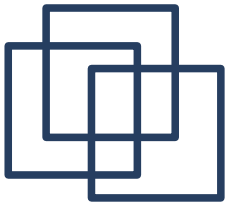
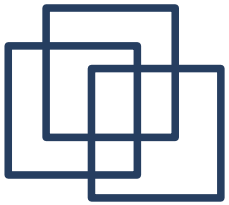


SOAP i alternatywy

1. WSDL.
2. Protokoły tekstowe
- XML-RPC.
- JSON-RPC.
- SOAPjr.
3. Protokoły binarne
- Google Protocol Buffers.
- Apache Thrift.



WSDL

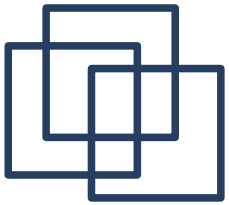
WSDL (Web Services Description Language) jest standardem opisu usług udostępnionych przez protokół SOAP. Obecna wersja protokołu (2.0) została zatwierdzona przez W3C w 2007 roku.

Do opisu usług sieciowych wykorzystywany jest format XML.

Typowy dokument składa się z kilku części:

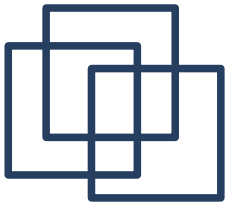
• nagłówek:

```
<?xml version="1.0" encoding="UTF-8"?>
<description xmlns="http://www.w3.org/ns/wsdl"
             xmlns:tns="http://www.tmsws.com/wsdl20sample"
             xmlns:whttp="http://schemas.xmlsoap.org/wsdl/http/"
             xmlns:wsoap="http://schemas.xmlsoap.org/wsdl/soap/"
             targetNamespace="http://www.tmsws.com/wsdl20sample">
```



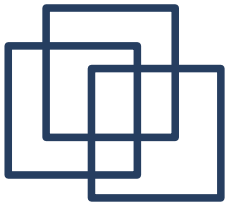
WSDL

```
<!-- Definicja typów: wywołanie metody i odpowiedź -->
<types>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
             xmlns="http://www.tmsws.com/wsdl20sample"
             targetNamespace="http://www.example.com/wsdl20sample">
  <xs:element name="request">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="header" maxOccurs="unbounded">
         <xs:complexType><xs:simpleContent>
          <xs:extension base="xs:string">
           <xs:attribute name="name" type="xs:string" use="required"/>
          </xs:extension>
         </xs:simpleContent></xs:complexType>
        </xs:element>
        <xs:element name="body" type="xs:anyType" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="method" type="xs:string" use="required"/>
      <xs:attribute name="uri" type="xs:anyURI" use="required"/>
    </xs:complexType>
  </xs:element>
```



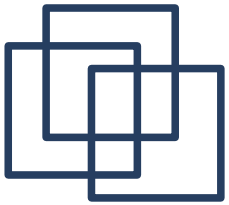
WSDL

```
    <xs:element name="response">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="header" maxOccurs="unbounded">
            <xs:complexType>
              <xs:simpleContent>
                <xs:extension base="xs:string">
                  <xs:attribute name="name"  use="required"/>
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>
          <xs:element name="body" type="xs:anyType" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="status-code" type="xs:anySimpleType"
                      use="required"/>
        <xs:attribute name="response-phrase" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:schema>
</types>
```



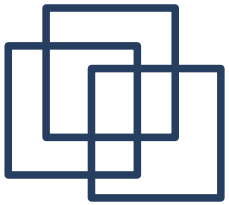
WSDL

```
<!-- Definicja interfejsu -->
<interface name="JakisInterfejs">
  <fault name="ClientError" element="tns:response"/>
  <fault name="ServerError" element="tns:response"/>
  <fault name="Redirection" element="tns:response"/>

  <operation name="Put" pattern="http://www.w3.org/ns/wsdl/in-out">
    <input messageLabel="In" element="tns:request"/>
    <output messageLabel="Out" element="tns:response"/>
  </operation>

  <operation name="Get" pattern="http://www.w3.org/ns/wsdl/in-out">
    <input messageLabel="In" element="tns:request"/>
    <output messageLabel="Out" element="tns:response"/>
  </operation>
</interface>
```



WSDL

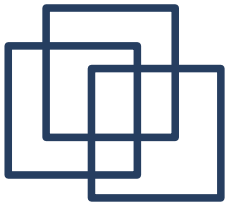
```
  <!-- definiowanie operacji udostępnianych przez SOAP -->
 <binding name="JakasUslugaSoapBinding"
          interface="tns:RESTfulInterface"
          type="http://www.w3.org/ns/wsdl/soap"
          wsoap:protocol=
               "http://www.w3.org/2003/05/soap/bindings/HTTP/"
          wsoap:mepDefault=
               "http://www.w3.org/2003/05/soap/mep/request-response">
   <operation ref="tns:Put" />
   <operation ref="tns:Get" />
 </binding>


<!-- Określenie punktu końcowego usługi sieciowej -->
 <service name="JakasUsluga" interface="tns:JakisInterfejs">
   <endpoint name="JakasUslugaSoapEndpoint"
             binding="tns:JakasUslugaSoapBinding"
             address="http://www.example.com/soap/"/>
   </service>
</description>
```



WSDL

Przedstawiony dokument definiuje usługę składającą się z dwóch zdalnych metod. Struktura przesyłanych wiadomości jest określona w ekcji <type> natomiast adres, pod który należy kierować żądania w sekcji <service>.

WSDL ułatwia pracę z usługami sieciowymi. Na podstawie opisu w WSDL'u można automatycznie tworzyć klasy/procedury łącznikowe realizujące zdalne wywołania SOAP.

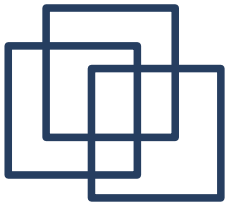


XML-RPC

XML-RPC to protokół zdalnego wywołania procedur używający XML'a do kodowania tych wywołań i protokołu HTTP do ich przesyłania. XML-RPC został opracowany w 1998 roku, a jego ewolucja doprowadziła do powstania SOAP.

XML-RPC jest prostszy w użyciu od SOAP głównie dlatego, że

- dopuszcza tylko jeden rodzaj serializacji,
- ma prostszy system zabezpieczeń (korzysta z HTTP)
- nie wymaga (ani nie wspiera) korzystania z WSDL'a. Opis usług jest możliwy poprzez XRDL, który jest znacznie prostszy.



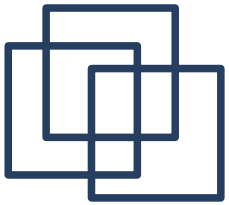
XML-RPC

Przykładowe typy podstawowe:

- **int:** `<i4>65</i4> lub <int>65</int>`
- **double:** `<double>-12.34</double>`
- **string:** `<string>Hello world!</string>`
- **boolean:** `<boolean>1</boolean> lub <boolean>0</boolean>`
- **date/time:**

```
<dateTime.iso8601>19980717T14:08:55</dateTime.iso8601>
```

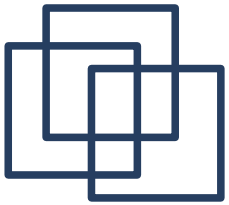
- **base64:** `<base64>SGVsbG8gV29ybGQh</base64>`
- `<nil/>`



XML-RPC

Tabele i struktury:

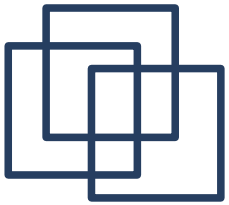
```
<array>
  <data>
    <value><i4>1234</i4></value>
    <value><string>Jakis tekst</string></value>
    <value><boolean>0</boolean></value>
  </data>
</array>

<struct>
  <member>
    <name>klucz</name>
    <value><i4>123</i4></value>
  </member>
  <member>
    <name>wartosc</name>
    <value><i4>2765</i4></value>
  </member>
</struct>
```



XML-RPC

Przykład wywołania metody:

```
<?xml version="1.0"?>
<methodCall>
  <methodName>przyklad.getName</methodName>
  <params>
    <param>
        <value><i4>1234</i4></value>
    </param>
  </params>
</methodCall>

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
        <value><string>Programowanie sieciowe</string></value>
    </param>
  </params>
</methodResponse>
```

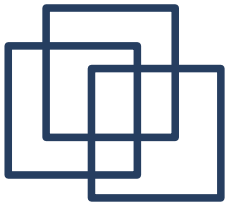


XML-RPC

Większość popularnych języków programowania posiada biblioteki do obsługi XML-RPC. Najistotniejsze zarzuty do tego standardu:

• brak możliwości przesyłania danych poza XML'em (np. załączniki w SOAP,

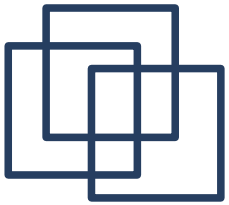
• nieefektywny format zapisu danych w porównaniu do innych technologii (np. JSON)



JSON-RPC

JSON-RPC to protokół podobny do XML-RPC jednak do transportu danych wykorzystywany jest format JSON (JavaScript Object Notation). Przykład notacji JSON:

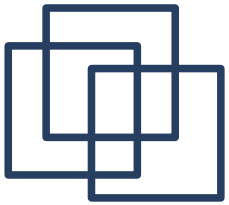
```
{
      "imię"      : "Jan",
      "nazwisko" : "Kowalski",
      "adres"     :
      {
          "ulica"          : "Reymonta 4",
      },
      "telefon":
      [
          {
            "numer": "1234567890"
          },
          {
            "number": "646 555-4567"
          }
      ]
}
```



JSON-RPC

Zgodnie z najnowszą (2010) propozycją specyfikacji (2.0) w trakcie komunikacji przekazywane są następujące, przykładowe obiekty:

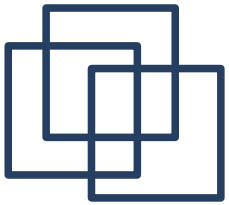
```
--> {"jsonrpc": "2.0", "method": "subtract", "params": [46, 20],
"id": 1}
<-- {"jsonrpc": "2.0", "result": 26, "id": 1}

--> {"jsonrpc": "2.0", "method": "subtract", "params": [23, 42],
"id": 2}
<-- {"jsonrpc": "2.0", "result": -19, "id": 2}

--> {"jsonrpc": "2.0", "method": "subtract", "params":
{"subtrahend": 23, "minuend": 42}, "id": 3}
<-- {"jsonrpc": "2.0", "result": 19, "id": 3}

--> {"jsonrpc": "2.0", "method": "subtract", "params":
{"minuend": 42, "subtrahend": 23}, "id": 4}
<-- {"jsonrpc": "2.0", "result": 19, "id": 4}
```



SOAPjr

SOAPjr to połączenie SOAP z JSON-RPC. Przekazywana wiadomość ma nadal strukturę charakterystyczną dla klasycznej wiadomości SOAP, jednak do jej reprezentacji, w miejsce XML'a wykorzystuje się format JSON. Przykład:

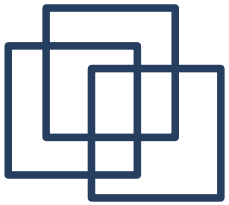
```
{
    "HEAD" : {
        "service_type" : "contacts",
        "action_type" : "view",
        "sid" : "80e5b8a8b9cbf3a79fe8d624628a0fe5"
    },
    "BODY" : {
        "username" : "jbloggs"
    }
}
```



SOAPjr

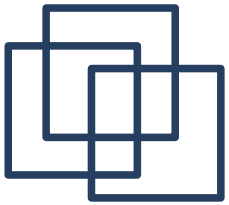
Dzięki swojej prostej strukturze SOAPjr idealnie nadaje się do dostępu do usług sieciowych w aplikacjach wykorzystujących technologię AJAX. Przykładowo, wiadomość z poprzedniego slajdu, przesłana jako argument w komendzie GET (HTTP) ma postać:

```
http://..../Service?json={"HEAD":
{"service_type":"contacts","action_type":"view",
"sid":"80e5b8a8b9cbf3a79fe8d624628a0fe5"},"BODY":
{"username":"jbloggs"}}
```



Google Protocol Buffers

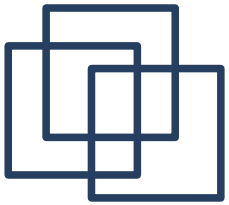
Protocol Buffers (2011) jest protokołem binarnej serializacji danych i jako taki może być podstawą do tworzenia rozwiązań z zakresu RPC. Został zaprojektowany tak, aby być wydajniejszym i szybszym od XML'a. Podobnie jak inne, analogiczne technologie (CORBA), korzysta z języka opisu interfejsów IDL i na jego podstawie tworzy kod źródłowy operujący na transferowanych danych. Obecnie Protocol Buffer jest dostępny (opensource) dla języków C++, Java i Python.



Google Protocol Buffers

Przykład (zapis w IDL):

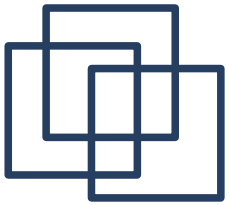
```
message Point {
  required int32 x = 1;
  required int32 y = 2;
  optional string label = 3;
}

message Line {
  required Point start = 1;
  required Point end = 2;
  optional string label = 3;
}

message Polyline {
  repeated Point point = 1;
  optional string label = 2;
}
```



Google Protocol Buffers

Przykład operowania na danych (C++):

```
#include "polyline.pb.h" // wygenerowane z interfejsu
Line* createNewLine(const std::string& name) {
  Line* line = new Line;
  line->mutable_start()->set_x(10);
  line->mutable_start()->set_y(20);
  line->mutable_end()->set_x(30);
  line->mutable_end()->set_y(40);
  line->set_label(name);
  return line;
}
Polyline* createNewPolyline() {
  Polyline* polyline = new Polyline;
  Point* point1 = polyline->add_point();
  point1->set_x(10);
  point1->set_y(10);
  Point* point2 = polyline->add_point();
  point2->set_x(10);
  point2->set_y(10);
  return polyline;
}
```

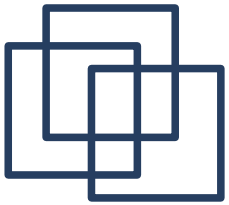


Apache Thrift

Thrift jest podobnym rozwiązaniem do Protocol Buffers. Jego zaletą jest wsparcie dla większej liczby języków programowania (w tym języków skryptowych jak np. PHP, JavaScript, Ruby, Perl).

Przykład (Interfejs):

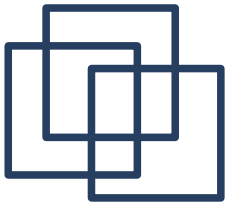
```
struct UserProfile {
        1: i32 uid,
        2: string name,
        3: string blurb
      }

      service UserStorage {
        void store(1: UserProfile user),
        UserProfile retrieve(1: i32 uid)
      }
```



Apache Thrift

Przykład (klient w Pythonie):

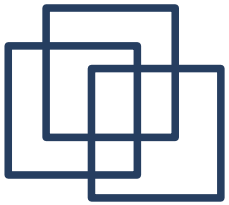
```
# Steworzenie obiektu
      up = UserProfile(uid=1,
                       name="Test User",
                       blurb="Thrift is great")

      # Nawiazanie polaczenia
      transport = TSocket.TSocket("localhost", 9090)
      transport.open()
      protocol = TBinaryProtocol.TBinaryProtocol(transport)

      # korzystanie z uslugi - wysylanie
      service = UserStorage.Client(protocol)
      service.store(up)

      # korzystanie z uslugi - odbieranie
      up2 = service.retrieve(2)
```



Apache Thrift

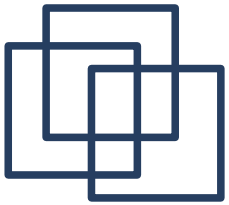
Przykład (serwer w C++):

```
class UserStorageHandler : virtual public UserStorageIf {
        public:
         UserStorageHandler() {
           // inicjalizacja
         }

         void store(const UserProfile& user) {
           // implementacja
           printf("store\n");
         }

         void retrieve(UserProfile& _return, const int32_t uid) {
           // implementacja
           printf("retrieve\n");
         }
       };
```



Apache Thrift

```
int main(int argc, char **argv) {
  int port = 9090;
  shared_ptr handler(new UserStorageHandler());
  shared_ptr processor(new UserStorageProcessor(handler));
  shared_ptr serverTransport(new TServerSocket(port));
  shared_ptr transportFactory(
                              new TBufferedTransportFactory());
  shared_ptr protocolFactory(new TBinaryProtocolFactory());
  TSimpleServer server(processor, serverTransport,
                           transportFactory, protocolFactory);
  server.serve();
  return 0;
}
```



Podsumowanie

Pomimo, że SOAP jest obecnie najpopularniejszym protokołem do realizacji usług sieciowych, istnieje szereg innych, nowszych technologii. Niektóre z nich mają na celu zastąpienie SOAP, inne natomiast pozwalają efektywniej wykorzystać możliwości związane z programowaniem rozproszonym.