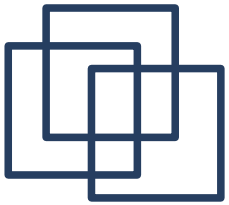
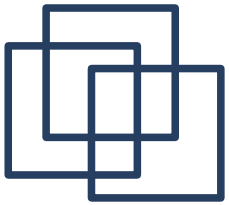


XML i SOAP

1. XML (eXtensive Markup Language).
- wprowadzenie,
- zastosowania,
- przykłady,
- specyfikacje DTD (Document Type Definition).

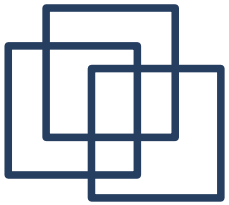
2. SOAP (Simple Object Access Protocol)
- struktura wiadomości,
- zdalne wywołanie metody z wykorzystaniem SOAP,
- przykłady.



XML - wprowadzenie

XML (eXtensive Markup Language) jest tekstowym językiem znaczników. Współcześnie jest on używany do wymiany danych między programami działającymi w środowisku sieciowym. Podobnie jak HTML do opisu danych używa tagów (identyfikatorów otoczonych nawiasami „<” i „>”, np. „<tag>”). W przeciwieństwie do HTML'a tagi służą raczej do identyfikacji danych a nie do określenia sposobu ich prezentacji.

```
<message>
    <to>odbiorac@adres.net</to>
    <from>nadawca@innyadres.net</from>
    <subject>coś o XML'u</subject>
    <text>
        Ciekawe to czego to jest potrzebne.
    </text>
</message>
```



XML - wprowadzenie

Najważniejsza różnica pomiędzy XML i HTML polega na przestrzeganiu hierarchicznego formatu dokumentu w pierwszym przypadku:

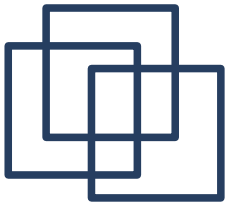
- dla każdego tagu musi istnieć tag zamykający: (**<text>...</text>**),
- tag **<text>** może być zamknięty tylko przez **</text>**,
- tagi nie zawierające treści można zapisywać w postaci **<text/>**.

```
<message>
    <text>
        Dobrze
    </text>
</message>
```

```
<message>
    <text/>
</message>
<!-- Dobrze (komentarz) -->
```

```
<message>
    <text>
        Źle
    </message>
</text>
```

```
<message>
    <text>
        Źle
</message>
```

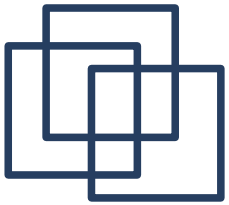


XML tagi i atrybuty

Tagi mogą posiadać atrybuty – dodatkową informację zawartą wewnątrz tagu:

```
<message to="odbiorac@adres.net" from="nadawca@innyadres.net"
            subject="coś o XML'u">
    <text>
        Ciekawe do czego to jest potrzebne.
    </text>
</message>
```

Używanie atrybutów i tagów wewnętrznych jest niemal równoważne tzn. wiadomości (messages) opisywane przez dwa przedstawione dokumenty XML są takie same. Wybór sposobu opisu uzależniony jest od upodobania i konkretnej sytuacji.



XML nagłówek

Każdy dokument XML rozpoczyna się (opcjonalnym) nagłówkiem.

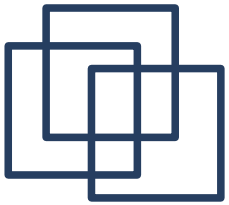
W najprostszej wersji wygląda on następująco:

```
<?xml version="1.0"?>
```

Często jednak zawiera on dodatkowe informacje, np.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

- **`version`** – określa wersję XML wykorzystywaną do opisu danych,
- **`encoding`** – kodowanie znaków wewnątrz dokumentu. Domyślnie UTF-8 (tzw. compressed unicode).
- **`standalone`** – określa, czy dokument jest zależny od jakiś zewnętrznych dokumentów.



XML – zastosowania

Jest wiele powodów dla których warto używać XML'a. Niektóre z nich:

• format tekstowy – można wykorzystać dowolny edytor tekstowy do obsługi dokumentu XML,

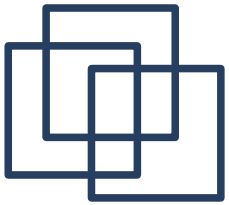
• identyfikacja (opis) danych – główny nacisk na strukturę danych a nie na sposób ich prezentacji,

• prosta konwersja do formatów korzystających ze styli – możliwość łatwego zdefiniowania przekształceń dokument XML w formaty określające sposób prezentacji dokumentu (HTML, PDF, TeX, RTF),

• modularność – poszczególne moduły mogą być używane wielokrotnie

• odwołania do innych dokumentów,

• hierarchiczna struktura – pozwala zwykle na szybsze przeszukiwanie dokumentu.



XML - zastosowania

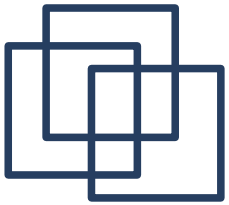
Podstawowe zastosowania XML'a.

• tradycyjne przetwarzanie danych – dane zapisane w formacie XML są przetwarzane przez odpowiedni program,

• programowanie opisane dokumentami (*document-driven programming*) – sposób działania programu jest opisany w dokumencie XML, który opisuje używane obiekty, logikę biznesową, interfejs użytkownika. Na podstawie tych informacji aplikacja jest tworzona „w locie”,

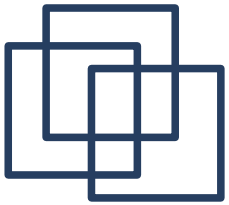
• archiwizacja – fundament dla document-driven programming. Odpowiednio skonfigurowane komponenty mogą zostać zapisane do późniejszego wykorzystania,

• powiązywanie (binding) – dokument opisujący strukturę XML'a (DTD) może być wykorzystany do budowy szkieletu aplikacji przetwarzającej dane.



XML - przykład

```
<?xml version='1.0' encoding='utf-8'?>

<!-- A SAMPLE set of slides -->
<slideshow title="Przykładowy pokaz slajdów"
           date="24.01.2005"
           author="Anonim"
>

    <!-- TITLE SLIDE -->
    <slide type="all">
        <title>Obudź się!</title>
    </slide>

    <!-- OVERVIEW -->
    <slide type="tech">
        <title>Przegląd</title>
        <item>Dlaczego <em>XML</em> jest wspaniały</item>
        <item/>
        <item>Technologia <em>SOAP</em> z załącznikami</item>
    </slide>

</slideshow>
```



XML - DTD

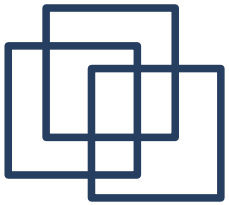
Aby poprawnie zinterpretować dane zapisane w XML'u korzysta się ze specyfikacji zapisanej w plikau DTD (*Document Type Definition*). Dzięki temu program analizujący dokument XML (parser) potrafi sprawdzić formalną poprawność pliku oraz utworzyć odpowiednie obiekty. I tak:

```
<!ELEMENT slideshow (slide+)>
```

mówi, że element **`slideshow`** zawiera z co najmniej jedną strukturę **`slide`**.

Kwalifikatory:

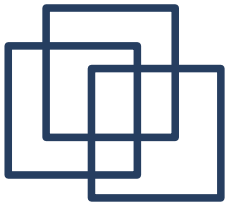
- ? – zero lub jedną,
- * – zero lub więcej,
- + – jedną lub więcej.



XML - DTD

```
<!ATTLIST slideshow
            title    CDATA    #REQUIRED
            date     CDATA    #IMPLIED
            author   CDATA    "unknown"
>
```

określa trzy atrybuty pojawiające się w tagu **slideshow**. Każdy z nich jest typu **CDATA** – (nieparsowalne) pole tekstowe. Podanie **title** jest wymagane w przeciwieństwie do **date** i **author**. Jeśli **date** nie zostanie podana wartość tego atrybutu jest określona przez aplikację przetwarzającą dane. W przypadku braku określenia pola **author** zostanie mu przypisana domyślna wartość „**unknown**”. Inny często używany typ to **#PCDATA** – parsowalne pole tekstowe.

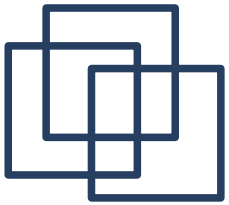


XML - DTD

```
<!ELEMENT slide (title, item*)>
<!ATTLIST slide
            type   (tech | exec | all) #IMPLIED
```

Struktura **slide** składa się z title oraz dowolnej liczby obiektów **item**. Każdy slide może posiadać atrybut **type** przybierający jedną z trzech wartości: **tech**, **exec**, **all**.

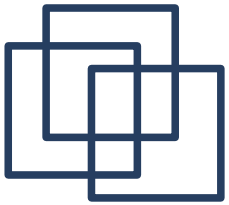
Warto zwrócić uwagę, że dokument ani dokument XML ani odpowiadający mu DTD nie zawiera interpretacji odpowiadającej tym trzem stałym wartościom.



XML - DTD

Przykładowy plik DTD dla pokazanego wcześniej dokumentu mógłby wyglądać następująco:

```
<?xml version='1.0' encoding='us-ascii'?>

<!-- DTD dla przykładowego pokazu slajdów -->

<!ELEMENT slideshow (slide+)>

<!ATTLIST slideshow
            title    CDATA    #REQUIRED
            date     CDATA    #IMPLIED
            author   CDATA    "unknown"
>
<!ELEMENT slide (title, item*)>
<!ATTLIST slide
            type   (tech | exec | all) #IMPLIED
>
<!ELEMENT title (#PCDATA)>
<!ELEMENT item (#PCDATA | item)* >
```

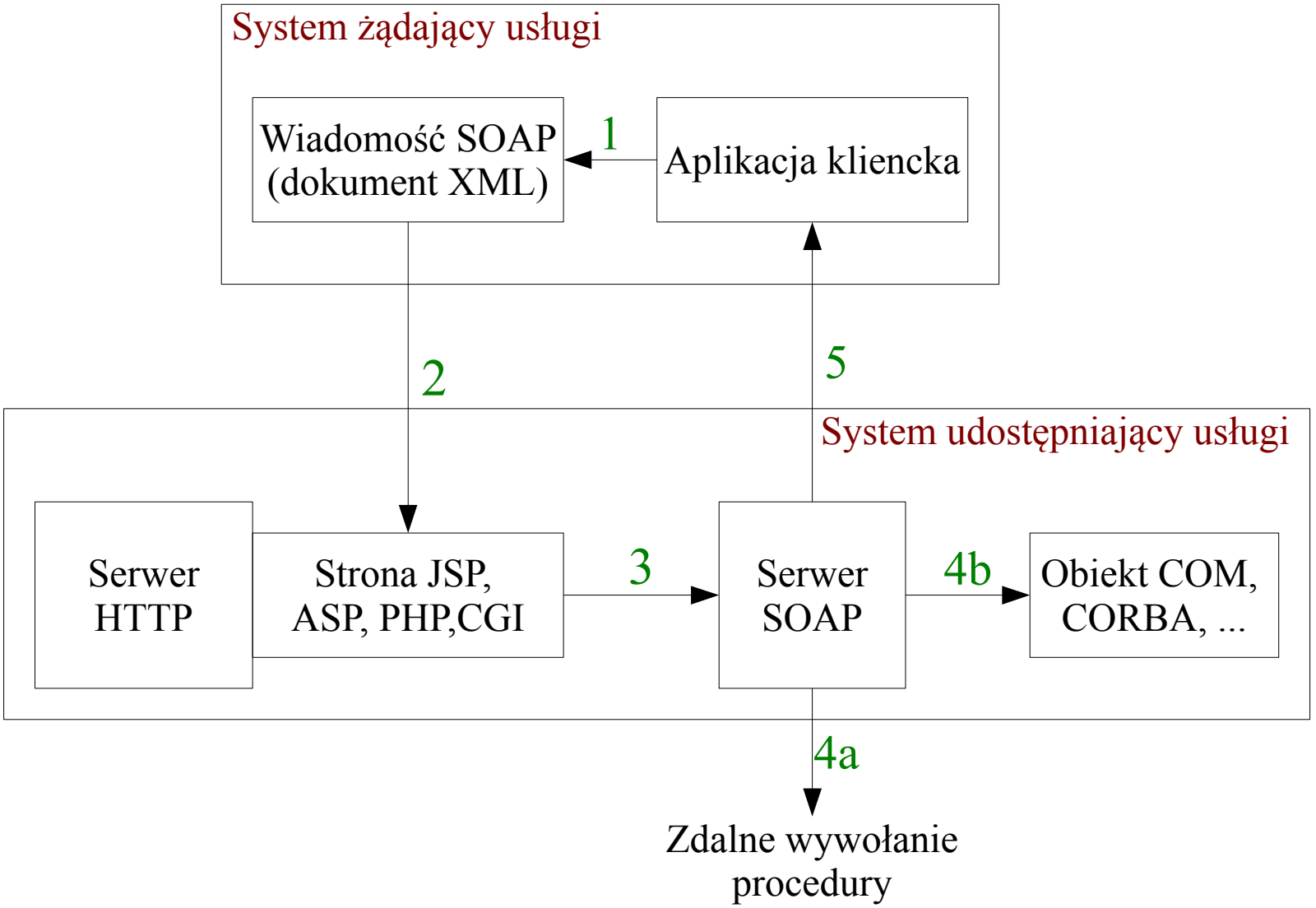


SOAP

SOAP (*Simple Object Access Protocol*) jest ogólnym protokołem do przesyłania wiadomości (*messages*). Jest też standardem wykorzystywanym obecnie do komunikacji z usługami sieciowymi (*web services*). Wiadomość to dokument w formacie XML, który może być przesyłany za pośrednictwem dowolnego, popularnego protokołu z warstwy zastosowań (najczęściej HTTP lub HTTPS).

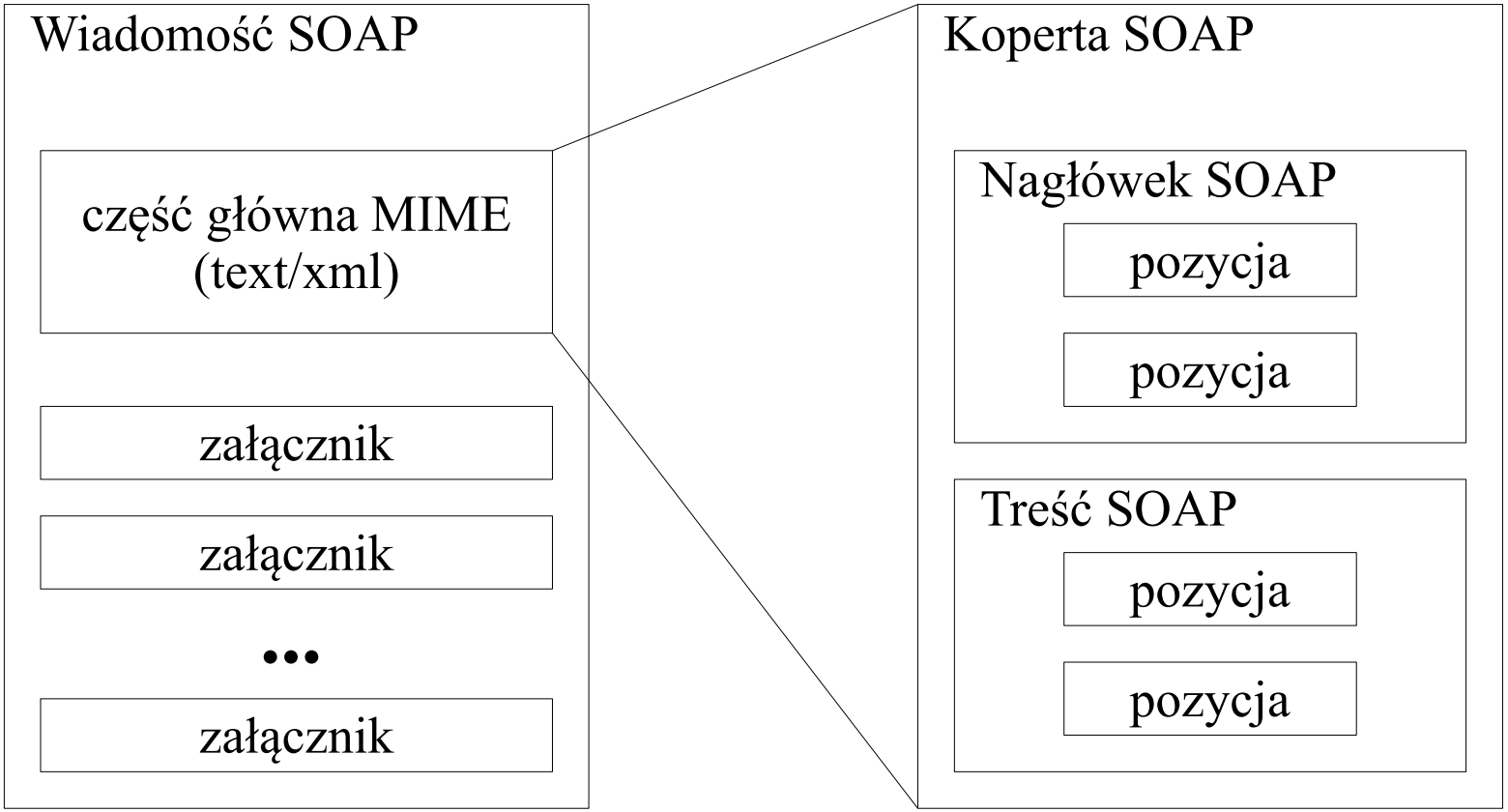
SOAP jest wykorzystywany w ramach architektury orientowanej na usługi, umożliwiając przekształcenie aplikacji rozproszonej na usługi SOAP, które mogą być zdalnie wywołane za pomocą dowolnego urządzenia.

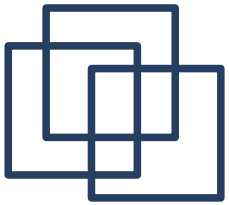
SOAP – schemat działania



Wiadomość SOAP

Podstawową jednostką komunikacji pomiędzy klientem SOAP a serwisem udostępniającym usługi jest wiadomość SOAP.

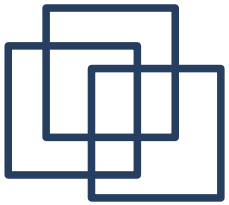




Wiadomość SOAP

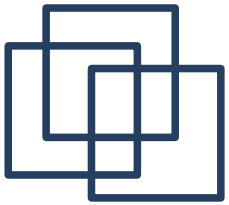
Przykładowa postać wiadomości SOAP:

```
<?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/1999/XMLSchema-instance"
     xmlns:xsd="http://www.w3.org/1999/XMLSchema">
 <SOAP-ENV:Body>
 <ns1:doGoogleSearch xmlns:ns1="urn:GoogleSearch"
     SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <key xsi:type="xsd:string">00000000000000000000000000000000</key>
  <q xsi:type="xsd:string">Learning Wireless Java</q>
  <start xsi:type="xsd:int">0</start>
  <maxResults xsi:type="xsd:int">10</maxResults>
  <filter xsi:type="xsd:boolean">true</filter>
  <restrict xsi:type="xsd:string" />
  <safeSearch xsi:type="xsd:boolean">false</safeSearch>
  <lr xsi:type="xsd:string" />
  <ie xsi:type="xsd:string">latin1</ie>
  <oe xsi:type="xsd:string">latin1</oe>
  </ns1:doGoogleSearch>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



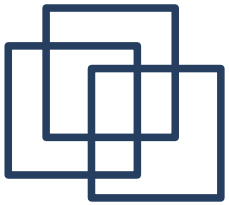
Klient SOAP

```
import java.net.*;
import javax.xml.soap.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;

public class SoapClient {
    public static void main(String args[]) {
        try {
            // Utwórz połączenie
            SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
            SOAPConnection connection = scf.createConnection();
            SOAPFactory sf = SOAPFactory.newInstance();

            // Stwórz wiadomość
            MessageFactory mf = MessageFactory.newInstance();
            SOAPMessage message = mf.createMessage();

            // Stwórz części wiadomości
            SOAPPart soapPart = message.getSOAPPart();
            // Wstaw treść wiadomości
            StreamSource msg = new StreamSource(new FileInputStream
                                                ("c:/request.xml"));
            soapPart.setContent(msg);
```



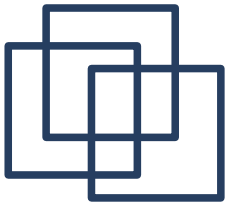
Klient SOAP

```
/* alternatywnie
SOAPEnvelope envelope = soapPart.getEnvelope();
SOAPBody body = envelope.getBody();
Name bodyName = sf.createName("doGoogleSearch", "ns1",
                                        "urn:GoogleSearch");
SOAPBodyElement bodyElement = body.addBodyElement(bodyName);
Name name = sf.createName("key");
SOAPElement key = bodyElement.addChildElement(name);
key.addTextNode("00000000000000000000000000000000");
...
*/

// Wyświetl wysyłaną kopertę
System.out.println("SOAP Request Sent:");
message.writeTo(System.out);
// Ustaw punkt docelowy
URL endpoint = new URL
        ("http://api.google.com/search/beta2");
// Wyślij wiadomość
SOAPMessage response = connection.call(message, endpoint);
// Zamknij połączenie
connection.close();
```



Klient SOAP

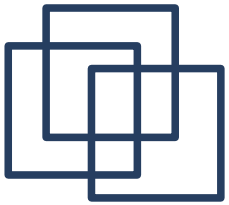
```
            // Wyświetl odpowiedź
            System.out.println("SOAP Response Received:");

            // Utwórz transformer
            TransformerFactory tf = TransformerFactory.newInstance();
            Transformer transformer = tf.newTransformer();

            // Odbierz obpowiedź
            Source content = response.getSOAPPart().getContent();

            // Wyświetl odpowiedź w konsoli
            StreamResult result = new StreamResult(System.out);
            transformer.transform(content, result);
            System.out.println();

        } catch(Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```



Załączniki SOAP

Załączniki w wiadomości SOAP są umieszczane w sposób identyczny jak w wiadomościach email.

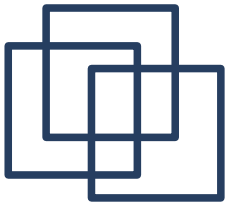
```
...
</SOAP-ENV:Envelope>

--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: base64
Content-ID: <claim061400a.tiff@claiming-it.com>

...Base64 encoded TIFF image...
--MIME_boundary
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <claim061400a.jpeg@claiming-it.com>

...Raw JPEG image..
--MIME_boundary--
```



Załączniki SOAP

Dostęp do załączników w odebranej wiadomości można uzyskać posługując się instancją klasy **`AttachmentPart`**:

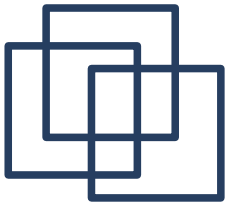
```
...
java.util.Iterator iterator = message.getAttachments();
while (iterator.hasNext()) {
    AttachmentPart attachment = (AttachmentPart)iterator.next();
    String id = attachment.getContentId();
    String type = attachment.getContentType();
    System.out.print("Attachment " + id + " has content type " + type);
    if (type == "text/plain") {
        Object content = attachment.getContent();
        System.out.println("Attachment " + "contains:\n" + content);
    }
}
...
```



Dodatkowe możliwości

Funkcjonalność SOAP zawiera dodatkowo kilka nieomawianych tu elementów. Najważniejsze z nich to:

1. Atrybuty w nagłówku – określają sposób przetwarzania wiadomości przez odbiorcę. Umożliwiają one m. in.: identyfikację klienta, zarządzanie tranzakcjami, itp.
2. Obsługa błędów – w przypadku błędu klient otrzymuje kopertę SOAP zawierającą: kod błędu, opis błędu, nazwę usługi, która zwróciła błąd, szczegóły błędu.



Podsumowanie

SOAP umożliwia jednorodny mechanizm dostępu do zdalnych usług. Jego działanie opiera się na wiadomościach zapisanych w formacie XML, które są wymieniane między klientem a serwerem za pomocą jakiegokolwiek protokołu warstwy zastosowań. Dzięki temu aplikacja kliencka może być napisana w dowolnym języku programowania, który umożliwia nawiązanie połączenia z serwerem SOAP. Istnieje możliwość zbudowania serwera SOAP, który umożliwi dostęp do zdalnych obiektów wykorzystujących dowolną z wcześniej omówionych technologii (RMI, CORBA, COM, RPC).