Projektowanie architektury systemu, część II

Cele wykładu

- Wprowadzenie pojęć: wzorzec architektoniczny/styl architektoniczny
- Prezentacja wybranych styli architektonicznych

Projekt – przegląd

- Cel:
  - Propozycja architektury systemu oprogramowania z uwzględnieniem **wymagań funkcjonalnych i niefunkcjonalnych**
  - Prezentacja architektury z różnych perspektyw
- Uwaga:
  - Architektura systemu może być zbudowana w oparciu o istniejące wzorce architektoniczne/style architektoniczne i/lub architektury referencyjne

Wzorce

- Wzorzec – powszechnie stosowane rozwiązanie często występującego problemu zaadoptowane do specyficznego kontekstu
- Rodzaje wzorców (względem ich wielkości/ziarnistości):
  - Wzorce architektury (style architektoniczne)
  - Wzorce projektowe
  - Idiomy językowe

Style architektoniczne

- Styl architektoniczny:
  - (Często występująca w praktyce) specjalizacja elementów należących do widoku architektonicznego wraz ze zbiorem ograniczeń, jak ich używać
  - Rodzina architektur, która spełnia zdefiniowane dla stylu ograniczenia
  - Może być używany przez wiele systemów
  - System może bazować na wielu stylach

Przykłady styli architektonicznych

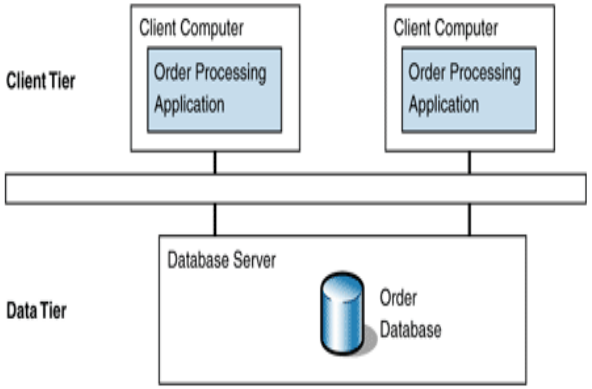
Problem:	Styl:
• Rozmieszczenie	• Klient/serwer, Architektura wielopoziomowa (3-Tier, N-tier)
• Struktura	• Architektura warstwowa, Architektura heksagonalna, Architektura czysta
• Integracja	• Różne wzorce integracyjne
• Interakcja	• MVC, MVP, MVVM

Architektura klient-serwer

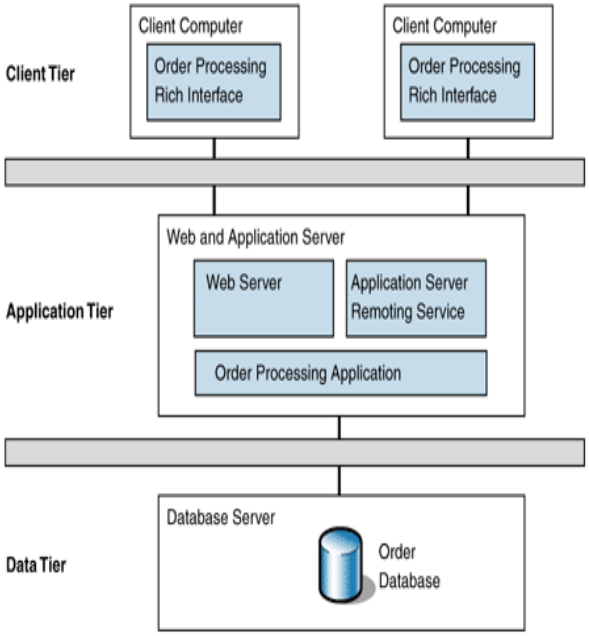
- Architektura systemu rozproszonego
- Składowe: Klient (wiele), Serwer
  - Połączenia:
    - Jednego typu: żądanie/odpowiedź
    - Asymetryczne (tylko klient wykonuje żądania)
    - Komunikacja zwykle synchroniczna
- Wariacje:
  - N-tier (architektura n-poziomowa)
  - Client-Queue-Client; kolejka jest zarządzana przez serwer
  - P2P
- Zalety:
  - Wysokie bezpieczeństwo
  - Łatwa pielęgnacja (zwłaszcza strony serwerowej)
- Wady:
  - Pojedyncze serwery mogą powodować wąskie gardła → negatywny wpływ na wydajność i skalowalność

Architektura klient-serwer i architektura wielopoziomowa

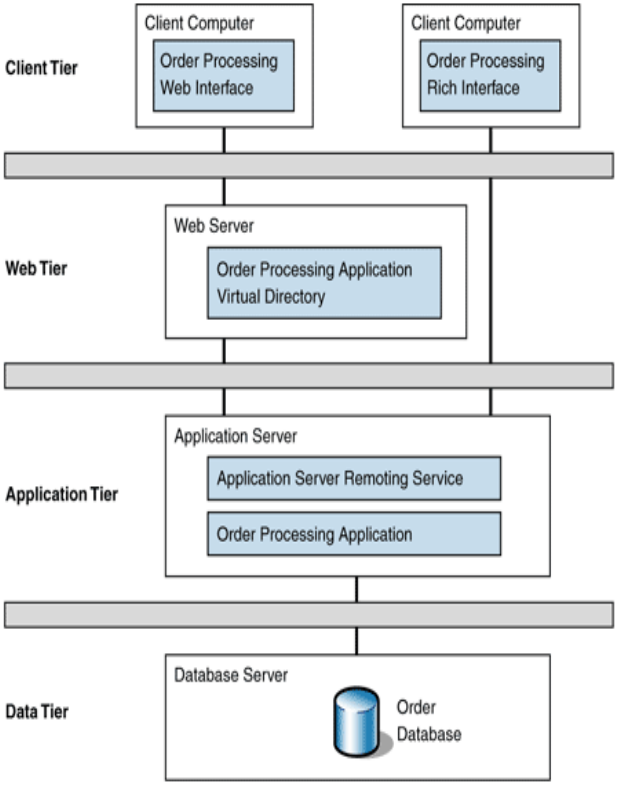
Architektura dwupoziomowa (Two tired) [5]



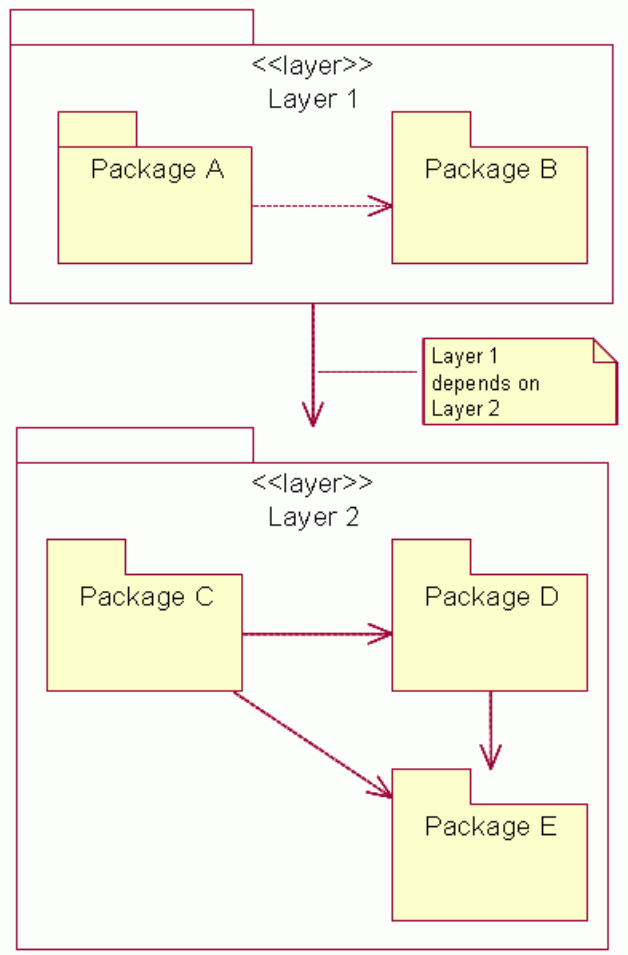
Architektura trzypoziomowa



Architektura wielopoziomowa



Architektura warstwowa



Wzorzec stosowany często w połączeniu z:

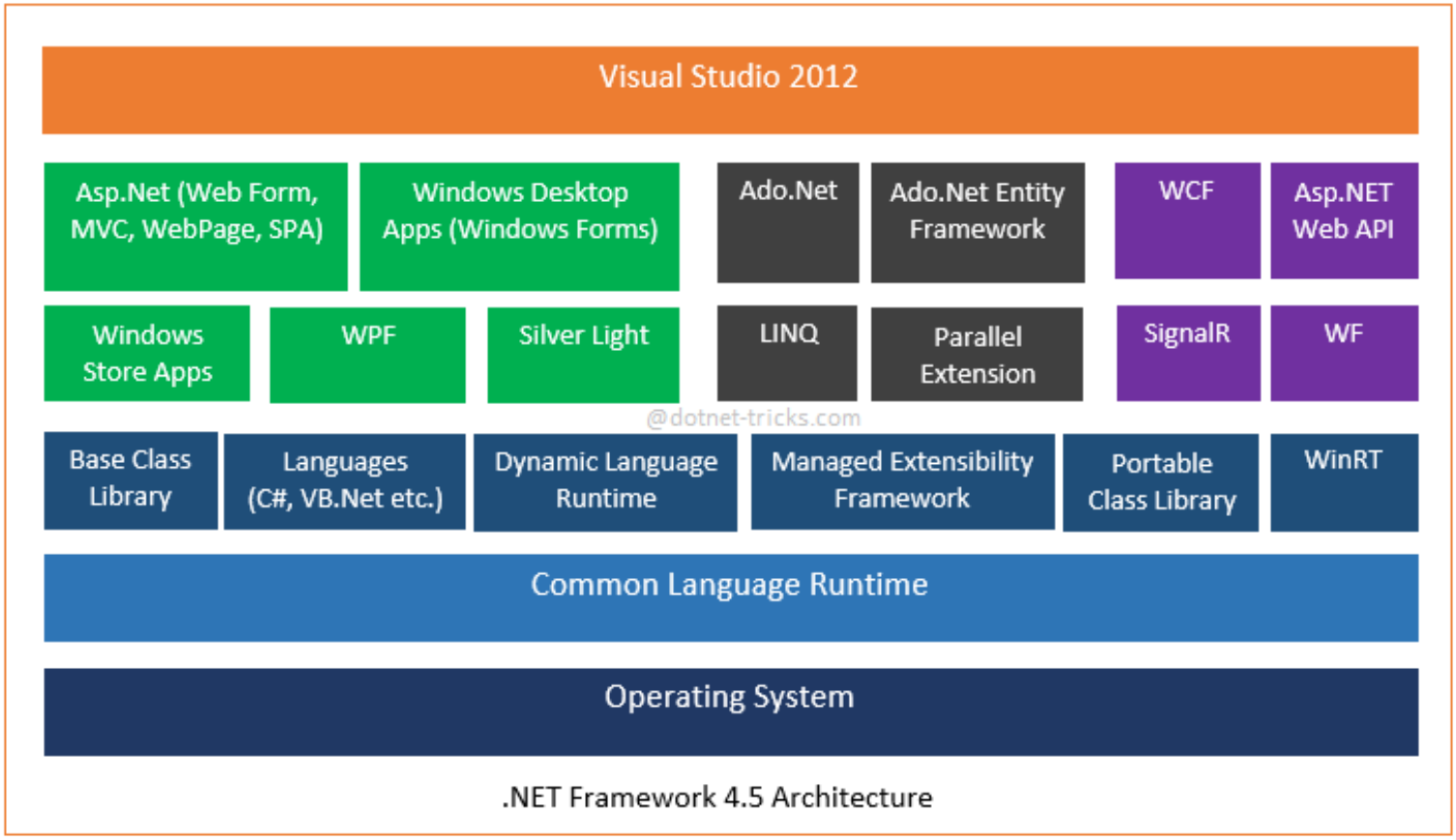
- Zależność od interfejsów zamiast bezpośrednio od elementów warstwy niższej
- Wzorcem Fasady – pojedynczy punkt dostępu do podsystemu
- Wzorcem DTO (Data Transfer Object) – kontener przenoszący dane między warstwami (abstrakcje jednego poziomu są tłumaczone na abstrakcje drugiego poziomu)

Dwie formy interakcji:

- Top down (zwykle przesyłanie komunikatów/wywoływanie metod)
- Bottom-up (sterowane zdarzeniami)

Architektura warstwowa, przykład 1

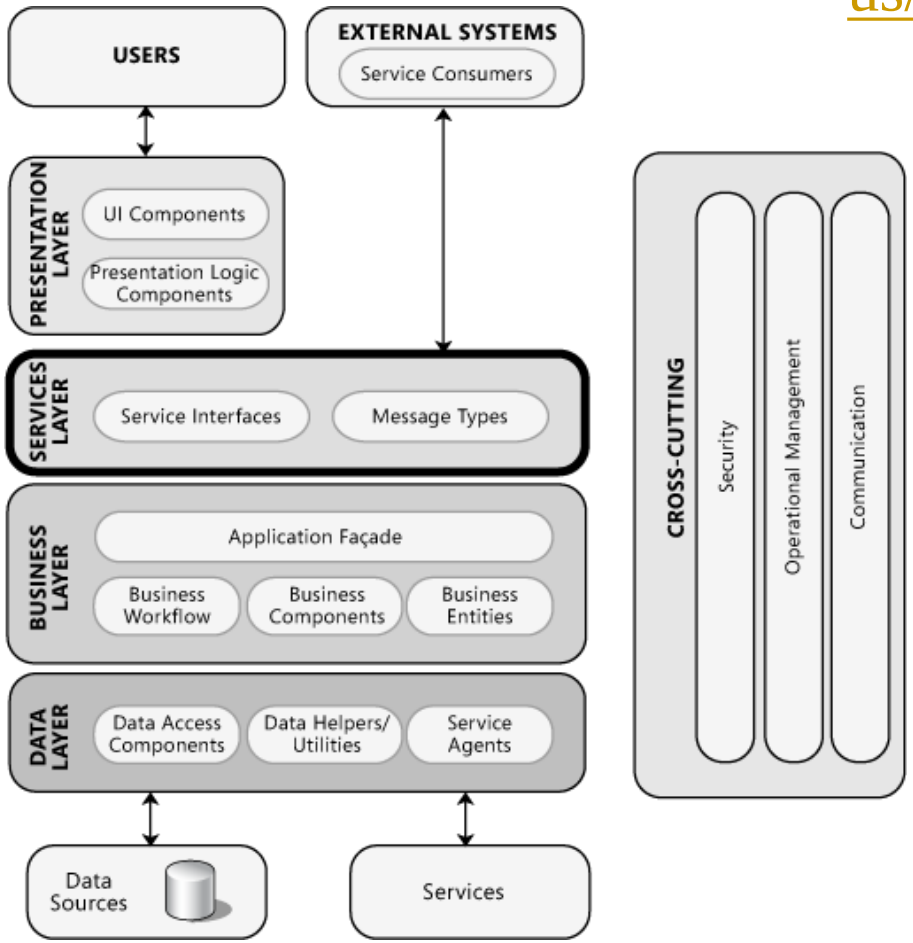
- Definicje frameworków programistycznych



http://www.dotnet-tricks.com/Tutorial/netframework/NcaT161013-Understanding-.Net-Framework-4.5-Architecture.html

Architektura warstwowa, przykład 2

http://msdn.microsoft.com/en-us/library/ee658109.aspx



Zasady projektowania:

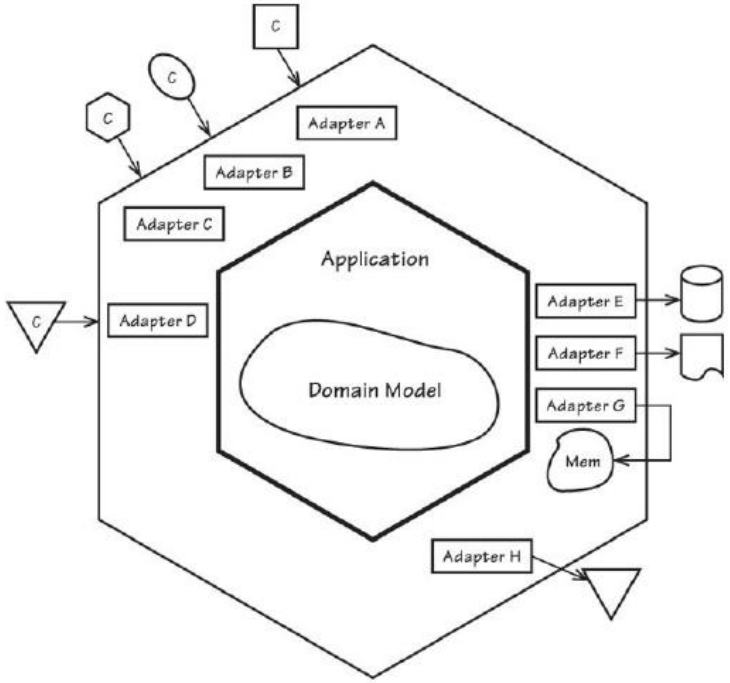
1. Określ warstwy, których potrzebujesz na podstawie ich odpowiedzialności
2. Podejmij decyzje co do rozmieszczenia fizycznego warstw i określ interfejsy
3. Zidentyfikuj „zagadnienia przecinające” i sposób ich obsługi

Architektura warstwowa

- Zalety:
  - Zarządzalność – warstwy reprezentują dekompozycję funkcjonalną (separacja pojęć)
  - Możliwość ponownego użycia warstw lub ich elementów
  - Łatwość testowania
- Wady:
  - Obniżona wydajność – komunikacja między warstwami

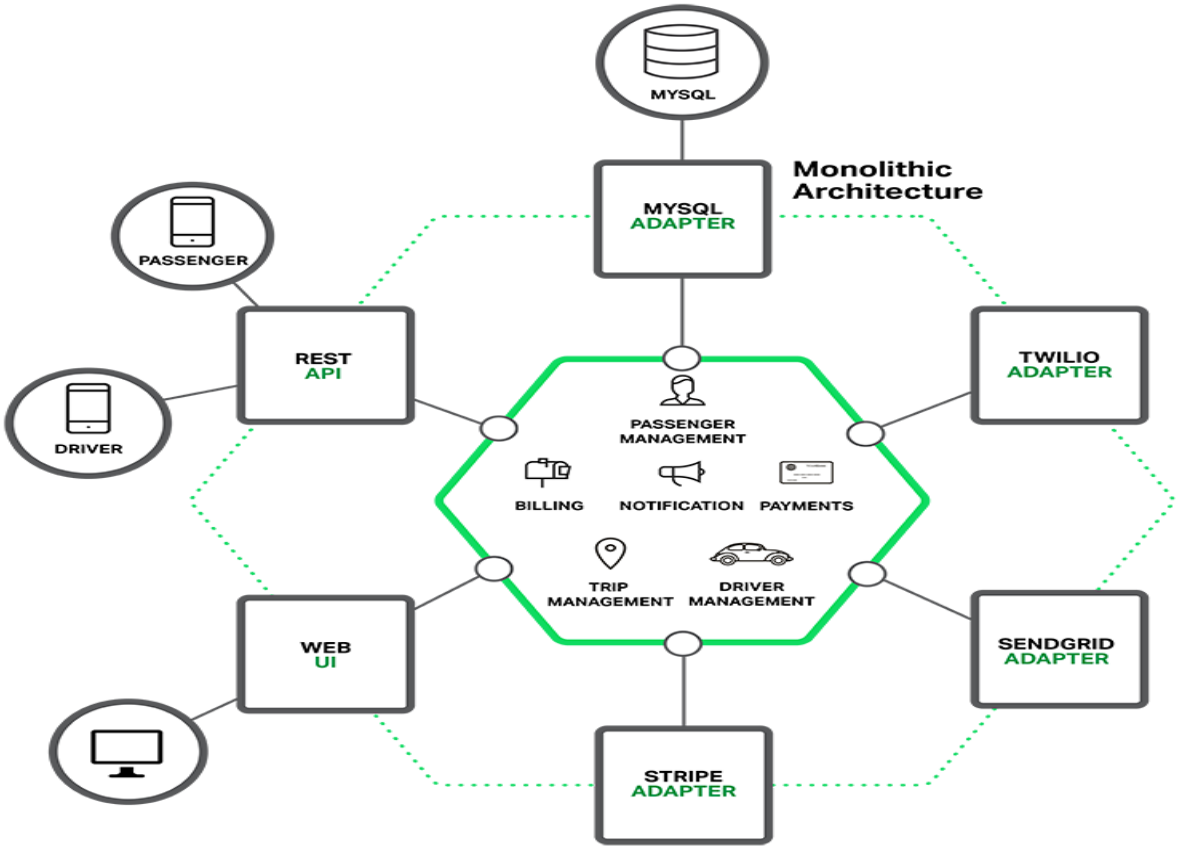
Architektura heksagonalna („Hexagonal architecture”, ‘’Ports & Adapters’’)

- Propozycja A. Cockubrn’a.
- Cel: oddzielenie logiki biznesowej od elementów specyficznych dla platformy (interfejs użytkownika, baza danych).
- Komunikacja z otoczeniem z wykorzystaniem adapterów komunikujących się na portach (API)
- Dla danego portu (np. komunikacja z bazą) można mieć wiele adapterów (komunikacja z bazą SQL, z bazą plikową, z bazą w pamięci komputera)

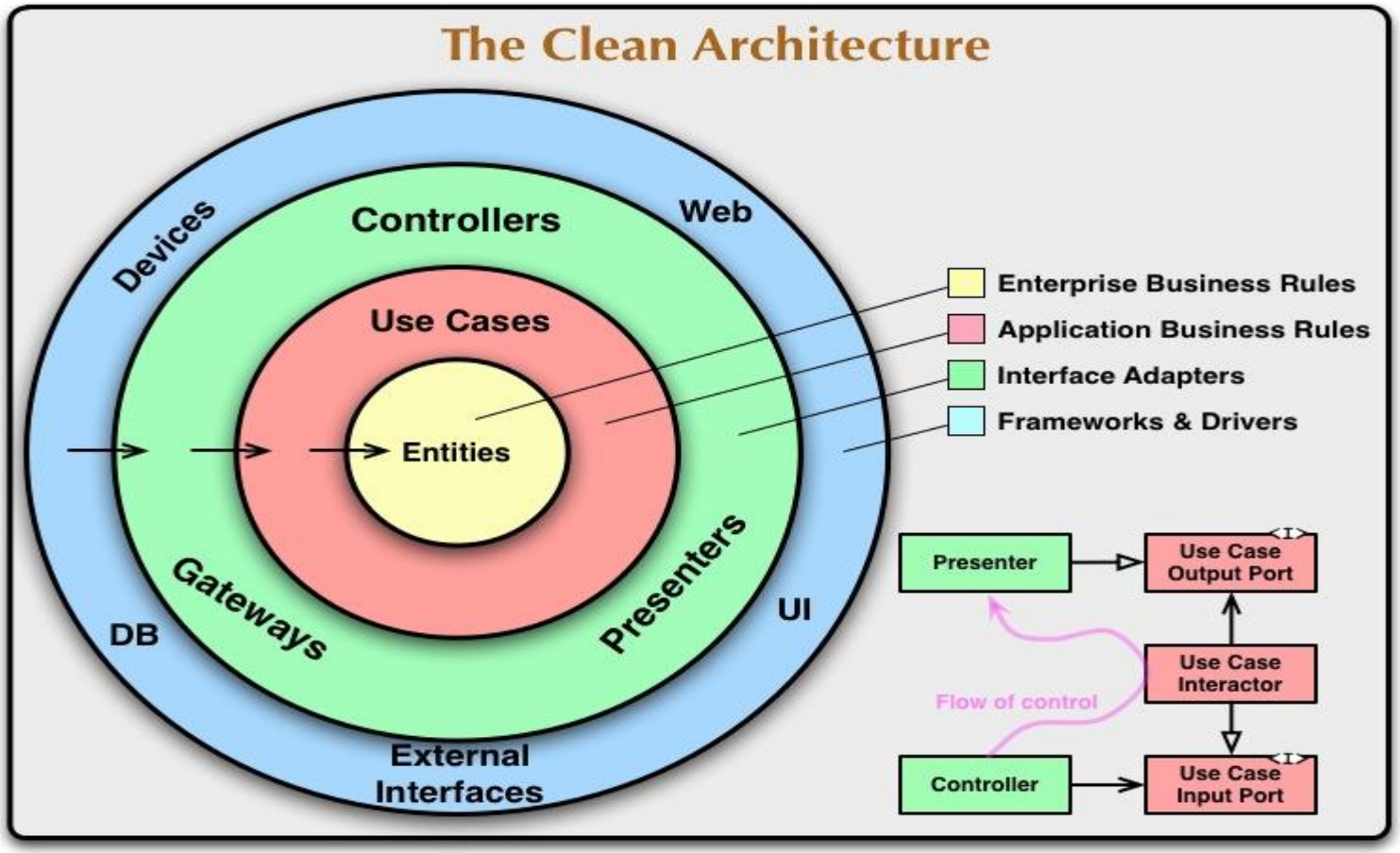


http://alistair.cockburn.us/Hexagonal+architecture

Architektura heksagonalna, przykład



Architektura „czysta” (szczególny przypadek architektury heksagonalnej)



https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html

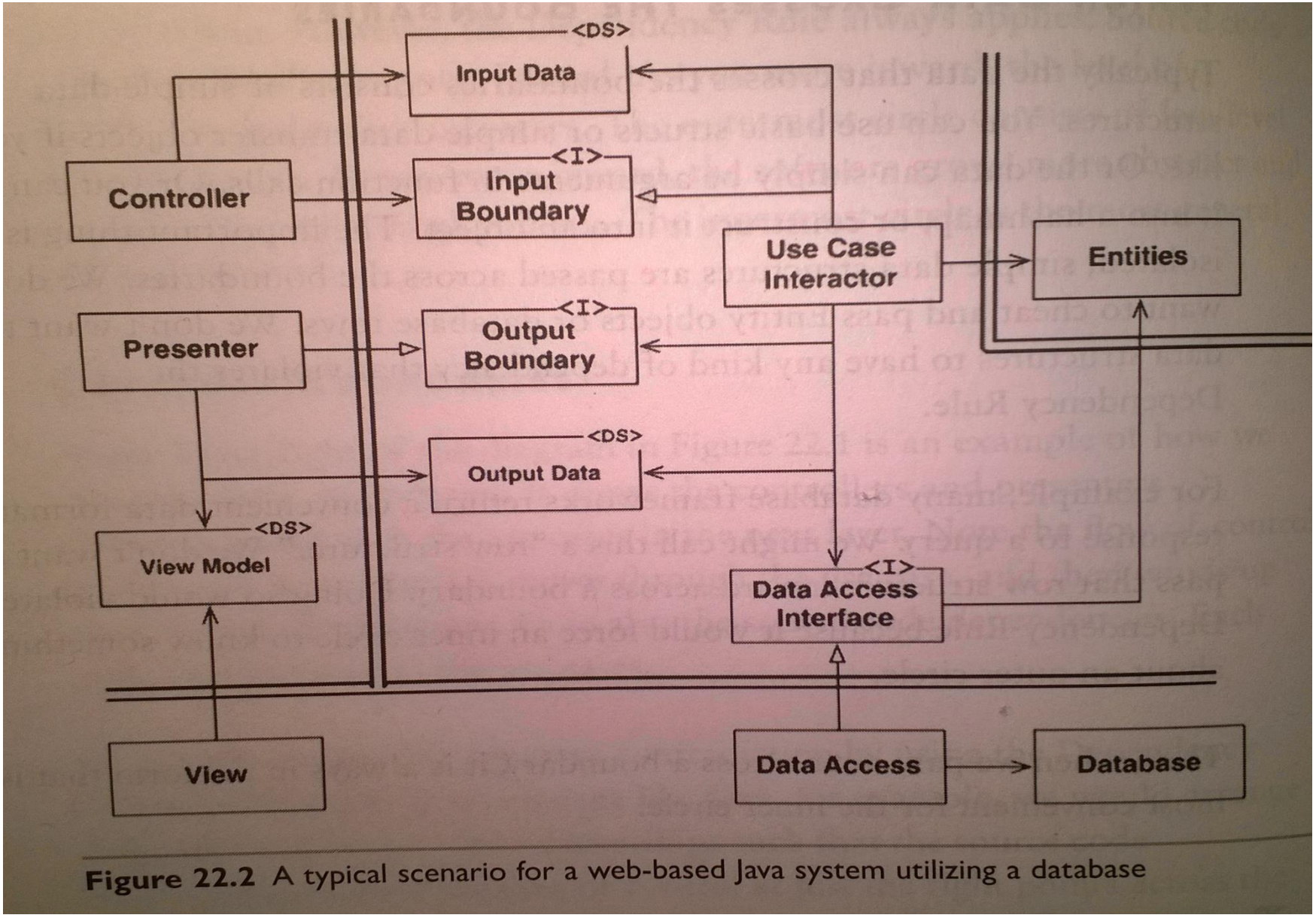


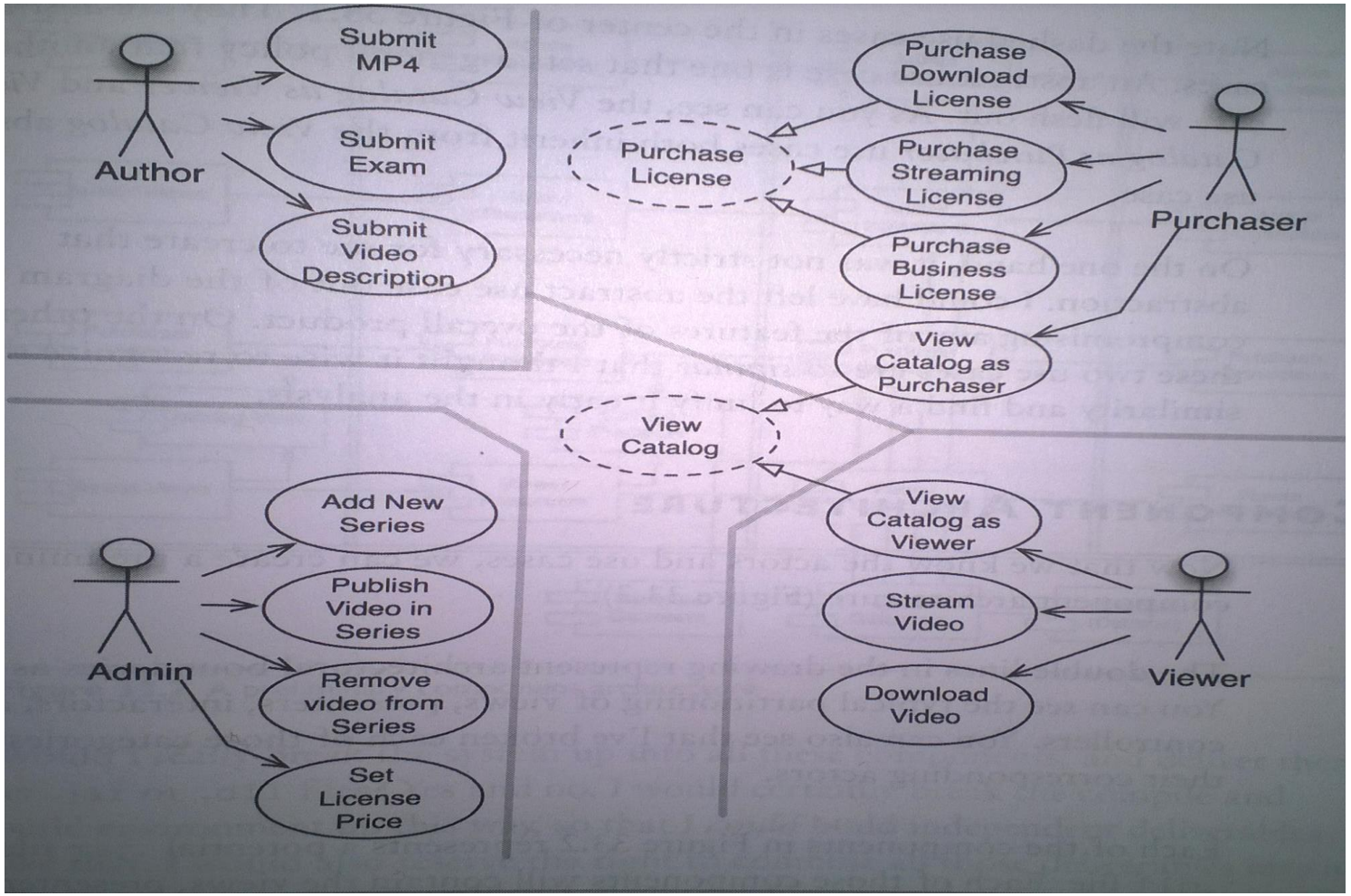
Figure 22.2 A typical scenario for a web-based Java system utilizing a database

R. C. Martn, „Clean Architecture”

Architektura „czysta”

- Zalety (charakterystyka):
  - Niezależność od frameworków i UI (części wewnętrzne)
  - Reguły biznesowe nie wiedzą nic o zewnętrznym świecie → Testowalność
  - Niezależność od bazy – można łatwo zmienić bazę

Architektura „czysta” - przykład



R. C. Martn, „Clean Architecture”

Architektura „czysta” – przykład, c.d.

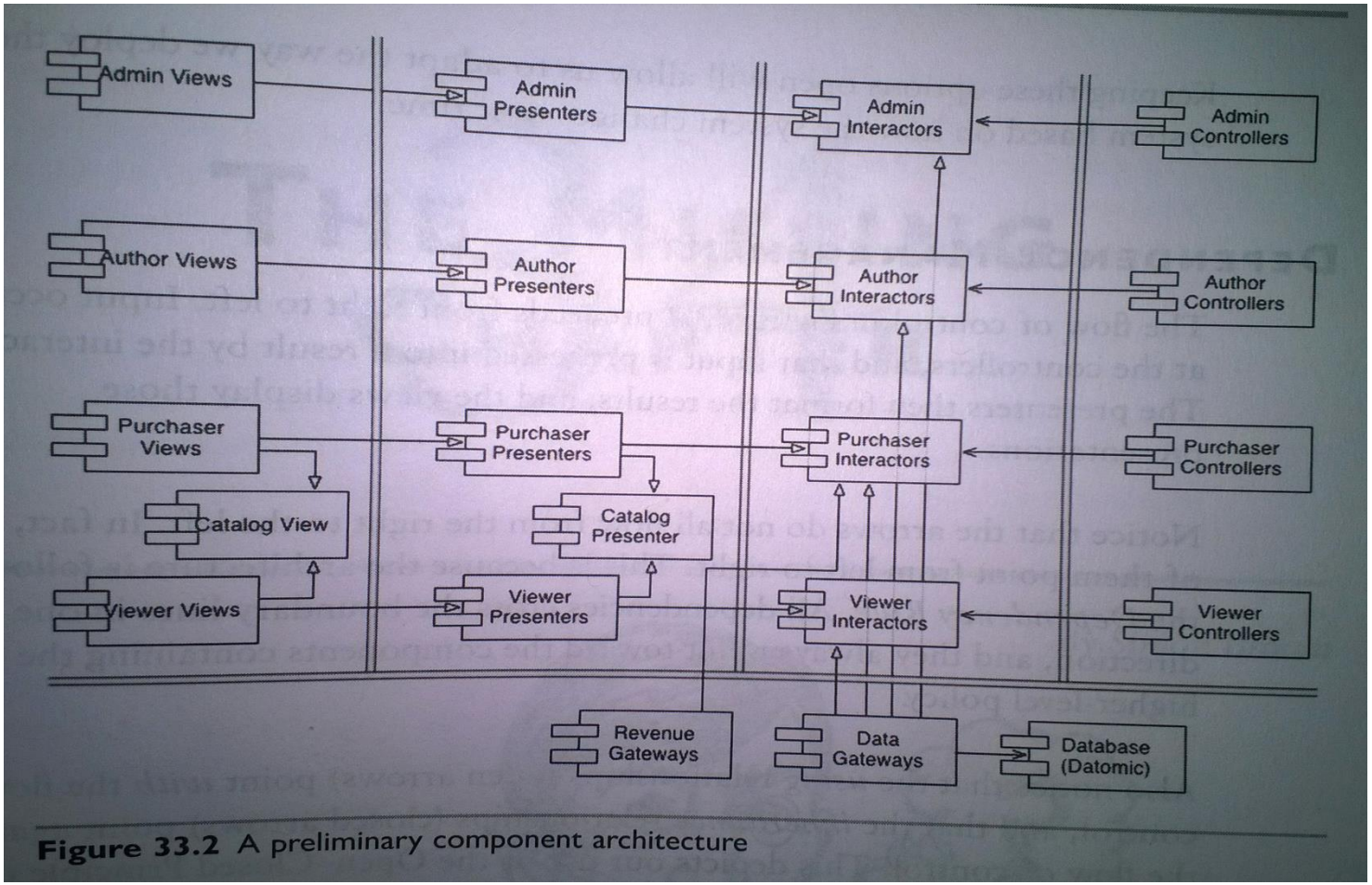
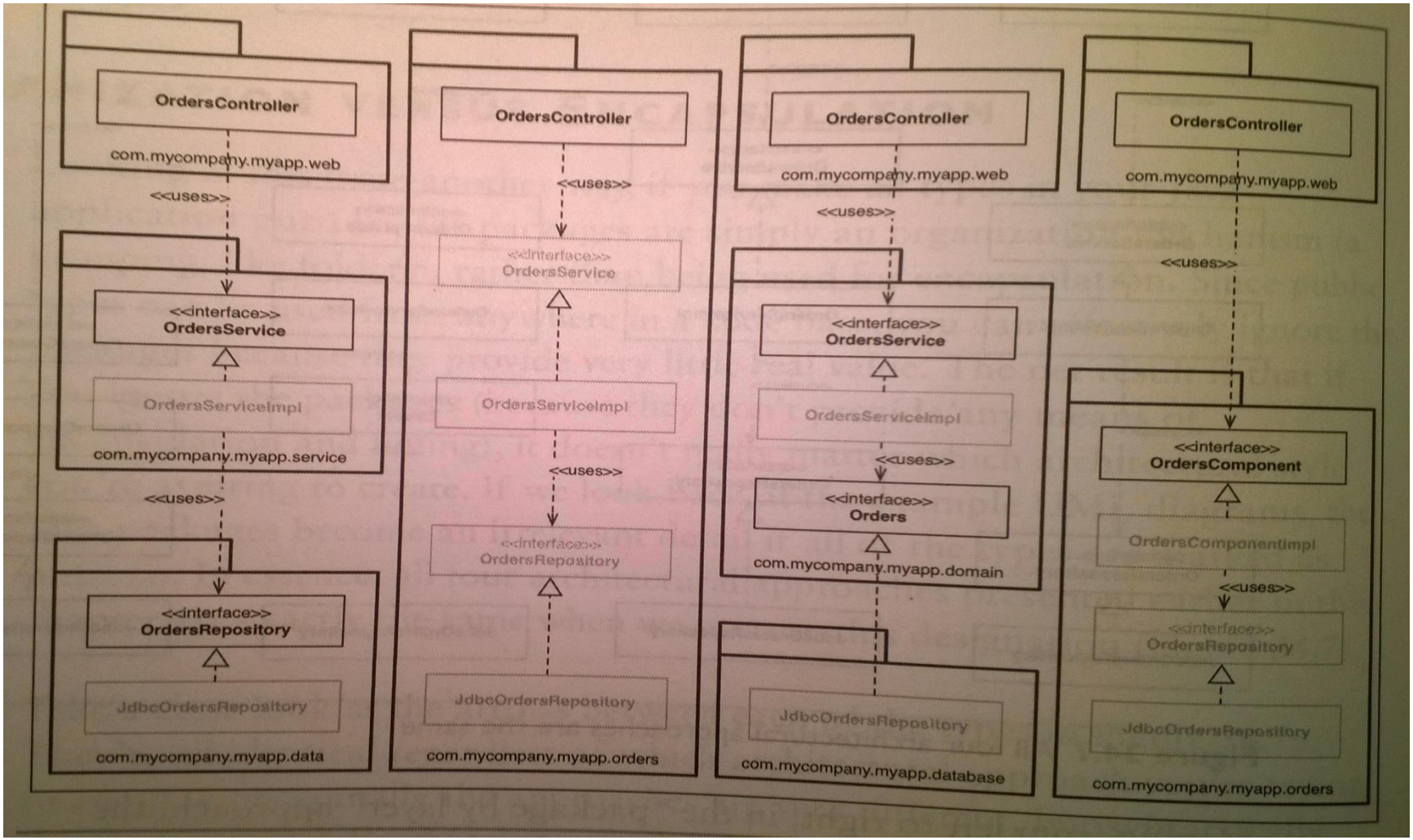


Figure 33.2 A preliminary component architecture

R. C. Martn, „Clean Architecture”

Różne sposoby pakietyzacji



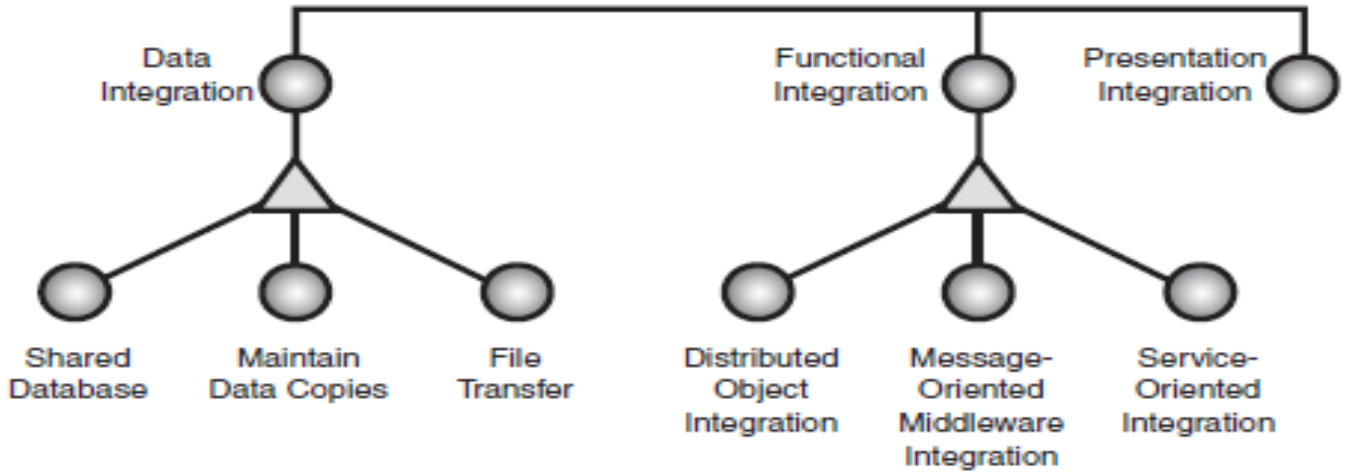
R. C. Martn, „Clean Architecture”

Integracja systemów

- Integracja – proces połączenia składowych (systemów) w jeden system (system systemów, SoS)
- Motywacja – efekt synergii
- Charakterystyka zintegrowanego systemu:
  - Niezależność komponentów
  - Dostarcza funkcji nieobecnych w żadnej składowej
  - Rozproszenie komponentów

Integracja systemów, cd.

- Klasyfikacja integracji ze względu na położenie mechanizmów służących do integracji:
  - Integracja warstwy prezentacji
  - Integracja funkcjonalna
  - Integracja danych

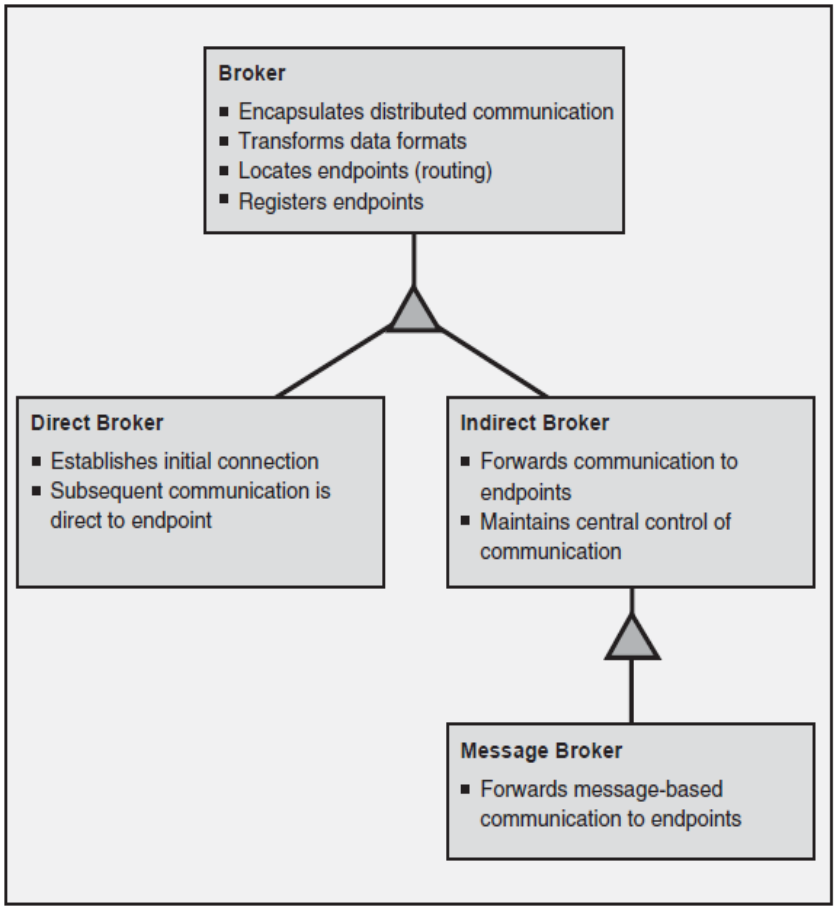


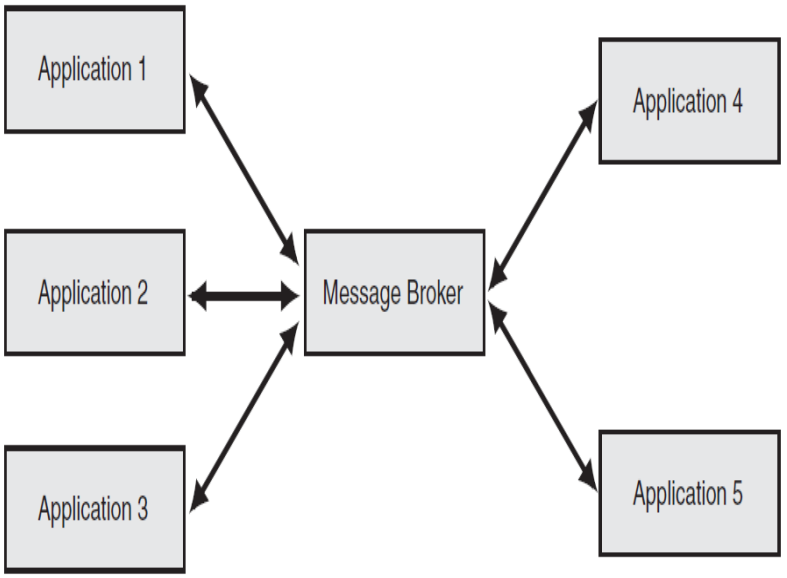
- Inne wzorce integracyjne:
  - Filtry i potoki (Pipes and Filters)
  - Bramy (Gateway)

Integracja systemów, cd.

- Integracja funkcjonalna:
  - Integracja obiektów rozproszonych (API): COM+, RPC
  - Integracja za pomocą śródprogramu zorientowanego na komunikaty: Broker komunikatów: bezpośredni, pośredni, wiadomości; szyna komunikatów
  - Integracja z wykorzystaniem usług: SOA, mikro-usługi
- Zalety:
  - Enkapsulacja (komunikacja z dobrze zdefiniowanym API)
  - Elastyczność
  - Duże prawdopodobieństwo sprawdzenia reguł biznesowych
- Wady:
  - Ograniczony do istniejącego API
  - „Odsłonięcie” warstwy logiki biznesowej
  - Uzależnienie od specyficznej technologii

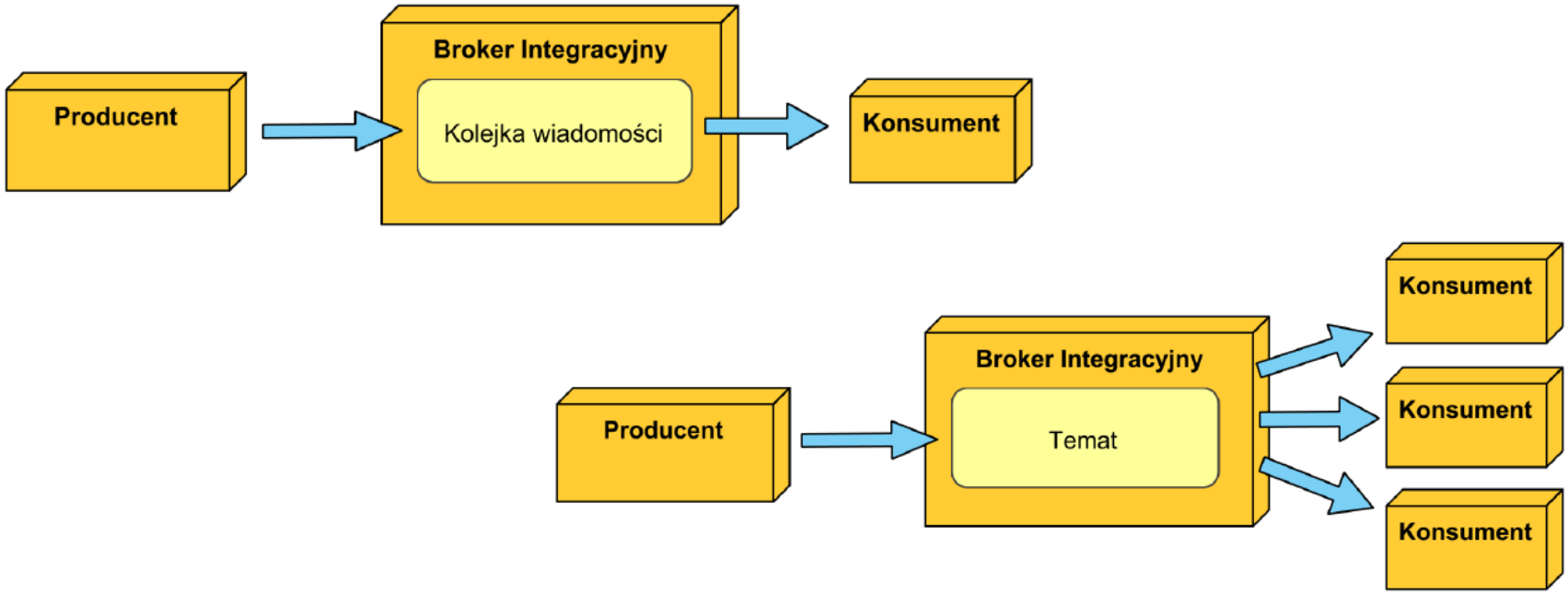
Broker


Broker
Encapsulates distributed communication
Transforms data formats
Locates endpoints (routing)
Registers endpoints
Direct Broker
Establishes initial connection
Subsequent communication is direct to endpoint
Indirect Broker
Forwards communication to endpoints
Maintains central control of communication
Message Broker
Forwards message-based communication to endpoints

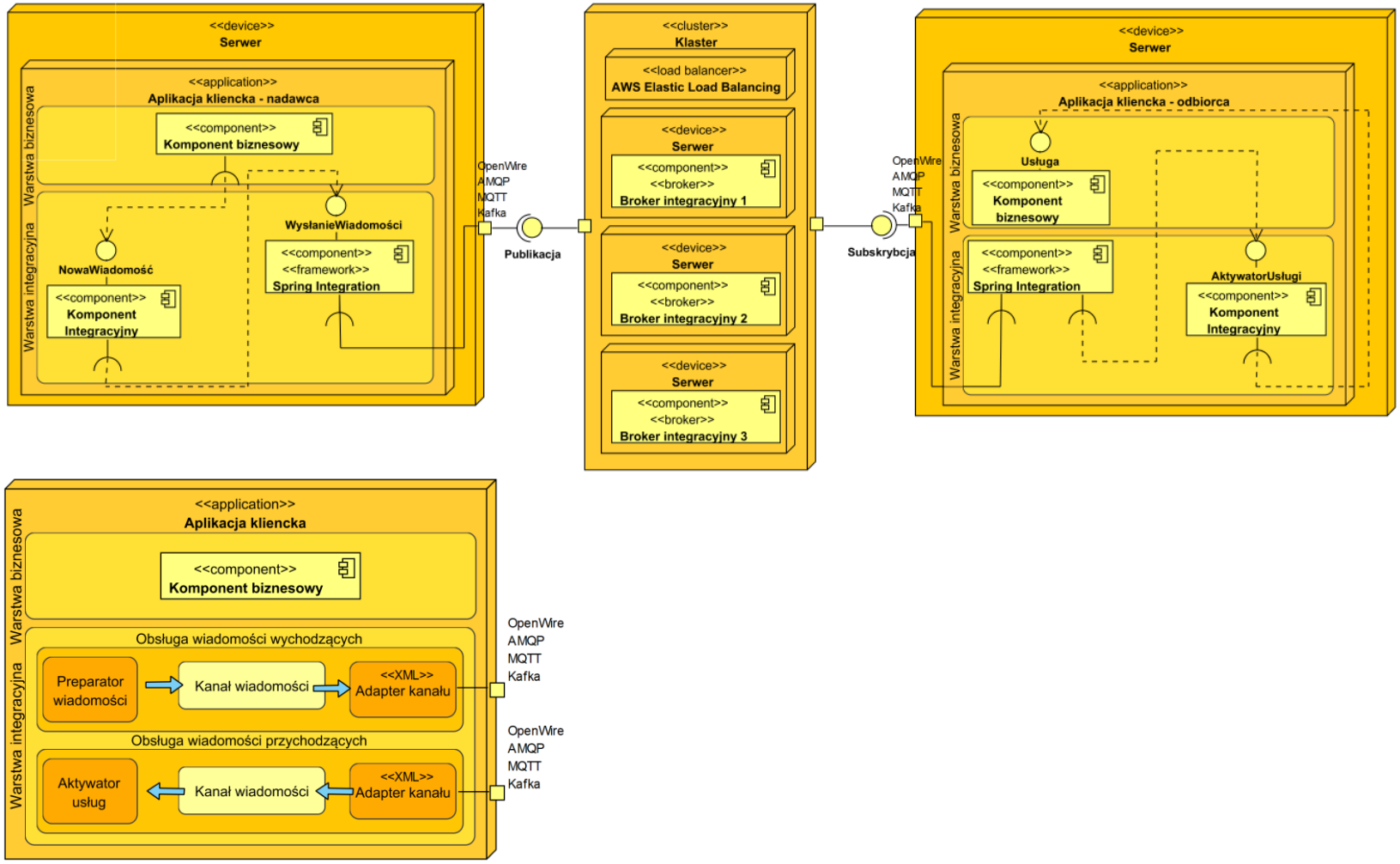

Application 1
Application 2
Application 3
Message Broker
Application 4
Application 5

Przykłady brokerów integracji

- Spring Integration + RabbitMQ, ActiveMQ, Apache Kafka
- Podstawowe modele przesyłania komunikatów:
  - Punkt-punkt
  - Publikuj/Subskrybuj

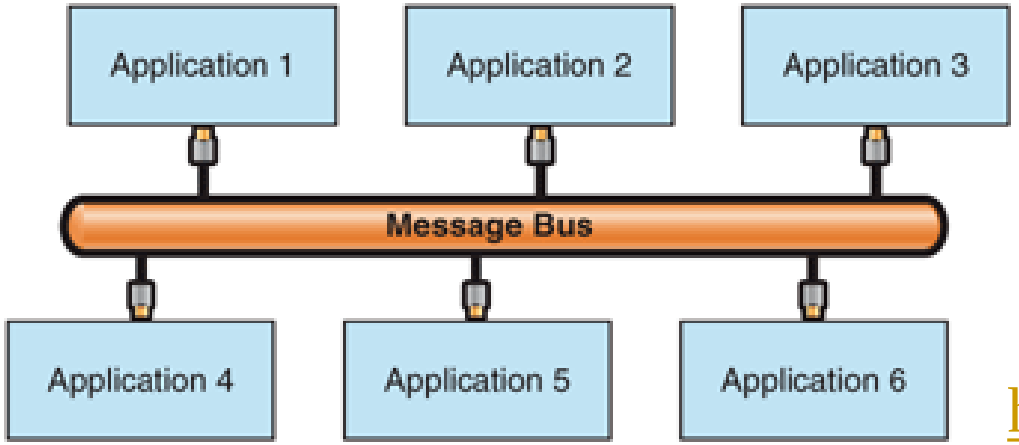


Przykład zastosowania brokera



Szyna komunikatów (Message Bus)

- Architektura integrująca wiele aplikacji (być może działających na innych komputerach i platformach), komunikujących się z wykorzystaniem specjalnego kanału komunikacji – szyny komunikatów (middleware); uogólnienie brokera integracji



http://msdn.microsoft.com/en-us/library/ff647328.aspx

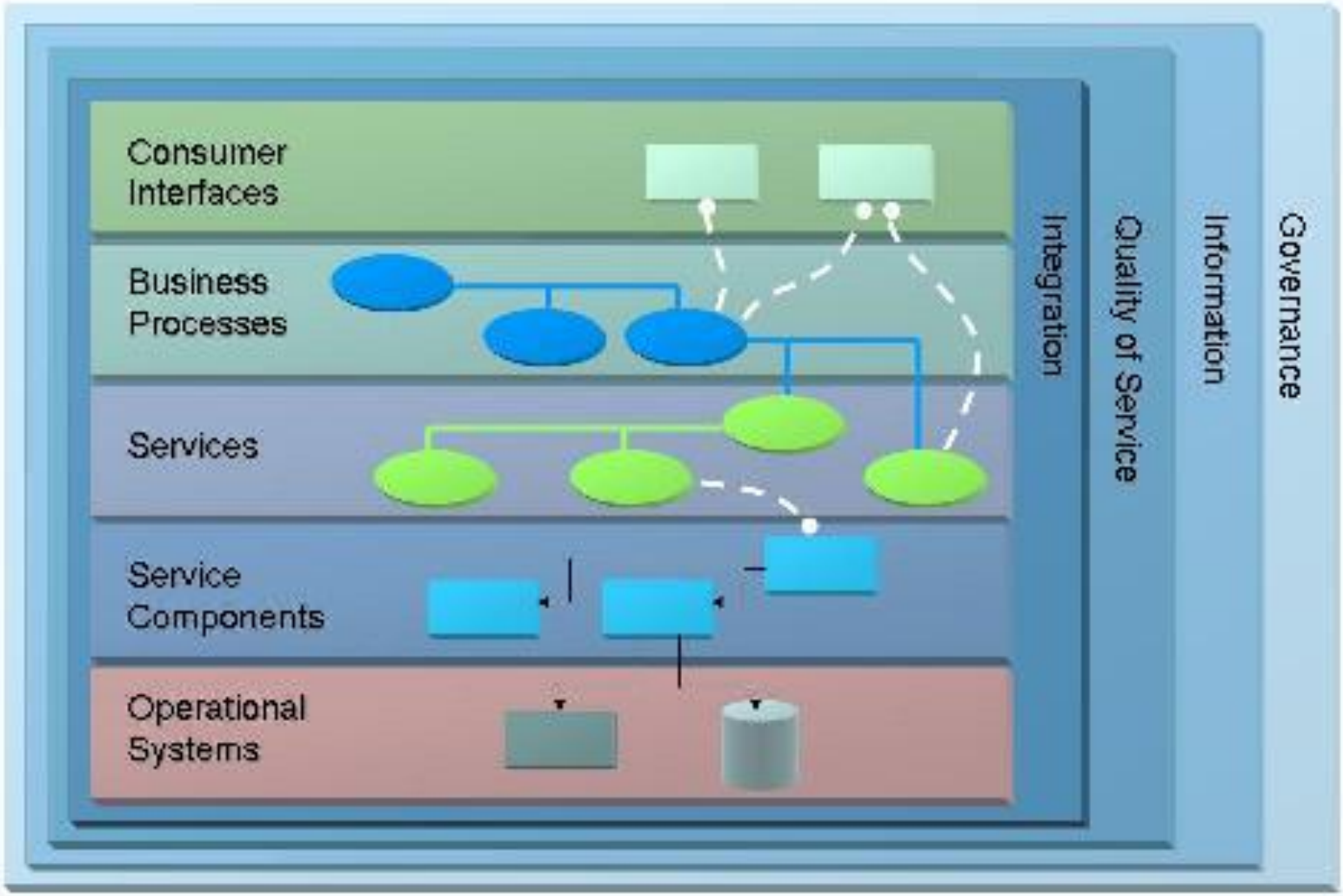
Szyna komunikatów (Message Bus)

- Kluczowe zalety:
  - Zwiększona modyfikowalność (dodanie/usunięcie aplikacji nie wpływa na pozostałe)
  - Zmniejszona złożoność aplikacji (nadawcy nie muszą wchodzić w interakcje z różnymi odbiorcami)
  - Zwiększona skalowalność (łatwo dodawać nowe aplikacje do szyny)
- Kluczowe wady:
  - Złożoność integracji
  - Zmniejszona integrowalność – wszystkie aplikacje muszą używać interfejsu szyny i są od niego zależne

Architektura SOA (Service-oriented architecture)

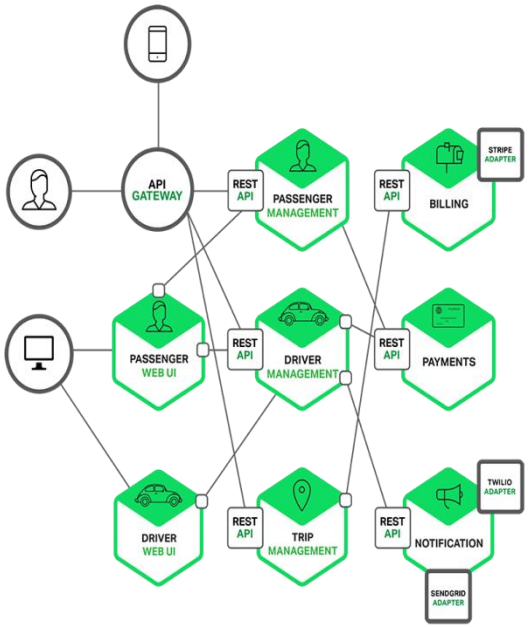
- Architektura, zbudowana z rozproszonych usług wymieniających komunikaty zgodnie ze zdefiniowanymi kontraktami oraz klientów (które używają usług)
- Kluczowe zasady SOA:
  - Usługi są autonomiczne
  - Usługi mogą być umieszczane na różnych serwerach
  - Usługi są luźno powiązane
  - Usługi współdzielą kontrakty, nie klasy
  - Kompatybilność w pracy usług jest uzyskiwana dzięki politykom (definicja cech takich jak protokół, szyfrowanie etc.)
- Kluczowe zalety SOA:
  - Wielokrotne używalność usług
  - Interoperacyjność
- Kompromis projektowy:
  - Wydajność
- Typowe technologie: WSDL, SOAP (XML), REST (JSON)

Architektura SOA, c.d.



Mikrousługi (Microservices)

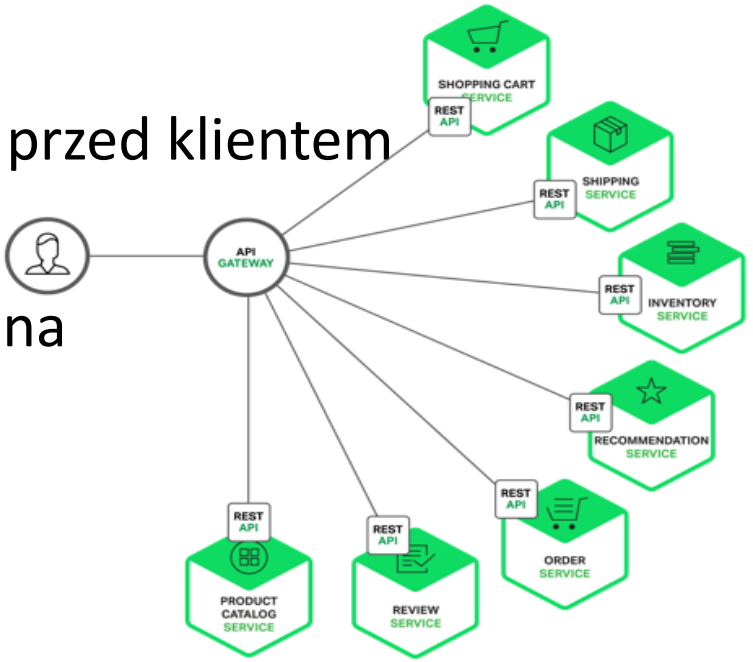
- Mikrousługa – jest mini-aplikacją, odpowiadającą za pewien obszar funkcjonalny, z własną architekturą i dziedzinową bazą danych
- Mikrousługa – może eksponować API, wykorzystywane przez inne usługi
- W środowisku produkcyjnym można mieć wiele instancji danej usługi
- Mikrousługi – to „lekkie” SOA, wykorzystujące zwykle prostsze protokoły komunikacyjne



- Usługi mogą komunikować się asynchronicznie (np. JMS) lub synchronicznie (np. REST)

Mikrousługi (Microservices), c.d.

- Często stosowane wzorce – Gateway API (Brama)
- Brama – odpowiada za przekierowanie żądań do innych usług, tłumaczenie protokołów i agregację wyników
- Zalety:
  - Ukrycie wewnętrznej struktury przed klientem
- Wady:
  - Brama musi być wysoce dostępna



Mikrousługi (Microservices), c.d.

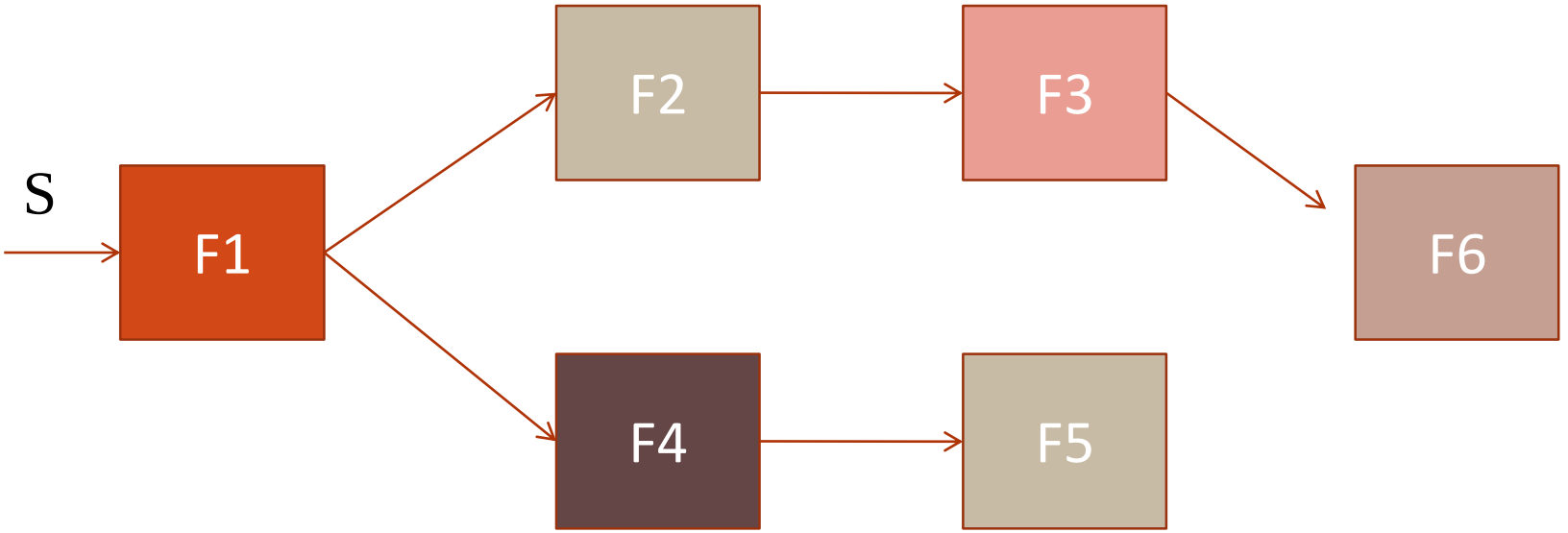
- Zalety:
  - Zarządzanie złożonością – podejście wymusza modularyzację na zbiór serwisów (architektura łatwiejsza w pielęgnacji i rozwoju)
  - Usługi mogą być zbudowane w oparciu o różne technologie, pod warunkiem implementacji kontraktów
  - Usługi mogą być niezależnie instalowane i wersjonowane
- Wady:
  - Partycjonowanie bazy danych (trudność implementacji transakcji biznesowych)
  - Bardziej złożone testowanie
  - Bardziej złożone wdrożenie, niż w przypadku aplikacji monolitycznych

Filtry i potoki (Pipes and filters)

- Komponenty (filtry):
  - Stopniowo dokonują transformacji danych wejściowych w dane wyjściowe
  - Transformacja może polegać na:
    - Dodaniu danych
    - Uściśleniu lub usunięciu danych
    - Zmianie reprezentacji danych
  - Mogą działać niezależnie (być może współbieżnie)
  - Zwykle są bezstanowe
  - Filtry:
    - Aktywne
    - Pasywne
- Połączenia (potoki):
  - Przenoszą dane z wyjścia filtra A do wejścia filtra B
  - Jednokierunkowe, nie zmieniają danych ani ich porządku
- Typowe zastosowania:
  - Przetwarzanie tekstów/obrazów
  - Przetwarzanie sygnałów

Filtry i potoki, c.d.

- F6(F3(F2(F1(s)))) – filtry typu „pull”
- F5(F4(F1(s)))



Filtry i potoki, c.d.

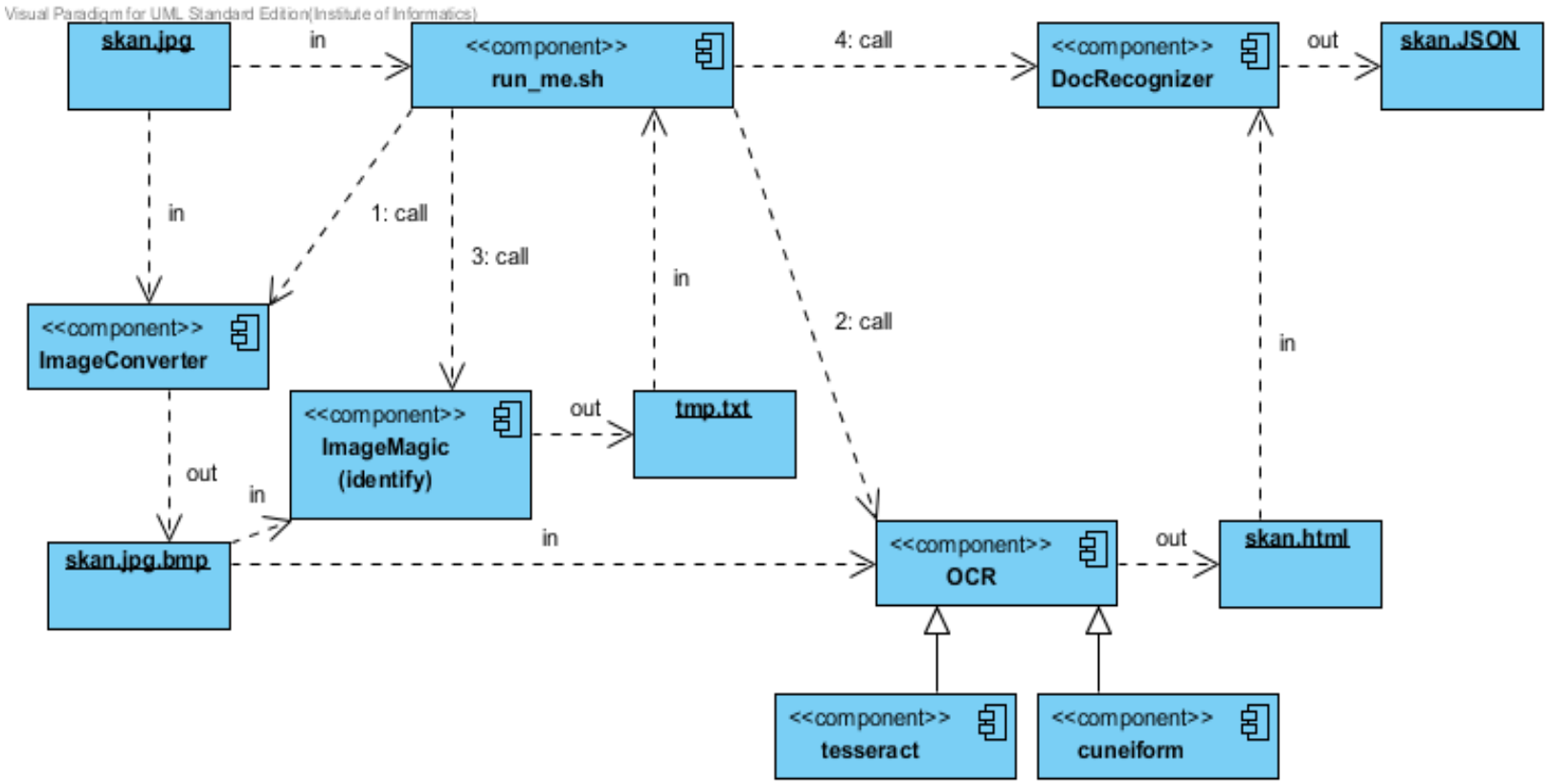
- Przetwarzanie wsadowe, jako szczególny przykład filtrów i potoków:
  - Komponenty – niezależne programy; jeden program musi się zakończyć, aby rozpocząć kolejny
  - Potok – sekwencja kroków obliczeniowych wykonywanych przez program wsadowy
  - Przykład (C++): preprocesor → kompilator → linker
  - Przykład (TeX) : (plik.tex) → tex → (plik.dvi) → divps → (plik.ps)

Filtry i potoki, c.d.

- Kiedy używać tego stylu:
  - Zadania wymagają dostępności danych
  - Dane są przenoszone od procesu do procesu
- Zalety:
  - Łatwość ponownego wykorzystania i rekonfiguracji filtrów
  - Można testować filtry osobno (mogą być tworzone niezależnie, być noże w innych technologiach)
  - Można rozważyć zrównoleglenie filtrów
- Wady:
  - Negatywny wpływ na wydajność przy przetwarzaniu wsadowym

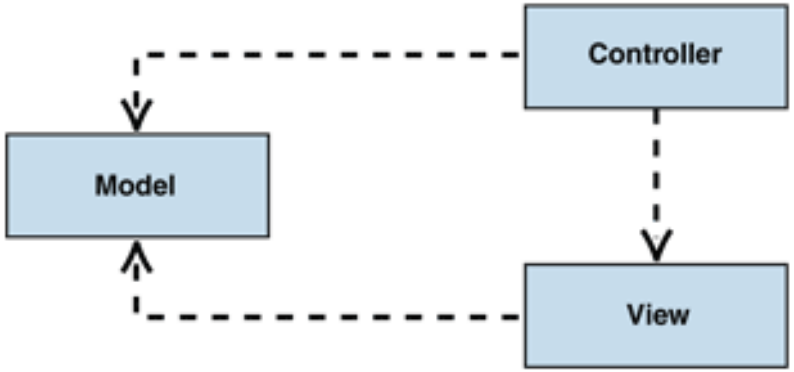
Filtry i potoki - przykład

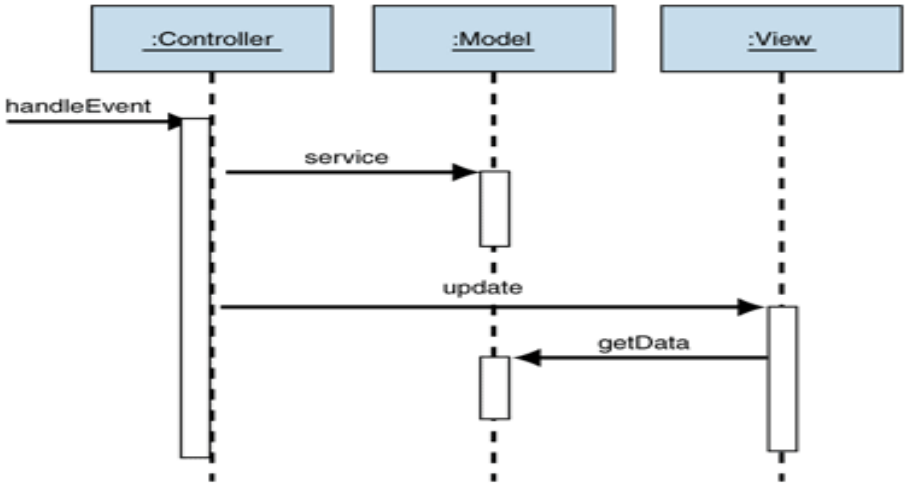
- Filtry: ImageConverter → OCR → DocRecognizer ; ImageConverter → ImageMagic (osobne programy napisane w innych j. programowania)
- Run_me: skrypt organizujący potok (przetwarzanie wsadowe)



MVC

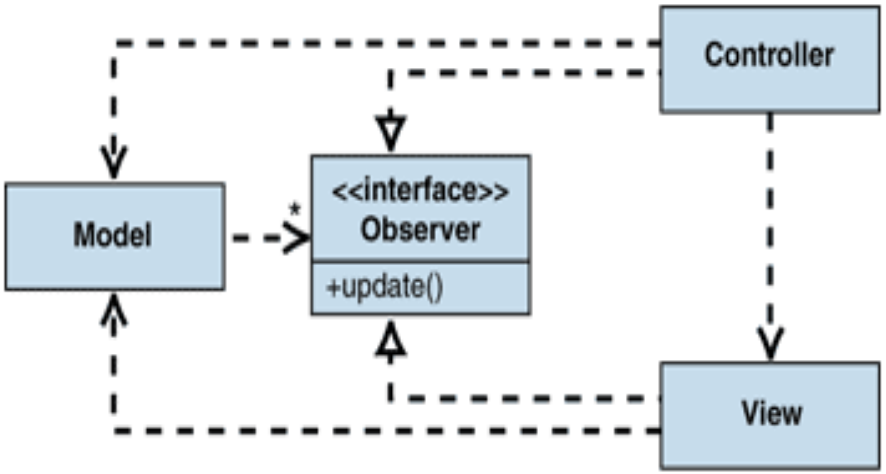
- Model pasywny [5]

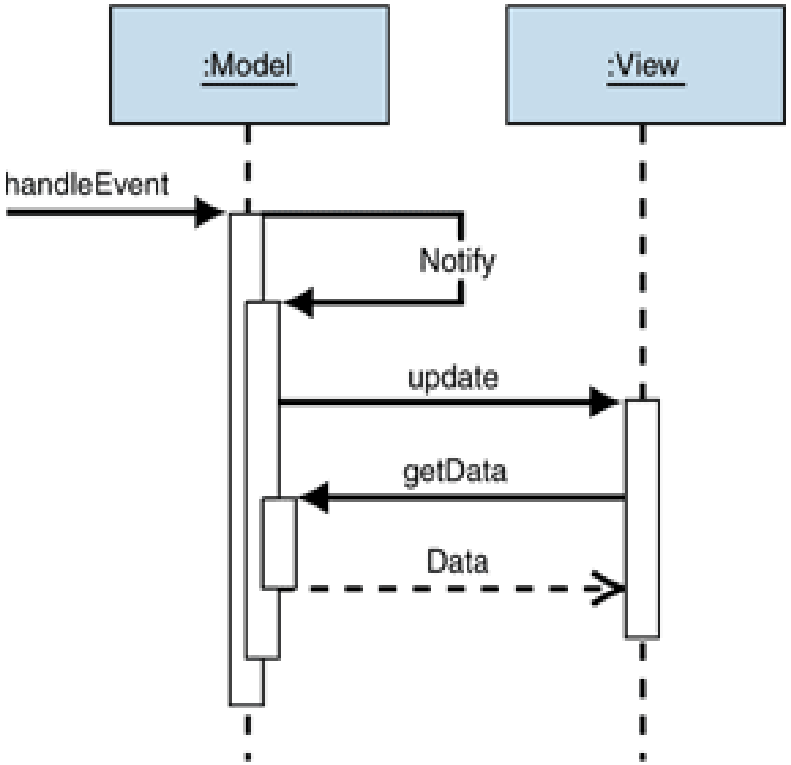




MVC, c.d.

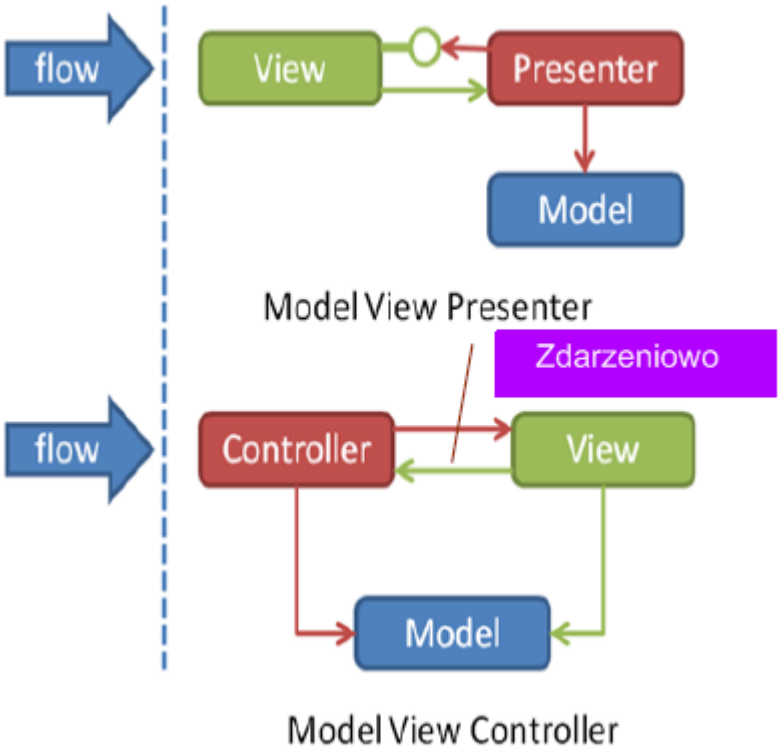
- Model aktywny [5]





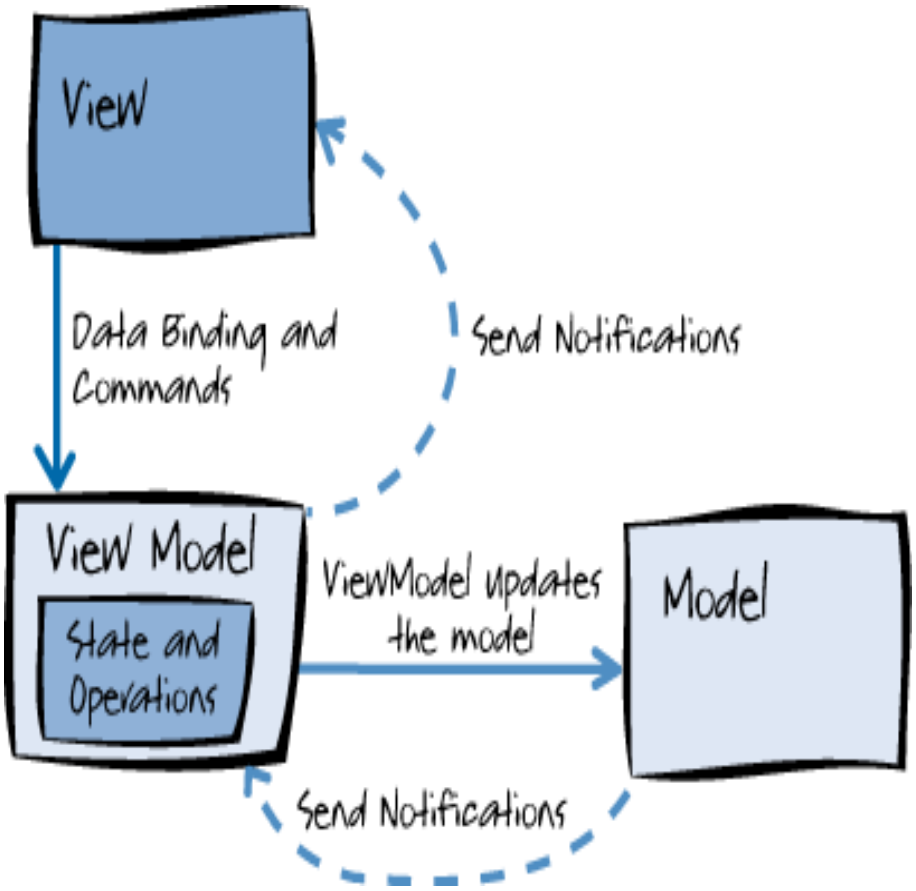
MVP (Model View Presenter)

- Model – pełni dokładnie tę samą funkcję, co w MVC
- Widok – graficzny interfejs odpowiedzialny za wyświetlanie danych; deleguje do prezentera obsługę żądań użytkow.
- Prezenter – odpowiada za logikę interfejsu; modyfikuje widok



MVVM (Model View View Model)

- Model – pełni dokładnie tę samą funkcję, co w MVC
- Widok – graficzny interfejs odpowiedzialny za wyświetlanie danych; odpowiada za logikę interfejsu (kod behind); deleguje wykonanie poleceń do ViewModelu
- ViewModel – odpowiada za logikę prezentacji zarządza modelem



Pytania kontrolne

- Czym są style architektoniczne?
- Podaj przykładowe klasyfikacje widoków architektonicznych
- Omów wybrane style architektoniczne

Materiały źródłowe

1. OMG Unified Modeling LanguageTM (OMG UML), Superstructure , Version 2.3, 2010
2. L. Maciaszek, B.L. Liong, Practical software engineering: a case study approach, Pearson Addison Wesley, 2005
3. Frank Buschmann, Kevlin Henney, Douglas C. Schmidt, Pattern-oriented software architecture: On patterns and pattern languages, John Wiley and Sons, 2007
4. Paris Avgeriou, Uwe Zdun, Architectural Patterns Revisited – A Pattern Language, http://www.infosys.tuwien.ac.at/staff/zdun/publications/ArchPatterns.pdf
5. http://msdn.microsoft.com/en-us/library/
6. http://rup.hops-fp6.org/
7. Integration strategy based on Oracle experiences, Marek Sokołowski, KKIO 2010 presentation
8. Materiały konferencji ISAT 2011