Projektowanie obiektowe oprogramowania

Architektura systemów (2)

Enterprise Service Bus

Command-Query Responsibility Segregation

Wykład 15

Wiktor Zychla 2013

---

1 Modele integracji aplikacji

1.1 Integracja przez wymianę plików

- Możliwe do implementacji w niejednolitym środowisku
- Konieczność ręcznej ingerencji użytkownika w proces czyni ten proces bezużytecznym tam gdzie oczekuje się automatyczności przepływów danych

1.2 Integracja przez wspólną bazę danych

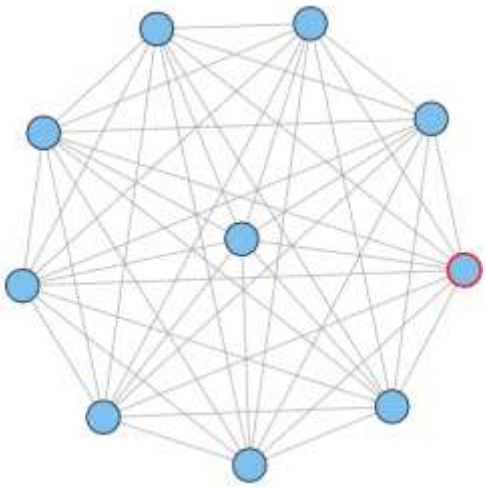
- Integracja jest automatyczna i natychmiastowa
- Duża struktura danych paraliżuje swój rozwój
- W ten sposób nie da się integrować aplikacji pochodzących od różnych dostawców

Hohpe/Woolf:

> *If the technical difficulties of designing a unified schema aren't enough, there are also severe political difficulties. If a critical application is likely to suffer delays in order to work with a unified schema, then often there is irresistable pressure to separate. Human conflicts between departments often exacerbate this problem*

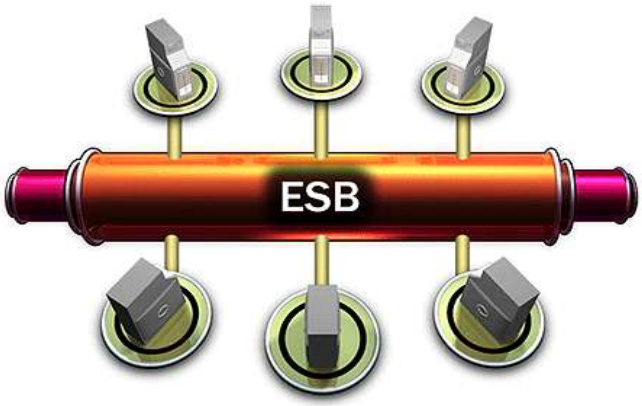
1.3 Integracja przez usługi aplikacyjne (Web Services)

- Możliwe do implementacji w niejednolitym środowisku
- Można integrować aplikacje różnych producentów
- Właściwie jedyna wada – przypadłość „sieci połączeń”



1.4 Integracja za pośrednictwem szyny usług

- Połączenie pomysłu integracji przez usługi aplikacyjne z ideą wzorca Observer
- Każda aplikacja łączy się z szyną usług i publikuje lub subskrybuje powiadomienia
- Złożoność sieci połączeń jest liniowa – każda aplikacja integruje się tylko z szyną usług
- Szyna dostarcza mechanizmów wiarygodnego dostarczania komunikatów



2 Enterprise Service Bus (Integracyjna szyna usług)

Message-oriented architecture – model architektury zorientowany na komunikaty wymieniane między modułami. Ciekawie zaczyna robić się wtedy, kiedy mówimy o architekturze **systemów** ponieważ wtedy do wymiany komunikatów niezbędny staje się zewnątrzprocesowy pośrednik – **Message-oriented Middleware** (**MOM**).

Enterprise Service Bus – usługa zewnątrzprocesowa typu MOM, dostarczająca narzędzi komunikacyjnych w rozległym środowisku aplikacyjnym. Szczegóły niżej.

Przykład – wiele niezależnych systemów bankowych, integracja przelewów i innych operacji finansowych – natychmiastowa aktualizacja między systemami.

W praktyce wzorzec ESB tak samo dobrze organizuje architekturę systemu, jak Event Aggregator organizował architekturę aplikacji.

2.1 Szyna usług - podstawowe pojęcia

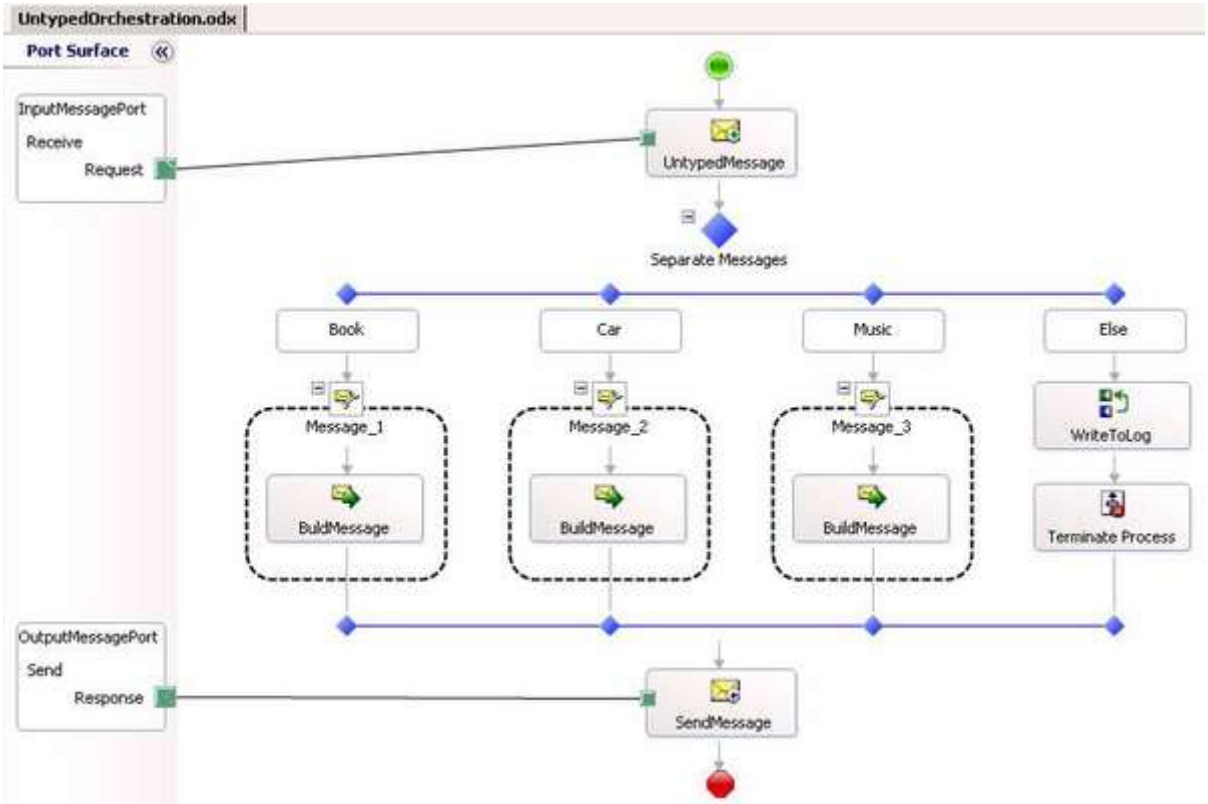
Publish/Subscribe – wzorzec wymiany komunikatów w szynie usług, w którym wybrane aplikacje pełnią rolę publikatorów komunikatów (powiadomień), a inne pełnią rolę subskrybentów tych komunikatów. Na tym atomowym wzorcu oparte są pozostałe wzorce złożone – Request/Reply i Saga.

Request/Reply – wzorzec wymiany komunikatów w szynie usług, w którym aplikacja zadaje zapytanie do szyny usług, a jedna (zwyczajowo) z aplikacji jest zarejestrowana jako dostawca takich danych i odpowiada na żądanie. Należy zauważyć, że R/R implementuje się z wykorzystaniem wzorca P/S – R/R dla aplikacji A i B wygląda następująco:

- B subskrybuje komunikatu typu „Żądam danych typu X”
- A subskrybuje komunikat typu „Odpowiedź na zapytanie o dane typu X”
- A publikuje komunikat typu „Żądam danych typu X”
- B otrzymuje komunikat i go przetwarza; publikuje komunikat typu „Odpowiedź na zapytanie o dane typu X” i w adresata wstawia A
- (opcjonlanie) Szyna filtruje komunikat tak żeby dostał go tylko A
- A otrzymuje odpowiedź

Saga (inaczej *long running transaction*; transakcja długoterminowa) – wzorzec wymiany komunikatów w szynie usług, oparty o złożony proces biznesowy, którego koordynatorem jest szyna. Proces zwykle składa się z atomowych kroków, najważniejsze z nich to komunikacja z portami wejścia/wyjścia.

Przykładowa saga: komunikat od konsultanta do spraw sprzedaży z wnioskiem o kredyt dla klienta trafia do szyny. Jeśli kwota kredytu jest mniejsza niż 1000$, szyna tworzy komunikat typu „aprobata wniosku” i przekazuje nowy komunikat do serwera departamentu wypłacania kredytów. Jeśli kwota kredytu jest większa lub równa 1000$, szyna przekazuje ten komunikat dalej, do serwera departamentu kredytowego, gdzie komunikat oczekuje w systemie na akceptację kierownika działu. Szyna oczekuje na odpowiedź nie dłużej niż 48h. Jeśli w tym czasie nadchodzi odpowiedź pozytywna – szyna konwertuje ją na komunikat typu „aprobata wniosku” i przekazuje nowy komunikat do serwera departamentu wypłacania kredytów. Jeśli odpowiedź jest odmowna lub upłynęło więcej niż 48h, szyna tworzy komunikat typu „odmowa” i przekazuje zwrotnie do systemu, w którym zobaczy go konsultant.



Rysunek 1 (za http://www.codeproject.com/Articles/13277/Configuring-BizTalk-Orchestrations-to-handle-un-ty)

(Uwaga! Diagram nie odpowiada treści przykładowego procesu!)

Asynchroniczna komunikacja – dlaczego rozproszonych usług integracyjnych nie można zaimplementować tak samo jak wzorca Observer? Dlatego że subskrybenci mają różne tempo przetwarzania powiadomień. Sto powiadomień od A do B i C może być przez B przetwarzane w ciągu 1 sekundy, a przez C w ciągu 1 godziny. Szyna musi wiarygodnie koordynować przekazywanie komunikatów i wymaga wsparcia jakiegoś repozytorium trwałego, np. relacyjnej bazy danych lub usługi kolejkowania wiadomości.

Specyfikacje formatów wymiany danych – zwyczajowo szyny usług do integracji aplikacji niejednolitych technologicznie wykorzystują nośnik XML, wsparty opcjonalnie walidacją XSD

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema     xmlns="http://ito.vulcan.edu.pl"     targetNamespace="http://ito.vulcan.edu.pl"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:complexType name="ITOJednostkaSamodzielna">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="1" name="Adres" type="ITOAdres"/>
      <xs:element         minOccurs="0"         maxOccurs="1"         name="JednostkiSkladowe"
type="ArrayOfITOJednostkaSkladowa"/>
      <xs:element minOccurs="1" maxOccurs="1" name="ID" type="xs:int"/>
      <xs:element minOccurs="0" maxOccurs="1" name="Nazwa" type="xs:string"/>
      <xs:element minOccurs="0" maxOccurs="1" name="NazwaZUS" type="xs:string"/>
      <xs:element minOccurs="0" maxOccurs="1" name="Kod" type="xs:string"/>
      <xs:element minOccurs="0" maxOccurs="1" name="Regon" type="xs:string"/>
      <xs:element minOccurs="0" maxOccurs="1" name="NIP" type="xs:string"/>
      <xs:element minOccurs="0" maxOccurs="1" name="Patron" type="xs:string"/>
      <xs:element minOccurs="0" maxOccurs="1" name="Siedziba" type="xs:string"/>
      <xs:element minOccurs="0" maxOccurs="1" name="Opis" type="xs:string"/>
      <xs:element minOccurs="0" maxOccurs="1" name="Uwagi" type="xs:string"/>
      <xs:element      minOccurs="1"      maxOccurs="1"      name="DataOd"      nillable="true"
type="xs:dateTime"/>
      <xs:element      minOccurs="1"      maxOccurs="1"      name="DataDo"      nillable="true"
type="xs:dateTime"/>
      <xs:element minOccurs="1" maxOccurs="1" name="PlatnikVAT" type="xs:boolean"/>
      <xs:element minOccurs="1" maxOccurs="1" name="PlatnikFGSP" type="xs:boolean"/>
      <xs:element minOccurs="0" maxOccurs="1" name="Rodzaj" type="xs:string"/>
      <xs:element minOccurs="0" maxOccurs="1" name="OrganProwadzacy" type="xs:string"/>
      <xs:element minOccurs="0" maxOccurs="1" name="OrganRejestrujacy" type="xs:string"/>
      <xs:element   minOccurs="1"   maxOccurs="1"   name="IdOrganProwadzacy"   nillable="true"
type="xs:int"/>
      <xs:element  minOccurs="1"  maxOccurs="1"  name="IdOrganRejestrujacy"  nillable="true"
type="xs:int"/>
      <xs:element minOccurs="1" maxOccurs="1" name="S4" type="xs:int"/>
      <xs:element minOccurs="0" maxOccurs="1" name="IdOsw" type="xs:string"/>
      <xs:element minOccurs="1" maxOccurs="1" name="Moja" type="xs:boolean"/>
      <xs:element minOccurs="1" maxOccurs="1" name="IsDeleted" type="xs:boolean"/>
      <xs:element minOccurs="1" maxOccurs="1" name="DataDodania" type="xs:dateTime"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Port wejścia – szyny są usługami zewnątrzprocesowymi; typowa szyna usług jest równocześnie usługą systemową (demonem systemowym). To oznacza, że komunikat trzeba do szyny jakoś dostarczyć. Port wejścia jest abstrakcją źródła danych, a typowe tzw. *adaptery* (implementacje konkretne) obejmują: relacyjne bazy danych, system plików czy usługi aplikacyjne. Na przykład więc port wejścia używający adaptera – systemu plików przyjmuje komunikat w postaci pliku pojawiającego się w określonym folderze. Aplikacja aby wysłać komunikat do szyny tworzy plik XML o określonej składni w tymże wybranym folderze, a szyna aktywnie podejmuje i przetwarza komunikat.

Mówimy że port wejścia pracuje w trybie **pull**, kiedy szyna musi aktywnie monitować jakiś zasób w celu stwierdzenia, że pojawiają się tam dane. W trybie pull pracują adaptery – system plików i relacyjna baza danych. Mówimy że port wejścia pracuje w trybie **push**, kiedy usługa przekazania komunikatu jest wzbudzana „na żądanie", bez potrzeby monitorowania zasobu. Tak działa adapter – usługa aplikacyjna – dane są przekazane do szyny przez wywołanie jakiejś jej usługi aplikacyjnej.

Port wyjścia – abstrakcja miejsca, w które szyna dostarcza dane. Typowe adaptery są podobne jak w przypadku portów wejścia. Na przykład port wyjścia używający adaptera – usługi aplikacyjnej dostarczy komunikat do aplikacji w ten sposób, że wywoła jakąś określoną usługę aplikacyjną (web service) po stronie aplikacji.

Mówimy że port wyjścia działa w trybie **pull**, kiedy aplikacja-subskrybent musi aktywnie monitować jakiś zasób, żeby zorientować się że szyna ma dla niej jakieś dane. Na przykład port wyjścia używający adaptera – usługa aplikacyjna po stronie szyny, wymaga od aplikacji regularnego monitorowania stanu portu (wywoływania określonej metody) w celu stwierdzenia czy pojawiły się jakieś dane. Mówimy że port wyjścia działa w trybie **push**, jeśli szyna danych aktywnie przekazuje komunikat do subskrybenta. Na przykład port wyjścia używający adaptera – usługa aplikacyjna po stronie subskrybenta, po pojawieniu się komunikatu po prostu wywoła aktywnie wskazaną usługę aplikacyjną.

Oczywista obserwacja – porty wejścia z adapterami w trybie push są wydajniejsze i pozwalają na niemal on-line przekazywanie powiadomień. Jedyny problem jest taki, że to nie zawsze technicznie jest możliwe, na przykład tryb push dla portów wyjścia nie jest możliwy jeśli subskrybent nie jest aplikacją serwerową, posiadającą własne usługi aplikacyjne; z kolei tryb pull zawsze jest możliwy na portach wyjścia, bo szyna jest usługą serwerową.

2.2 Przegląd implementacji

Usługi ESB mają dziesiątki mniejszych i większych implementacji. Trochę wyobrażenia można nabrać przeglądając (niepełny!) rejestr na wiki:

http://en.wikipedia.org/wiki/Comparison_of_business_integration_software

Możliwe są proste i tanie implementacje wykorzystujące np. mechanizmy Pub/Sub w WCF, podsystem Workflow Foundation czy wprost usługi kolejkowania wiadomości (MSMQ, JMS) opakowane jakąś implementacją adapterów dla wybranych portów wejścia wyjścia.

2.3 Przykład

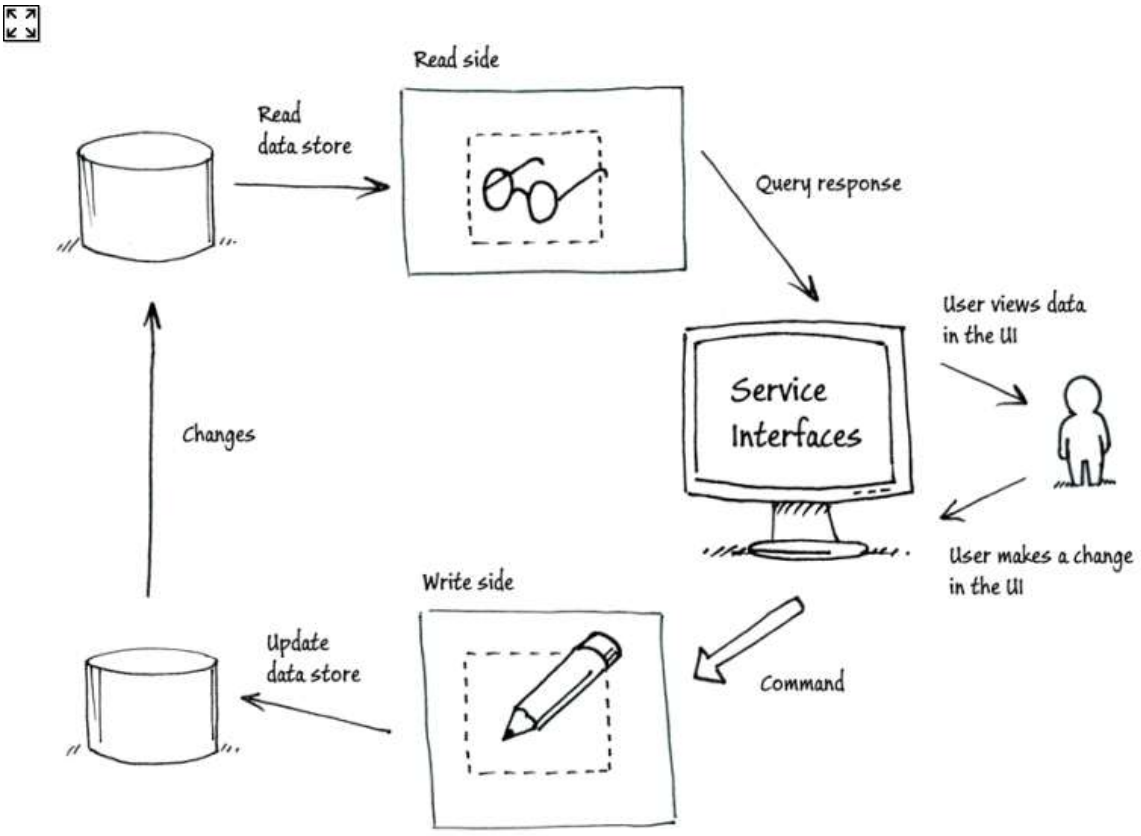
Interaktywny przykład najprostszej możliwej implementacji szyny integracyjnej opartej o wieloplatformowy i popularny system typu MOM, RabbitMQ. Omówimy pojęcia:

- Kolejki (Queue)
- skrzynki nadawczej (Exchange),
- Powiązań (Binding)
- Routing typu „fan”
- Routing typu „direct”
- Routing typu „topic”

http://rabbitmq.com

3 Command-Query Responsibility Segregation

Command-Query Responsibilibty Separation – filozofia budowy systemu transakcyjnego, w którym rola odczytu i zapisu (modyfikacji) jest wyraźnie koncepcyjnie rozdzielona.



Rysunek 2 Za http://msdn.microsoft.com/en-us/library/jj591573.aspx

Podstawowe spostrzeżenia:

- W dużej części systemów dane głównie się ogląda – rozrzut między odczytami a zapisami to nawet 1000:1
- Zapis jest najwygodniejszy do struktury silnie znormalizowanej, odczyt jest najwygodniejszy ze struktury mocno zdenormalizowanej (idealnie: całkowicie spłaszczczonej). To mogą być nawet dwa różne magazyny danych (np. baza relacyjna do zapisu i baza typu noSQL do odczytu)
- W praktyce do implementacji architektury CQRS można użyć systemu kolejkowego – każda operacja po stronie zapisu powoduje powstanie komunikatu, który jest przetwarzany po stronie odczytu i używany do zaktualizowania danych. System składa się wtedy z części do przetwarzania zapisów/modyfikacji, systemu kolejkowego i części do odczytu. Mając tak rozdzielone odpowiedzialności można łatwiej dostosować system do rzeczywistego obciążenia.

4 Literatura

1. Hohpe, Woolf – Enterprise Integration Patterns, http://www.eaipatterns.com/ - podręcznik wzorców integracyjnych, szczegółowo omawia wzorzec ESB i szereg jego podwzorców
2. Videla, Williams – RabbitMQ in Action