**39. Systemy pośredniczące w wymianie danych. / 31. Narzędzia integracji SI**

***Spis treści:***

1. ***Świat przed Web Servicami:***
   1. ***EDI***
   2. ***RPC***
      1. ***Java RMI***
      2. ***Xml RPC***
      3. ***JSON RPC***
      4. ***CORBA***
      5. ***DCOM***
2. ***Web Service’y:***
   1. ***XML, JSON***
   2. ***SOAP, WSDL, UDDI***
   3. ***Mashupy***
   4. ***Szyna ESB***

**EDI**

* **EDI** to skrót pochodzący od angielskiej nazwy *Electronic Data Interchange* oznaczający w bezpośrednim tłumaczeniu Elektroniczną Wymianę Danych.
* **EDI** to wymiana danych w **formatach** opisanych **międzynarodowymi standardami**, między systemami informatycznymi partnerów handlowych, przy minimalnej interwencji człowieka.
* **EDI** łączy możliwości informatyki i telekomunikacji. Umożliwia **eliminację** dokumentów papierowych zwiększając efektywność wszystkich działań związanych z handlem.
* **EDI** jest najprostszym sposobem **realizacji transakcji handlowych** z pominięciem żmudnej pracy przy tworzeniu, kopiowaniu i przesyłaniu dokumentów papierowych. EDI tworzy pomost, który łączy bezpośrednio systemy informatyczne współpracujących ze sobą firm.



* Celem EDI jest **wyeliminowanie** wielokrotnego **wprowadzania danych** oraz przyspieszenie i zwiększenie dokładności przepływu informacji dzięki połączeniu odpowiednich aplikacji komputerowych   
  w firmach uczestniczących w wymianie.
* Użycie EDI pozwala poprawić czasową dostępność informacji logistycznej, poszerzyć i uściślić dane, a także zmniejszyć pracochłonność procesu.
* Aby w pełni wykorzystać zalety EDI, uczestnicy kanału logistycznego powinni się komunikować za pośrednictwem komputera. Innymi słowy, efektywne wdrożenie EDI wymaga bezpośredniej komunikacji między systemami komputerowymi zarówno nabywców, jak i sprzedawców produktu..
* **EDI** umożliwia natychmiastowe przekazywanie informacji, które są zawarte w typowych dokumentach handlowych. Zastosowanie standardowych i akceptowanych na całym świecie formatów danych zapewnia, że wszyscy uczestnicy wymiany używają tego samego języka.
* **Dokument EDI** jest **odpowiednikiem papierowego dokumentu** handlowego o ustalonej międzynarodowej postaci, który został przystosowany do celów elektronicznej transmisji danych.
* **EDI** funkcjonuje **niezależnie od rodzaju oprogramowania** użytkownika. Stosowanie EDI nie jest ograniczone różnicami w oprogramowaniu, jakie partnerzy handlowi używają w swoich przedsiębiorstwach.



* Standardy EDI **określają ściśle format** przesyłanych dokumentów.
* EDI **nie określa sposobu przesyłania komunikatów** – mogą one być przesyłane przez dowolne medium, którym posługują się obie strony transmisji. Może to być transmisja modemowa, poprzez FTP, HTTP, AS1, AS2.
* Ujednolicenie formatu przesyłanych informacji i danych jest najważniejszym i niezbędnym czynnikiem warunkującym rozwój i korzystanie z EDI.
  1. Partnerzy wykorzystują do tego celu jednakowe standardy i oprogramowanie umożliwiające ich tworzenie.
  2. Handel elektroniczny ma charakter globalny, niezbędna jest więc standaryzacja o zasięgu międzynarodowym.
  3. W ramach różnych organizacji międzynarodowych (UE, WTO, OECD itd.) powołano specjalne komórki zajmujące się nadzorowaniem EDI oraz jej standaryzacją.
* **Standardy EDI.**
  1. EDIFACT - (popierany przez UE i ONZ) - standard międzynarodowy
  2. ANSI X12 - standard amerykański
* Generacja **I**
  1. stosowane głównie własne rozwiązania oprogramowania
  2. transmisja przez kosztowne sieci prywatne,
* Generacja **II**
  1. postępująca standaryzacja: EDIFACT, ANSI X12
  2. sieci VAN (*Value-Added Network*) - medium transportowe umożliwiające translację między różnymi systemami i typami komputerów
  3. lepsza i tańsza transmisja danych
* Generacja **III**
  1. wykorzystanie sieci Internet do transmisji (najtańsze medium, zasięg ogólnoświatowy).
* Oszczędności - spadek kosztów własnych ponoszonych na łączności i wymianie informacji między firmami.

**Korzyści z EDI**

* Wzrost efektywności:
  1. system produkcji JIT (*just-in-time*) - dostarczanie części do produkcji wprost na linie produkcyjne.
  2. system precyzyjnego zaopatrzenia i dostaw - *Lean Supply* - realizacja dostaw na żądanie.
* Uzyskanie przewagi nad konkurencją.
* Szybsza reakcja na zmiany i potrzeby.
* Wprowadzenie EDI w węższym zakresie (jedynie przesyłanie dokumentów i informacji) nie wymaga żadnych zmian ani w strukturze organizacyjnej firmy, ani w metodach jej działania.
* Znaczenie symboliczne - korzystanie z EDI jest wyrazem zaufania do partnera handlowego i chęci długoletniej współpracy.

**Bariery stosowania EDI**

* **Zastosowanie EDI w wymianie handlowej między małą lub średnią firmą a dużą przynosiło na ogół korzyści tylko tej ostatniej.**
* **Mała (średnia) firma zmuszona była, w celu osiągnięcia korzyści ze stosowania EDI, do zintegrowania swojego systemu komputerowego z aplikacją EDI, co było zadaniem kosztownym.**
* **Małe firmy ograniczały często używanie komputera do prowadzenia tylko własnej bazy danych, albo nawet tylko jako maszyny do pisania.**
* **Tak więc mała firma, uczestnicząc w scenariuszu Zamówienie - Faktura, otrzymuje zamówienie drogą elektroniczną (e-mail), drukuje je i być może nawet przepisuje.**
* **Po otrzymaniu faktury informacja jest drukowana i wprowadzana do systemu komputerowego.**
* **Z tych powodów jedynie około 10% firm było zainteresowanych stosowaniem EDI w wymianie handlowej.**
* Zasady tworzenia komunikatów EDI określa międzynarodowy standard UN/EDIFACT – United Nations Rules for Electronic Data Interchange for Administration, Commerce and Transport, czyli zasady ONZ dla elektronicznej wymiany danych w administracji, handlu i transporcie, które zostały uregulowane normą ISO 9735 w 1987 r. Komunikaty EDIFACT są elektronicznymi odpowiednikami dokumentów papierowych.
* Komunikaty EDIFACT są bardzo złożone, przeładowane z punktu widzenia funkcjonalności oraz trudne do zrozumienia i wdrażania. W praktyce każdy komunikat EDIFACT musi być wspierany poradnikiem wdrażania dla użytkownika, takim jak np. EANCOM.

Przykład komunikatu EANCOM

* Faktura wysłana przez dostawcę, określonego kodem lokalizacyjnym EAN 5454321000010 oraz numerem VAT 41012345894, do kupującego o kodzie lokalizacyjnym 5412345000010 i numerze VAT 427247863.
* Faktura o numerze IN112233 jest wysłana 8.10.1997 o godz. 10.25 rano. Dotyczy ona towarów zamawianych dnia 2.10.1997 zgodnie z zamówieniem nr PO112233 oraz cennikiem z dnia 01.10.1997. Walutą są franki belgijskie. Warunki płatności to dwa miesiące i dodatkowe skonto 1.8% przy otrzymaniu zapłaty w ciągu 10 dni od daty faktury. Zapłata dokonywana jest na konto bankowe dostawcy.
* Faktura zawiera dwie pozycje:
* Itd..

|  |  |
| --- | --- |
| **UNH+ME0001+INVOIC:2:901:UN:EAN005’** | nagłówek komunikatu, numer wymiany |
| **BGM+380+IN112233+971008:1025’** | nazwa komunikatu, data |
| **REF+PO+PO112233+971002’** | numer i data zamówienia |
| **REF+PL+PL123+971001’** | numer i data cennika |
| **NAD+BY+5412345000010:14’** | kupujący |
| **REF+VA+427247863’** | numer VAT |
| **NAD+SU+5454321000010:14’** | dostawca |
| **REF+VA+410125894’** | numer VAT |
| **DTM+011+971004’** | data wysyłki |

**Remote Procedure Call** (**RPC** - zdalne wywołanie procedury) to [protokół](http://pl.wikipedia.org/wiki/Protoko%C5%82y_komunikacyjne) zdalnego wywoływania procedur, stworzony przez firmę [Sun](http://pl.wikipedia.org/wiki/Sun_Microsystems) i swego czasu dość popularny na [Uniksach](http://pl.wikipedia.org/wiki/Unix), obsługiwany w bibliotekach języka [Java](http://pl.wikipedia.org/wiki/Java). W czasach, gdy w Internecie królować zaczęła coraz większa liczba oficjalnych standardów, architektura RPC okazała się niewystarczająca. Niemniej jednak na przestrzeni lat powstało wiele implementacji tego standardu. (CORBA, RMI, DCOM, XML-RPC – później SOAP).

W modelu tym (RPC) programiści znali **jedynie interfejs** zdalnej metody, którą chcieli wywołać. Z ich punktu widzenia wywołanie takiej metody nie różniło się od wywołania metody lokalnej.

Podejście to miało jednak poważną wadę. Otóż było uzależnione od zastosowanego języka programowania. Oznacza to, że przykładowo klient posiadając aplikację napisaną w języku JAVA mógł wywoływać jedynie te zdalne metody, które zostały napisane w tym samym języku.

Sun’owski, pierwotny RPC zakłada, że:

* Dane będą zapisane w języku XDR,
* Komunikaty są wysyłane przez protokół TCP lub UDP,
* Klient musi znać IP serwera,
* Program rezydentny cały czas działa zarówno u klienta jak i na serwerze. Przy połączeniu informuje klienta o porcie, na który trzeba wysyłać dane.
* **Zaleta: prosty interfejs**
* **Wady: niskopoziomowa konstrukcja nieprzyjazna dla Firewalli (XML RPC jest wyjątkiem)**

Protokoły tego typu (jak RPC, CORBA, [DCOM](http://pl.wikipedia.org/wiki/Distributed_Component_Object_Model), czy XML-RPC) mają na celu ułatwienie komunikacji pomiędzy komputerami. Na typowy scenariusz użytkowania składają się:

* [Serwer](http://pl.wikipedia.org/wiki/Serwer) (czyli program oferujący usługi, np. drukowania) przez cały czas nasłuchuje na wybranym porcie, czy ktoś się z nim nie łączy.
* [Klient](http://pl.wikipedia.org/wiki/Klient_%28informatyka%29) (czyli program który potrzebuje jakiejś usługi od serwera na innym komputerze) nawiązuje z nim łączność poprzez [sieć komputerową](http://pl.wikipedia.org/wiki/Sie%C4%87_komputerowa).
* Klient wysyła swoje dane we wcześniej ustalonym przez programistów klienta i serwera formacie.
* Serwer realizuje usługę i odsyła potwierdzenie lub kod błędu.

Powyższe protokoły same zapewniają cały powyższy mechanizm, ukrywając go przed klientem. Może on nawet nie wiedzieć, że łączy się z innym komputerem - z punktu widzenia programisty zdalne wywołanie procedury serwera wygląda jak wywołanie dowolnej innej [procedury](http://pl.wikipedia.org/wiki/Podprogram) z programu klienta.

**Java RMI**

Implementacja RPC, działa na maszynach wirtualnych Javy.

**RMI** ([ang.](http://pl.wikipedia.org/wiki/J%C4%99zyk_angielski) *Remote Method Invocation* - zdalne wywołanie metod) to mechanizm umożliwiający zdalne wywołanie metod obiektów. Obiekty te mogą znajdować się w innych maszynach wirtualnych Javy, które mogą znajdować się na innych [komputerach](http://pl.wikipedia.org/wiki/Komputer).

**Zasada działania:**

Obiekty zdalne rejestrowane są pod wybranymi nazwami w serwisie RMI Registry. Aplikacja kliencka ściąga z RMI Registry tzw. stub tego obiektu, który umożliwia komunikację z obiektem zdalnym przy użyciu wyeksportowanych metod w ten sam sposób, jakby chodziło o obiekt lokalny. Rola RMI Registry w tym miejscu kończy się - nie pośredniczy on w komunikacji pomiędzy aplikacją kliencką a obiektem zdalnym. Parametry metod będące obiektami przy wywołaniu zdalnym są [serializowane](http://pl.wikipedia.org/wiki/Serializacja).

**XML-RPC**

Protokół RPC, który używa XMLa do przesyłania żądań. Przesył odbywa się poprzez **protokół HTTP**.

XML-RPC jest szeroko wspierany w takich językach jak Java, C++, Erlang, JavaScript (node.js) i inne.

|  |  |
| --- | --- |
| Przykładowe żądanie (wywołanie procedury) | Przykładowa odpowiedź |
| <?xml version="1.0"?>  <methodCall> <methodName>examples.getStateName</methodName>  <params>  <param>  <value><i4>40</i4></value>  </param>  </params>  </methodCall> | <?xml version="1.0"?>  <methodResponse>  <params>  <param>  <value><string>South Dakota</string></value>  </param>  </params>  </methodResponse> |

**JSON-RPC**

Praktycznie to samo co XML-RPC, tylko że używa JSONa. Gdy przesył odbywa się przez HTTP, pole content-type zawiera ‘application/json’.

**CORBA**

**Otwarty standard** zbudowany na bazie RPC przez organizację OMG

**Niezależny** od języka programowania.

**Niezależny** od platformy

**Skalowalny**

**DCOM**

Odpowiedź Microsoftu na CORBĘ.

**Web service**  
„Web Service to oprogramowanie stworzone w celu wspierania sieciowej współpracy między systemami informatycznymi. Posiada interfejs opisany „maszynowym” formatem (np. WSDL). Inne systemy współpracują z Web Service w sposób opisany w jego dokumentacji, z wykorzystaniem wiadomości SOAP, najczęściej transportowanymi przez protokół HTTP z wykorzystaniem XML oraz w połączeniu z innymi sieciowymi standardami.

Usługa internetowa jest w istocie składnikiem oprogramowania, niezależnym od platformy sprzętowej oraz implementacji, dostarczającym określonej funkcjonalności.

Ten rodzaj usług okazał się skuteczny w sieciach korporacyjnych, za pomocą których przedsiębiorstwa lub instytucje, budowały systemy wymiany danych między swoimi oddziałami, jak również do celów łączności z partnerami i klientami. W takich mniejszych, dobrze kontrolowanych środowiskach, łatwiej jest uzyskać zgodność danych przesyłanych między poszczególnymi komponentami usług a otwartość standardów ułatwia tworzenie rozwiązań klienckich, niezależnie od platformy. Wykorzystanie usług internetowych pozwala komponentom programowym współdziałać ze sobą przez Internet, niezależnie od swojej lokalizacji i szczegółów implementacji. Dzięki temu będą w stanie zastąpić starsze rozwiązania, opracowane dla sieci prywatnych, jak CORBA czy DCOM, zaś dzięki stosunkowo prostej konstrukcji, mogą uzyskać znacznie większą popularność.

Pierwszą oraz jedną z ważniejszych jest odejście od podejścia aplikacyjnego i zastąpienie go podejściem usługowym. Oznacza to, że po stworzeniu usługi sieciowej, nie trzeba jej instalować na wszystkich stanowiskach, na których będzie ona wykorzystywana. Do uruchomienia tego typu usługi zazwyczaj wystarcza jeden serwer, który posiada zainstalowaną usługę na wybranym porcie oraz potrafi komunikować się z otoczeniem poprzez standardowy protokół HTTP.

Heterogeniczność środowiska pozwala na wykorzystanie innego typu procesora czy innego systemu operacyjnego niż ten, który zarządza innymi usługami sieciowymi.

Niewielkimi wadami użycia web service’ów może być niewielki spadek wydajności w porównaniu ze scentralizowaną architekturą. Konieczność komunikacji poszczególnych usług sieciowych poprzez protokół HTTP wpływa również na nadmiarowość danych generowanych oraz przetwarzanych przez aplikacje. Co za tym idzie, wydłuża się czas przesłania danych z jednego modułu do drugiego. Czasem wadą może być również bezstanowość, która wynika bezpośrednio z użycia bezstanowego protokołu HTTP.

**XML i JSON**

Usługi sieciowe wykorzystują do komunikacji ustandaryzowane dane tak, by możliwa była ich wymiana bez potrzeby implementowania wielu metod kodujących i dekodujących te dane. Najpowszechniejszymi formatami danych używanymi w technologii web service są XML oraz JSON. W poniższej tabeli zaprezentowany jest przykład danych zapisanych w obu formatach:

|  |  |
| --- | --- |
| XML | JSON |
| <?xml version="1.0" encoding="UTF-8"?>  <newOrderSingle type="8">  <accountNumber>55-55</accountNumber>  <expireDate>2011-08-16</expireDate>  <orderQty>10.0</orderQty>  <price>120.5</price>  <currency>PLN</currency>  <side>1</side>  <symbol>  <shortName>COMARCH</shortName>  <shortName>CMR</shortName>  </symbol>  </newOrderSingle> | {  newordersingle:{  type:8,  accountnumber: 55-55',  expiredate:'2011-08-16',  orderqty:10,  price:120.5,  currency:'PLN',  side:1,  symbol:{  shortname:[  'COMARCH',  'CMR'  ]  }  }  } |

Format XML używa do opisu znaczników znanych z języka HTML. Początki języka XML sięgają jednak SGMLa, który został znacznie uproszczony tak, by przetwarzanie przez parsery XML odbywało się szybciej. W pierwszej linii XML zawiera deklarację wersji oraz kodowania. Obecnie istnieją dwie wersje XML-a: 1.0 oraz 1.1. Definicja kodowania znaków pozwala poprawnie interpretować zapisane znaki. W kolejnych liniach zapisana jest właściwa treść dokumentu. Każdy element posiada znacznik otwierający jak i zamykający (np. odpowiednio: <price> oraz </price>). Każdy element może posiadać dowolną liczbę atrybutów, które są oddzielone spacją obok nazwy elementu. W powyższym przekładzie atrybutem jest type, natomiast wartość (w tym wypadku - liczbowa) wynosi 8. Dany element może zawierać kilka podelementów o identycznych nazwach. Tworzona jest wtedy swego rodzaju tablica. W powyższym przykładzie tablicę tą stanowią elementy o nazwie <shortName>. Pliki XML mogą być z powodzeniem konwertowane do innej postaci XML-a bądź dowolnego innego typu plików przy użyciu transformacji XSLT. Najczęściej używane jest to do prezentacji danych na stronie internetowej. Ciekawym narzędziem są również schematy XML Schema, które pozwalają walidować dany dokument XML według podanego w schemacie wzorca. W schemacie można określać nie tylko, że dany element musi przechowywać wartość tekstową czy liczbową, ale także zakres wartości czy maksymalną długość przechowywanego ciągu znaków.

Format JSON zorientowany jest na prostotę. Twórcy tego formatu uznali, że zamykające znaczniki są nadmiarowe, dlatego zrezygnowano z nich. Początek i koniec danego elementu oznaczone są przy użyciu nawiasów klamrowych. Przed nawiasem otwierającym znajduje się dwukropek, natomiast przed dwukropkiem nazwa elementu. Tablice rozpoczyna się nawiasem kwadratowym a poszczególne elementy oddziela przecinkiem. Warto zauważyć, że poszczególne elementy nie posiadają żadnych atrybutów. Konwertując dane z XML-a do JSON-a, wszelkie atrybuty elementu stają się jego podelementami. JSON wspierany jest przez wiele języków programowania. W JavaScript jest on obsługiwany natywnie bez potrzeby korzystania z dodatkowych bibliotek. Bardzo częstym zastosowaniem JSON-a jest serializacja obiektów. Kiedy chcemy zapisać obiekt będący instancją danej klasy programistycznej, kompilator często wręcz bez naszej wiedzy dokonuje serializacji do JSON-a a następnie zapisuje go do pliku bądź bazy danych. Istnieją również odmiany tego formatu, które mają większe możliwości niż prosty JSON. Przykładowo JsonML potrafi rozróżniać elementy od atrybutów, jednak rozmiar pliku wygenerowanego w tym formacie znacząco wzrasta. Ponadto obsługa tego języka nie jest już tak prosta w wielu językach programowania jak obsługa czystego JSON-a.

**Analiza porównawcza wad i zalet XML i JSON.**

Różnice omówionych standardów zostały przedstawione w poniższej tabeli:

|  |  |  |
| --- | --- | --- |
|  | **XML** | **JSON** |
| **Kodowanie znaków** | Zapisane w deklaracji  (1. linia) | Określane ręcznie lub odczytywane z nagłówka HTTP (*Content-Encoding*) |
| **Objętość** | Nadmiarowość: 312B | Prostota: 159B |
| **Możliwości** | Bardziej złożone struktury | Proste struktury zapisu |
| **Walidacja** | Poprzez procesor XML Schema | Mało popularny JSON Schema |

Korzystając z formatu JSON pewnym problemem może być określanie kodowania znaków użytych w dokumencie. O ile XML jasno i wyraźnie ma zapisane kodowanie w swym nagłówku, tak JSON musi posiłkować się innymi, nieco mniej skutecznymi sposobami. Niekiedy kodowanie zapisuje w elemencie stworzonym specjalnie do tego celu. Można również polegać na wartości zapisanej w nagłówku wiadomości HTTP (*Content Encoding*), jednak praktyka pokazuje, że niewiele usług sieciowych ustawia prawidłowe wartości w tym nagłówku.

Zdecydowaną zaletą JSON-a w porównaniu z XML-em jest bez wątpienia prostota, która prowadzi do znacznego zmniejszenia rozmiaru dokumentu zapisanego w formacie JSON. Przykładowo informacje zapisane w poprzednim rozdziale w formacie XML zajmują 312 B, natomiast te zapisane w formacie JSON jedynie 159 B. Dla niektórych prostota może być jednak wadą, ponieważ struktura formatu JSON jest uboższa. Naszym zdaniem w większości przypadków JSON jest zupełnie wystarczający, a korzyści przewyższają pewne ograniczenia tego formatu. Do niedawna największą zaletą XML-a w stosunku do JSON-a było istnienie wielu różnych walidatorów plików XML korzystających ze schematów XML Schema. Ostatnio powstaje jednak coraz więcej narzędzi dokonujących podobną walidację przy użyciu schematów JSON Schema.

W użyciu istnieje również wiele innych formatów innych niż XML, JSON oraz ich pochodne. Poniższy wykres pokazuje udział poszczególnych formatów na rynku usług sieciowych (dane pochodzą z końca 2011 roku). Widać wyraźnie, że znakomita większość usług sieciowych korzysta z XML-a lub JSON-a. Pozostałe formaty danych wykorzystywane są w pewnych specjalistycznych zastosowaniach, które jednak ze względu na małą popularność nie będę tu szerzej omawiane.

Warto również wspomnieć, że format JSON zdobywa coraz większą popularność a jego udział za 1-2 lata powinien być największy w środowisku usług sieciowych.

**Złote zasady przy projektowaniu WebService’ów:**

- Używaj standardowych kodów odpowiedzi HTTP

**- Używaj przyjaznych kodów, skrótów/Nie używaj nic nieznaczących oznaczeń.**

**- Np.: s (zamiast 1) - sprzedaż, k (zamiast 2) - kupno**

**- Pozwól ograniczyć rozmiar odpowiedzi**

**- Utrzymuj zawsze aktualną dokumentację**

- Udostępniaj przykładowe zapytania do WebService’u

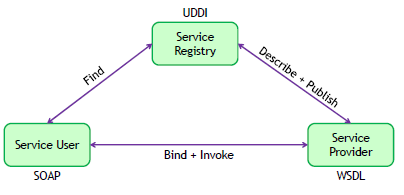
- Informuj o nowościach/zmianach

**SOAP, WSDL, UDDI**

*SOAP* jest w istocie protokołem opartym na *XML*, który służy do wywoływania usług sieciowych. Wspiera wywołania synchroniczne i asynchroniczne oraz może być wykorzystywany wraz z protokołem *HTTP*, bądź innym. W jego skład wchodzą trzy najważniejsze elementy:

* Koperta (ang. *envelope*) określającej co znajduje się wewnątrz komunikatu
* Zbiór reguł do rozszyfrowywania danych
* Reguły dotyczące wywoływania zdalnych metod

Powyższe składniki są częścią oficjalnej specyfikacji protokołu. Natomiast strukturę przesyłanych komunikatów tworzą dwa typy elementów: wymagane i opcjonalne. Elementami wymaganymi są *Envelope*, który stanowi opakowanie wszystkich pozostałych elementów oraz *Body* będący kontenerem na podstawową zawartość wiadomości. Elementami opcjonalnymi są *Header*, w którym umieszczone są metadane oraz *Fault*, gdzie znajdują się informacje dotyczące obsługi wyjątków i błędów.



Poniżej przedstawiony został przykład żądania protokołu *SOAP* korzystającego z metody *POST* protokołu *HTTP.* Oprócz standardowego nagłówka żądania *POST* oraz deklaracji *XML* w listingu znajduje się wywołanie metody *GetStockPrice*, która jest wywoływana z parametrem *StockName* o wartości *IBM.*

POST /InStock HTTP/1.1

Host: www.example.org

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

<?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 xmlns:m="http://www.example.org/stock">

<m:GetStockPrice>

<m:StockName>IBM</m:StockName>

</m:GetStockPrice>

</soap:Body>

</soap:Envelope>

W kolejnym przykładzie przedstawiono odpowiedź na powyższe żądanie. Zostało ono również przesłane z wykorzystaniem metody *POST*, stąd charakterystyczny nagłówek typowy dla tego typu wiadomości.

HTTP/1.1 200 OK

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

<?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 xmlns:m="http://www.example.org/stock">

<m:GetStockPriceResponse>

<m:Price>34.5</m:Price>

</m:GetStockPriceResponse>

</soap:Body>

</soap:Envelope>

Przedstawiona odpowiedź *SOAP* zawiera element *GetStockPriceResponse*, który jest wiadomością zawierająca parametr *Price* o wartości 34.5. Ten prosty przykład obrazuje podstawowe możliwości protokołu *SOAP.*

*WSDL* jest opartym również na *XML* słownikiem służącym do opisywania usług sieciowych. Działa on w oparciu o dobrze znaną w świecie informatycznym zasadę kontraktu pomiędzy twórcą a odbiorcą – zwanym konsumentem – usługi. Jest podobnie jak pozostałe standardy definiowany w oparciu o *XML*. Protokół *WSDL* określa zadania realizowane przez usługę oraz precyzuje gdzie można ją odnaleźć i w jaki sposób można ją uruchomić i z niej korzystać, aby uzyskać poprawne wyniki.

Dokumenty *WSDL* mają prostą strukturę opartą na czterech najważniejszych elementach:

<types> - Typy danych używanych przez WS

<message> - Wykorzystywane wiadomości

<portType> - Operacje wykonywane przez WS

<binding> - Obsługiwane protokoły komunikacyjne

Zgodnie ze specyfikacją najważniejszym elementem jest <portType> określający operacje wykonywane przez usługę sieciową. Gdyby odnieść ten element do standardowych języków programowania, można by go było porównać do biblioteki funkcji.

Kolejnym elementem jest <types>, który określa jakie typy danych są wykorzystywane w operacjach świadczonych przez usługę. Jest on jest standardowo specyfikowany w *XML Schema.*

Element <message> charakteryzuje wykorzystywane przez usługę wiadomości. Opisuje on elementy danych. Należy zauważyć, że każda operacja może składać się z wielu elementów. Analogicznie do poprzedniego przypadku odnosząc ten element do standardowych języków programowania, można go uznać za odpowiednik parametrów wywołania funkcji.

Poniżej przedstawiony został przykład wiadomości *WSDL.*

<message name="daneAkcji">

<part name="nazwa" type="xs:string"/>

<part name="ilosc" type="xs:integer"/>

</message>

<portType name="gielda">

<operation name="sprzedajAkcje">

<input name="akcjeDoSprzedania" message="daneAkcji"/>

</operation>

</portType>

W przykładowej wiadomości zdefiniowany został port o nazwie „gielda”, w którym zdefiniowano operację „sprzedajAkcje”. Operacja ta przyjmuje dane akcji do sprzedania w postaci wiadomości „daneAkcji” z parametrami „nazwa” oraz „ilość”. Dane akcji zostały zdefiniowane w elemencie <message> w pierwszej części wiadomości. Operacja „sprzedajAkcje” nie zwraca wyniku. Gdyby było inaczej, obok elementu <input> pojawiłby się również element <output> zdefiniowany w analogiczny sposób.

*UDDI* to rejestr służący do odnajdywania usług w sieci. Odpowiada on na pytanie jak możemy odnaleźć odpowiednią usługę sieciową spełniającą odpowiednie wymagania. Rejestr *UDDI* jest katalogiem zawierającym informacje o usługach sieciowych. Zatem jest katalogiem interfejsów usług zapisanych w *WSDL.* Został oparty na *XML*. Wykorzystuje również protokół *SOAP* do transmisji żądań. Dzięki temu możliwe jest korzystanie ze skomplikowanego API rejestru. Warto nadmienić, że istnieje oddzielne API dla programistów chcących: uzyskać informacje dotyczące usług: *Inquire,* publikować informacje: *Publishers,* otrzymać informacje o bezpieczeństwie: *Security,* być informowanymi o zmianach w rejestrze: *Subsciption,* zrealizować operacje replikacji rejestru: *Replication*.

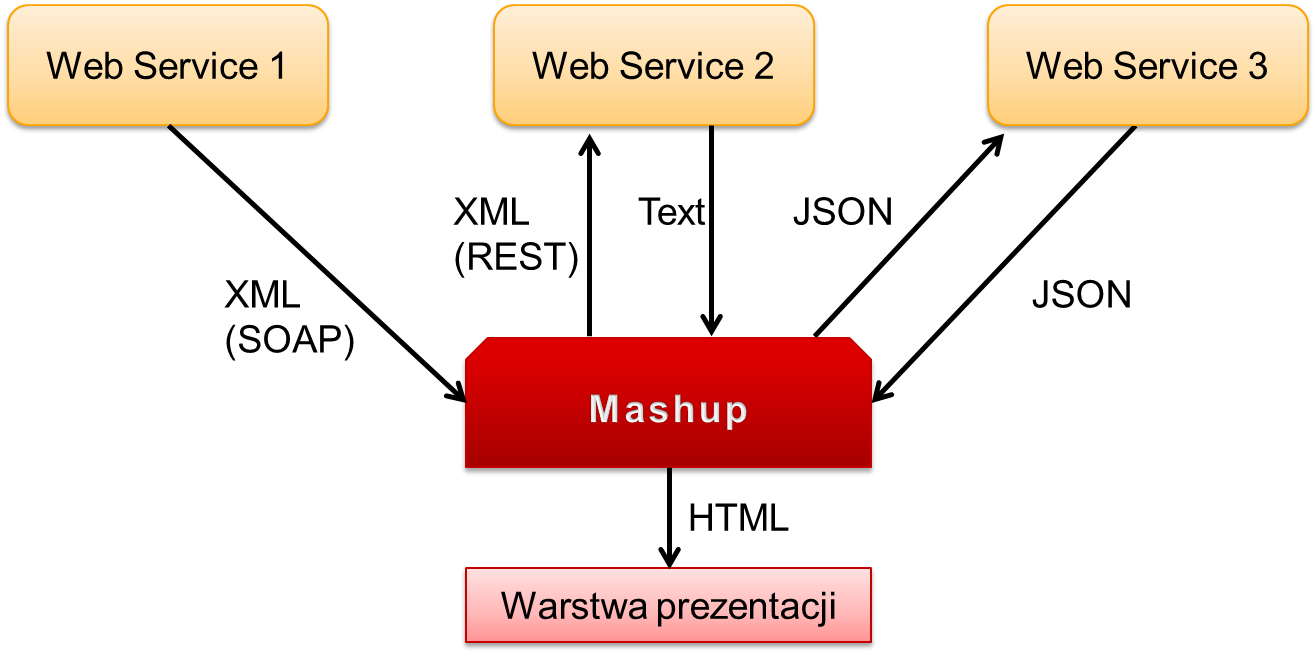
Przez skomplikowane *API* oraz złożoną budowę rejestru zdecydowaliśmy się przedstawić jedynie najważniejsze informacje o jego strukturze, pomijając drobiazgowe szczegóły architektoniczne.

Rejestr *UDDI* jest podzielony na trzy najważniejsze części: *white pages, yellow pages* oraz *green pages.* Pierwsza z części – *white pages* – zawiera opublikowane adresy kontaktowe firm oraz podstawowe informacje o nich. Można w niej również odszukać usługi świadczone przez podobne firmy. *Yellow pages* zawiera klasyfikacje przemysłowe . Usługi są podzielone według kategorii. W ostatniej części – *green pages* – umieszczane są opisy usług i sposobu ich uruchamiania. Znajdują się tutaj również odnośniki do dokumentacji i standardów znajdujących się na zewnętrznych serwerach. Warto zaznaczyć, że już ponad 220 największych firm informatycznych świata jest członkami społeczności *UDDI.*

soapUI – Narzędzie, którym można odpytywać wybrane Web service’y.

**MASHUPY:**

Mashup’y to aplikacje internetowe, które łączą dane pochodzące z co najmniej dwóch różnych web service’ów. Najczęściej prezentowane są one w postaci stron internetowych, na których wypełniamy krótki formularz, a następnie wyświetlone zostają dane pochodzące z kilku różnych web service’ów. Architektura usług sieciowych pozwala całkowicie pominąć warstwy prezentacji poszczególnych usług, zebrać je w jedno miejsce, przetworzyć oraz na koniec wyświetlić je na jednej stronie w wygodnej w użyciu postaci. Poniższy schemat ilustruje zasadę działania oraz zbierania informacji przykładowego mashup’a.



Jak widać na powyższym schemacie, format danych oraz technika dostępu nie ma żadnego znaczenia, ponieważ architektura usług sieciowych pozwala na wykorzystanie wielu różnych źródeł i połączenie ich w jedną całość.

Mashup’y zazwyczaj tworzone są w celu wizualizacji pewnych informacji bądź konwersji danych podawanych przez jedną usługę do innego formatu. Przykładowym zastosowaniem może być przeliczanie wartości walutowych przy użyciu danych pochodzących z usługi sieciowej kantoru.

**Szyny Integracyjne**

Korporacyjna Magistrala Usług (ang. Enterprise Service Bus) - dodatkowa warstwa pośrednia w wielowarstwowej architekturze systemów informatycznych umożliwiająca zastosowanie koncepcji SOA (Architektura zorientowana na usługi) w środowisku korporacyjnym. Umożliwia dynamiczne przyłączanie i odłączanie usług wchodzących w skład korporacyjnego systemu informacyjnego.

ESB (Enterprise Service Buses) - architektura oprogramowania używana do projektowania i implementowania interakcji i komunikacji między aplikacjami przy użyciu SOA. Promuje asynchroniczny sposób przesyłu wiadomości pomiędzy aplikacjami. Głównym zastosowaniem jest integracja heterogenicznych i złożonych aplikacji.

* Właściwości:
  + **inwokacja** - zdolność do wysyłania zapytań i odbierania odpowiedzi od serwisów i zintegrowanych zasobów
  + routing
    - oparty na treści - nie trzeba definiować celu wysyłanych wiadomości, wiadomości są filtrowane i wysyłane w określone miejsce, gdy treść spełnia dane kryteria
    - oparty na liście adresatów - wysyłanie do wielu, podział wiadomości i wysyłanie różnych części do różnych osób
    - oparty na agregacji - zbieranie powiązanych wiadomości z wielu miejsc
  + **mediacja** - ESB może wykonać pożądane transformacje i “przetłumaczenia” protokołu, formatu i treści wiadomości
  + adaptery zmniejszające nakład pracy związane z integracją
  + **bezpieczeństwo**: szyfrowanie, uwierzytelnianie, kontrola dostępu
* Wspierane standardy:
  + SOAP
  + WSDL
  + JMS (Java Message Service)
  + JCA (Java EE Connector Architecture)
  + BPEL (Business Process Execution Language) - język opisujący typy, kolejność akcji, z których składają się całe prcesy biznesowe
  + JBI (Java Business Integration) - specyfikacja SOA dla Javy
  + JDBC (Java Database Connectivity)