Projektowanie obiektowe oprogramowania
Wykład 15 – Elementy architektury Enterprise (2)
Single Sign-on
Enterprise Service Bus
Wiktor Zychla 2012

---

1 Single sign-on – porównanie protokołów

Protokół SSO określa sekwencję komunikatów, prowadzącą do wymiany informacji o tożsamości użytkownika między aplikacją a usługą poświadczania tożsamości. Dla każdego z protokołów ta sekwencja się różni.

Z uwagi na podobieństwa, można wyróżnić pewne cechy wspólne różnych protokołów:

- Protokoły oparte o **SAML** (np. WS-Federation) – sekwencja wymiany komunikatów między aplikacją (RP) a usługą (STS) odbywa się zawsze za pośrednictwem przeglądarki i kończy się przesłaniem dużego, podpisanego XMLa, który zawiera komplet informacji o użytkowniku
- Protokoły oparte o **token dostępowy** (ang. access token) (np. OAuth2) – sekwencja wymiany komunikatów między aplikacją a usługą prowadzi do ustalenia pewnego długiego ciągu znaków, który pozwala aplikacji na wchodzenie w imieniu użytkownika do różnych usług za pośrednictwem dowolnie bogatego API po stronie usługi. W praktyce w ten sposób można precyzyjniej przekazywać dane między aplikacją a usługą, ponieważ aplikacja może wielokrotnie używać tego samego tokena do dostępu do różnych danych po stronie usługi.
- Protokoły oparte o zbudowanie wzajemnego **zaufania** (ang. association) (np. OpenID) – sekwencja wymiany komunikatów obejmuje pewne komunikaty wymieniane bezpośrednio między aplikacją a usługą, umożliwiające zbudowanie relacji zaufania i w konsekwencji „podpisanie” informacji o tożsamości użytkownika (odpowiednik certyfikatu podpisującego token w imieniu usługi w protokole WS-Federation)

2 Enterprise Service Bus

Enterprise Service Bus (Integracyjna Szyna Usług) – jeden z modeli architektury aplikacji, w którym moduły komunikują się udostępniając sobie nawzajem usługi za pośrednictwem dodatkowego komponentu integracyjnego.

W praktyce wzorca tego używa się najczęściej do zbudowania środowiska ze zintegrowanymi przepływami usług tam, gdzie wcześniej działały tylko rozproszone aplikacje.

Przykład – system płacowy i system księgowy. Moduł płacowy przygotowuje paski wypłat, księgowość księguje wypłaty, przygotowuje i wysyła przelewy przez bank, zwraca do płac zwrotnie informację o zaksięgowaniu poleceń wypłaty.

Inny przykład – dane osobowe użytkownika modyfikowane są w jednej aplikacji, źródłowej i są automatycznie aktualizowane w każdej innej aplikacji systemu.

Jeszcze inny przykład – dwa niezależne systemy bankowe, integracja przelewów i innych operacji finansowych – natychmiastowa aktualizacja między systemami.

2.1 Modele integracji aplikacji

2.1.1 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

2.1.2 Integracja przez wspólną bazę danych

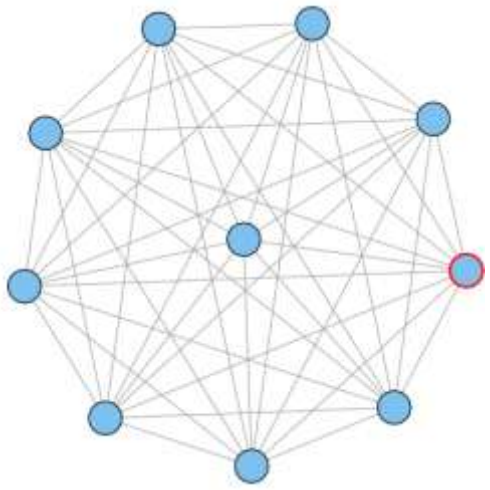
- Integracja jest automatyczna i natychmiastowa
- Duża struktura danych paraliżuje swój rozwój
- W ten sposób nie da się integrować aplikacji pochodzących od różnych dostawców

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*

2.1.3 Integracja przez usługi aplikacyjne (Web Services)

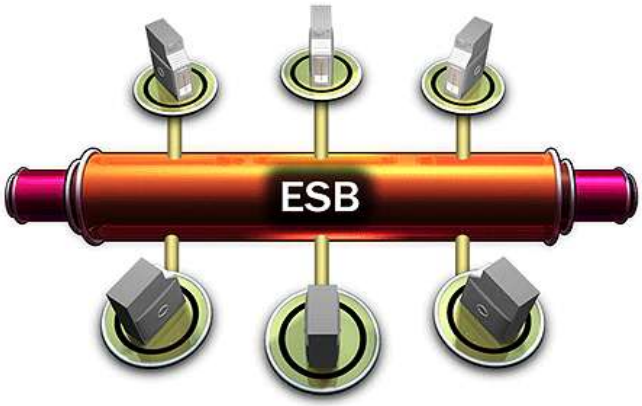
- Możliwe do implementacji w niejednolitym środowisku
- Można integrować aplikacje różnych producentów
- Właściwie jedyna wada – przypadłość „sieci połączeń”



2.1.4 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



2.2 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.

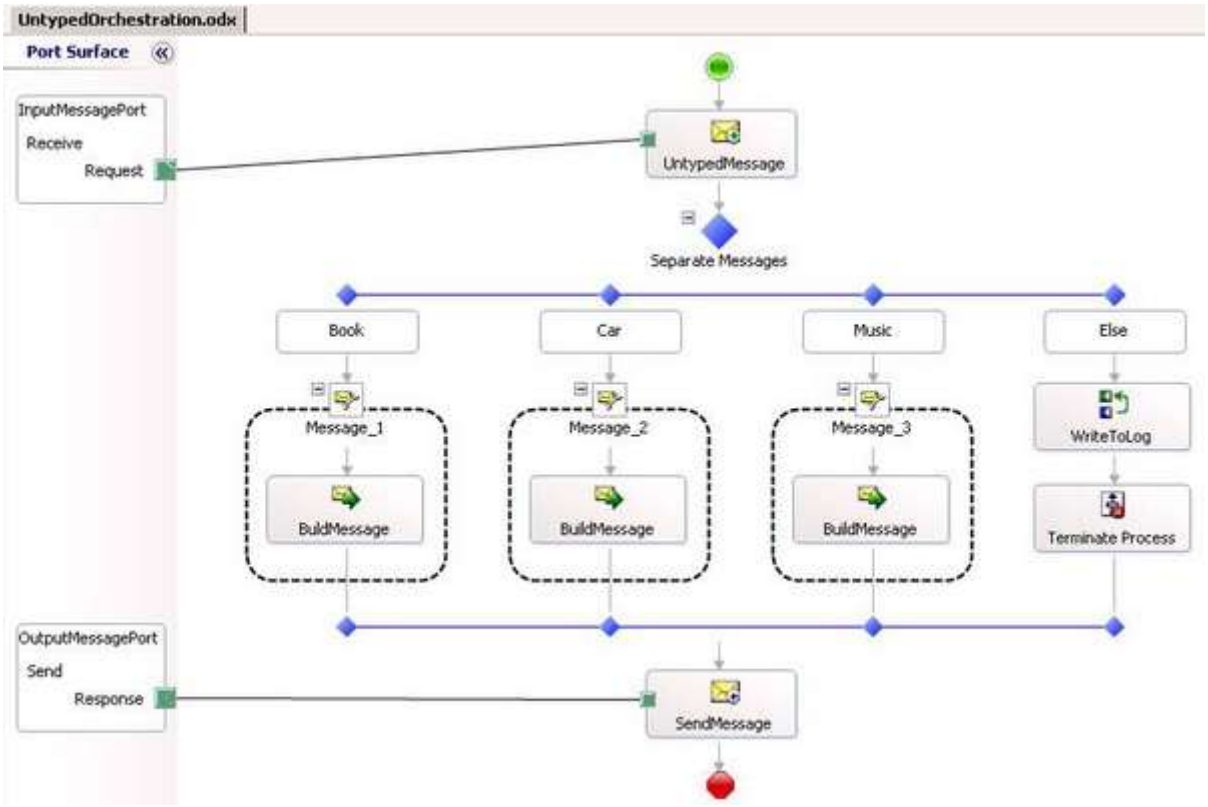
Request/Reply – wzorzec wymiany komunikatów w szynie usług, w którym aplikacja zadaje zapytanie do szyny usług, a jedna (zwyczajowo) z aplikacji jest zarejestrowana jako dostawca takich danych i odpowiada na żądanie. Należy zauważyć, że R/R implementuje się z wykorzystaniem wzorca P/S – R/R dla aplikacji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

2.3 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.

2.4 Przykład

Interaktywny przykład najprostszej możliwej implementacji szyny integracyjnej opartej o MSMQ i darmową technologię Mass Transit.

http://masstransit-project.com/

2.5 Literatura

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