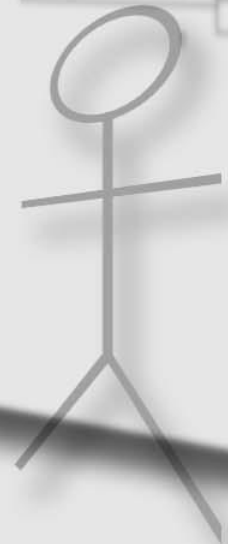


# Modelowanie i Analiza Systemów informacyjnych (MAS)

dr inż. Mariusz Trzaska  
[mtrzaska@pjwstk.edu.pl](mailto:mtrzaska@pjwstk.edu.pl)

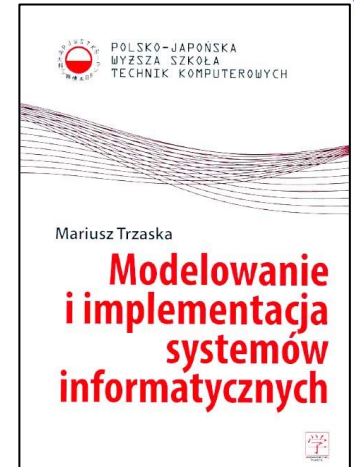
## Wykład 1

### Projektowanie i modelowanie architektury systemu



# Informacje

- Tematyka wykładu
- Mariusz Trzaska: Modelowanie i implementacja systemów informatycznych.
  - Wydawnictwo PJWSTK. Stron 299. ISBN 978-83-89244-71-3: [sklep PJATK](#).
  - Wersja elektroniczna (eBook działający na Windows, iOS, Android, urządzeniach mobilnych oraz czytnikach) w księgarni internetowej Ibuk: [wersja PDF](#).
  - Fragmenty: <http://www.mtrzaska.com/mas-ksiazka>



# Informacje (2)

- Literatura – c.d.
  - Książki o modelowaniu, UML, obiektowości;
  - Książki dotyczące programowania:
    - Java,
    - C#,
    - C++.
- Zaliczenie
- Przykładowe zadania programistyczne:

<https://www.mtrzaska.com/mas-zadania-programistyczne/>

# Zagadnienia

- Wprowadzenie
- Problemy i ich rozwiązanie
- Architektura - widoki
- Różne rodzaje architektur
  - Mainframe,
  - Cienki klient,
  - Dwuwarstwowa,
  - Trójwarstwowa
- Architektura trójwarstwowa
- Model – View – Controller
- Podsumowanie

*Wykorzystano:*

- *wykłady Software Architecture autorstwa Chandika Mendis (University of Moratuwa)*
- *artykuł „Java SE Application Design With MVC” autorstwa Robert Eckstein*

# Architektura

- Klucz do sprawienia by skomplikowane systemy stały się możliwe do zbudowania.
- Efektywne zarządzanie tworzeniem skomplikowanych systemów
- Łatwiejsze kierowanie dużymi grupami pracowników
- Zwiększone ponowne użycie (*reuse*)
- Lepsza spójność i integracja w ramach systemu
- Realizacja strategii biznesowych

# Architektura – dlaczego?

- Nie możesz wybudować drapacza chmur w ten sam sposób co budy dla psa...

*Booch, SD'99*



# Architektura – potencjalne problemy

- Dekompozycja systemu
  - Jak podzielić system na mniejsze kawałki?
  - Czy uwzględniliśmy wszystkie elementy?
  - Czy poszczególne elementy pasują do siebie?
- Kontrola i zarządzanie
  - Problemy zrównoleglenia
  - Własności obejmujące wiele aspektów systemu
  - Kompromisy
- Zapewnienie
  - Ponownego użycia
  - Redukcji kosztów
  - Krótkiego czasu do uzyskania gotowego rozwiązania (*time to market*)
  - Integralności systemu
  - Rozszerzalności
  - Pielęgnowalności
  - Zgodności z celami biznesowymi

# Architektura - definicja

- Architektura obejmuje opis struktury systemu składającego się z:
  - Komponentów lub „bloków”,
  - Ich właściwości widocznych na zewnątrz,
  - Zależności pomiędzy nimi.

*Na podstawie Bass, Clements, and Kazman. Software Architecture in Practice, Addison-Wesley 1997*



# Dekompozycja systemu

- Odpowiedź na problemy ze złożonością
  - Intelktualną
  - Związaną z zarządzaniem
- Jak podzielić system na mniejsze elementy?
  - Enkapsulacja
  - Luźne wiązanie
  - Przejrzyste interfejsy i usługi
- Czy mamy wszystkie niezbędne elementy?
  - Czy zapewniłmy spełnienie wymagań
    - Funkcjonalnych
    - Niefunkcjonalnych
- Czy elementy pasują do siebie?
- Dobrze zdefiniowane interfejsy
- Rygorystyczne wytyczne

# Decyzje architektoniczne - zakres

- Na architekturę ma wpływ wiele decyzji, które nie mogą być podjęte bez kompromisów związanych z ogólnymi celami systemu.
- Decyzje powinny być podejmowane z perspektywy całego systemu lub nawet jeszcze szerzej.
- Decyzja, która może być podjęta korzystając z węższej perspektywy powinna być przekazana właściwej jednostce lub grupie roboczej (architektura minimalistyczna; *minimalist architecture*)
- Decyzje architektoniczne powinny się skupiać na ważnych obszarach o dużym wpływie na resztę systemu, pozostając w zgodzie z biznesowym celem (strategią)
- Cele strategiczne
- Ważne usługi
- Cechy systemu jako całości

# Architektura - widoki

- Konceptyjny
  - Diagram,
  - Identyfikacja komponentów i przydział odpowiedzialności
- Logiczny
  - Uaktualniony diagram koncepcyjny
    - z interfejsami
    - specyfikujący komponenty
  - Projekt współpracy komponentów, mechanizmów i protokołów połączeń
- Wykonawczy (execution)
  - Procesy (diagramy interakcji)
  - Przydział działających instancji komponentów dla procesów,
  - Ich współpraca
  - Wykorzystanie zasobów

# Widok koncepcyjny

- Identyfikuje wysokopoziomowe komponenty z uwzględnieniem właściwej dekompozycji.
  - Zależności,
  - Sposoby współpracy,
  - Odpowiedzialności,

# Widok logiczny

- Definiuje interfejsy i specyfikację komponentów:
  - Precyzyjnie,
  - Jednoznacznie
- Określa dokładne sposoby:
  - interakcji,
  - współpracy
- Powinien zapewniać w miarę niezależny sposób pracy dla poszczególnych programistów.
- Tworzy dokładne diagramy architektury systemu

# Widok wykonawczy

---

- Uwzględnia:
  - fizyczny (implementacyjny) model
  - Wdrożenie
  - Odzwierciedlenie procesów w działającym systemie
  - Skalowalność

# Mainframe (systemy spadkowe)

- Nie jest to współczesny klient/serwer, ale są pewne podobieństwa
- Użytkownik łączy się z komputerem centralnym za pomocą prostego terminala, który przechwytyje polecenia klawiatury i przesyła je do jednostki centralnej
- Całe przetwarzanie odbywa się w komputerze centralnym
- Można wykorzystywać różne platformy sprzętowe i systemowe

# Architektury współdzielenia plików

- Nie jest to współczesny klient/serwer.
- Pliki były udostępniane przez serwer i pobierane na stacje roboczą.
- Następnie całe przetwarzanie (włączając w to logikę biznesową i dane) odbywało się na stacji roboczej
- Takie podejście działa w miarę dobrze jeżeli objętość danych nie jest duża.



# Klient - Serwer

- Podejście wprowadzone w celu zastąpienia serwera plików.
- Korzystając z (relacyjnego) systemu zarządzania bazą danych (DBMS), polecenia (kwerendy, ang. *queries*) użytkownika mogą być wykonywane natychmiast.
- Redukcja ruchu sieciowego ponieważ przesyłam tylko dane będące odpowiedzią na kwerendę (a nie wszystkie potrzebne do obliczenia/ustalenia wyniku).
- Ułatwienie współpracy wielu użytkowników dzięki graficznemu interfejsowi dostępu do danych.
- Wykorzystanie zdalnego wywoływania procedur (*Remote Procedure Calls - RPCs*) lub języka zapytań (*Standard Query Language - SQL*) do komunikacji pomiędzy klientem, a serwerem.
- Jedna z najbardziej (najbardziej popularna?) architektur wykorzystywanych współcześnie.

# Architektura „cienkiego klienta” (thin client)

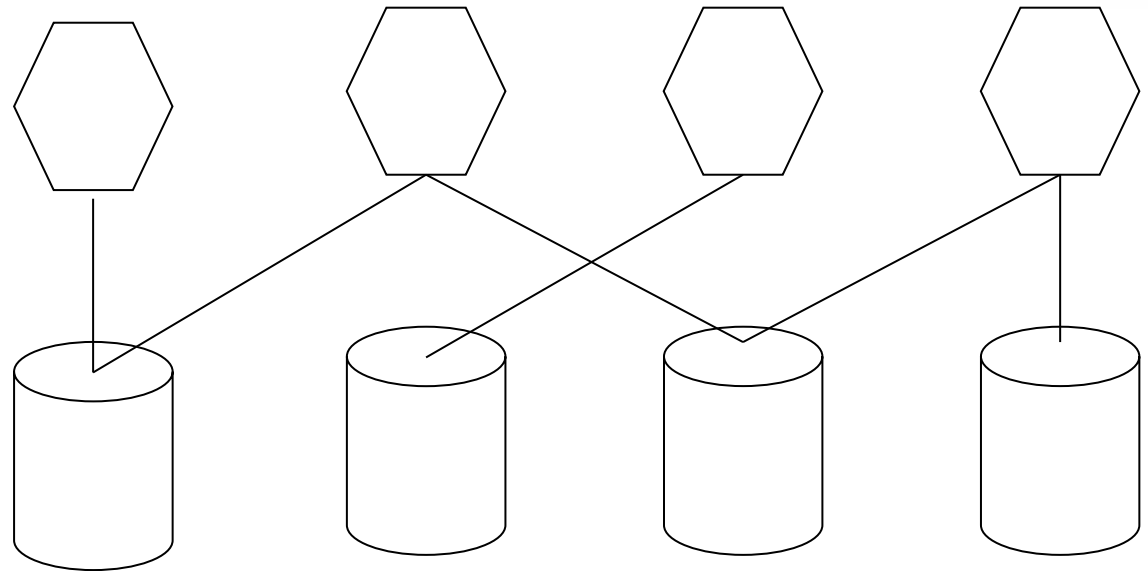
- Podobna do klient-serwer
- Klient jest maszyną o bardzo słabych parametrach
- Całe przetwarzanie odbywa się na serwerze
- Nie zdobyła wielkiej popularności

# Architektura dwuwarstwowa

- Składa się z trzech komponentów rozmieszczonych w dwóch warstwach:
  - Klient (interfejs użytkownika),
  - Serwer (dane, logika biznesowa)

Interfejs użytkownika +  
część funkcjonalności

Zarządzanie bazą  
danych + (większa)  
część funkcjonalności



- Interfejs dostępu do aplikacji jest zlokalizowany na komputerze użytkownika, a funkcjonalność bazodanowa na serwerze (zwykle maszyna dużo mocniejsza).

# Architektura dwuwarstwowa (2)

- Zarządzanie przetwarzaniem jest podzielone pomiędzy środowisko użytkownika, a środowisko systemu zarządzania bazą danych.
- Serwer bazy danych udostępnia:
  - Dane,
  - Procedury,
  - Wyzwalacze (ang. triggers),
  - ...
- Sposób implementacji zarządzaniem danych i usługami przez dostawcę BD ogranicza elastyczność i wybór danego systemu DBMS
  - Limitowana zdolność do przenoszenia systemu z jednego środowiska bazodanowego do innego.
  - Zakres funkcji

# Architektura trójwarstwowa

- Główna zmiana w stosunku do architektury dwuwarstwowej:
  - Dodano warstwę pośredniczącą (*Middleware*) pomiędzy interfejsem użytkownika oraz bazą danych
- Wiele sposobów implementacji oraz zadań dla warstwy pośredniczącej:
  - Monitorowanie transakcji,
  - Wymiana komunikatów pomiędzy pozostałymi warstwami,
  - Serwer aplikacji (Logika biznesowa)
- Dodatkowe możliwości
- Kolejowanie,
  - Uruchamianie aplikacji,
  - Specjalna propagacja komunikatów.

# Architektura trójwarstwowa (2)

- Monitorowanie transakcji
- Zdolność do uaktualniania wielu różnych DBMS'ów w czasie jednej (logicznej) transakcji
- Przezroczysta praca z różnymi źródłami danych:
  - Pliki,
  - Relacyjne DBMS,
  - Obiektowe DBMS,
- Nadawanie (oraz modyfikowanie) priorytetów transakcjom
- Bezpieczeństwo

# Architektura trójwarstwowa (3)

- Wymiana komunikatów
- Przetwarzanie asynchroniczne
  - Korzyści,
  - Problemy
- Przetwarzanie synchroniczne
- Przezroczysta komunikacja pomiędzy aplikacjami heterogenicznymi
- „Inteligentne” zarządzanie ruchem:
  - Priorytety,
  - Kolejowanie,
  - Automatyczne powiadamianie o zdarzeniach
- Separuje od problemów związanych z komunikacją

# Serwer aplikacji

- Aplikacja jest uruchamiana na współdzielonym komputerze, zamiast na kliencie
- Przetwarzanie GUI odbywa się poza serwerem
- Obsługa:
  - Logiki biznesowej,
  - Dostępu do danych
- Poprawa bezpieczeństwa (klient nie ma bezpośredniego dostępu do aplikacji)



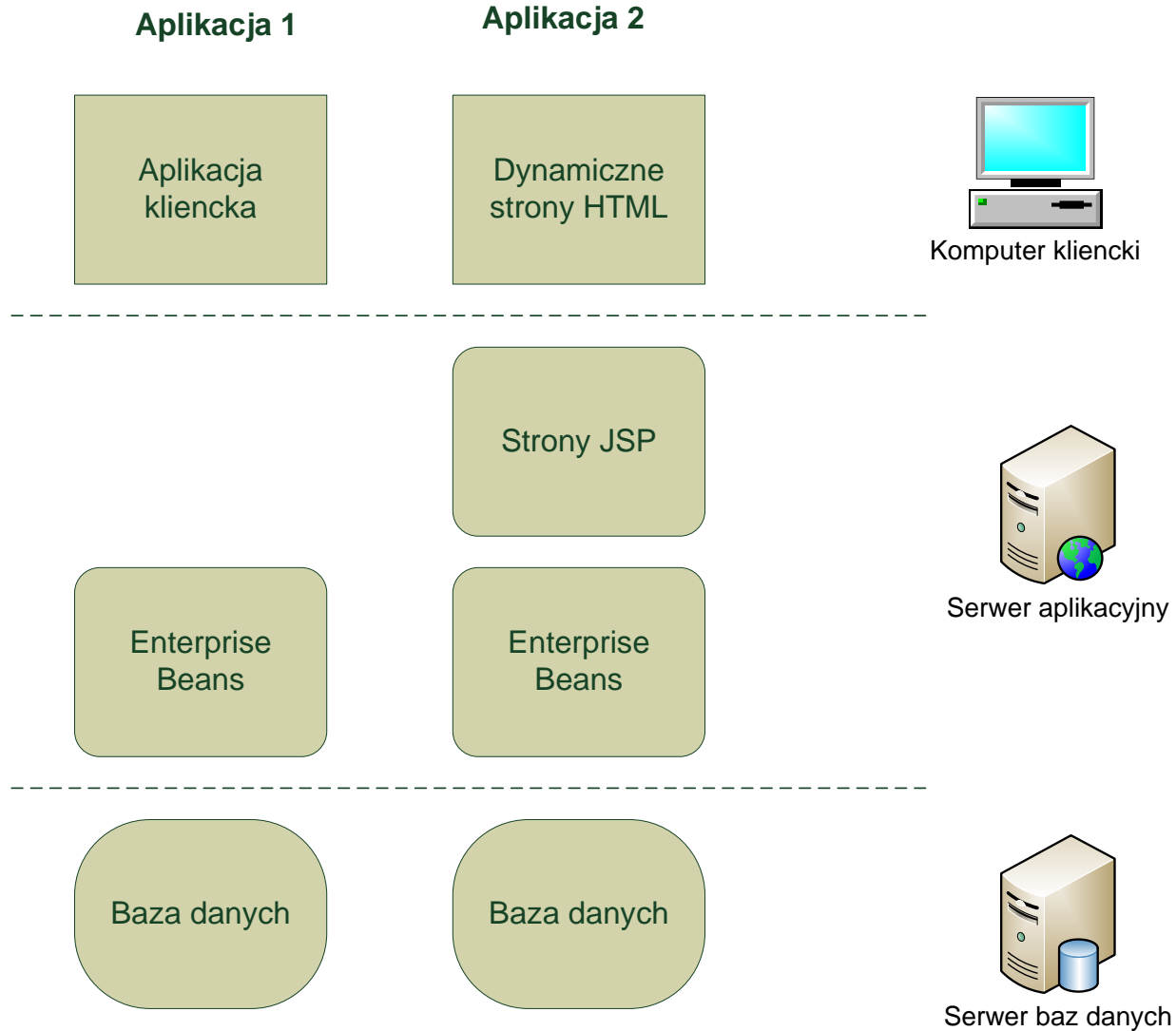
# Serwer aplikacji (2)

- Lepsza skalowalność
- Niższe koszty pielęgnacji aplikacji na jednym serwerze niż na wielu komputerach klienckich
- Przykłady:
  - Apache Tomcat
  - JBoss
  - Serwer aplikacji J2EE
  - Plone,
  - Zope
  - Websphere Application Server

# Serwer aplikacji (3)

- Typowe komponenty:
  - Bezpieczeństwo
  - Połączenie do baz(y) danych
  - Dostęp do zasobów
  - Interfejsy do komponentów zewnętrznych
  - Zarządzanie transakcjami
  - Komponenty sterowane zdarzeniami (lub reagujące na zdarzenia)

# Rozproszona aplikacja wielowarstwowa

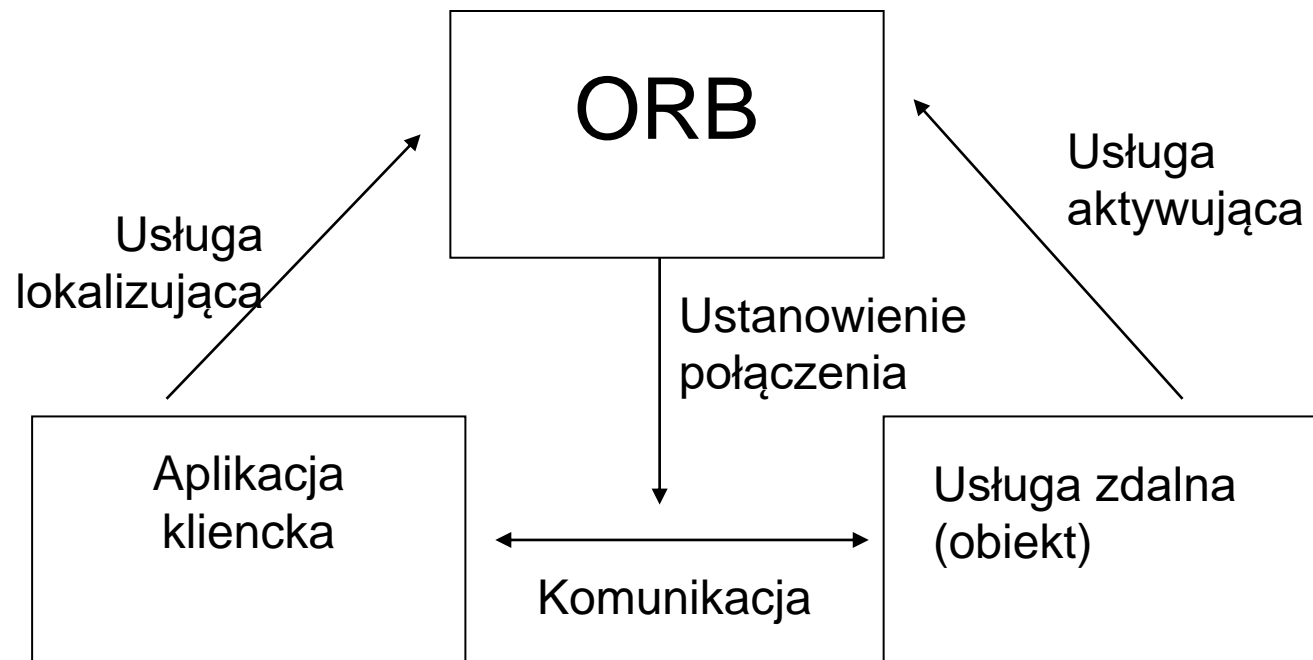


# Architektura trójwarstwowa typu ORB (Object Request Broker)

- Systemy typu klient/serwer wykorzystujące technologie wspierające rozproszone obiekty
- CORBA, RMI, DCOM, itd.
- Wspierają współpracę rozproszonych obiektów, umożliwiając użytkownikom budowę systemów przez łączenie obiektów pochodzących z różnych źródeł (i wykonanych w różnych technologiach)
- Szczegóły implementacyjne wykorzystywanych obiektów są ukryte przed użytkownikiem.
- Programista musi tylko znać interfejs opisujący dostęp do obiektu
- Mimo wielu zalet wydaje się, że nie są zbyt popularne (?)

# ORB

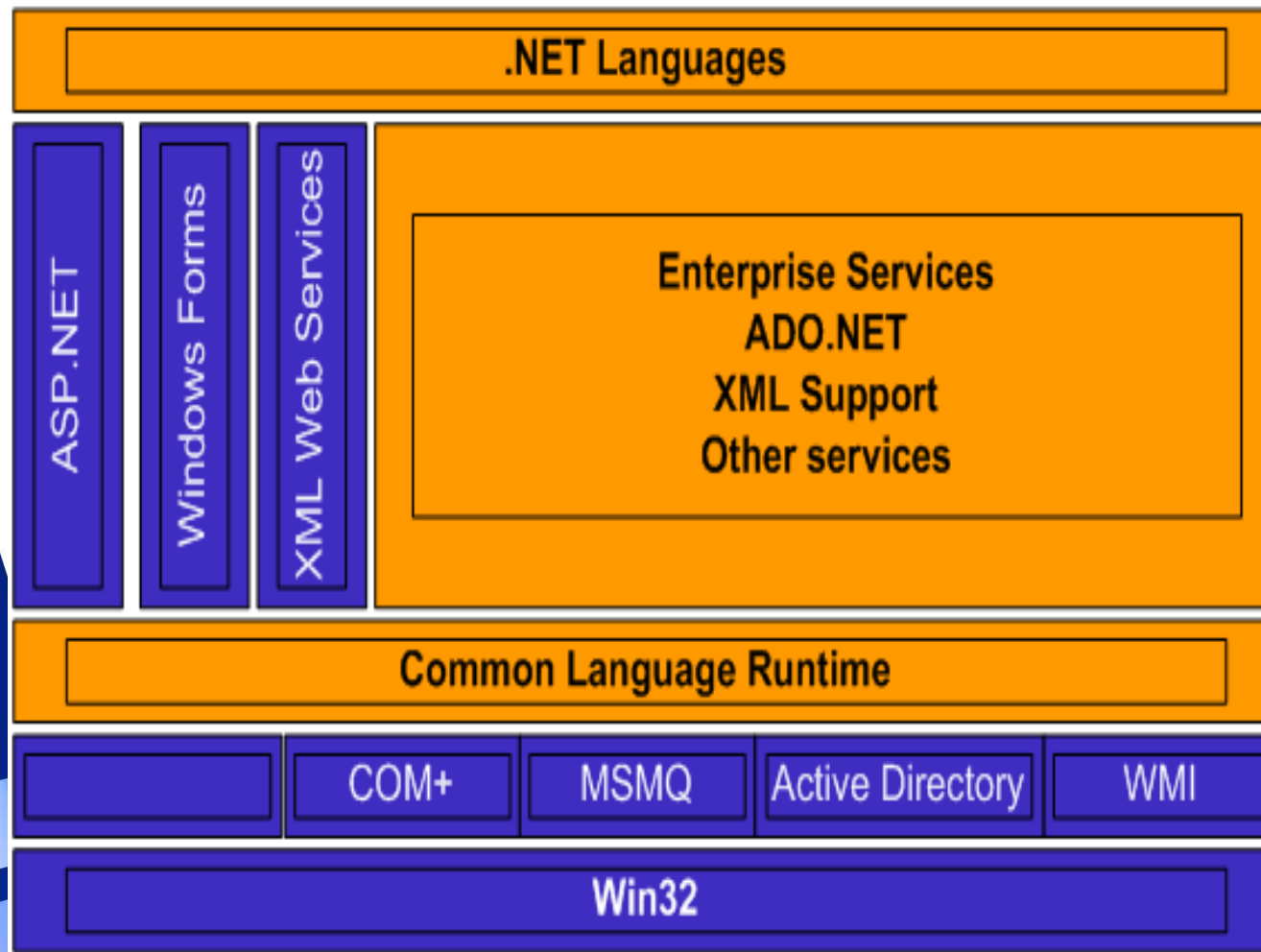
- Najważniejsze funkcje (usługi) technologii ORB:
  - Definiowanie interfejsów,
  - Lokalizacja i ewentualna aktywacja zdalnych obiektów,
  - Komunikacja pomiędzy obiektami.



# ORB (2)

- Inne funkcje:
  - Bezpieczeństwo,
  - Transakcje,
  - Wyszukiwanie,
  - ...
- Wiele sposobów realizacji ORB:
  - Implementacja w kliencie,
  - Oddzielne procesy
  - Część systemu operacyjnego czy serwera aplikacji

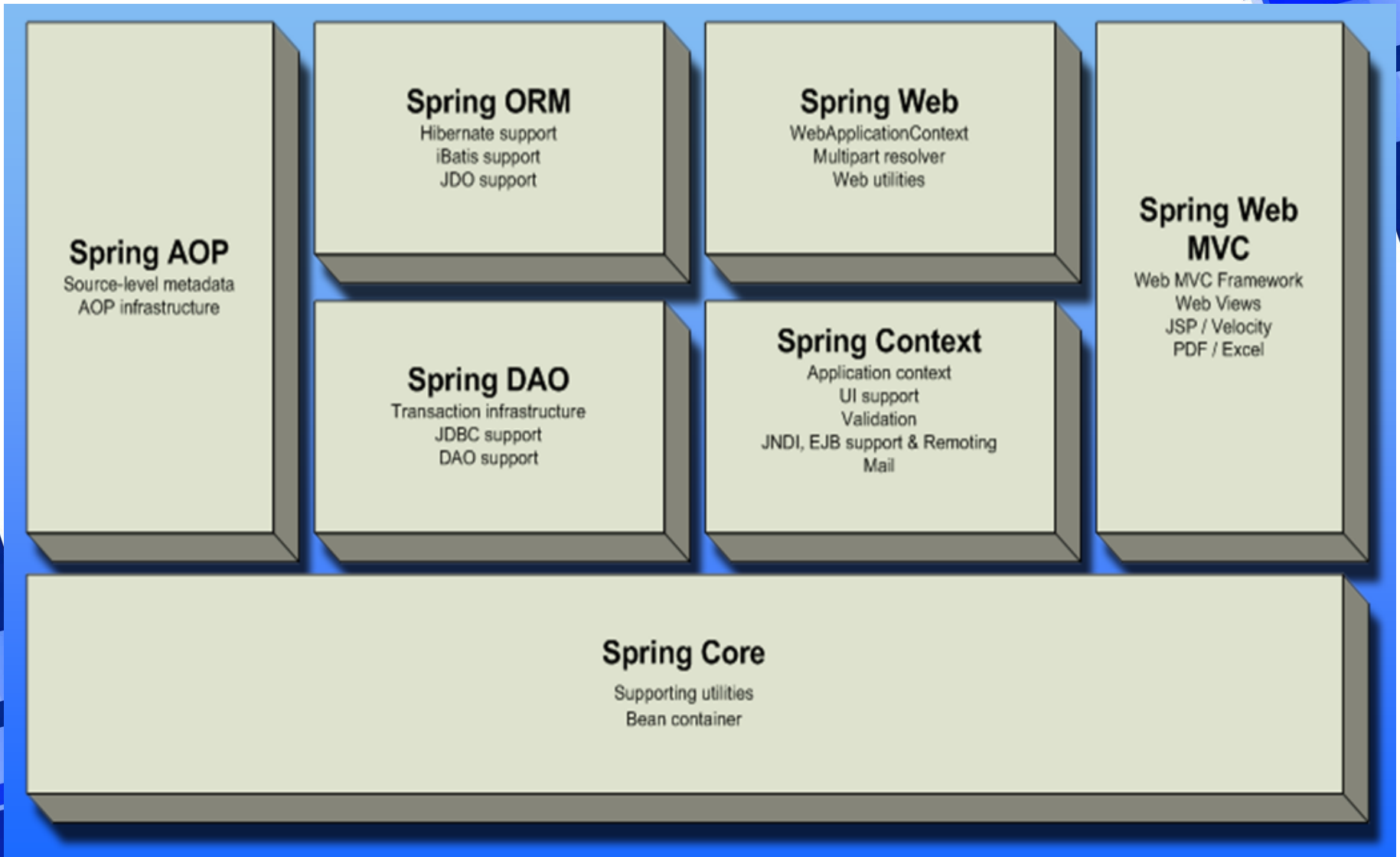
# .NET Framework - Windows Serwer jako serwer aplikacji



- Enterprise Services
- Connection Pooling
- Distributed Transactions
- Security
- Messaging (MSMQ)
- Management/Monitoring

Źródło: MS Windows Technical Overview of Application Services, MS Corporation

# Szkielet aplikacji – Spring



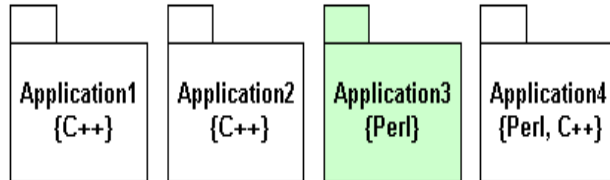


# Architektura warstwowa

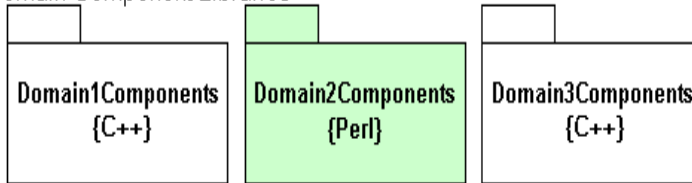
- Wyzwania związane z tworzeniem dużych systemów
  - Ogromne ilości kodu źródłowego (setki tysięcy lub miliony linii)
  - Wysokie skomplikowanie interakcji pomiędzy komponentami
  - Rozległe wykorzystanie gotowych elementów („z półki”)
  - Programowanie w wielu językach
  - Duża liczba programistów (często są to setki osób rozproszonych geograficznie)
  - Różne sposoby pracy z danymi (w tym zapewnianie trwałości):
    - Pliki,
    - Bazy relacyjna,
    - Bazy obiektowe,
  - Rozproszenie komponentów na różne platformy sprzętowe
  - Rozbudowana wielozadaniowość
- Warstwowe diagramy są dobrym sposobem na podzielenie dużych systemów na zarządzalne kawałki

# Architektura warstwowa (2)

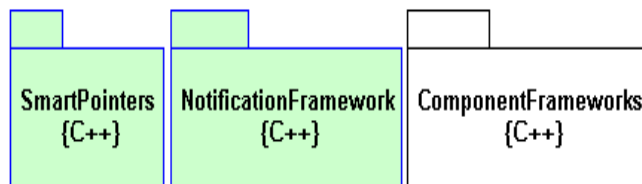
## Applications



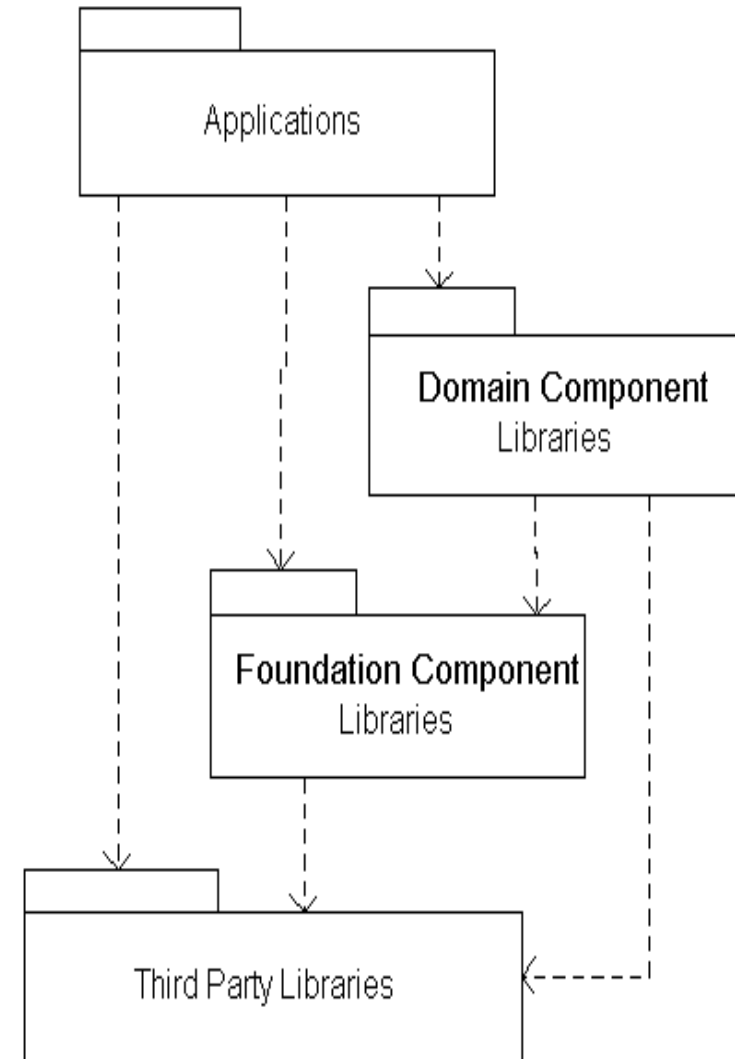
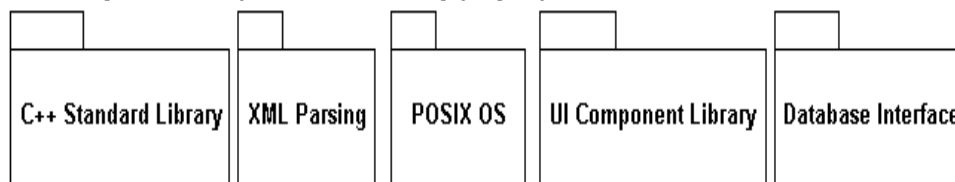
## Domain Component Libraries



## Foundation Component Libraries (maintained by project)

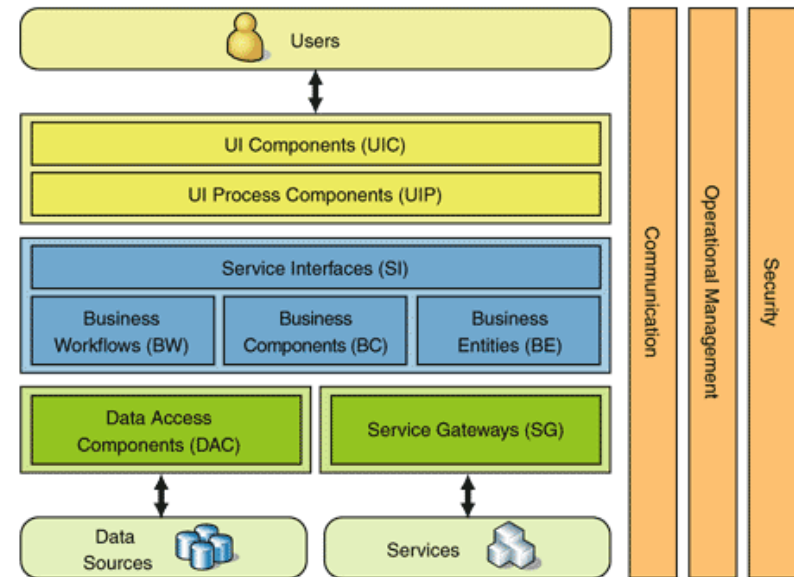


## Third Party Libraries (not maintained by project)



# Diagram architektury warstwowej dla .NET (przykład)

- Warstwa Prezentacji zapewnia (graficzny) interfejs użytkownika. Przeważnie wykorzystuje Windows Forms lub ASP.NET.
- Warstwa Biznesowa realizuje biznesową funkcjonalność aplikacji. Realizowane jest to za pomocą jednego z języków udostępnianych przez platformę .NET. Stworzone komponenty mogą być połączone z Microsoft .NET Enterprise Services i/lub z Microsoft BizTalk Server.
- Warstwa Danych zapewnia dostęp do zewnętrznych źródeł danych takich jak bazy danych (np. MS Access, MS SQL Server) czy pliki XML.



Źródło: MSDN – Patterns and Practices

# Model – View – Controller

- Wzorzec projektowy(?) opracowany w 1979 przez Trygve Reenskaug w Xerox Palo Alto Research Center.
- Bardzo mocno wykorzystywany przy programowaniu z wykorzystaniem GUI na platformie Java.
- Popularny w aplikacjach webowych, np. ASP .NET MVC, Ruby on Rails.
- Jest pewną modyfikacją architektury trójwarstwowej.

# Model – View – Controller (2)

- Składa się z trzech elementów (warstw):
  - Model (*model*) - dane
  - Widok (*view*) - wizualizacja
  - Kontroler (*controller*) - zarządzanie

# Model

- Reprezentacja danych (w tym metadanych),
- Definicja zasad rządzących dostępem do nich,
- Właściwa modyfikacja danych (z uwzględnieniem np. powiadamiania o zmianach)
- Często jest też pewnego rodzaju odwzorowaniem procesów ze świata rzeczywistego

# Widok (view)

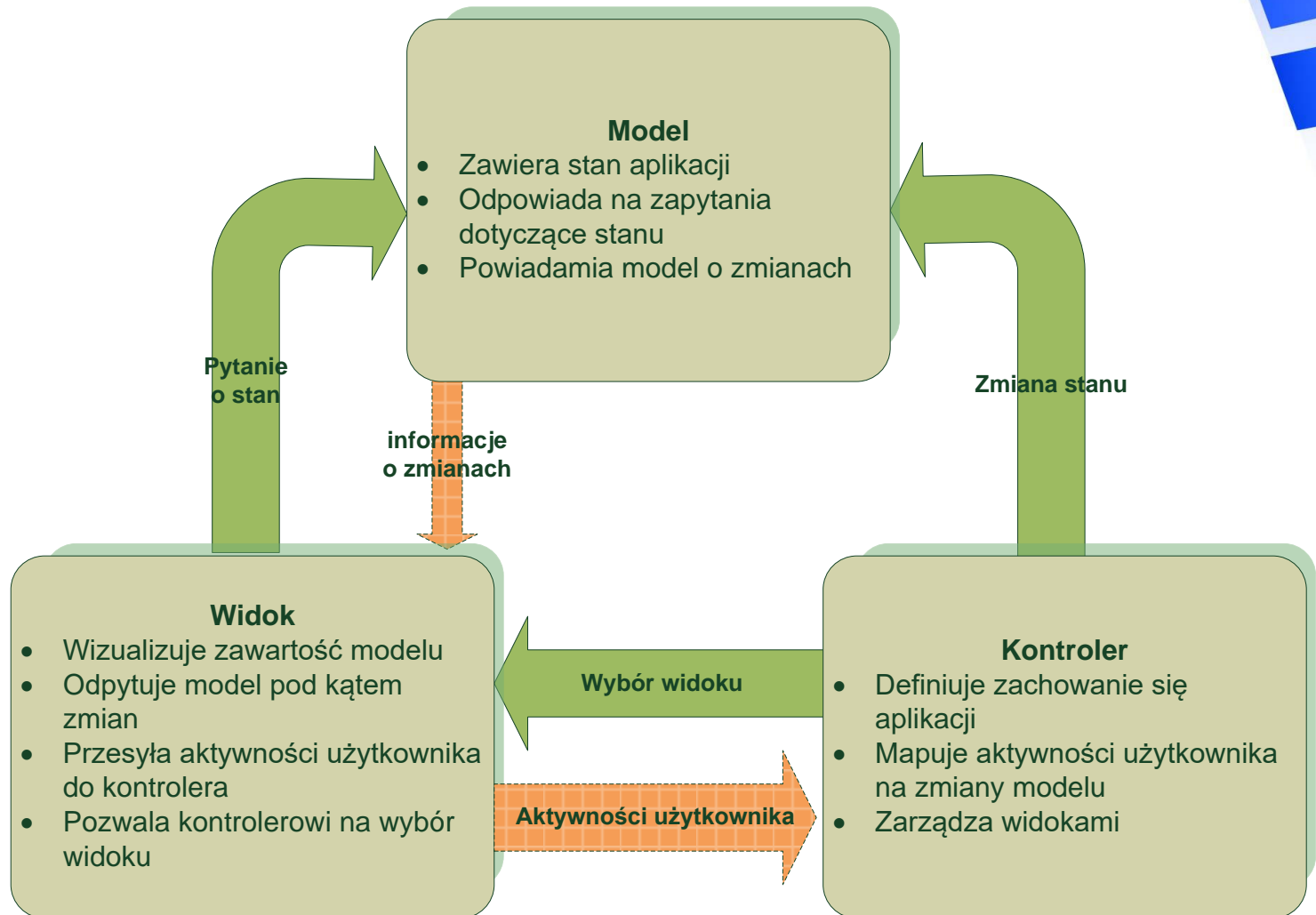
- Wizualizacja zawartości modelu (danych).
- Dokładnie określa jak dane mają zostać wyświetlone.
- Musi właściwie reagować na zmiany modelu (danych).
  - Technologia *push* – widok rejestruje się w modelu celem otrzymywania powiadomień o zmianach,
  - Technologia *pull* – widok sam odpowiada za odpytywanie modelu.

# Kontroler (controller)

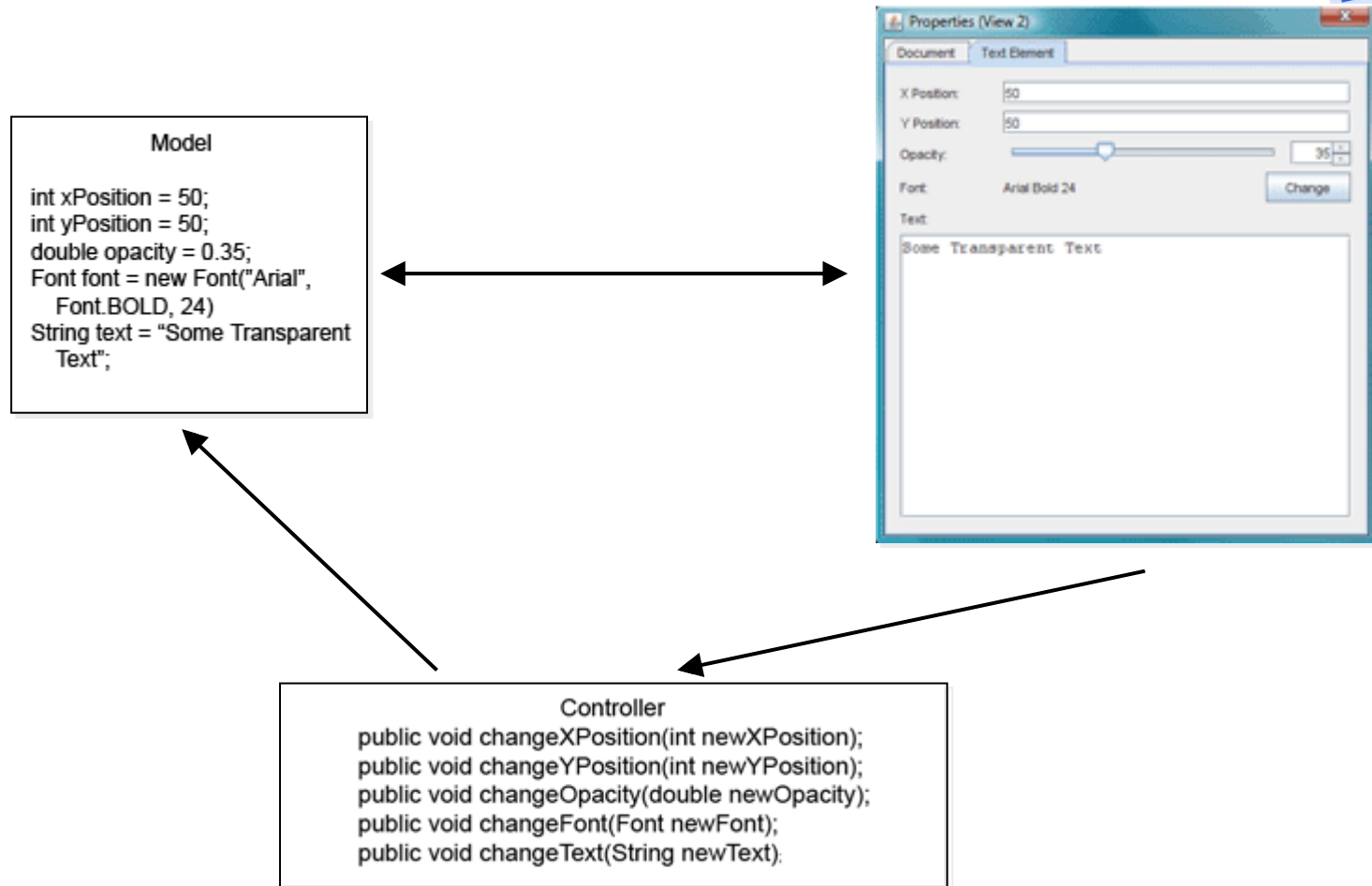
- Tłumaczy działania użytkownika na akcje, które odzwierciedlane będą w modelu.
- W zależności od rodzaju aplikacji:
  - Aplikacja desktopowa: działaniami użytkownika będą kliknięcia myszką lub wybory pozycji z menu,
  - Aplikacja internetowa dostępna przez www: odpowiednie żądania *HTTP GET* oraz *POST*.
- Kontroler może też utworzyć nowy widok.



# MVC - diagram



# MVC – przykład dla Javy SE 6



# MVC – przykład dla Javy SE 6 (2)

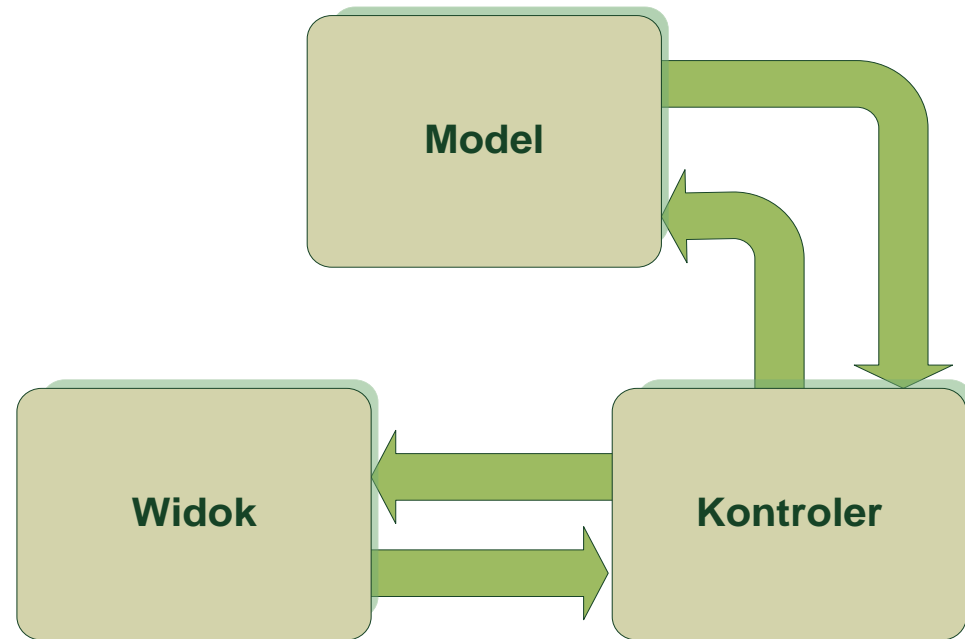
- Po utworzeniu modelu, widoku oraz kontrolera:
  - Widok rejestruje się w modelu celem otrzymywania powiadomień o zmianach (push). Model nie jest świadomy istnienia kontrolera.
  - Kontroler jest łączony z widokiem (korzystając z listener'a).
  - Kontroler otrzymuje referencję pokazującą na model.

# MVC – przykład dla Javy SE 6 (3)

- Gdy użytkownik wykona jakąś aktywność:
  - Widok rozpoznaje aktywność GUI, np. kliknięcie na przycisku
  - Widok wywołuje odpowiednią metodę z kontrolera
  - Kontroler łączy się z modelem ewentualnie modyfikując jego stan.
  - Jeżeli model został zmodyfikowany, informuje o tym widok(i) (korzystając z odpowiednich listener'ów).

# MVC – nowe podejście

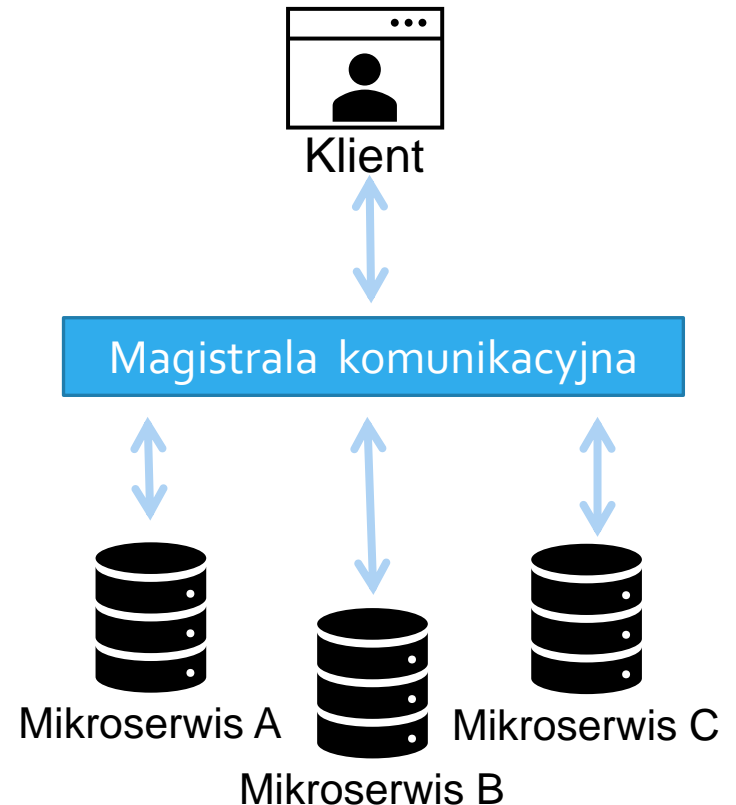
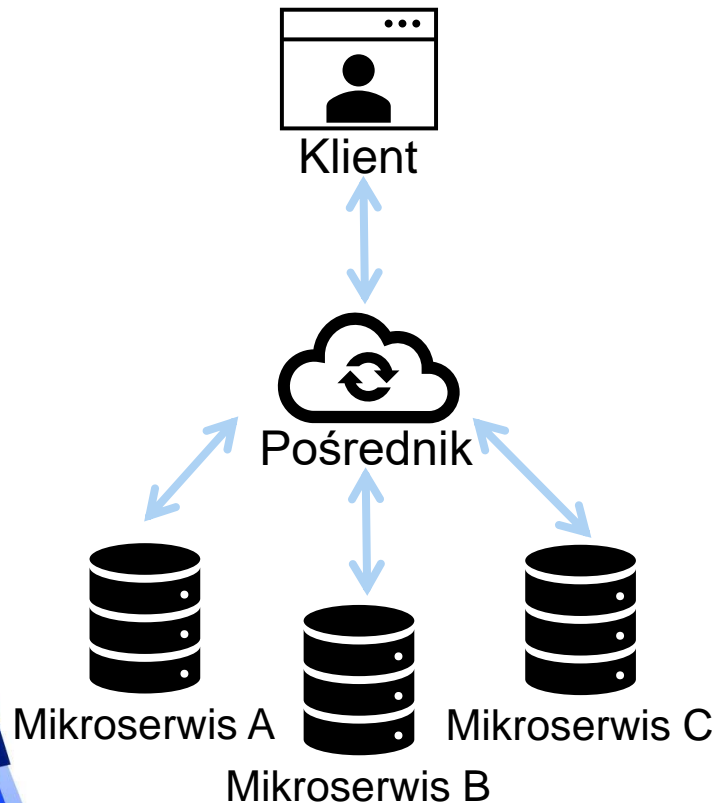
- Różnice w stosunku do poprzedniej wersji
  - Powiadomienia o zmianach stanu w modelu są przekazywane za pomocą kontrolera.
  - Czyli, kontroler kontroluje przepływ danych pomiędzy widokiem, a modelem
- Takie podejście pozwala na lepsze oddzielenie modelu od widoku.
- Klasyczna architektura trójwarstwowa?



<http://java.sun.com/developer/technicalArticles/javase/mvc/index.html>

# Architektura mikroserwisów

- Różne warianty i rozwiązania, m. in.
  - indywidualna komunikacja,
  - systemy kolejkowe.



# Architektura mikroservisów (2)

- Zalety, m. in.:
  - Łatwa dekompozycja problemów,
  - Skalowanie,
  - Heterogeniczność,
  - Elastyczność,
  - Odporność na awarie,
  - Łatwiejsze modernizacje części,
  - Przenaszalność.



# Architektura mikroservisów (3)

- Wady, m. in.:
  - Skomplikowana infrastruktura,
  - Koszty,
  - Spójność danych,
  - Komunikacja,
  - Złożoność technologiczna,
  - Bezpieczeństwo (m. in. komunikacji)





# Podsumowanie

- Właściwie zaprojektowana architektura znacząco ułatwia (umożliwia) budowę dużych systemów komputerowych
- Obecnie najpopularniejsza jest architektura trójwarstwowa
  - Interfejs użytkownika,
  - Logika biznesowa,
  - Źródło danych.
- Istnieje wiele gotowych rozwiązań wspierających tworzenie aplikacji w takiej architekturze.