Projektowanie obiektowe oprogramowania

Architektura systemów (3)

Service Oriented Architecture

Wykład 15

Wiktor Zychla 2019

Spis treści

1 Modele integracji danych

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
- - wymiana automatyczna jest możliwa, ale nadal wymaga dostępu systemów do tego samego nośnika (pliku), co często jest fizycznie niemożliwe

1.2 Integracja przez wspólną bazę danych

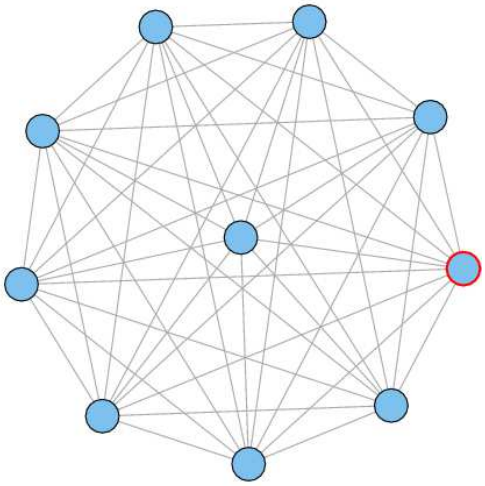
- + Integracja jest automatyczna i natychmiastowa
- - Duża struktura danych paraliżuje swój rozwój
- - W ten sposób nie zawsze da się integrować aplikacji pochodzących od różnych dostawców / wykonanych w różnych technologiach / nie mających dostępu do tej samej bazy danych

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 po http/https)

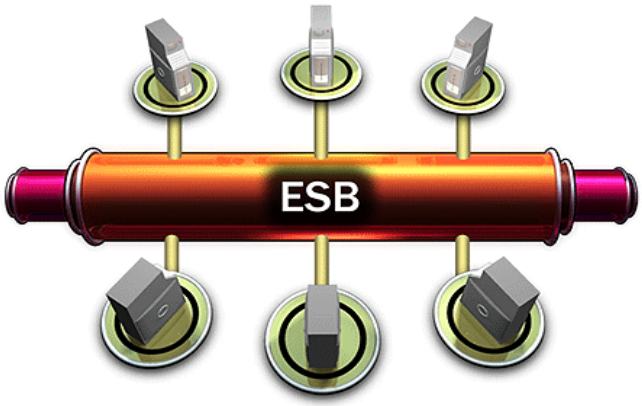
- + możliwe do implementacji w niejednolitym środowisku
- + można integrować aplikacje różnych producentów
- + protokoły komunikacyjne oparte o XML/JSON są standardami interoperacyjnymi
- - właściwie jedyna wada – przypadłość „sieci połączeń” (każdy z każdym; przestaje się to sprawdzać po 3 i więcej aplikacjach)



1.4 Integracja za pośrednictwem brokera/szyny usług

- Połączenie pomysłu integracji przez usługi aplikacyjne z ideą wzorca Event Aggregator
- Każda aplikacja łączy się z podsystemem komunikacyjnym i publikuje lub subskrybuje powiadomienia

- Złożoność sieci połączeń jest liniowa, nie ma problemu „sieci powiązań” – każda aplikacja integruje się tylko z podsystemem komunikacyjnym
- Szyna dostarcza mechanizmów wiarygodnego dostarczania komunikatów



2 Message Oriented Architecture

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**).

Gdyby każdy system typu MOM zbudowany był w taki sposób, że komunikacja z nim odbywałaby się w jego własnym „dialekcie", to budowa rozwiązań integracyjnych byłaby utrudniona. Próbuje się więc budować abstrakcje, które gwarantują wymienność implementacji infrastruktury oraz możliwość łączenia różnych rozwiązań typu MOM.

Z punktu widzenia implementacji mechanizmów integracyjnych po stronie aplikacji-nadawców i aplikacji-odbiorców danych, istnieją dwie filozofie budowy abstrakcji

- **API Driven** (zorientowane na interfejs programistyczny) – podstawą abstrakcji mechanizmów integracyjnych jest API udostępniane przez mechanizm integracyjny. API jest ustandaryzowane, na zewnątrz gwarantuje stały kontrakt, a szczegóły wewnętrznej implementacji są nieistotne.

  Przykład – JMS, Java Message.

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

  JMS jest zbiorem API, to znaczy że posiada abstrakcje takie jak połączenie, nadawca, odbiorca, komunikat wyrażone w języku obiektowej specyfikacji. JMS ma implementacje różnych „dostawców", trochę tak jak podsystem ORM ma implementacje dostawców dialektów dla różnych baz danych. Wikipedia wymienia kilkanaście istniejących implementacji.

  Plus abstrakcji typu API driven jest taki że raz napisany kod zawsze pozostaje niezmienny, zmienia się tylko **konfiguracja** dostawcy.

  Minus jest taki, że JMS jako API jest częścią J2EE, więc nie występuje w środowiskach .NET, PHP itd. W efekcie, aby zaimplementować mechanizm integracji z dostawcą specyfikacji JMS w innej technologii niż J2EE i tak trzeba poznać szczegóły implementacji protokołu komunikacyjnego konkretnego dostawcy.

  Nie da się więc powiedzieć, że takie specyfikacje zorientowane na interfejs są interoperacyjne – kodu napisanego w JMS nie da się łatwo przenieść do .NET/PHP/itd.

  Przykład kodu nadawcy komunikatów, ilustrujący model pojęciowy JMS

```
/* za https://examples.javacodegeeks.com/enterprise-java/jms/jms-
client-example/ */
public static void main(String[] args) throws URISyntaxException,
Exception {
```

```
    Connection connection = null;
    try {
        // factory połączeń - specyficzne dla dostawcy
        ConnectionFactory connectionFactory = new
ActiveMQConnectionFactory(
                "tcp://localhost:61616");
        connection = connectionFactory.createConnection();
        // od tego miejsca API jest takie samo, bez względu na
dostawcę
        Session session = connection.createSession(false,
                Session.AUTO_ACKNOWLEDGE);
        Queue queue = session.createQueue("customerQueue");
        MessageProducer producer = session.createProducer(queue);
        // publikacja wiadomości
        String task = "Task";
        for (int i = 0; i < 10; i++) {
            String payload = task + i;
            Message msg = session.createTextMessage(payload);
            System.out.println("Sending text '" + payload + "'");
            producer.send(msg);
        }
        // zakończenie sesji
        // (uwaga - wysłanie "END" to tylko konwencja!)
        producer.send(session.createTextMessage("END"));
        session.close();
    } finally {
        if (connection != null) {
            connection.close();
        }
    }
}
```

- **Protocol Driven** (zorientowane na protokół komunikacyjny) – podstawą abstrakcji mechanizmów integracyjnych jest gwarantowany protokół komunikacyjny między aplikacją a podsystemem MOM. Protokół jest gwarantowany, może być oparty o TCP albo o protokół wyższej warstwy, np. HTTP/HTTPS.

  Przykład – AMQP Advanced Message Queuing Protocol.

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

  AMQP jest otwartym standardem protokołu komunikacyjnego, opartego o TCP, który specyfikuje formaty ramek TCP w komunikacji klienta (aplikacji) z serwerem (brokerem) MOM. Istnieją gotowe implementacje AMQP dla większości przemysłowych technologii wytwarzania oprogramowania, ale jeśli z jakiegoś powodu takiej implementacji brak – w oparciu o specyfikację protokołu zawsze można dostarczyć własnej implementacji. Wikipedia wymienia kilka (mniej niż 10) przemysłowych implementacji, w tym open-source.

  Plus abstrakcji typu Protocol driven jest taki że jest rzeczywiście interoperacyjna, to znaczy aplikacje integrujące się z brokerem mogą być napisane w różnych technologiach/językach.

Współcześnie preferuje się specyfikacje typu Protocol Driven, specyfikacje typu API Driven są uznawane za problematyczne z uwagi na problemy z interoperacyjnością.

Więcej informacji:

- Mark Richards, Understanding the differences between AMQP & JMS

Taki „niskopoziomowy" protokół taki jak AMQP nie zawsze jest jednak dobrym wyborem. Już wiemy kiedy tak jest – na przykład wtedy, kiedy oprogramowanie musi spełniać zewnętrzne wymagania, takie jak Krajowe Ramy Interoperacyjności, gdzie wprost mówi się o interoperacyjności opartej o XML/WSDL.

W takich wypadkach można z powodzeniem ukryć implementację opartą o AMQP za warstwą komunikacyjną opartą o http/https.

3 Enterprise Service Bus

3.1 Szyna usług - podstawowe pojęcia

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.

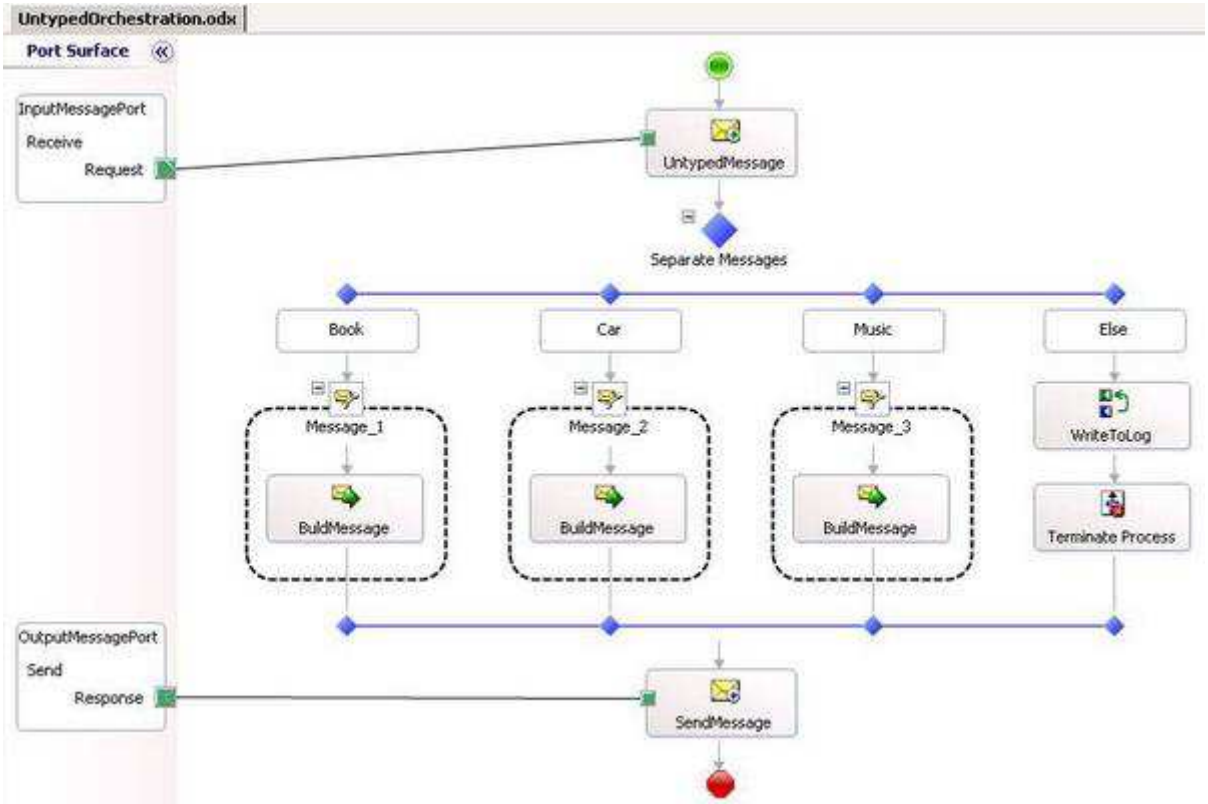
Publish/Subscribe – wzorzec wymiany komunikatów w szynie usług, w którym wybrane aplikacje pełnią rolę publikatorów komunikatów (powiadomień), a inne pełnią rolę subskrybentów tych komunikatów. Na tym atomowym wzorcu oparte są pozostałe wzorce złożone – Request/Reply i Saga.

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
- (opcjonalnie) Szyna filtruje komunikat tak żeby dostał go tylko A
- A otrzymuje odpowiedź

Saga (inaczej *long running transaction*; **transakcja długoterminowa**) (także: **proces biznesowy**) – 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 formalną XSD, transformacjami XSLT i mechanizmami autentykacji/autoryzacji opartymi o XmlDsig (patrz poprzedni wykład).

```
<?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.

Szyna usług a silnik procesów biznesowych - o oprogramowaniu serwerowym zdolnym do aktywnej koordynacji procesów biznesowych powiemy że jest silnikiem procesów biznesowych (Business Process Engine, BPE). W praktyce o takich rozwiązaniach najczęściej mówi się w kontekście specyfikacji procesów, którymi można się posługiwać.

W praktyce są dwie standaryzacje opisu procesów:

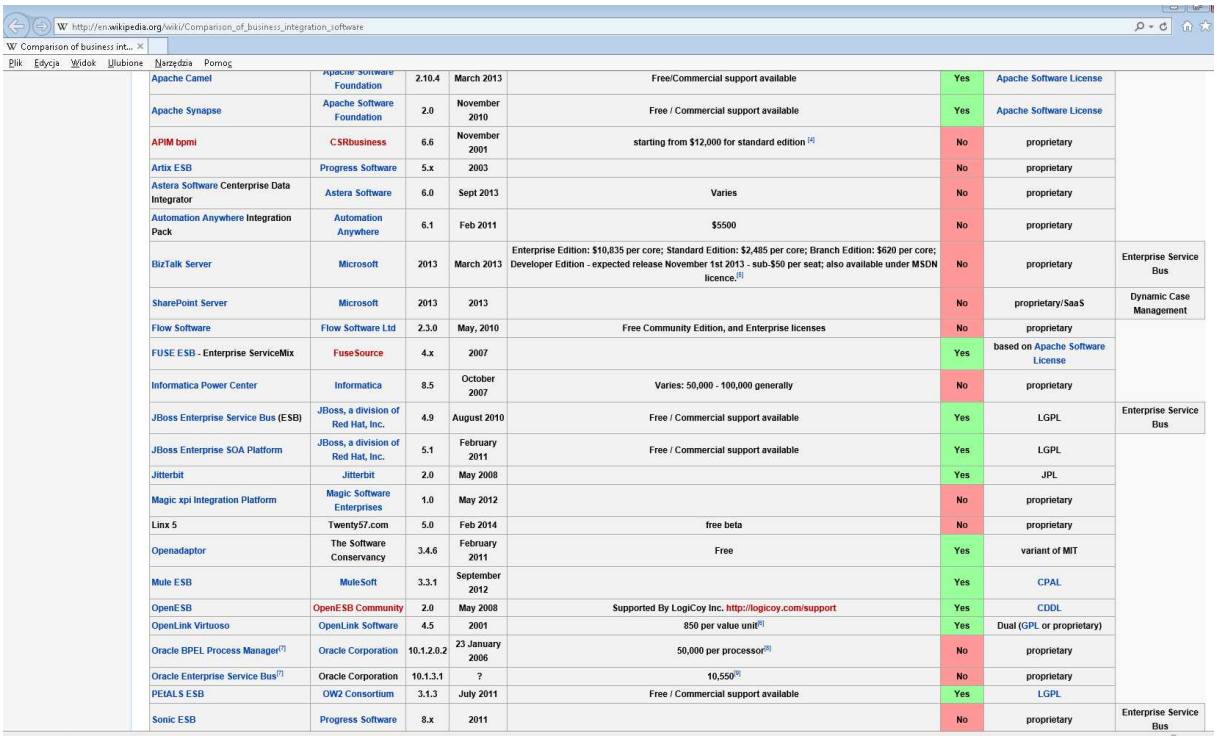
- BPEL – Business Process Execution Language, to oparta o XML standaryzacja z 2003 aktualnie pozostająca pod opieką konsorcjum OASIS
- BMPN – Business Process Model and Notation – początkowo powstał jako język do graficznego reprezentowania BPEL (chodziło o standaryzację aplikacji służących do projektowania BPEL). W 2008 przeszedł pod opiekę OMG i w 2011 w wersji 2.0 otrzymał również wsparcie dla notacji niegraficznej (XML)

Z tych dwóch, wydaje się że BPEL ma szczyt zainteresowania już za sobą, z uwagi na rozliczne kontrowersje, natomiast BPMN ma szansę być wspierany przez narzędzia projektowe.

3.2 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



Apache Camel	Apache Software Foundation	2.10.4	March 2013	Free/Commercial support available	Yes	Apache Software License	
Apache Synapse	Apache Software Foundation	2.0	November 2010	Free / Commercial support available	Yes	Apache Software License	
APIM bpmi	CSRbusiness	6.6	November 2001	starting from $12,000 for standard edition [4]	No	proprietary	
Artix ESB	Progress Software	5.x	2003		No	proprietary	
Astera Software Centerprise Data Integrator	Astera Software	6.0	Sept 2013	Varies	No	proprietary	
Automation Anywhere Integration Pack	Automation Anywhere	6.1	Feb 2011	$5500	No	proprietary	
BizTalk Server	Microsoft	2013	March 2013	Enterprise Edition: $10,835 per core; Standard Edition: $2,485 per core; Branch Edition: $620 per core; Developer Edition - expected release November 1st 2013 - sub-$50 per seat; also available under MSDN licence.[5]	No	proprietary	Enterprise Service Bus
SharePoint Server	Microsoft	2013	2013		No	proprietary/SaaS	Dynamic Case Management
Flow Software	Flow Software Ltd	2.3.0	May, 2010	Free Community Edition, and Enterprise licenses	No	proprietary	
FUSE ESB - Enterprise ServiceMix	FuseSource	4.x	2007		Yes	based on Apache Software License	
Informatica Power Center	Informatica	8.5	October 2007	Varies: 50,000 - 100,000 generally	No	proprietary	
JBoss Enterprise Service Bus (ESB)	JBoss, a division of Red Hat, Inc.	4.9	August 2010	Free / Commercial support available	Yes	LGPL	Enterprise Service Bus
JBoss Enterprise SOA Platform	JBoss, a division of Red Hat, Inc.	5.1	February 2011	Free / Commercial support available	Yes	LGPL	
Jitterbit	Jitterbit	2.0	May 2008		Yes	JPL	
Magic xpi Integration Platform	Magic Software Enterprises	1.0	May 2012		No	proprietary	
Linx 5	Twenty57.com	5.0	Feb 2014	free beta	No	proprietary	
Openadaptor	The Software Conservancy	3.4.6	February 2011	Free	Yes	variant of MIT	
Mule ESB	MuleSoft	3.3.1	September 2012		Yes	CPAL	
OpenESB	OpenESB Community	2.0	May 2008	Supported By LogiCoy Inc. http://logicoy.com/support	Yes	CDDL	
OpenLink Virtuoso	OpenLink Software	4.5	2001	850 per value unit[6]	Yes	Dual (GPL or proprietary)	
Oracle BPEL Process Manager[7]	Oracle Corporation	10.1.2.0.2	23 January 2006	50,000 per processor[8]	No	proprietary	
Oracle Enterprise Service Bus[7]	Oracle Corporation	10.1.3.1	?	10,550[9]	No	proprietary	
PEtALS ESB	OW2 Consortium	3.1.3	July 2011	Free / Commercial support available	Yes	LGPL	
Sonic ESB	Progress Software	8.x	2011		No	proprietary	Enterprise Service Bus

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

3.3 Przykład

Interaktywny przykład najprostszej możliwej implementacji szyny integracyjnej opartej o wieloplatformowy system MOM typu open-source, **RabbitMQ**. Omówimy pojęcia:

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

http://rabbitmq.com

4 Service Oriented Architecture

Service Oriented Architecture (SOA) to sposób myślenia o architekturze systemu, w którym poszczególne podsystemy dostarczają dobrze określonych funkcjonalności biznesowych (gromadzenie danych, wyszukiwanie danych, przetwarzanie danych, mechanizmy wydruków, usługi komunikacyjne itp.) ale zachowują dużą autonomię jako osobne “aplikacje” w rozumieniu klasycznych stostów aplikacyjnych.

Architektura SOA zwykle oznacza więc też Szynę Integracyjną (lub inny component typu MOM), model komunikacji oparty o XML/JSON i/lub jakiś mechanism przetwarzania długotrwałych transakcji.

Architekturę typu SOA zwykle wybiera się dla dużych, heterogenicznych systemów, często wytwarzanych w różnym czasie i przez różnych dostawców.

Zaletą architektury SOA jest duża podatność na autonomiczne zmiany w ramach każdego z podsystemów, w tym całkowita wymiana pewnych podsystemów na inne/nowsze, bez utraty spójności systemu..

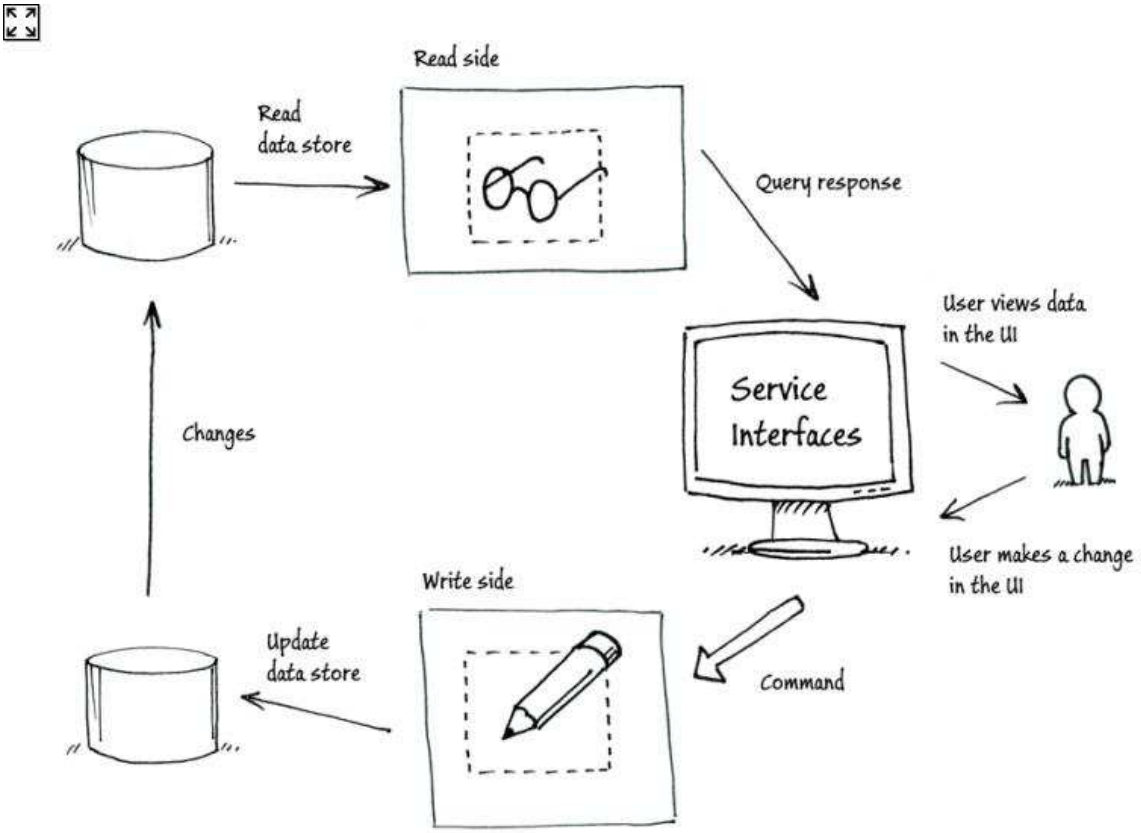
Wadą SOA jest duży nakład początkowy, niezbędny chociażby do zapewnienia odpowiedniej standaryzacji poszczególnych usług oraz implementacji całej infrastruktury typu MOM.

Współcześnie rozważa sie również tzw. **architekurę mikrousług** (Fowler: microservices, http://martinfowler.com/articles/microservices.html), w której podobne pojęcia jak w SOA pojawiają się na poziomie architektury pojedynczej aplikacji.

5 Command-Query Responsibility Segregation

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

Darmowy podręcznik CQRS od Microsoft Patterns & Practices : Exploring CQRS and Event Sourcing



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 (Command), który jest używany do zaktualizowania danych. Następnie w miejscu aktualizacji danych generuje się dodatkowe komunikaty o zmianach (Changes), które również za pomocą systemu kolejkowego przesyła się do części systemu odpowiedzialnej za generowanie widoków dla użytkowników.

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.

6 Literatura

1. Hohpe, Woolf – Enterprise Integration Patterns, http://www.eaipatterns.com/ - podręcznik wzorców integracyjnych, szczegółowo omawia wzorzec ESB i szereg jego podwzorców
2. A. Rotem-Gal-Ol – Wzorce SOA
3. Videla, Williams – RabbitMQ in Action
4. Microsoft Patterns & Practices – Exploring CQRS and Event Sourcing, https://msdn.microsoft.com/en-us/library/jj554200.aspx