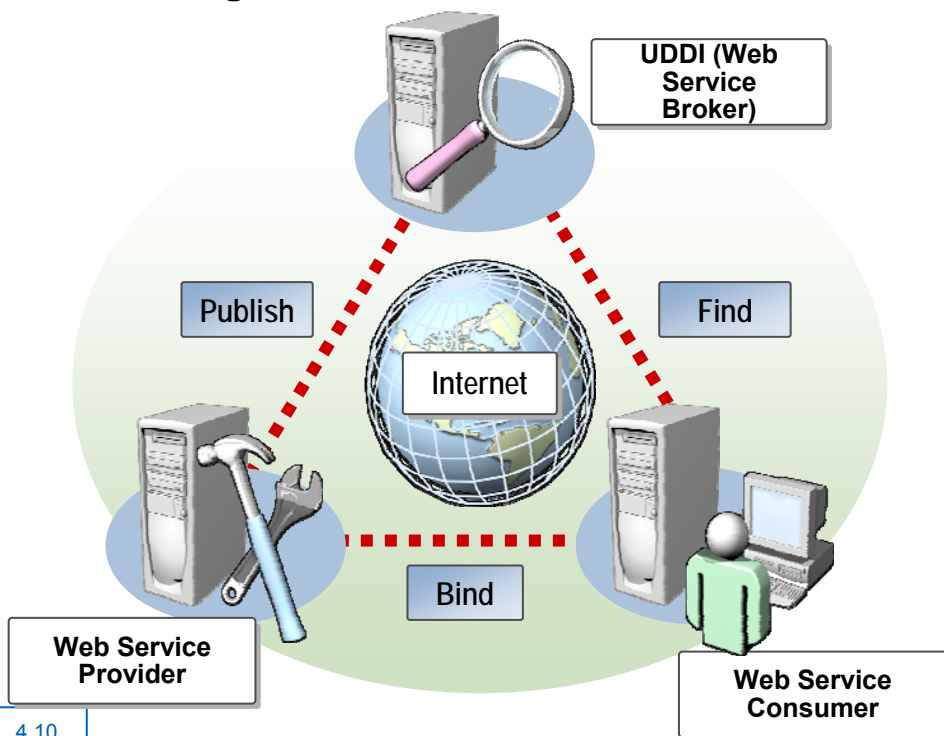
- **Jednostka (komponent)** udostępniona (**adresowalna**) przez standardowe protokoły internetowe... która realizuje jakiś wartościowy proces biznesowy. Dostępna globalnie poprzez użycie protokołu HTTP i formatu danych opartych na XML
- Przykłady usług WWW:
 - Wyliczenie kosztu przesyłki kurierskiej, zlecenie odebrania przesyłki, udzielenie pożyczki, prognoza pogody, kursy akcji, oferty sklepów internetowych, przechowywanie i dostęp do współdzielonych informacji o użytkowniku, ...
- Łatwa do odkrycia i zrozumienia dzięki standardowi **UDDI (Universal Description, Discovery and Integration)**, który jest specyfikacją interfejsu odpowiedzialnego za katalogowanie usług. Umożliwia przeszukiwanie rejestru usług, ich analizę i zarządzanie wpisami.
- **Kontrakt usługi** opisuje dostępne usługi jako wiadomości, które dostawca Web Service akceptuje i generuje
 - **WSDL (Web Services Definition Language)** służy do **opisu usług**, definiuje oferowane interfejsy i fizyczną lokalizację punktów końcowych
- **Komunikacja** – użycie standardowych protokołów i formatów danych
 - HTTP (GET i POST), XML, SOAP (Simple Object Access Protocol),
 - Dowolny system, który rozumie te protokoły będzie może współpracować z serwisem



Usługi WWW (2)

UDDI- Universal Description, Discovery and Integration

- Definiuje sposób publikowania i odkrywania Web Services.
- Bazuje na usługach katalogowych lub rejestrach biznesowych
- Wyszukiwanie według biznesu, rodzaju usługi

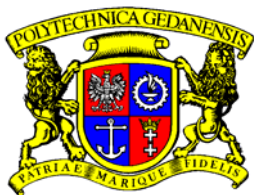


Oferowanie usługi

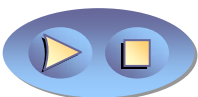
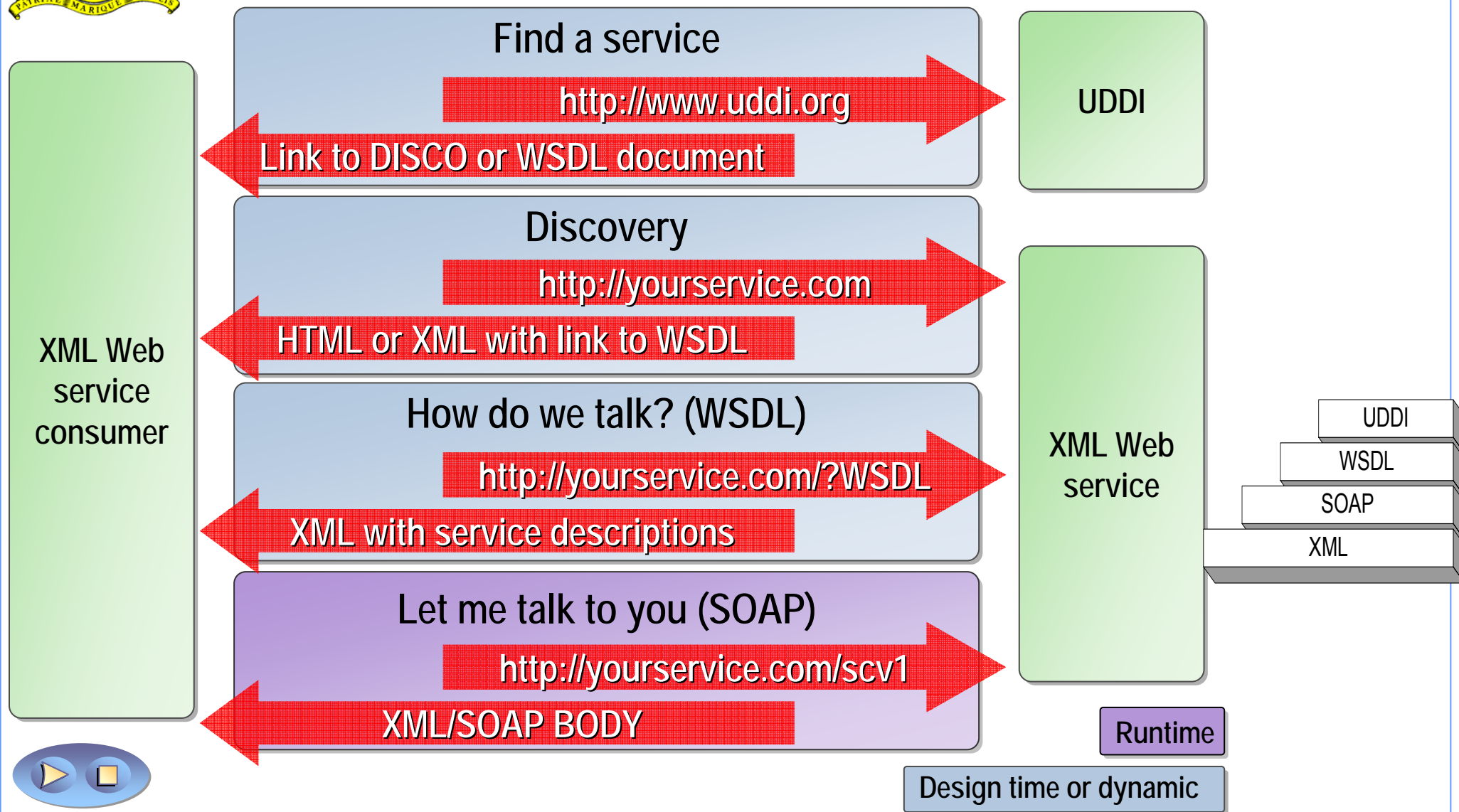
- Opisz usługi w WSDL
- Utwórz usługę Web Service
- Zarejestruj w katalogu UDDI
- Inni mogą:
 - Znaleźć Wasz serwis, Zrozumieć jak korzystać z tego serwisu, ...

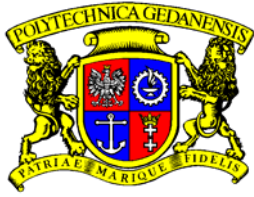
Korzystanie z usług

- Klient korzystający z usługi musi wiedzieć
 - Jakie są dostępne metody, jakie są argumenty, jakie są zwracane typy wartości
- Informacje te można uzyskać za pomocą usługi zapytania Web Service
 - Odpowiedź zawiera opis usługi Web Service
- Aby skorzystać z usługi należy:
 - Wysłać do usługi Web Service komunikat (synchronicznie lub asynchronicznie) i odebrać odpowiedź

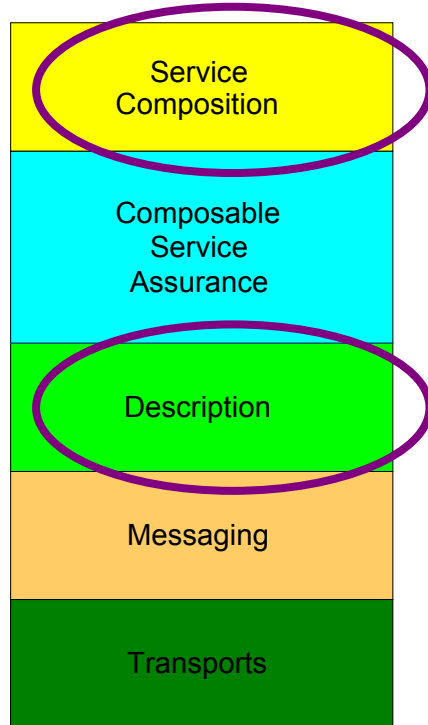


Infrastruktura Web Service – stos protokołów

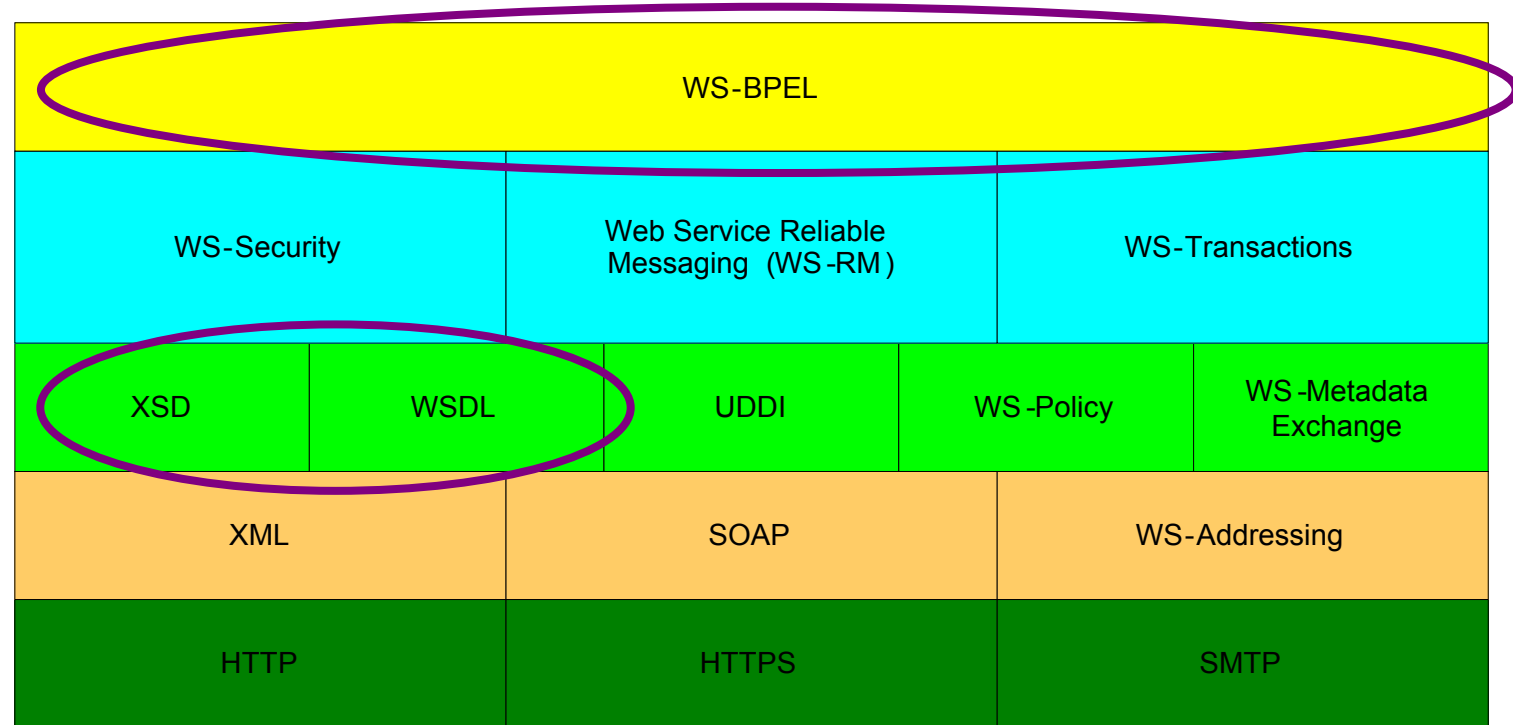




Web services stack



Conceptual stack



Technology stack

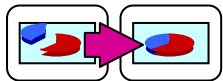
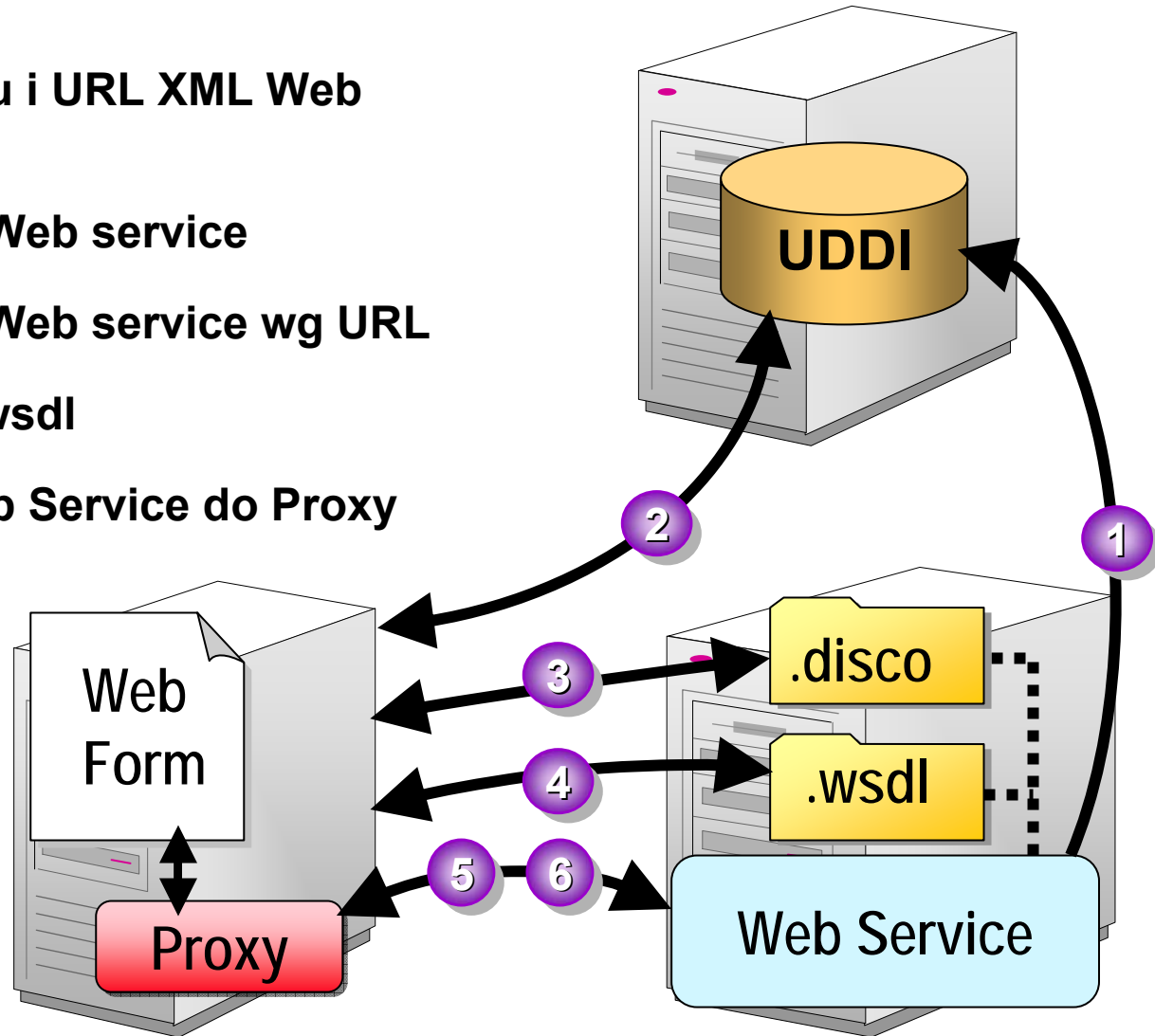
WS-* specifications

Organization for the Advancement of Structured Information Standards (OASIS)



Wyszukiwanie konkretnej usługi

- 1 Publikacja opisu i URL XML Web service
- 2 Wyszukaj XML Web service
- 3 Zlokalizuj XML Web service wg URL
- 4 Odczytaj opis .wsdl
- 5 Połącz XML Web Service do Proxy
- 6 Wywołaj XML Web Service z formularza Web Form przez Proxy

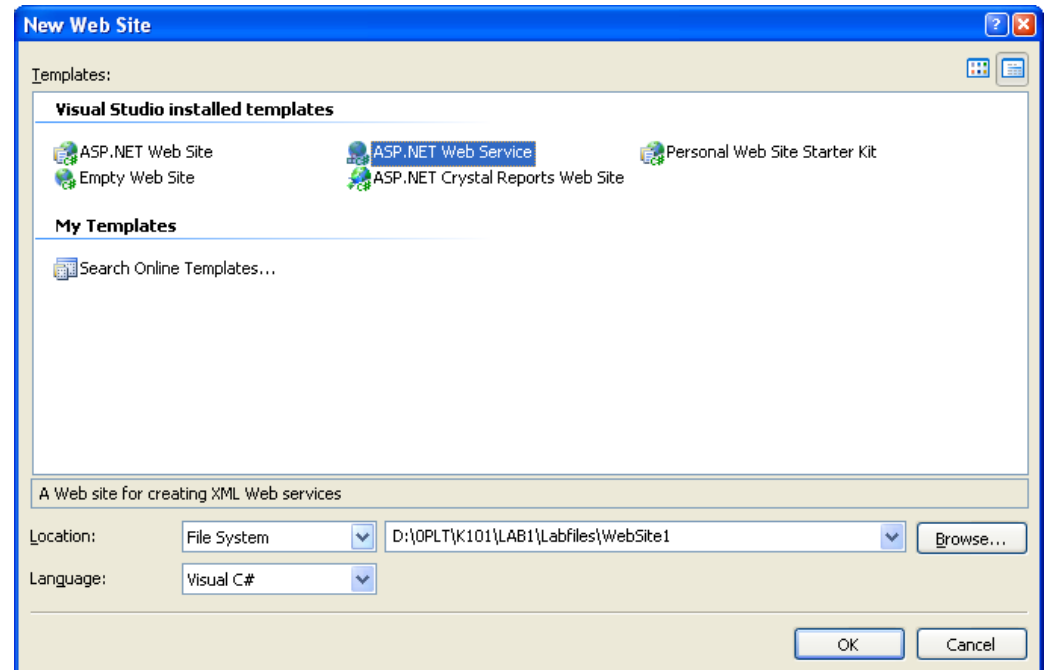


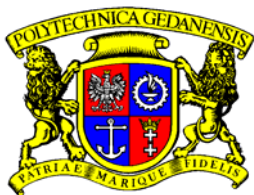


4.2 Tworzenie i korzystanie z Web Services

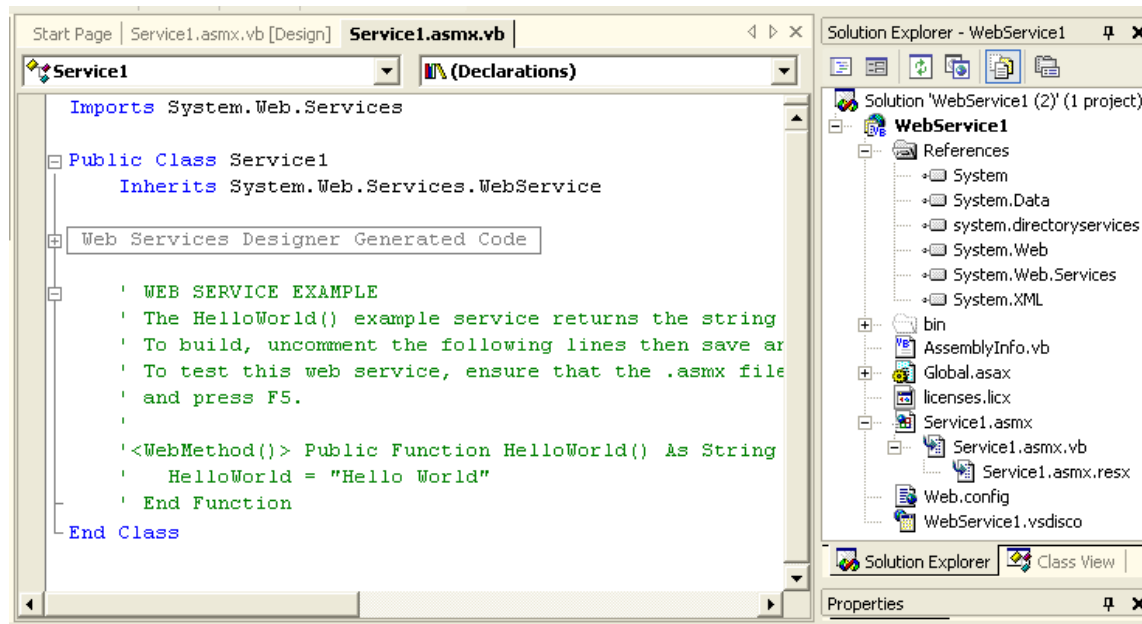
- IIS and ASP.NET wspierają usługi WWW
- Utwórz wirtualny katalog w IIS
- Pliki projektu typu ASP.NET Web Service zostaną utworzone w odpowiadającym katalogu fizycznym

- Utwórz nowy projekt typu XML Web Service przy pomocy Visual Studio .NET
- Zadeklaruj funkcje typu WebMethod
- Zbuduj XML Web Service projekt
- Przetestuj w przeglądarce





Tworzenie usługi XML Web Service



XML Web Service Code

•.asmx page

```
<%@ WebService Language="c#"
    Codebehind="Service1.asmx.cs"
    Class="XMLWebServiceName.Service1" %>
```

•.asmx.cs page

```
using System;
using System.Web.Services;
class Service1
{
    [WebMethod]
    public type function1()
    {
        //function_here
    }
}
```



WebService and WebMethod Attributes

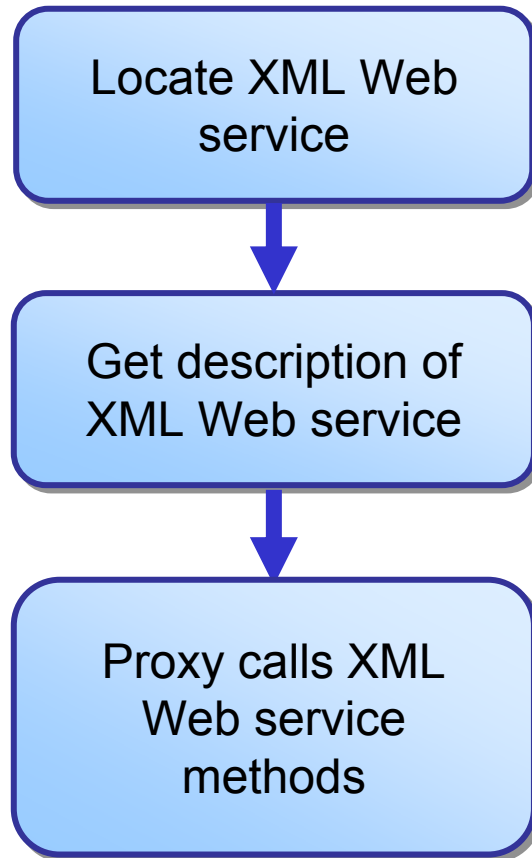
- **The WebService Attribute** - the optional class attribute is applied to the class implementing the XML Web Service.
 - *Description*, which describes the overall Web Service, and *Namespace*, which sets the default XML namespace for the service to avoid naming conflicts among XML elements and attributes.

```
[WebService(Namespace="http://www.dateservices.com/ws/",  
    Description="<b>Web Service that Provides Date Functions.</b>")]  
public class Birthday : System.Web.Services.WebService
```

- **The WebMethod attribute** is required to expose a method as part of a Web Service. Its properties include:
 - *BufferResponse* – whether the response for this request is buffered
 - *CacheDuration* – the number of seconds the response should be held in the cache
 - *Description* – a descriptive message
 - *EnableSession* – whether session state is enabled for an XML Web service method
 - *MessageName* – can be used to alias method or property names (e.g. to uniquely identify polymorphic methods). The default is the name of the XML Web service method
 - *TransactionOption* – indicates the transaction support



Korzystanie z usług WWW – mechanizm Proxy



Static or dynamic

Methods, arguments, return values

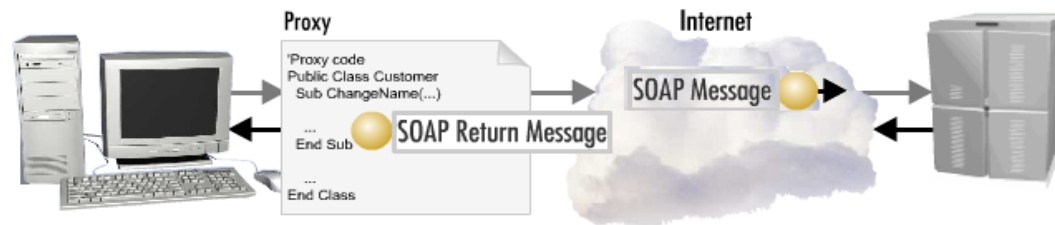
Proxy transparently handles network communication

Discovery document

```
<?xml version="1.0"?>
<discovery ...
  <contactRef ref=
    "http://.../Person.asmx?wsdl"
    docRef="..."?/>
</discovery>
```

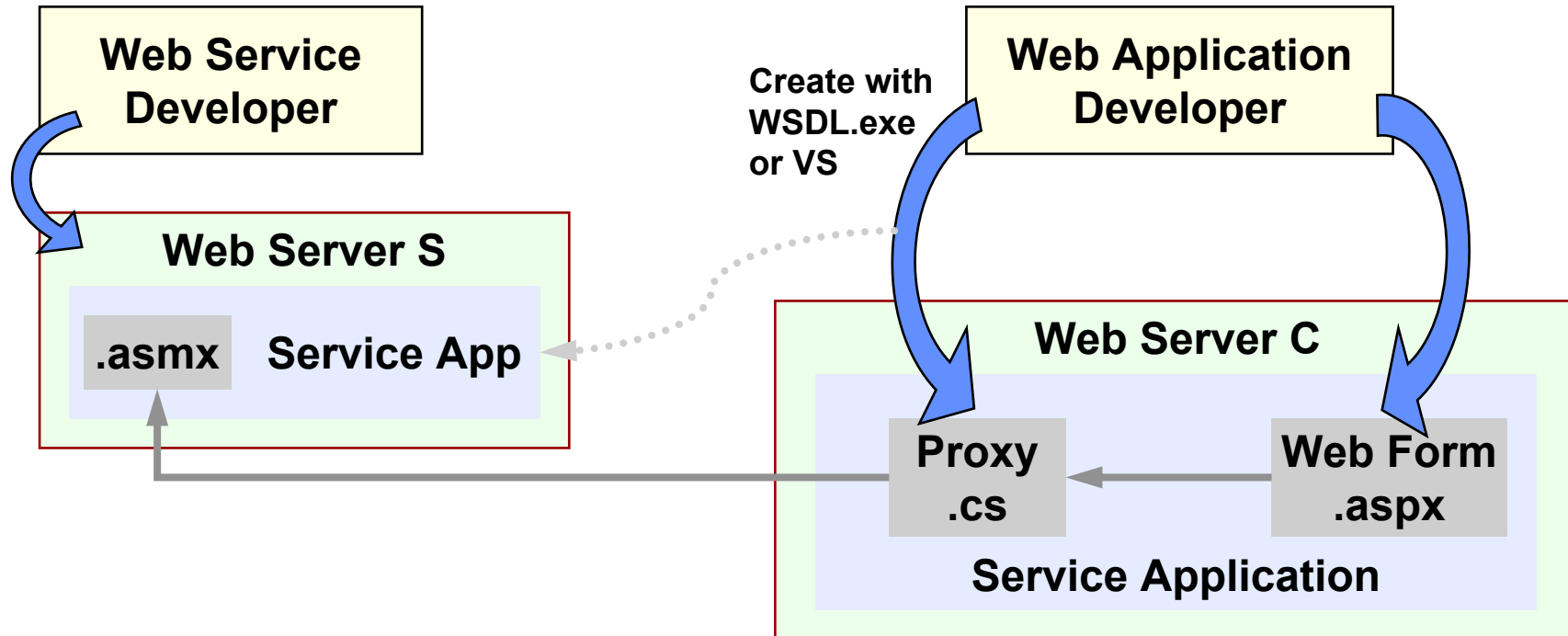
WSDL

```
<?xml version="1.0"?>
<serviceDescription ...>
  <soap ...>...</soap>
  <httppost ...>...</httppost>
  <httpget ...>...</httpget>
  <xsd:schema ...>...</xsd:schema>
</serviceDescription>
```





Consuming Web Services





Consuming Web Services Invoking via HTTP-GET, HTTP-POST

- **HTTP-GET**

```
http://localhost/MathService.asmx/Multiply?a=11&b=11
```

– Result is an XML document

```
<?xml version="1.0" encoding="utf-8" ?>  
<int xmlns="http://www.microsoft.com/MathService/">121</int>
```

- **HTTP-POST**

```
POST /MathService.asmx/Multiply HTTP/1.1  
Host: localhost  
Content-Type: application/x-www-form-urlencoded  
Content-Length: length
```

```
a=11&b=11
```

– Result is an XML document

```
<?xml version="1.0" encoding="utf-8" ?>  
<int xmlns="http://www.microsoft.com/MathService/">121</int>
```



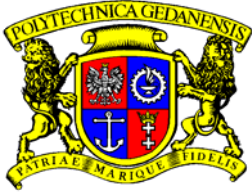
Korzystanie z XML Web Service - mechanizm Proxy (2)

1. Utwórz **Web reference** dla XML Web Service
2. Utwórz instancję XML Web Service
3. Wywołaj metodę XML Web Service
4. Utwórz (Build) aplikację ASP.NET Web Application

```
private void Button1_Click(object sender, EventArgs e)
{
    GetStocks.Local host. service1 ProxyGetStocks = new GetStocks.Local host. Servi ce1();
    lblResults.Text = ProxyGetStocks.GetRating("Contoso");
}
```

XML Web Service Error Handling

```
GetStocks.StockWebRef.Service1 ProxyGetStocks = new GetStocks.StockWebRef.Service1();
ProxyGetStocks.Timeout = 10000;
try
{
    lblMessage.Text = ProxyGetStocks.GetRating(TextBox1.Text);
}
catch (Exception err)
{
    lblMessage.Text = err.Message;
}
```



Klasa pośrednicząca usługi Web

- Ma taką samą nazwę co klasa reprezentowanej przez nią usługi Web oraz zawiera metody o takich samych nazwach jak metody usługi Web
- Kod w klasie pośredniczącej służy do przekazywania wywołań i wyników; obliczenia są realizowane przez usługę Web
- Kod klasy proxy dziedziczy metody klasy ***System.Web.Services.Protocols.SoapHttpClientProtocol***

Wybrane składowe *System.Web.Services.Protocols.SoapHttpClientProtocol*

Member	Description
BeginInvoke()	Begins an asynchronous invocation of the Web method.
EndInvoke()	Ends an asynchronous invocation of the Web method.
Invoke()	Invokes the Web method synchronously.
CookieContainer	Gets or sets the collection of cookies. This permits a proxy-based client to accept cookies and allow a server to maintain state information.
Proxy	Gets or sets proxy information needed to make a Web Service call through a firewall.
Timeout	Gets or sets the timeout (in milliseconds) a client waits for a synchronous call to a Web Service to be completed.
Url	Gets or sets the URL used to access the Web server.



Synchroniczne/asynchroniczne wywoływania metod usług Web

- Synchroniczne operacje używają tylko jednego wątku, który zostaje zablokowany do czasu zwrócenia wyniku przez metodę. W kodzie klienta należy określić limit czasu oczekiwania na odpowiedź i obsługę wyjątków.
- Wywołania asynchroniczne tworzą nowy wątek roboczy w którym wykonywane są operacje wywoływanej metody, a sterowanie wraca do wątku głównego. Procedura wywołująca otrzymuje informacje w momencie zakończenia uruchomionego zadania.
- Asynchroniczna komunikacja odbywa się za pomocą udostępnionych przez pośrednika metod **Beginnazwametody()** i **Endnazwametody()**. Metody te wywołują *BeginInvoke()* oraz *EndInvoke()*
- **BeginInvoke()** – umieszcza żadaną metodę w kolejce, przygotowując do uruchomienia w odrębnym wątku. Zwraca ona obiekt typu [AsyncResult], który jest przekazywany do metody **EndInvoke()** w celu pobrania wyników.



4.3 SOAP = HTTP+XML

Simple Object Access Protocol

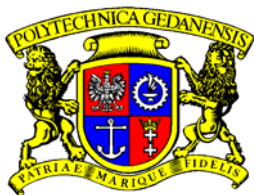
- Protokół do wymiany informacji w rozproszonym środowisku; przekazuje komunikaty żądanie/odpowieź
 - Wykorzystuje zwykle HTTP
 - Jest niezależny od platformy
- Używa gramatyki XML do:
 - Określania nazw metod
 - Definiowania typów parametrów i zwracanych wartości
 - Opisu typów
- Określa sposób wysyłania/odbierania komunikatów/odpowiedzi oraz kodowania

Cechy:

- rozszerzalność,
- możliwością zastosowania wielu różnych protokołów sieciowych (HTTP, SMTP lub MSMQ)
- niezależnością od zastosowanego modelu programistycznego

Żądanie - schemat SOAP

```
POST /WebCalculator/Calculator.asmx HTTP/1.1
Content-Type: text/xml
...
SOAPAction: "http://tempuri.org/Add"
Content-Length: 386
...
<?xml version="1.0"?>
<soap:Envelope ...>
  <soap:Header ...>
    ...
  </soap:Header>
  <soap:Body>
    <Add xmlns="http://tempuri.org/">
      <n1>12</n1>
      <n2>10</n2>
    </Add>
  </soap:Body>
</soap:Envelope>
```



Schemat XML definiujący SOAP 1.1 (1)

<xs:schema

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
targetNamespace="http://schemas.xmlsoap.org/soap/envelope/"
>
```

<!-- Envelope, header and body -->

```
<xs:element name="Envelope" type="tns:Envelope" />
```

```
<xs:complexType name="Envelope" >
```

```
<xs:sequence>
```

```
<xs:element ref="tns:Header" minOccurs="0" />
```

```
<xs:element ref="tns:Body" minOccurs="1" />
```

```
<xs:any namespace="##other" minOccurs="0"
  maxOccurs="unbounded" processContents="lax" />
```

```
</xs:sequence>
```

```
<xs:anyAttribute namespace="##other"
  processContents="lax" />
```

```
</xs:complexType>
```

```
<xs:element name="Header" type="tns:Header" />
```

```
<xs:complexType name="Header" >
```

```
<xs:sequence>
```

```
<xs:any namespace="##other" minOccurs="0"
  maxOccurs="unbounded" processContents="lax" />
```

```
</xs:sequence>
```

```
<xs:anyAttribute namespace="##other"
  processContents="lax" />
```

```
</xs:complexType>
```

```
<xs:element name="Body" type="tns:Body" />
```

```
<xs:complexType name="Body" >
```

```
<xs:sequence>
```

```
<xs:any namespace="##any" minOccurs="0"
  maxOccurs="unbounded" processContents="lax" />
```

```
</xs:sequence>
```

```
<xs:anyAttribute namespace="##any"
  processContents="lax" />
```

```
</xs:complexType>
```

Zestaw standardów WWW Consortium (www.w3c.org)

- SOAP 1.2, WSDL 1.1 (1.2 and 2.0)
- additional protocols based on SOAP and WSDL
- protocol bindings (HTTP)
- www.uddi.org, Ariba, IBM i Microsoft

Specyfikacja SOAP - <http://www.w3.org/TR/SOAP/>



Schemat XML definiujący SOAP 1.1 (2)

<!-- Global Attributes -->

```
<xs:attribute name="mustUnderstand"
  default="0" >
  <xs:simpleType>
    <xs:restriction base='xs:boolean'>
      <xs:pattern value='0|1' />
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
<xs:attribute name="actor" type="xs:anyURI" />
<xs:simpleType name="encodingStyle" >
  <xs:list itemType="xs:anyURI" />
</xs:simpleType>
<xs:attribute name="encodingStyle"
  type="tns:encodingStyle" />
<xs:attributeGroup name="encodingStyle" >
  <xs:attribute ref="tns:encodingStyle" />
</xs:attributeGroup>
```

```
<xs:element name="Fault" type="tns:Fault" />
<xs:complexType name="Fault" final="extension" >
  <xs:sequence>
    <xs:element name="faultcode" type="xs:QName" />
    <xs:element name="faultstring" type="xs:string" />
    <xs:element name="faultactor" type="xs:anyURI"
      minOccurs="0" />
    <xs:element name="detail" type="tns:detail"
      minOccurs="0" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="detail">
  <xs:sequence>
    <xs:any namespace="##any" minOccurs="0"
      maxOccurs="unbounded" processContents="lax" />
  </xs:sequence>
  <xs:anyAttribute namespace="##any"
    processContents="lax" />
</xs:complexType>
</xs:schema>
```



Budowa wiadomości SOAP

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header <!-- opcjonalny -->
    <!-- w tym miejscu znajdują się bloki nagłówka... -->
  </soap:Header>
  <soap:Body>
    <!-- treść komunikatu lub element Fault... -->
  </soap:Body>
</soap:Envelope>
```

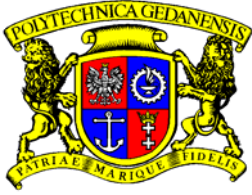
Infrastruktura komunikacyjna specyfikacji SOAP składa się z następujących podstawowych elementów XML: **Envelope** (koperta), **Header** (nagłówek), **Body** (treść) oraz **Fault** (błąd) z przestrzeni nazw <http://schemas.xmlsoap.org/soap/envelope/>

Element Header – informacje precyzujące sposób traktowania wiadomości **Mechanizm rozszerzeń**

- uwierzytelnianie – odbiorca może wymagać uwierzytelniania przed przystąpieniem do przetwarzania wiadomości
- streszczenie wiadomości – w celu zapewnienia ochrony przed przekłamaniami treści wiadomości podczas transportu poprzez wielu pośredników, można tu zastosować np.: podpis cyfrowy
- informacje o routingu – dotyczy to np.: miejsc przeznaczenia wiadomości oraz kolejności w jakiej musi ona tam dotrzeć
- transakcje – konieczność wykonania pewnych czynności w ramach transakcji nadawcy
- informacje o odpłatności – informacje niezbędne do obliczenia należności za świadczenie usług przez odbiorcę wiadomości w zależności od stopnia wykorzystania tych usług

Element Body reprezentuje **właściwą treść komunikatu**; to pojemnik, który może zawierać dowolną liczbę elementów (danych) dowolnej przestrzeni nazw, które chcemy przesłać w komunikacie. Zawiera dane przeznaczone dla ostatecznego odbiorcy.

Element Fault, umieszczony wewnątrz elementu Body — może informować o błędach



SOAP: Wymiana komunikatów (message exchange pattern)

- proste jednokierunkowe przekazanie komunikatu, gdzie nadawca nie otrzymuje odpowiedzi.
- żądanie/odpowieź (*request/response*)
- zaproszenie/odpowieź (*solicit/response*) - serwer wysyła zaproszenie a klient wysyła odpowiedź; jest to odwrotność standardowego wzorca żądanie/odpowieź),
- powiadomienia (*notification*)

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <x:TransferFundsResponse
xmlns:x="urn:examples-org:banking">
      <balances>
        <account>
          <id>22-342439</id> <!-- numer konta -->
          <balance>33.45</balance> <!-- saldo -->
        </account>
        <account>
          <id>98-283843</id>
          <balance>932.73</balance>
        </account>
      </balances>
    </x:TransferFundsResponse>
  </soap:Body>
</soap:Envelope>
```

Przykłady

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Server</faultcode>
      <faultstring>Niewystarczające środki</faultstring>
      <detail>
        <x:TransferError xmlns:x="urn:examples-org:banking">
          <sourceAccount>22-342439</sourceAccount> <!-- rachunek źródłowy -->
          <transferAmount>100.00</transferAmount> <!-- kwota przelewu -->
          <currentBalance>89.23</currentBalance> <!-- bieżące saldo -->
        </x:TransferError>
      </detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```



SOAP: Wymiana komunikatów (2) (message exchange pattern)

- WSDL defines four types:
 - **Request-response**: The operation can receive a request and will return a response

```
<operation name="checkPrice">
  <input message="pc:PriceCheckRequest"/>
  <output message="pc:PriceCheckResponse"/>
</operation>
```
 - **One-way**: The operation can receive a message but will not return a response.

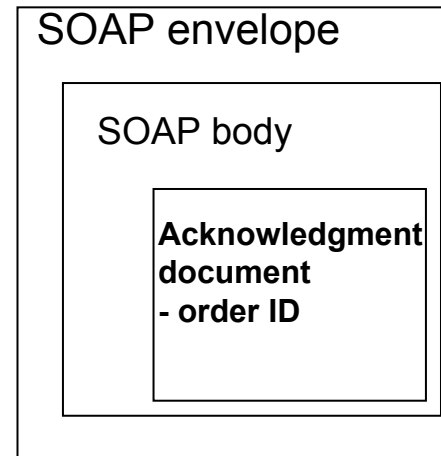
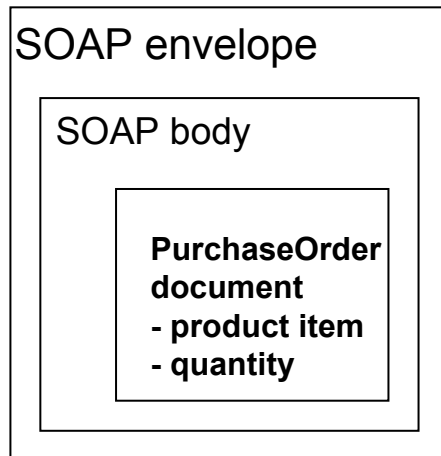
```
<operation name="cancellation">
  <input message="tns:orderCancellation"/>
</operation>
```
 - **Notification**: The operation can send a message but will not wait for a response

```
<operation name="notification">
  <output message="tns:promotionNotification"/>
</operation>
```
 - **Solicit-response**: The operation can send a request and will wait for a response

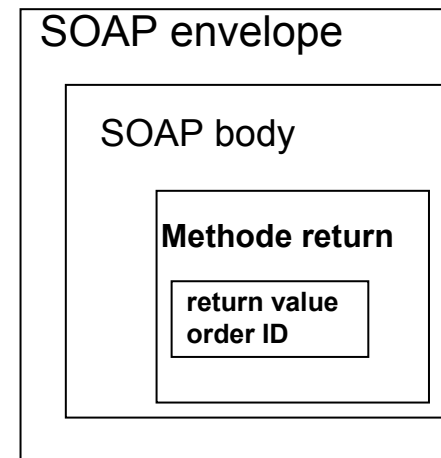
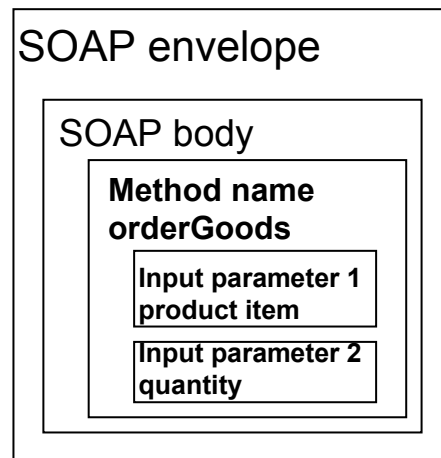
```
<operation name="cancellation">
  <output message="tns:pushThis"/>
  <input message="tns:reponseToPush"/>
</operation>
```
- Different types are defined by decided by the order/occurrences of input and output.



Formaty komunikatów SOAP



Uzgodnienie struktury dokumentu
Document-style interaction



Uzgodnienie sygnatur metod
RPC-style interaction

Dwie kategorie (style) komunikatów SOAP:

- **wiadomości dokumentacyjne** – przesyłanie dokumentów XML, których format jest uzgodniony pomiędzy nadawcą i odbiorcą
- **wiadomości proceduralne** – zapewniają komunikację dwukierunkową i są określane mianem zdalnego wywoływania procedur (RPC). Treść takiej wiadomości zawiera żądanie wykonania jakiejś operacji na serwerze wraz z wymaganymi argumentami oraz zwrócenia wyników.
 - Wiązanie RPC mówi, że wywołanie metody jest przedstawiane jako struktura **struct** nosząca nazwę tej metody. Dla każdego parametru wejściowego [in] lub wejściowo-wyjściowego [in/out] struktura będzie zawierać element, noszący nazwę danego parametru.

Sposoby zapisu serializowanych danych

- **Literal** - dane są serializowane zgodnie z regułami języka XML Schema
- **Encoded** – dane są serializowane zgodnie z określoną regułą kodowania (np. SOAP Encoding)

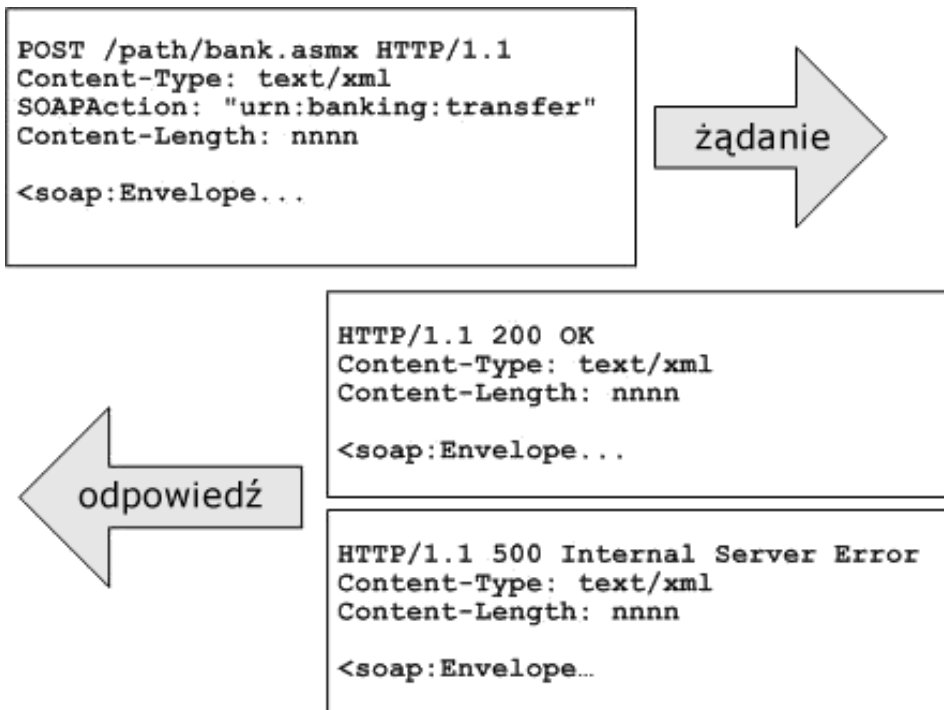


Wiązanie SOAP do protokołów

- **Wiązanie SOAP** dla danego protokołu definiuje, w jaki sposób komunikaty SOAP są przesyłane za pomocą tego protokołu.

MSMQ lub SMTP

- Specyfikacja SOAP 1.1 kodyfikuje wyłącznie wiązanie do protokołu HTTP
- Wzorzec komunikacji żądanie/odpowieź protokołu SOAP odpowiada modelowi żądanie/odpowieź protokołu HTTP



Nagłówek **Content-Type** dla komunikatów zarówno żądania jak i odpowiedzi, przesyłanych protokołem HTTP, musi mieć wartość `text/xml` (lub `application/soap+xml` w SOAP 1.2).

W komunikacie żądania należy stosować metodę **POST**, a URI powinno identyfikować węzeł, do którego kierowane jest żądanie.

Specyfikacja SOAP definiuje także nowy nagłówek HTTP o nazwie **SOAPAction**, który musi być obecny we wszystkich żądaniach SOAP HTTP. Nagłówek **SOAPAction** ma informować o przeznaczeniu komunikatu.

Jeśli nie wystąpiły żadne błędy przy przetwarzaniu komunikatu, odpowiedź HTTP powinna mieć przypisany kod statusu 200.



Example of a SOAP Request / Response

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI "
```

<SOAP-ENV: Envelope

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"

SOAP-ENV: encodingStyle = <http://schemas.xmlsoap.org/soap/encoding/> />

<SOAP-ENV: Body>

<m: GetLastTradePrice xmlns:m="Some-URI ">

<symbol>DIS</symbol>

</m: GetLastTradePrice>

</SOAP-ENV: Body>

</SOAP-ENV: Envelope>

```
HTTP/1.1 200 OK
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
```

<SOAP-ENV: Envelope

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"

SOAP-ENV: encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />

<SOAP-ENV: Body>

<m: GetLastTradePriceResponse xmlns:m="Some-URI ">

<Price>34.5</Price>

</m: GetLastTradePriceResponse>

</SOAP-ENV: Body>

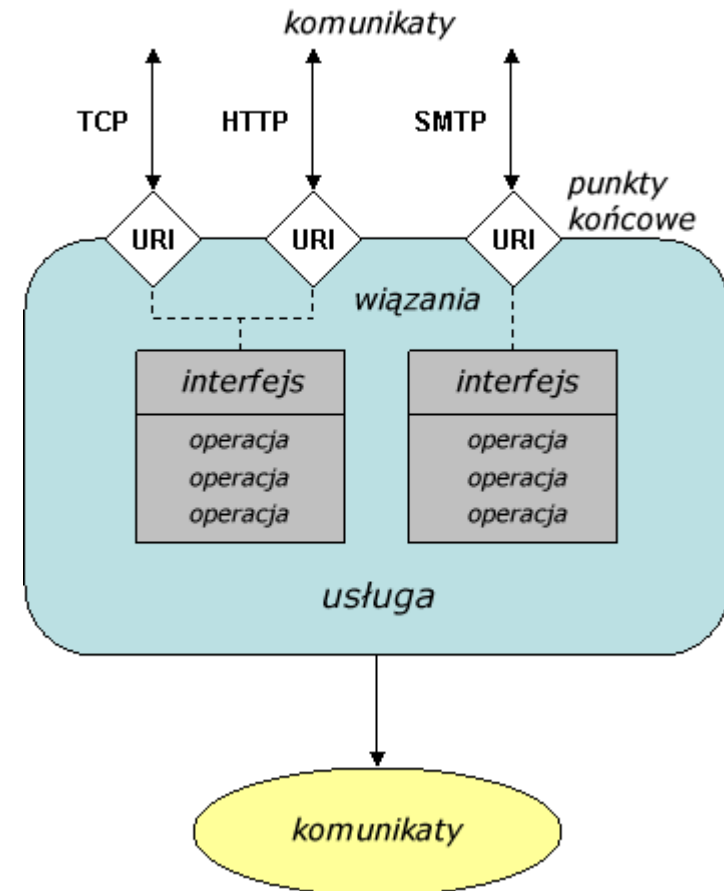
</SOAP-ENV: Envelope>



4.4 WSDL – Web Services Description Language

Język Web Services Description Language (WSDL)

- jakie **komunikaty XML** mogą być użyte
- możliwe metody komunikacji. Wymiana komunikatów jest nazywana **operacją**.
- grupowania powiązanych ze sobą operacji w **interfejsy**
- jaki protokół komunikacyjny należy stosować do komunikacji z usługą oraz mechanizmy związane z danym protokołem - stosowane polecenia, nagłówki i kody błędów
- opisanie sposobu korzystania z **interfejsu** w połączeniu z konkretnym **protokołem** komunikacyjnym - **wiązanie**
- każde **wiązanie** powinno być dostępne pod unikalnym, identyfikowanym za pomocą **adresu URI**, nazywanym także **punktem końcowym usługi XML Web Service**





Struktura dokumentu WSDL

```
<!-- struktura definicji WSDL -->
```

```
<definitions
```

```
  name="MathService,,  
  targetNamespace="http://example.org/math/"  
  xmlns="http://schemas.xmlsoap.org/wsdl/">
```

```
<!-- definicje abstrakcyjne -->
```

```
<types> ...
```

```
<message> ...
```

```
<portType> ...
```

```
<!-- definicje konkretne -->
```

```
<binding> ...
```

```
<service> ...
```

```
</definitions>
```

← Co

← Jak

← Gdzie

- Elementy **types**, **message** i **portType** - składają się na **interfejs programistyczny**, do którego będziemy uzyskiwali dostęp z naszej aplikacji
- Elementy **binding** i **service** opisują w jaki sposób wywołania abstrakcyjnego interfejsu tłumaczone są na przesyłane komunikaty

types	zawiera definicje typów abstrakcyjnych zdefiniowanych przy użyciu schematu XML
message	definicja abstrakcyjnego komunikatu , który może składać się z wielu części, a każda część może być innego typu
portType	abstrakcyjny zbiór operacji obsługiwanych przez jeden lub kilka punktów końcowych (nazywany interfejsem); operacje definiowane są poprzez określenie wymiany komunikatów
binding	specyfikacja konkretnego protokołu i formatu danych dla określonego portType
service	zbiór powiązanych punktów końcowych - punkt końcowy definiowany jest jako połączenie wiązania i adresu (URI)



Przykład: WSDL – elementy types

<definitions

```
xmlns="http://schemas.xmlsoap.org/wsdl/"  
xmlns:xs="http://www.w3.org/2001/XMLSchema"  
xmlns:y="http://example.org/math/"  
xmlns:ns="http://example.org/math/types/"  
targetNamespace="http://example.org/math/">
```

<types>

<xs:schema

```
targetNamespace="http://example.org/math/types/"  
xmlns="http://example.org/math/types/" >  
<xs:complexType name="MathInput">  
  <xs:sequence>  
    <xs:element name="x" type="xs:double"/>  
    <xs:element name="y" type="xs:double"/>  
  </xs:sequence>  
</xs:complexType>  
<xs:complexType name="MathOutput">  
  <xs:sequence>  
    <xs:element name="result" type="xs:double"/>  
  </xs:sequence>  
</xs:complexType>  
<xs:element name="Add" type="MathInput"/>  
<xs:element name="AddResponse" type="MathOutput"/>
```

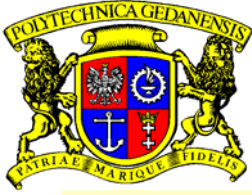
</xs:schema>

</types>

...

</definitions>

- Element **types** zawiera definicje typów w postaci schematów XML. Do umieszczonych w tym miejscu definicji typów odwołują się definicje wyższego poziomu, określające szczegóły strukturalne komunikatów.
- Element **types** może zawierać dowolną liczbę elementów **schema** z przestrzeni nazw <http://www.w3.org/2001/XMLSchema>



WSDL – elementy message

<definitions

```
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:y="http://example.org/math/"
xmlns:ns="http://example.org/math/types/"
targetNamespace="http://example.org/math/"
>
...
<message name="AddMessage">
  <part name="parameter" element="ns:Add"/>
</message>
<message name="AddResponseMessage">
  <part name="parameter" element="ns:AddResponse"/>
</message>
...
</definitions>
```

- Element **message** definiuje abstrakcyjny **komunikat**, który ma służyć jako dane wejściowe lub wyjściowe operacji.
- **Komunikaty** składają się z co najmniej jednego elementu **part** i z każdym takim elementem związany jest albo atrybut **element** (gdy treść komunikatu stanowi dokument XML), albo **atrybut type** (gdy treść komunikatu to wywołanie RPC).



WSDL – interfejsy: portType

<definitions

```
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:y="http://example.org/math/"
xmlns:ns="http://example.org/math/types/"
targetNamespace="http://example.org/math/"
>
...
<portType name="MathInterface">
  <operation name="Add">
    <input message="y:AddMessage"/>
    <output message="y:AddResponseMessage"/>
  </operation>
</portType>
...
</definitions>
```

- Element *portType* definiuje grupę operacji
- Każdy element *portType* musi posiadać unikalną **nazwę**, dzięki której można się do niego odwoływać z innych obszarów definicji WSDL.
- Każdy element *operation* zawiera kombinację elementów *input* i *output* (jeśli zawiera elementy *output*, to może także zawierać elementy *fault*).
- **Kolejność** tych elementów definiuje wzorzec wymiany komunikatów (*MEP*) obsługiwany przez daną operację (jednokierunkowy, żądanie-odpowiedź, zaproszenie-odpowiedź, powiadomienie)
- Elementy *input*, *output* i *fault*, stosowane w elemencie *operation*, muszą odwoływać się do definicji komunikatu



WSDL – wiązanie: binding

<definitions

```
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:y="http://example.org/math/"
xmlns:ns="http://example.org/math/types/"
targetNamespace="http://example.org/math/"
>
...
<binding name="MathSoapHttpBinding" type="y:MathInterface">
  ... <-- element rozszerzenia -->
  <operation name="Add">
    ... <-- element rozszerzenia -->
    <input>
      ... <-- element rozszerzenia -->
    </input>
    <output>
      ... <-- element rozszerzenia -->
    </output>
  </operation>
  ...
</binding>
...
</definitions>
```

- Element **binding** opisuje szczegóły dotyczące stosowania konkretnego elementu **portType** z wybranym protokołem.
- Dla każdej **operacji** w opisywanym elemencie **portType**, element binding zawiera element **operation**, a także kilka elementów rozszerzenia (opis szczegółów wiązania).



WSDL – wiązanie: binding (2)

```
<definitions
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:y="http://example.org/math/"
  xmlns:ns="http://example.org/math/types/"
  targetNamespace="http://example.org/math/"
>
...
<binding name="MathSoapHttpBinding" type="y:MathInterface">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="Add">
    <soap:operation
      soapAction="http://example.org/math/#Add"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
  ...
</binding>
...
</definitions>
```

wiązanie SOAP do HTTP dla elementu *portType* o nazwie **MathInterface**

Element **soap:binding** sygnalizuje, że jest to wiązanie w standardzie SOAP 1.1.

Atrybut **style** elementu informuje o domyślnym sposobie wykorzystania usługi (możliwe wartości to *document* i *rpc*), a atrybut **transport** o wymaganym protokole transportowym (w tym przypadku HTTP).

Element **soap:operation** definiuje wartość nagłówka HTTP SOAPAction dla każdej operacji.

Atrybut **use** elementu **soap:body** określa sposób kodowania poszczególnych fragmentów komunikatu wewnątrz elementu **body** (możliwe wartości *literal* i *encoded*).



WSDL – services

<definitions

```
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:y="http://example.org/math/"
xmlns:ns="http://example.org/math/types/"
targetNamespace="http://example.org/math/"
>
...
<service name="MathService">
  <port name="MathEndpoint" binding="y:MathSoapHttpBinding">
    <soap:address
      location="http://localhost/math/math.asmx"/>
  </port>
</service>
</definitions>
```

Element **service** definiuje kolekcję elementów *port* i *endpoint*, które udostępniają poszczególne wiązania w sieci

Każdemu portowi trzeba nadać nazwę i przypisać do niego wiązanie (atrybut binding).

Wewnątrz elementu **port** należy użyć elementu rozszerzenia, aby zdefiniować **adres wiązania**

Przykład: zdefiniowano usługę o nazwie **MathService**, która udostępnia wiązanie **MathSoapHttpBinding** pod adresem **http://localhost/math/math.asmx**

Generowanie klas Proxy na podstawie definicji WSDL.

.NET - **wSDL.exe** może wygenerować klasę umożliwiającą dostęp do usługi oraz klasę implementującą usługę.

Apache Axis - **WSDL2Java**, które przeprowadza takie same operacje, tyle że dla klas Java



Przykład 1 - podsumowanie

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:y="http://example.org/math/"
  xmlns:ns="http://example.org/math/types/"
  targetNamespace="http://example.org/math/">
```

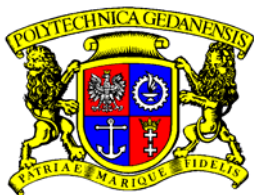
```
<types>
  <xs:schema targetNamespace="http://example.org/math/types/"
    xmlns="http://example.org/math/types/"
    elementFormDefault="unqualified" attributeFormDefault="unqualified">
    <xs:complexType name="MathInput">
      <xs:sequence>
        <xs:element name="x" type="xs:double"/>
        <xs:element name="y" type="xs:double"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="MathOutput">
      <xs:sequence>
        <xs:element name="result" type="xs:double"/>
      </xs:sequence>
    </xs:complexType>
    <xs:element name="Add" type="MathInput"/>
    <xs:element name="AddResponse" type="MathOutput"/>
  </xs:schema>
</types>
```

```
<message name="AddMessage">
  <part name="parameters" element="ns:Add"/>
</message>
<message name="AddResponseMessage">
  <part name="parameters" element="ns:AddResponse"/>
</message>
```

```
<portType name="MathInterface">
  <operation name="Add">
    <input message="y:AddMessage"/>
    <output message="y:AddResponseMessage"/>
  </operation>
</portType>
```

```
<binding name="MathSoapHttpBinding" type="y:MathInterface">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="Add">
    <soap:operation soapAction="http://example.org/math/#Add"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
<service name="MathService">
  <port name="MathEndpoint" binding="y:MathSoapHttpBinding">
    <soap:address location="http://localhost/math/math.asmx"/>
  </port>
</service>
```

```
</definitions>
```



4.5. UDDI

UDDI (Universal Description, Discovery and Integration) jest:

- jednym z podstawowych komponentów architektury SOA
- mechanizmem odnajdywania opisów usług, którego główną rolą jest standaryzacja profili usług, ich przechowywanie oraz umożliwienie odnalezienia potrzebnej usługi i skorzystania z niej
- katalogiem (rejestr) o dostawcach i ich usługach, o strukturze zgodnej ze standardowym **modelem danych**

OASIS

White Pages

Podstawowa informacja kontaktowa, identyfikator firmy lub dostawcy usługi.

Yellow Pages

Kategoryzacja usług web według taksonomii:

NAICS - North American Industry Classification System

SIC - Standard Industrial Classification

...

Green Pages

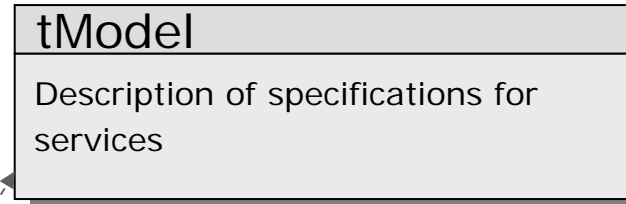
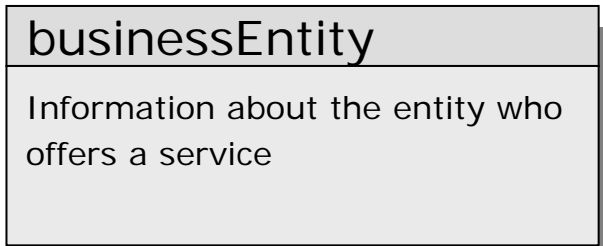
Informacja techniczne opisująca usługi web

<http://uddi.xml.org/>



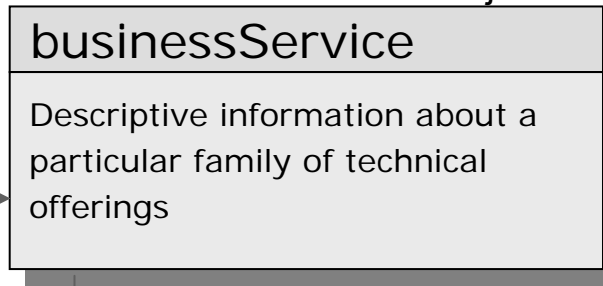
UDDI: Model danych

Podstawowe informacje identyfikujące firmę, także w kontekście branżowym, geograficznym

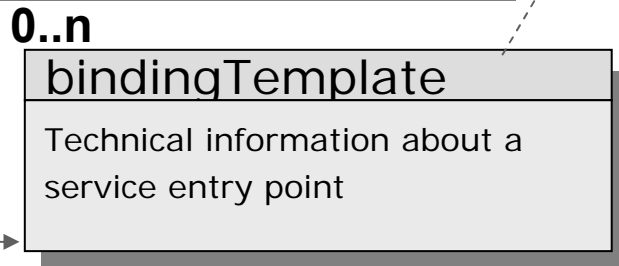


Technical model

0..n Informacje o usłudze



Referencje do tModel, której obiekty opisują (lokalizację) specyfikacji usługi WSDL



0..n

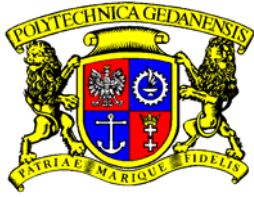
Informacje techniczne dla wywołania usługi. Zbiór odnośników do specyfikacji

test.uddi.microsoft.com

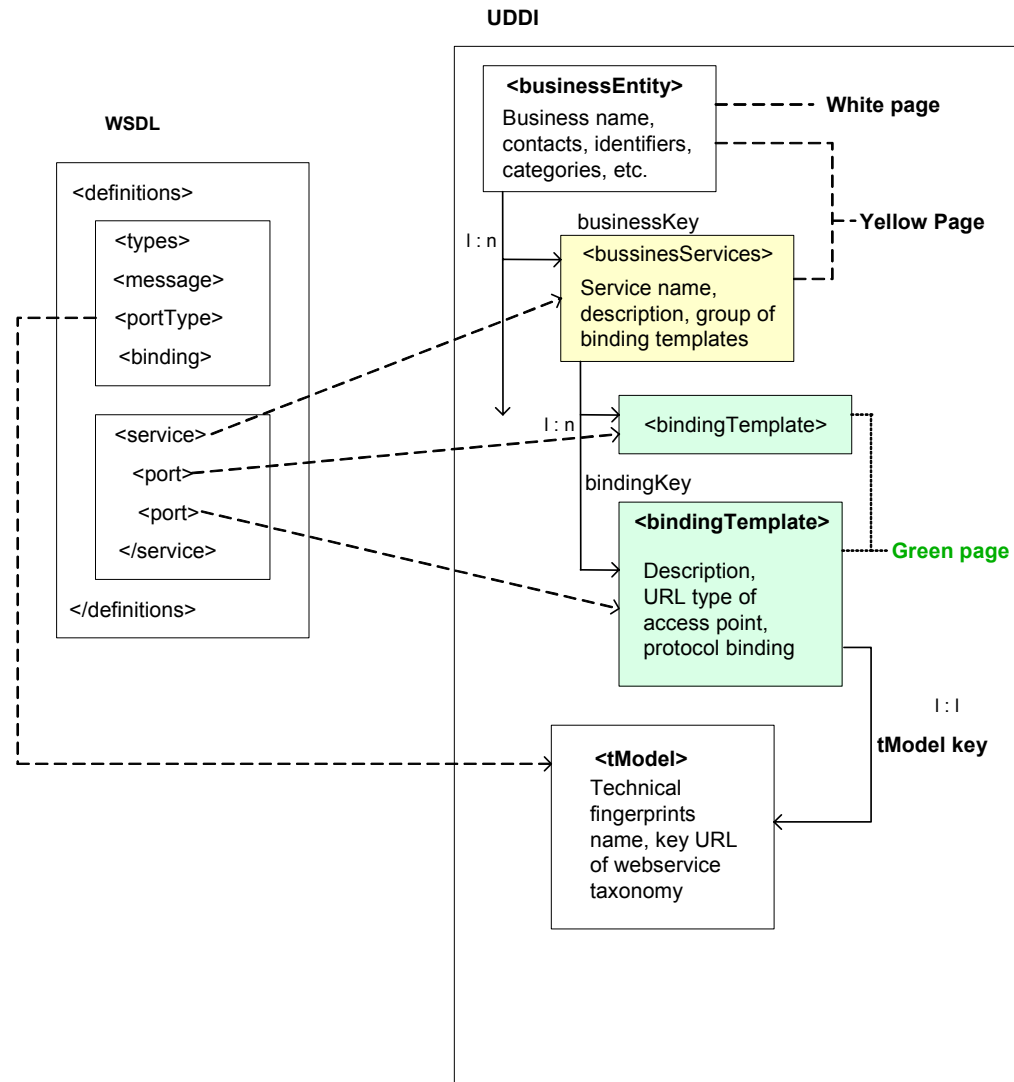
API rejestru UDII

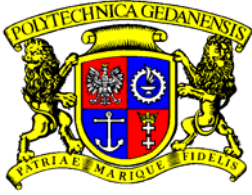
.NET

```
using System;
using Microsoft.Uddi; //Publish,SaveTModel,TModelDetail,DeleteTModel,
//SaveBusiness,BusinessDetail,DeleteBusiness
using Microsoft.Uddi.ServiceType; //TModel,TModelInstanceInfo
using Microsoft.Uddi.Business; //BusinessEntity,Contact
using Microsoft.Uddi.Service; //BusinessService
using Microsoft.Uddi.Binding; //BindingTemplate
using Microsoft.Uddi.Api; //URLType
```

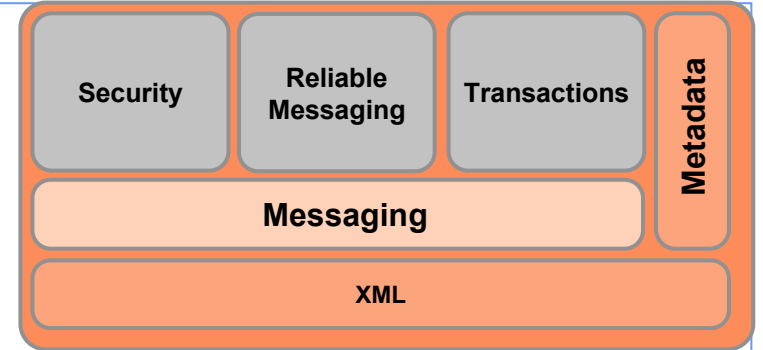


UDDI - WSDL

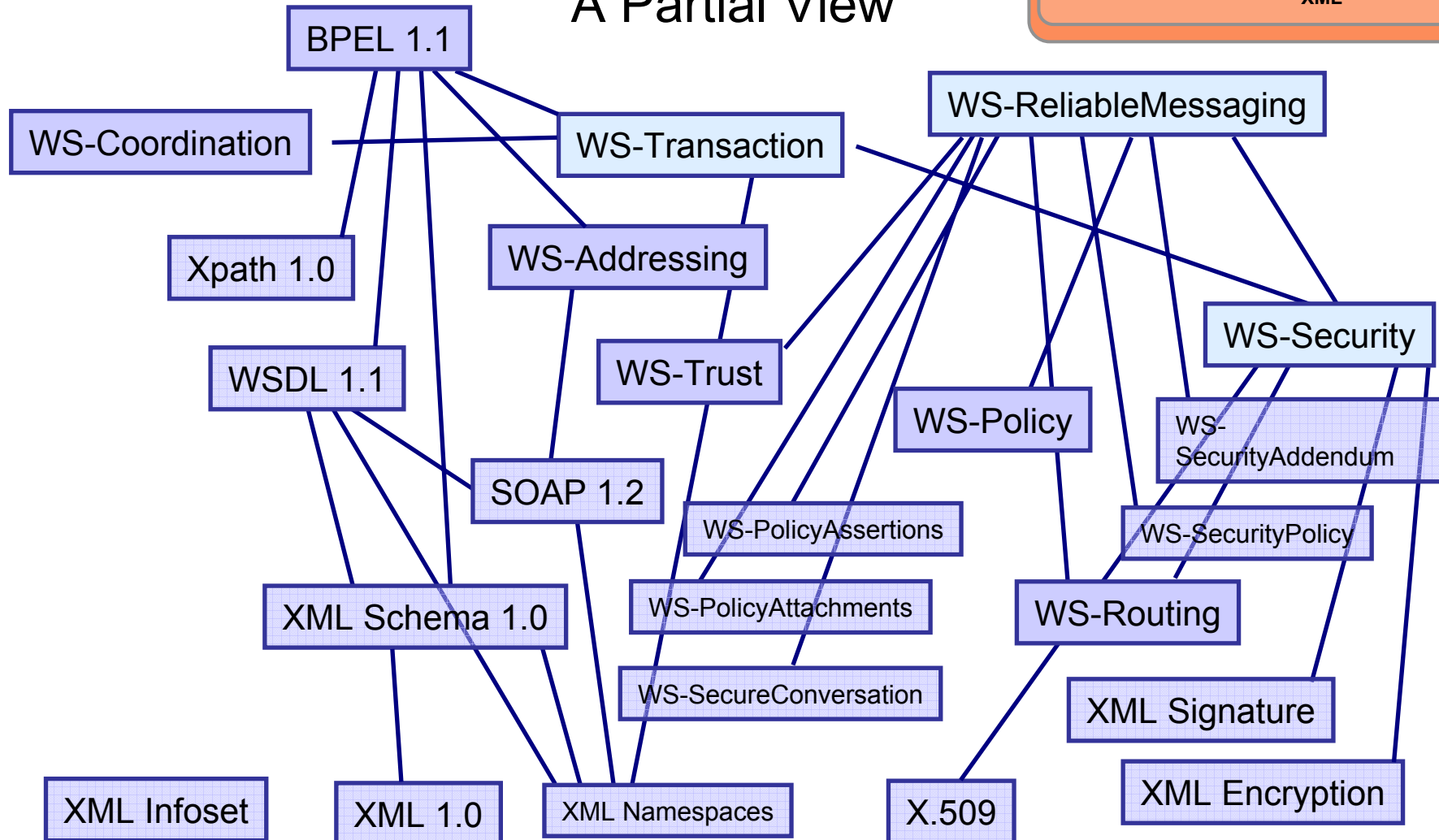




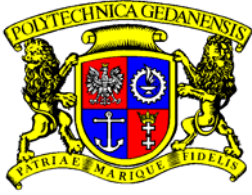
4.6 Standardy WS - zależności



A Partial View



Gartner



Web Services Interoperability

WSE3.0: *WS-Addressing, WS-Attachments, WS-Security, WS-SecurityPolicy, WS-Trust, WS-SecureConversation WS-Policy,*

Adresowanie zasobów

- Adresowanie komunikatów SOAP na poziomie transportowym (np. HTTP)

WS-Addressing

- Referencja punktu końcowego (endpoint reference) – adres zasobu
- Właściwości adresowania wiadomości (message addressing properties) – identyfikacja komunikatów przesyłanych pomiędzy klientami i usługami
- `<wsa: EndpointReference> ... </wsa: EndpointReference>`
- Definiuje zbiór nagłówków (np. `wsa:To`, `wsa:From`, `wsa:ReplyTo`, `wsa:FaultTo`, `wsa:MessageID`, ...)

WS-Attachments

Załączniki binarne

- Kodowanie Base64Encoding
- MTOM Message Transmission Optimization mechanism)
- XOP (XML-binary Optimized Packaging)

WS-Policy

- Wymagania biznesowe, rodzaj udostępnianych danych, procedury bezpieczeństwa, ograniczenia techniczne, niezawodnościowe

• WS-Security

- Definiuje oparte na standardach rozszerzenie SOAP zapewniające możliwość wymuszenia bezpieczeństwa podczas wywoływania usług Web Services.
- Profil podstawowy WS-I (WS-I Basic Profile) - organizacja WS-I
- spójność komunikatu (Message Integrity),
- autoryzacja pojedynczego komunikatu (Single Message Authentication),
- poufność komunikatu (Message Confidentiality).

• WS- Reliable Messaging

• WS-Transaction



Specyfikacje WS- Security

