Projektowanie i tworzenie usług sieciowych

Usługi sieciowe (ang. web services)

- Funkcjonalność dostępna przez sieć (najczęściej z wykorzystaniem protokołu http)
- Zbiór standardów zapewniających wysoką interoperacyjność

Założenia usług sieciowych

- Komunikacja w heterogenicznym środowisku
- Niezależność od platformy (sprzętowej, programowej, transportowej)
- Duża modularność i reużywalność

Usługi sieciowe - warstwa transportowa

- Protokół HTTP lub inne np. FTP, JMS, email

Usługi sieciowe - warstwa komunikacji

- Protokół SOAP - dialekt XML, definiujący sposób wymiany informacji między konsumentem i producentem
- Nazwa protokołu pochodzi od słów Simple Object Access Protocol jednak ze względu na rolę jaką usługi sieciowe pełnią w architekturze SOA często akronim ten rozumiany jest jako Service Oriented Architecture Protocol

Usługi sieciowe - opis / dokumentacja

- Dokument WSDL - dialekt XML, kontrakt między usługodawcą, a klientami usługi (konsumentami)
- Opisuje usługę, dostępne operacje oraz sposób ich użycia bez ujawniania szczegółów implementacyjnych

Usługi sieciowe - typowy scenariusz użycia

- Dostawca usługi publikuje dokument WSDL w specjalnym katalogu
- Konsument wyszukuje usługę w katalogu i pobiera jej WSDL, a następnie wykorzystując protokół SOAP wysyła żądanie na serwer producenta
- Producent przetwarza żądanie i wykorzystując protokół SOAP dostarcza odpowiedź do konsumenta

Dokument WSDL

- Kontrakt między producentem i konsumentem
- Opisuje usługi oraz informacje niezbędne do ich użycia (parametry wejściowe, zwracane wyniki, wyjątki, sposób dostępu)
- Ma formę dokumentu XML

Struktura WSDL

- definitions - element główny dokumentu
- types - definicja typów danych używanych w dokumencie
- message - definicja komunikatów wymienianych w ramach usługi
- portType - definicja operacji realizowanych w ramach usługi
- binding - opis protokołu transportowego używanego do wywołania usługi
- service - opis usługi i jej lokalizacji

Protokół SOAP

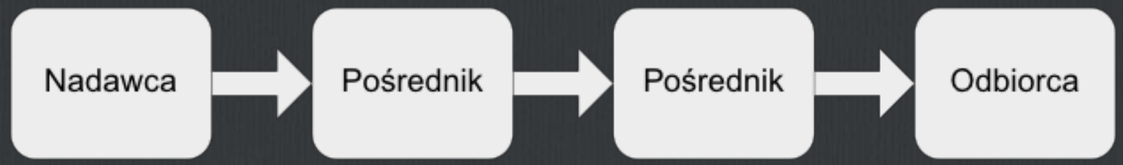
- Protokół komunikacji między producentem, a konsumentem (określa format przesyłanych wiadomości)
- Dialekt XML

Struktura komunikatu SOAP

- envelope - element główny
- header - przechowuje opcjonalne meta-dane
- body - właściwa treść komunikatu pozwalająca na wywołanie wskazanej operacji lub zawierająca zwrócony wynik

Separacja na poziomie protokołu

- Odbiorca i nadawca komunikatu nie są ze sobą powiązani
- Zanim komunikat trafi do odbiorcy może zostać przetworzony przez dowolną liczbę pośredników, którzy mogą ingerować w nagłówki i treść w celu weryfikacji reguł bezpieczeństwa, eliminacji powtórzonych żądań czy zapewnia odpowiedniej kolejności



Obsługa danych binarnych

- Protokół SOAP umożliwia przekazywanie danych binarnych takich jak obrazy czy pliki audio / video
- Transport może być realizowany na różne sposoby:
  - Poprzez zakodowanie danych binarnych w postaci tekstowej np. przy użyciu base64 i osadzeniu ich w ramach ciała wiadomości
  - Jako załącznik / załączniki do wiadomości SOAP

Kodowanie z użyciem base64

- Znacząco zwiększa ilość przesyłanych danych, dlatego powinno być używane tylko w przypadku małych załączników
- Wymaga wykonywania dodatkowych operacji kodowania / dekodowania po stronie klienta i usługodawcy

Załączniki SOAP - standardy

- SwA (SOAP with Attachments)
- DIME (Direct Internet Message Encapsulation)
- MTOM (Message Transmission Optimization Mechanism) oparty na XOP (XML-Binary Optimized Packaging)

Strategie projektowania

- Code first - rozpoczyna się od napisania kodu, a następnie wygenerowaniu kontraktu w postaci dokumentu WSDL
- Contract first - rozpoczyna się od stworzenia kontraktu, a dopiero w kolejnym kroku implementacji rozwiązania na poziomie języka programowania

Ćwiczenia praktyczne

- [ ] Omówienie przykładowych dokumentów WSDL i SOAP
- [ ] Przetestowanie udostępnionej usługi z użyciem SoapUI
- [ ] Generowanie dokumentu WSDL
- [ ] Zaprojektowanie usługi (kontraktu WSDL) realizującej wskazaną funkcjonalność

Architektura REST (REpresentational State Transfer)

- Założenia określone i opisane w pracy doktorskiej Roya Fieldinga pod tytułem „Architectural Styles and the Design of Network based Software Architectures”
  http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Jednorodny interfejs (ang. uniform interface)

- Definiuje sposób komunikacji między serwerem i klientem
- Upraszcza i rozluźnia powiązania zapewniając elastyczność oraz możliwość łatwego wprowadzania zmian
- Zakłada wykorzystanie zasobów (ang. resources)

Zasoby (ang. resources)

- Identyfikowane za pomocą unikalnych adresów URI
- Manipulacja jest możliwa tylko z użyciem predefiniowanych metod oraz reprezentacji stanu
- Niezależne od sposobu reprezentacji np. json, xml, yml

Hypermedia as the Engine of Application State (HATEOAS)

- Klient dostarcza stan poprzez ciało wiadomości, parametry żądania, nagłówki oraz URI
- Usługodawca poprzez ciało wiadomości, kody odpowiedzi i nagłówki
- Kiedy jest to wymagane odpowiedź może zawierać dodatkowe informacje - hipermedia określające dalsze możliwości pracy z zasobami

Bezstanowość (ang. stateless)

- Stan niezbędny do obsłużenia żądania zawarty jest w samej wiadomości (każda wiadomość jest samo opisująca i posiada wystarczającą ilość informacji do jej przetworzenia)
- Zapewnia wysoką skalowalność i wydajność (cache)

Możliwość wykorzystania cache (ang. cacheable)

- Odpowiedzi powinny dać się cachować lub przynajmniej informować czy jest to możliwe
- Wykorzystanie cache może częściowo lub całkowicie wyeliminować komunikację klient-server co pozytywnie wypływa na wydajność i skalowalność

Separacja klient-server (ang. klient-server separation)

- Klienci nie znają sposobu przechowywania danych, realizacji logiki biznesowej, wykorzystywanych bibliotek i technologii itd.
- Usługodawcy nie wiedzą nic o sposobie prezentowania informacji, typie klienta itp.
- Obie strony są prostsze w utrzymaniu i mogą być rozwijane niezależnie od siebie

System warstwowy (ang. layered system)

- Klient nie wie czy komunikuje się bezpośrednio z serwerem końcowym czy pośrednikiem
- Pośrednicy mogą poprawiać skalowalność, zapewniać load balancing, cache, security itd.

Wykonanie kodu na żądanie (ang. code on demand)

- Serwer jest w sanie rozszerzyć funkcjonalność klienta transferując do niego instrukcje / kod do wykonania
- Powyższe założenie jako jedyne nie jest obowiązkowe do zachowania architektury REST

REST a protokół HTTP

- Architektura REST nie narzuca protokołu komunikacyjnego jednak implementacja usług tego typu najczęściej wykorzystuje protokół HTTP (ang. Hypertext Transfer Protocol)

Protokół HTTP

- Synchroniczny i bezstanowy
- Konwersacja realizowana jest przy użyciu tekstowych komunikatów żądania i odpowiedzi
- Komunikaty składają się z:
  - lini początkowej - metoda protokołu i adres URI dla żądania oraz status i jego opis dla odpowiedzi
  - nagłówków - meta-informacji dotyczących zawartości wiadomości np. jej rozmiaru, typu, kodowania
  - ciała wiadomości oddzielonego separatorem (pusta linia)

Wybrane metody protokołu HTTP

- GET - odpytuje serwer o określony zasób, nie powinna skutkować zmianą stanu po stronie serwera, wywoływana wielokrotnie z reguły powinna zwracać te same wyniki
- HEAD - działa jak metoda GET, ale nie zwraca ciała odpowiedzi
- PUT - wykonuje żądanie zachowania ciała wiadomości na serwerze pod wskazaną lokalizacją, stosowana w celu wykonania operacji typu stwórz lub aktualizuj, wywoływana wielokrotnie powinna zwracać te same wyniki - być typu idempotent (ang. idempotent), zwykle udostępnia tożsamość tworzonego / aktualizowanego zasobu
- POST - umożliwia wykonanie modyfikacji zasobu, jako jedyna nie jest typu idempotent
- DELETE - służy do usuwania zasobów, powinna być typu idempotent
- OPTIONS - umożliwia weryfikację obsługiwanych metod bez konieczności wykonania konkretnej akcji

Negocjacja treści

- Współczesne aplikacje muszą być wystarczająco elastyczne, aby obsługiwać wiele rodzajów klientów i umożliwiać integrację odmiennych platform. Wymagania te są szczególnie istotne w kontekście architektury SOA
- Większość języków programowania pozwala na komunikację przy użyciu protokołu HTTP jednak każdy z klientów może wykorzystywać / rozumieć inny format danych np. XML, YAML, JSON, TEXT
- Dodatkowo niemal zawsze zachodzi potrzeba internacjonalizacji
- Protokół HTTP oferuje możliwość negocjacji dotyczących formatu ciała przesyłanych wiadomości, sposobu ich kodowania, a nawet używanego języka (lokalizacji). Cecha ta nazywana jest HTTP Content Negotiation lub w skrócie conneg
- Negocjacja treści odbywa się przy użyciu nagłówków Accept i Content-Type. Serwer akceptuje preferowany przez klienta typ treści lub zwraca status 406 (Not Acceptable) informujący o braku wsparcia
- Negocjacja preferowanego języka odbywa się poprzez nagłówek Accept-Language, a w przypadku kodowania Accept-Encoding

Dobre praktyki związane z projektowaniem REST

- [ ] Identyfikatory powinny być jasne, zwięzłe i osadzone w kontekście
- [ ] Identyfikatory powinny składać się jedynie z rzeczowników (rodzaj wykonywanej operacji określa metoda protokołu HTTP)
- [ ] Identyfikatory powinny mieć postać kolekcji albo instancji (nie należy mieszać liczby pojedynczej i mnogiej)
- [ ] Relacje powinny być mapowane w postaci podzasobów
- [ ] Należy używać metod protokołu HTTP zgodnie z ich przeznaczeniem

Dobre praktyki związane z projektowaniem REST

- [ ] Zarówno klient jak i serwer powinien być poinformowany o stosowanym formacie danych (odpowiednie nagłówki)
- [ ] Operacje takie jak filtrowanie, sortowanie, stronicowanie, grupowanie itd. powinny być realizowane z użyciem parametrów żądania
- [ ] Tam gdzie to możliwe należy stosować HATEOAS

Dobre praktyki związane z projektowaniem REST

- [ ] Od samego początku należy wersjonować API (klient podaje oczekiwaną wersję w nagłówku Accept)
- [ ] Obsługa błędów powinna być realizowana za pomocą odpowiednich statusów HTTP oraz ciała informującego zaistniałym błędzie (kod błędu, komunikat)
- [ ] Jeśli serwer nie obsługuje wszystkich potrzebnych metod HTTP należy wykorzystać nagłówek X-HTTP-Method-Override

Dobre praktyki związane z projektowaniem REST

- [ ] Ze względu na prostotę preferowanym formatem danych powinien być JSON
- [ ] Należy tworzyć zasoby zachowując odpowiedni poziom szczegółowości
- [ ] Należy zapewnić cachowanie (poprawnie zakończone operacje GET)
- [ ] Dokumentacja

Argumenty za usługami opartymi o SOAP

- [ ] Dobre wsparcie starszych systemów
- [ ] Sformalizowany kontrakt / dokumentacja API
- [ ] Możliwość przetwarzania stanowego (konwersacja)
- [ ] Dobre wsparcie z poziomu narzędzi i języków

Argumenty za usługami opartymi o architekturę REST

- [ ] Prostota i łatwość użycia
- [ ] Bardzo dobra skalowalność i wydajność (bezstanowość, cachowanie, format wiadomości)
- [ ] Wsparcie różnych formatów danych (standardem jest JSON)
- [ ] Popularność

Format JSON (JavaScript Object Notation)

- Lekki format wymiany danych
- Bazuje na składni pochodzącej z języka JavaScript (literał obiektu)
- Niezależny od platformy i języka
- Bardzo popularny (ogromne wsparcie narzędzi i bibliotek)

Argumenty za formatem JSON

- [ ] Popularność
- [ ] Czytelność
- [ ] Zwięzłość
- [ ] Prostota

WADL (Web Application Description Language)

- Dialekt XML opisujący usługę sieciową
- Opisuje zasoby oraz relacje dostępne w ramach usługi
- Niezależny od platformy i języka

WADL (Web Application Description Language)

```
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
 xmlns:tns="urn:yahoo:yn" xmlns:yn="urn:yahoo:yn" xmlns:ya="urn:yahoo:api"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://wadl.dev.java.net/2009/02">
  <grammars>
    <include href="NewsSearchResponse.xsd"/>
    <include href="Error.xsd"/>
  </grammars>

  <resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
    <resource path="newsSearch">
      <method name="GET" id="search">
        <request>
          <param name="appid" type="xsd:string" style="query" required="true"/>
          <param name="query" type="xsd:string" style="query" required="true"/>
          <param name="type" style="query" default="all">
            <option value="all"/>
            <option value="any"/>
            <option value="phrase"/>
          </param>
          <param name="results" style="query" type="xsd:int" default="10"/>
          <param name="start" style="query" type="xsd:int" default="1"/>
          <param name="sort" style="query" default="rank">
            <option value="rank"/>
            <option value="date"/>
          </param>
          <param name="language" style="query" type="xsd:string"/>
        </request>
        <response status="200">
```

Ćwiczenia praktyczne

- [ ] Wykorzystanie dostarczonej usługi REST z użyciem narzędzi Postman
- [ ] Zapoznanie się opisem usługi (WADL)
- [ ] Zaprojektowanie interfejsu i modelu danych usługi realizującej wskazaną funkcjonalność

Architektura tradycyjna (monolityczna)

- Kod aplikacji (poszczególne warstwy) stanowi zespojoną całość
- Podział na poszczególne jednostki organizacyjne jest realizowany głównie z użyciem takich mechanizmów jak klasy i pakiety / moduły

SOA (Service-oriented architecture)

- Aplikacja składa się z wielu, luźno powiązanych komponentów (czarne skrzynki z dobrze zdefiniowanym interfejsem), które w połączeniu w odpowiedni sposób realizują założenia / proces biznesowy
- Pojęcie SOA obejmuje tworzenie, zarządzanie i modelowanie procesów biznesowych

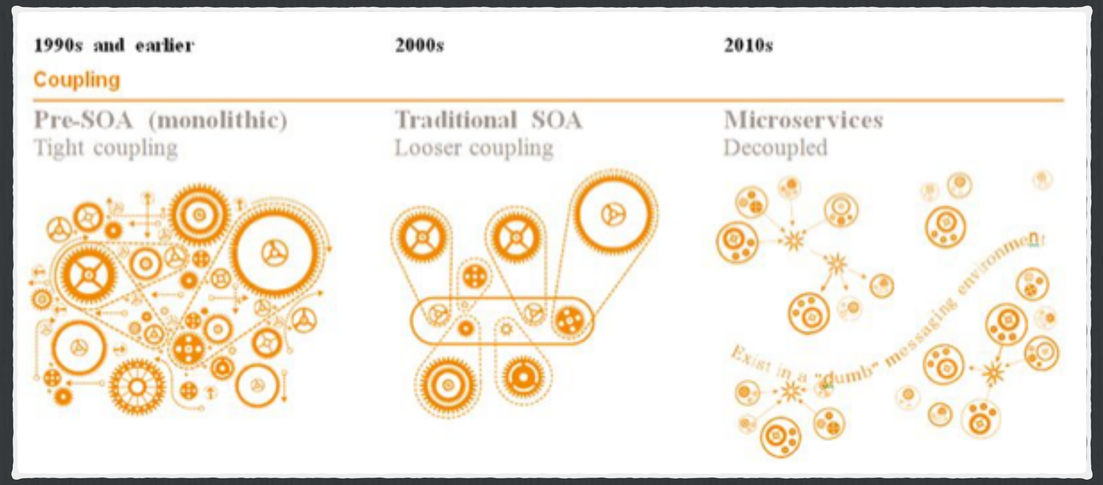
Microservices architecture

- Aplikacja składa się z wielu luźno powiązanych usług (serwisów)

Mikroserwisy

- [ ] Samowystarczalne, lekkie procesy, komunikujące się po HTTP
- [ ] Tworzone i wdrażane relatywnie małym nakładem pracy
- [ ] Skupione na realizacji określonej funkcjonalności
- [ ] Posiadają dobrze zdefiniowany interfejs

Ewolucja architektury opartej o usługi



Ćwiczenia praktyczne

- [ ] Uruchomienie i analiza przykładu opartego o mikroserwisy
- [ ] Zapoznanie się z działaniem takich rozwiązań jak service discovery, configuration server, client side load balancing, circuit braker, oauth2