Projektowanie Graficznych Interfejsów Użytkownika

Robert Szmurło

Wzorce Projektowe

Nie ma złotego środka spełniającego wszystkie wymagania.

- skala projektu
- liczba deweloperów
- stopień skomplikowania

Wzorzec projektowy – w inżynierii oprogramowania, uniwersalne, sprawdzone w praktyce rozwiązanie często pojawiających się, powtarzalnych problemów projektowych. (za wikipedią :-)

Po co mówimy o wzorcach?

- chcemy np. zrozumieć różne style architektoniczne
- chcemy w zespole architektów i programistów posługiwać się wspólną terminologią
  - przecież nie chce nam się za każdym razem tłumaczyć jak odseparować prezentację :-)

Wzorce projektowe

Przykłady:

Wzorzec (?) MVC – Model View Controller

- przetrwał próbę czasu (ponad 35 lat, lata 70)
- sprawdza się w separacji Logiki Biznesowej od Logiki Prezentacji.
- stosuje się go zazwyczaj do rozwiązań bazodanowych, a nie do specjalistycznych aplikacji opartych na technologii „bogaty klient” („rich client”)

Wzorzec singleton

Wzorzec fabryki i wiele innych...

Wzorce projektowe

Wzorce projektowe w GUI:

- **Model – Widok – Kontroler (MVC)** ***(osobny wzorzec projektowy czy raczej ich zbiór?)***
- Zawiadomienie (Notification)
- Kontroler zarządzający (Supervising controller)
- Pasywny widok (Passive View)
- Model prezentacji (Presentation Model)
- Sumator zdarzeń (scalacz) (Event aggregator)
- Sterownik okna (Window Driver)
- Synchronizacja przepływu (Flow Synchronization)
- Synchronizacja obserwatora (Observer Synchronization)
- Selektor prezentacji (Presentation Chooser)
- Oddzielona prezentacja (Separated Presentation)

Głównym źródłem niniejszego opracowania wzorców jest praca:
Martin Fowler, „Development of Further Patterns of Enterprise Application Architecture”
http://www.martinfowler.com/eaaDev/ (Link: 20.03.2007)

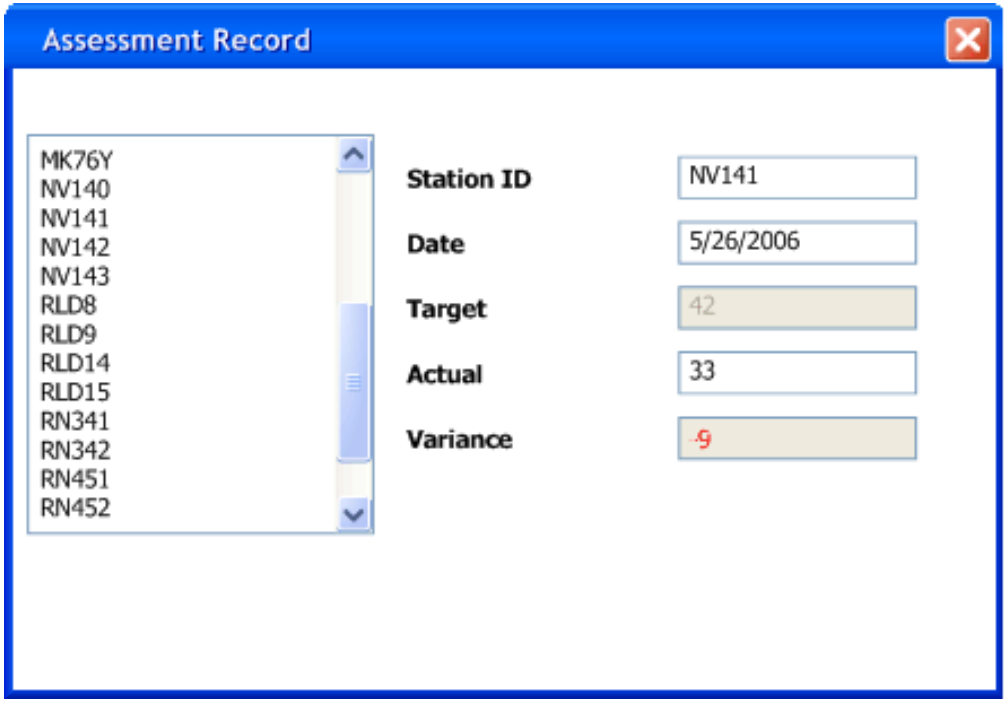
Wzorzec 1: Formularze i Kontrolki

Architektura promowana w latach 90-tych dla aplikacji klient-serwer głównie w narzędziach takich jak Visual Basic, Delphi czy PowerBuilder.

- Bardzo praktyczna podczas prototypowania, ze względu na natychmiastowe efekty.

Program przykładowy: *(na podstawie Martin Fowler, „Development of Further Patterns of Enterprise Application Architecture")*

- Zbiór stacji pomiarowych pewnej wartości,
  - raportujemy dla każdej stacji:
    - wartość zmierzoną
    - wartość progową
    - różnicę



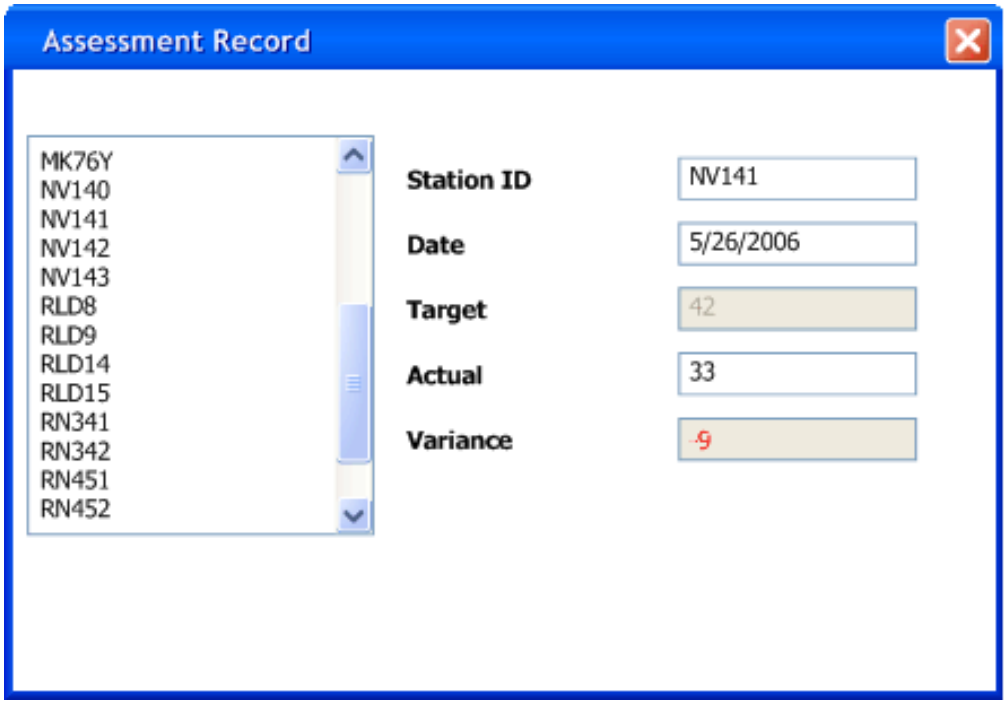
Zadania formularza

Formularz (widok) jest specyficzny dla naszej aplikacji,

- ale używa pewnych ogólnych kontrolek.

Formularz jest odpowiedzialny za:

- układ kontrolek na ekranie,
- logikę formularza (widoku), która nie może być w łatwy sposób zrealizowana wewnątrz kontrolek.



Wyświetlane dane

Najczęściej dane wyświetlane na ekranie pochodzą z jakiejś bazy danych (SQL).

Dane te występują w trzech zasadniczych kopiach (stanach):

- 1. w rzeczywistej bazie na serwerze (lub lokalnym pliku), kopię tę możemy nazwać **stanem rekordu**.
- 2. w pamięci operacyjnej związanej z naszą aplikacją w postaci jakiegoś zbioru danych (RecordSet), którą nazwiemy **stanem sesji**.
- 3. na ekranie w komponentach GUI, którą określimy **stanem ekranowym**.

Jednym z kluczowych elementów architektury GUI jest synchronizacja danych pomiędzy stanami ekranowym i sesji.

W naszym pierwszym wzorcu wykorzystuje się do tego wbudowany w niektóre kontrolki mechanizm: **DataBinding**.

Databinding – szczegóły

Podstawowa idea – data binding zapewnia synchronizację między tym co znajduje się na ekranie a tym co znajduje się w recordset'ie.

Każda zmiana na ekranie automatycznie propaguje się do recordsetu i odwrotnie.

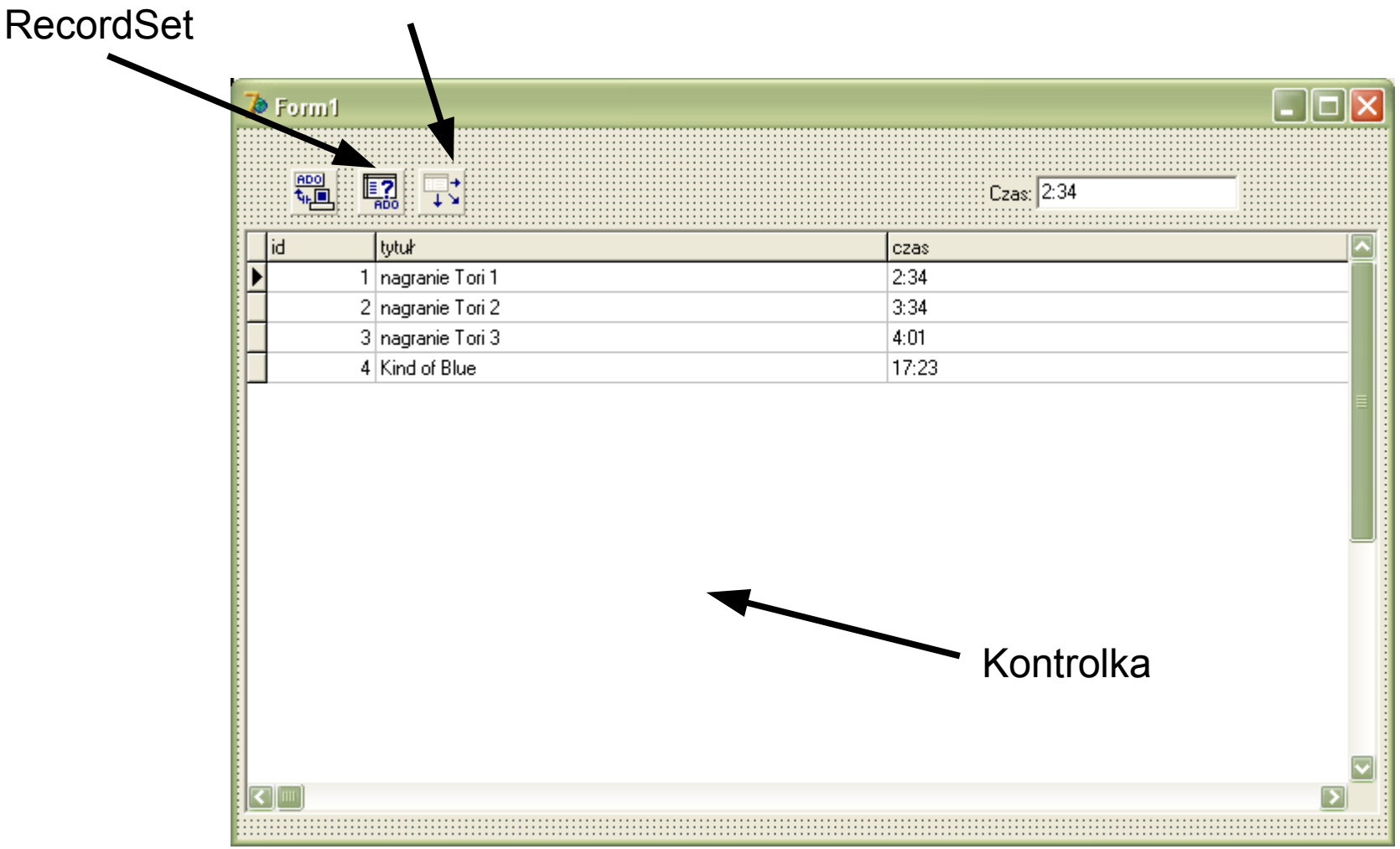
Koncepcja ładowania z sesji (recorsetu) do kontrolek tylko podczas pierwszego pokazywania. (zapobiega zapętleniu się uaktualnień :-)

Kontrolka jest podpięta do kolumny w tabeli lub kwerendzie zazwyczaj za pomocą pola atrybutu (property).

Większa część funkcjonalności związanej z Data Binding jest zaimplementowana w bibliotece, którą wykorzystujemy.

Demo w Delphi

Mechanizm Data Binding



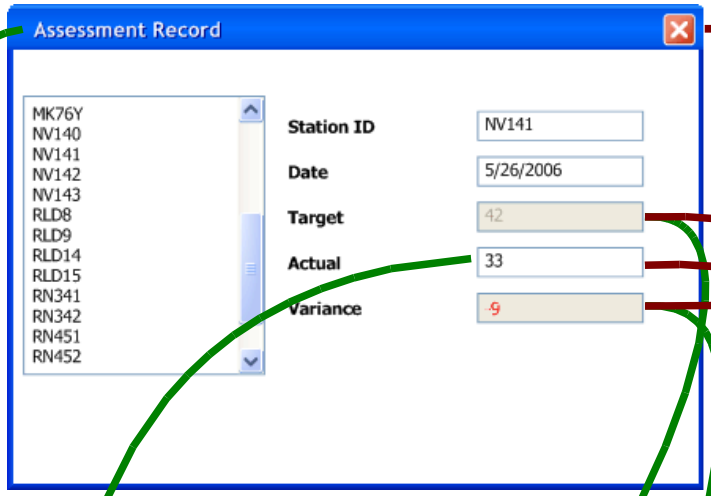
Specyficzne wymagania aplikacji

Niestety bardzo często standardowe kontrolki nie udostępniają jakiejś funkcjonalności, która jest związana z logiką naszej konkretnej aplikacji. W naszym przypadku jest to logika związana z kolorem w jakim ma być wyświetlona różnica między wartościami zmierzonymi i zadanymi. W przypadku znacznych odchyłek ma być to kolor czerwony.

Są dwa zasadnicze rozwiązania:

- 1. za każdym razem gdy zmieni się wartość kontrolki różnicy, możemy informować o tym formularz (klasę widoku), który następnie pobierze aktualną wartość z kontrolki i na jej podstawie podejmie odpowiednią decyzję o kolorze. (Wzorzec GUI: **Formularz i Kontrolka**)
  - do tego kontrolki mają specjalne zdarzenia (events), do których podpina się formularz. Takie rozwiązanie wykorzystuje wzorzec obserwatora.
- 2. tworzymy naszą własną kontrolkę, która rozszerza funkcjonalność kontrolki standardowej o dodatkową opcję koloru.

Analiza procesu edycji z wykorzystaniem Form



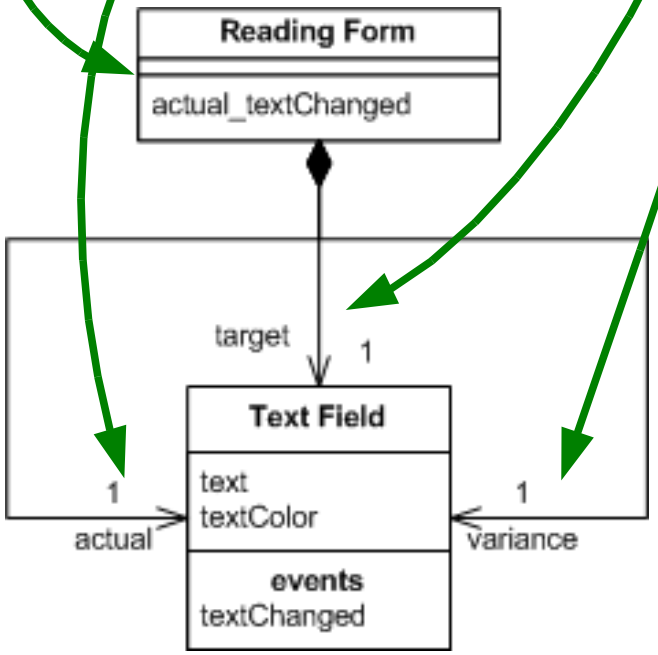


Diagram klas

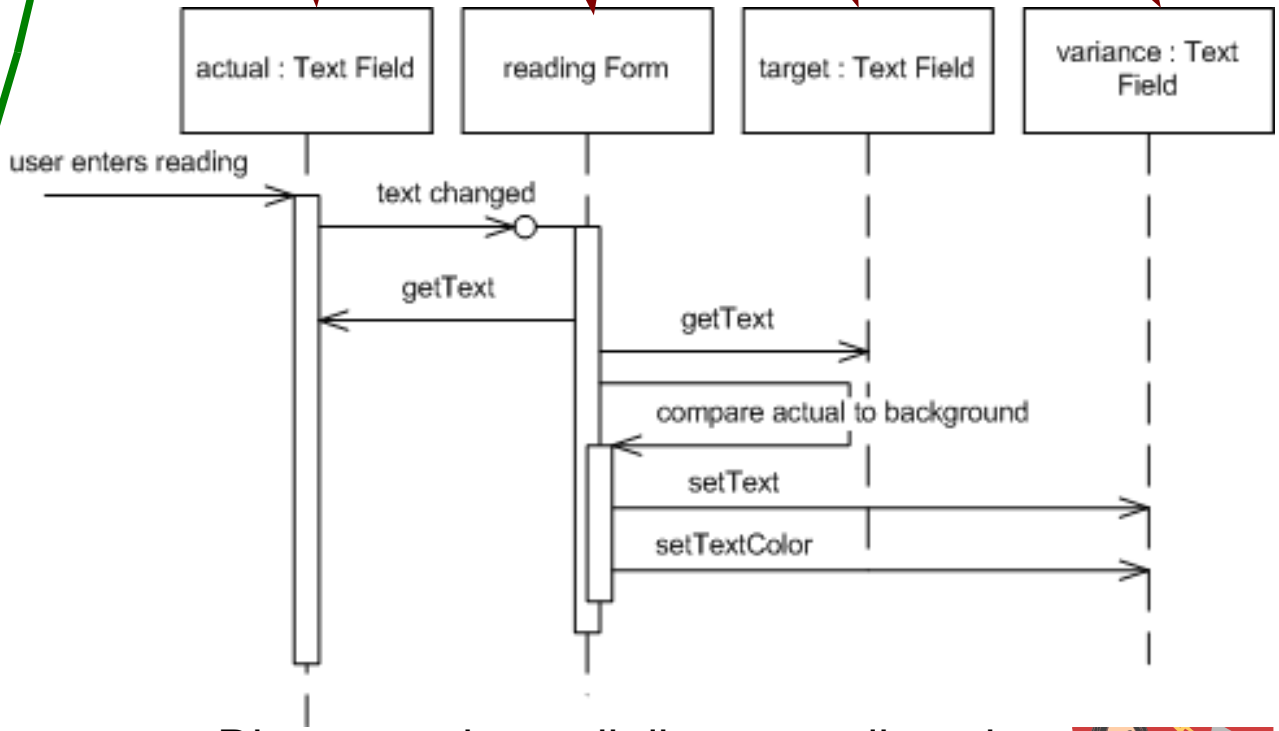
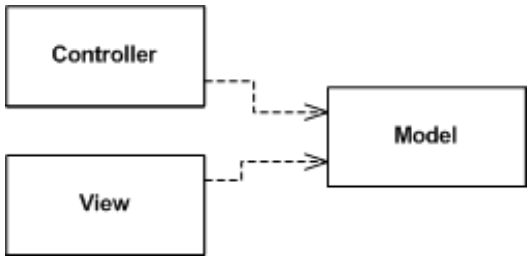


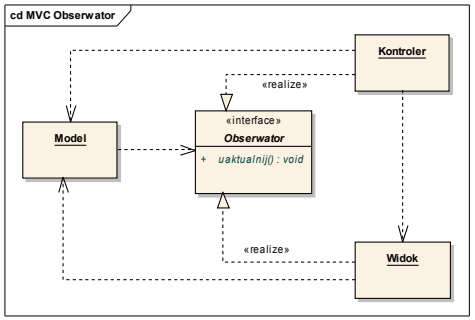
Diagram sekwencji dla przypadku gdy użytkownik wprowadza nową wartość.

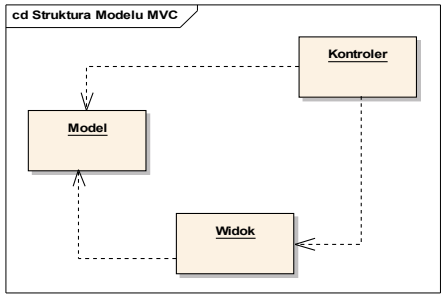
Podsumowanie – Formularz i Kontrolka

- Podsumujmy pierwszy wzorzec oparty na formularzu i kontrolkach:
  - Programiści tworzą formularze (widoki) specjalizowane dla naszej aplikacji, ale wykorzystują do tego uniwersalne kontrolki.
  - Formularz definiuje układ kontrolek na ekranie.
  - Formularz *obserwuje* kontrolki za pośrednictwem metod, które są połączone ze zdarzeniami generowanymi przez kontrolki.
  - Proste edycje danych na ekranie są obsługiwane przez mechanizm Data Binding.
  - Bardziej złożone modyfikacje są realizowane w metodach obsługi zdarzeń w formularzu.
- Problemy:
  - wydajność,
  - transakcje, wzajemne blokowanie,
  - walidacja,
  - dla klikaczy... :-)

Wzorzec MVC







Wiele odmian

Motywacja- czyli potrzeby rynku

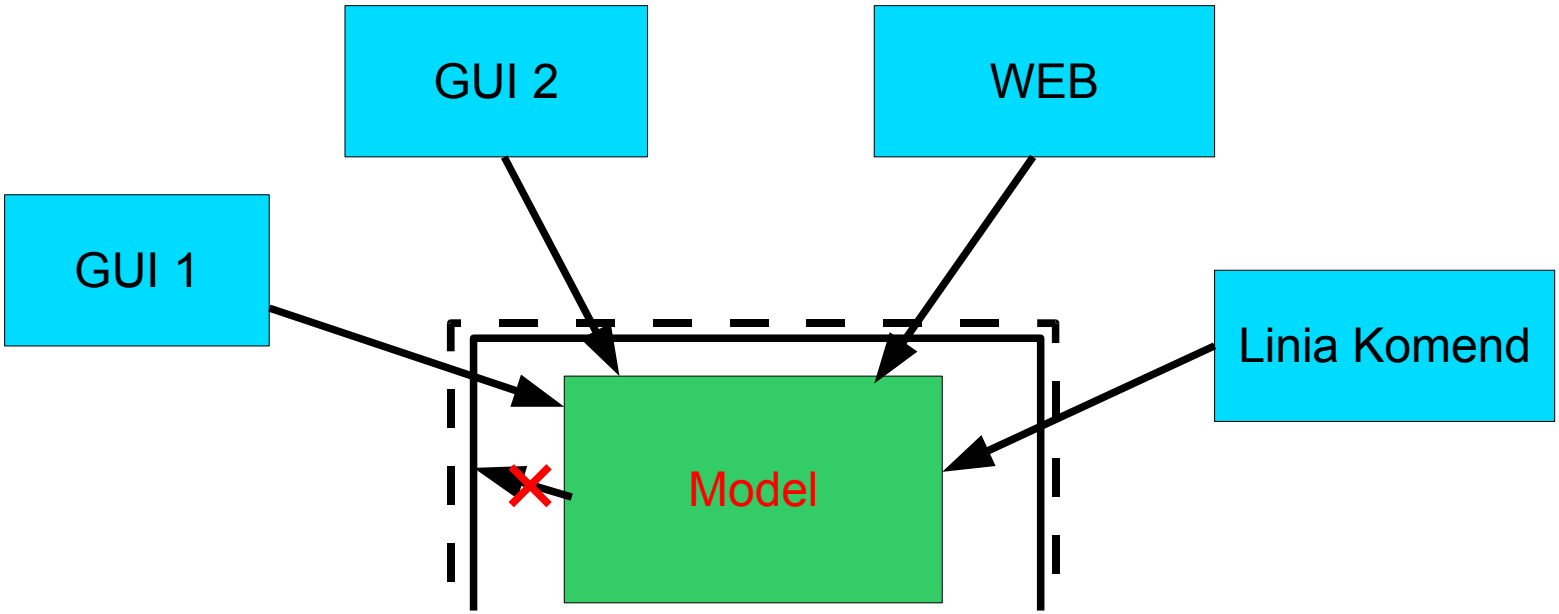
- Interfejs użytkownika ale i nie tylko interfejs zmienia się bardzo często. (nowe strony, zmiana kolejności, zmiana organizacji elementów na ekranie, nowe kolumny w bazie, nowe tabele, ...)
- W niektórych sytuacjach system wyświetla tą samą informację, ale w innej formie (widoki).
- Projekt efektywnego (też efektownego?) interfejsu w HTML wymaga specjalnych kwalifikacji? Rzadko spotyka się osoby, które potrafią projektować strony internetowe i jednocześnie logikę biznesową.
- Interfejs składa się zasadniczo z dwóch elementów: wyświetlania i uaktualniania.
- Kod interfejsu użytkownika zależy od specyfiki sprzętowej (np. PDA, WebForms, WindowForms)
- Tworzenie automatycznych testów interfejsu użytkownika jest pracochłonne i trudniejsze od testów logiki biznesowej.

Odseparowana Prezentacja *(ang. Separated Presentation)*

Koncepcja której podstawą jest odseparowanie obiektów dziedziny dla której stworzona jest aplikacja, czyli obiektów użytkownika, od widoków prezentacji realizowanych za pomocą elementów GUI.

Obiekty dziedziny (użytkownika) w koncepcji tej powinny być całkowicie niezależne i nie powinny odwoływać się do prezentacji.

Więcej, powinny w naturalny sposób obsługiwać wiele interfejsów jednocześnie.

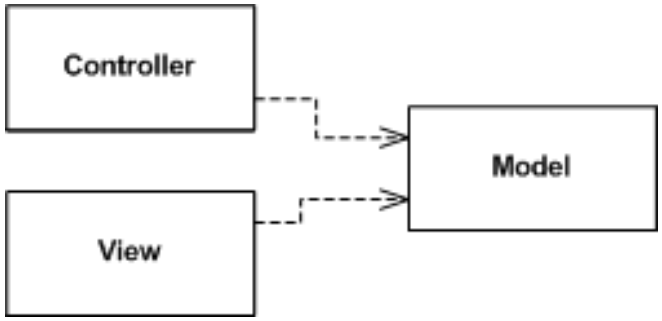


MVC – Moduły

Tu już operujemy na obiektach a nie na RecordSet'ie

Model. (obiekty dziedziny)

- Zarządza zachowaniem się danych w dziedzinie dla której została stworzona aplikacja.
- Potrafi poinformować o swoim stanie.
- Potrafi uaktualnić swój stan.



Widok.

- Zajmuje się tylko wyświetlaniem informacji.
- Ma wiedzę co trzeba wyświetlić i jak w danym momencie.

Kontroler.

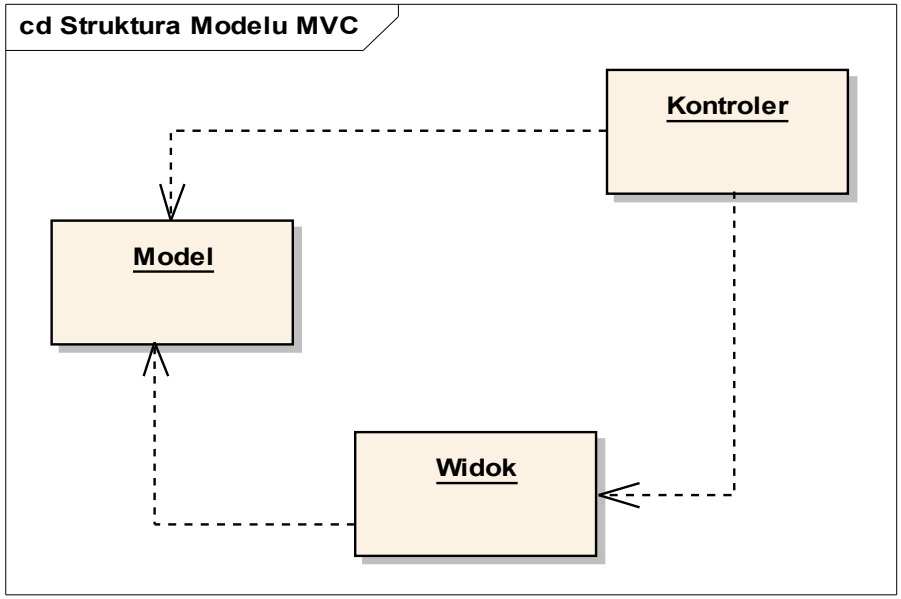
- Interpretuje zainicjowane akcje oraz wprowadzone przez użytkownika dane.
- Ma wiedzę potrzebną aby kontrolować sekwencje widoków.

Uwaga! W aplikacji jest wiele par widok, kontroler.

Model – Widok - Kontroler

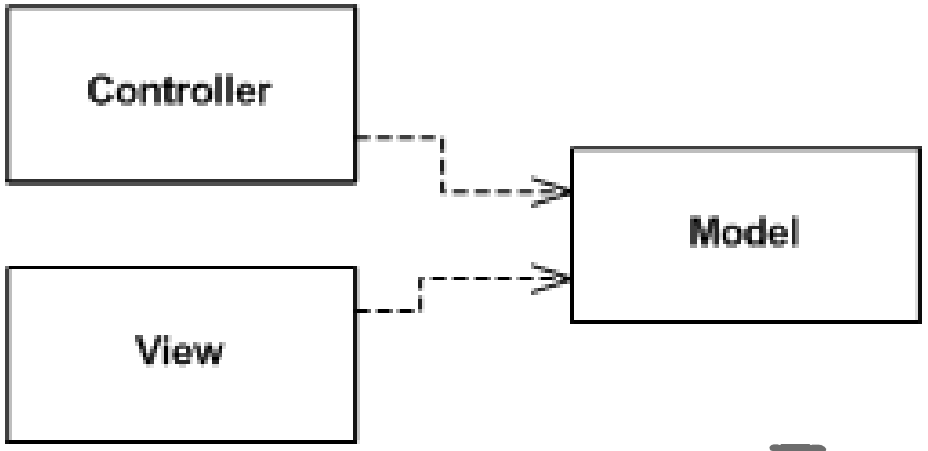
Koncepcja klasyczna

- model jest elementem niezależnym
- występuje dwustronna zależność między kontrolerem a widokiem



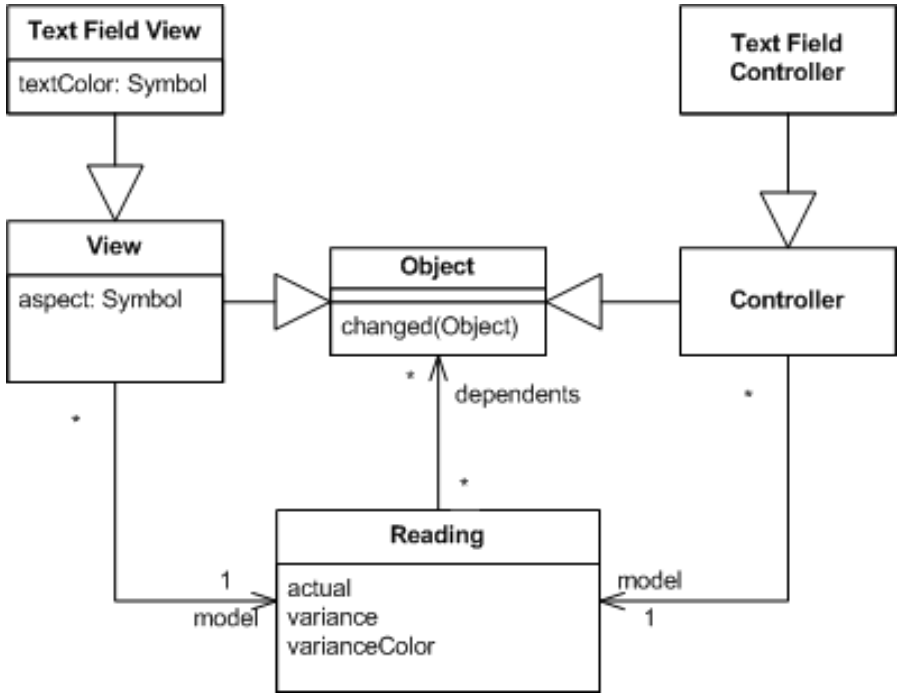
Koncepcja: Martin Fowler

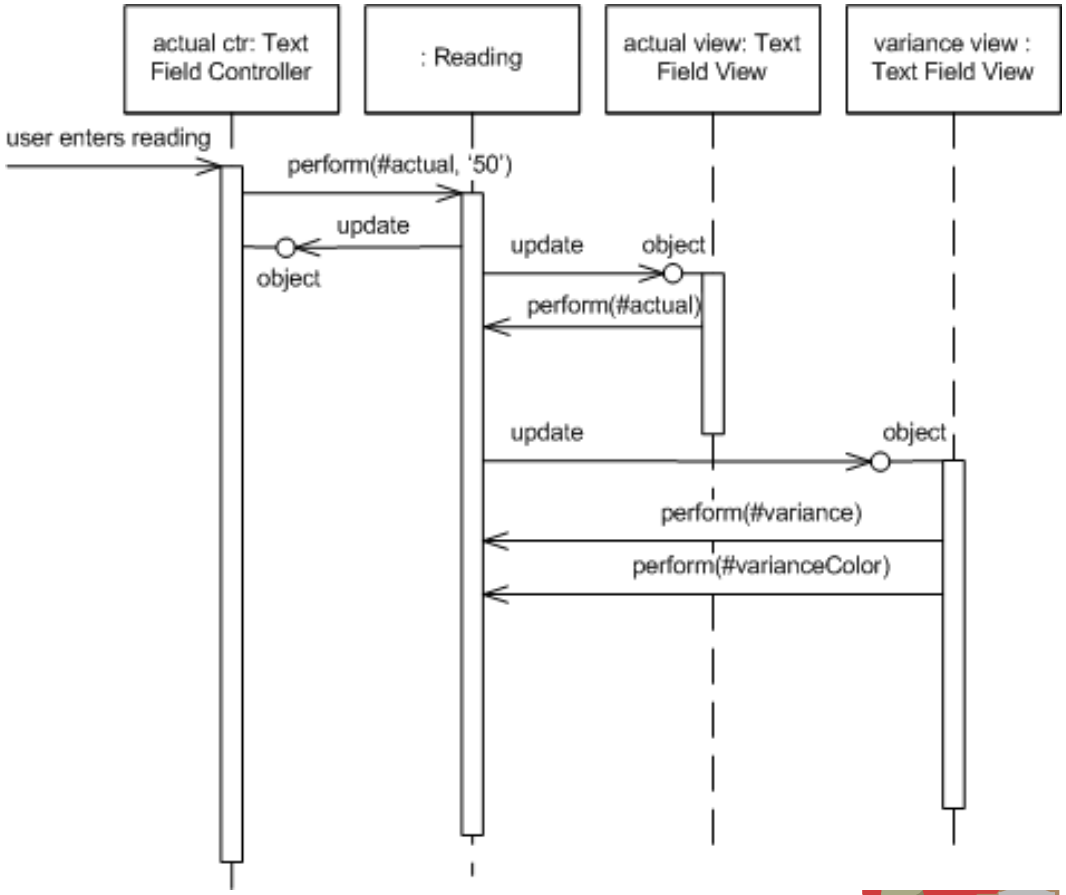
- model nadal niezależny
- programista zazwyczaj nie używa bezpośrednio zależności między kontrolerem a widokiem, komunikację automatyzują różne narzędzia i wzorce, które informują widok o zmianach modelu



Tradycyjny model MVC

- Model wywodzi się ze Smalltalka.
- Zakładamy, że mamy kontrolki dedykowane do naszej aplikacji.
- Bardzo podobne do Data Binding, ale bez zdarzeń (events), oraz tym razem widok obserwuje model, a nie formularz obserwuje kontrolki. (czyli też wzorzec Obserwator)

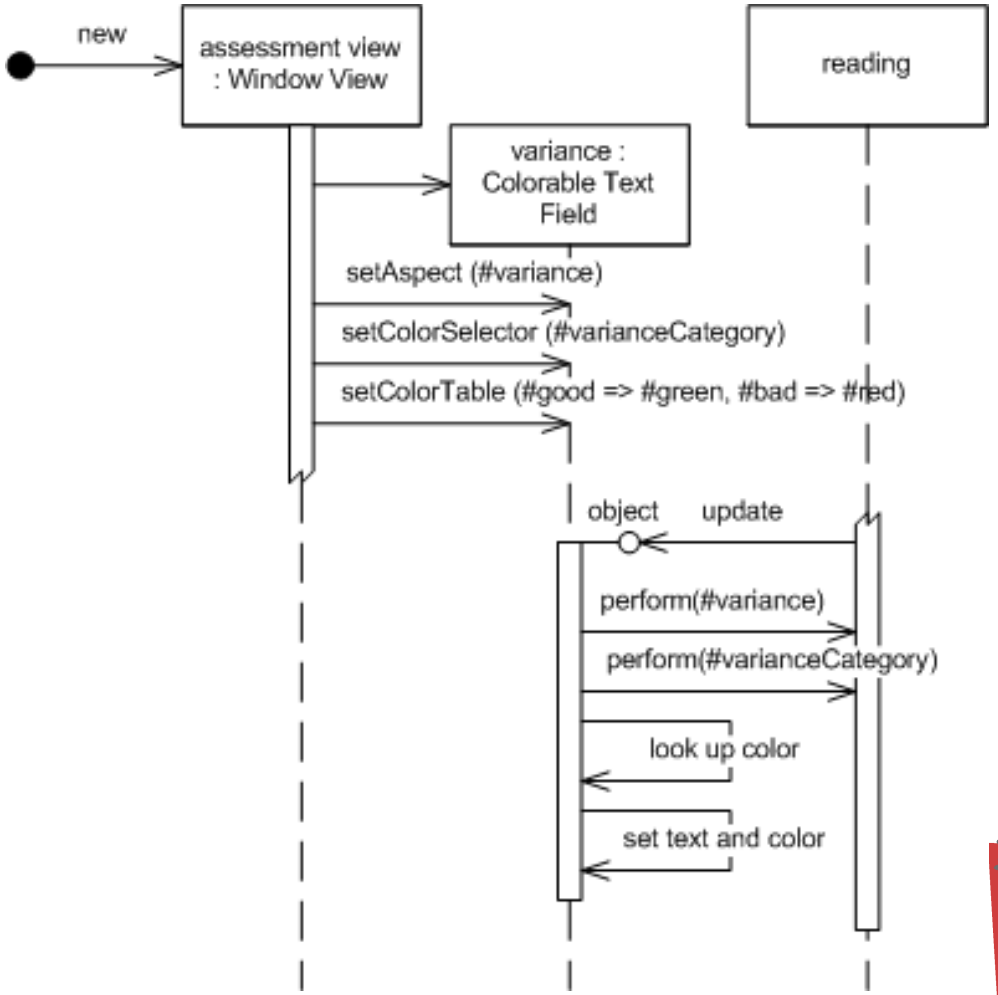




MVC – ze specyficznymi kontrolkami

W specjalistycznych kontrolkach umieszczamy dodatkową funkcjonalność:

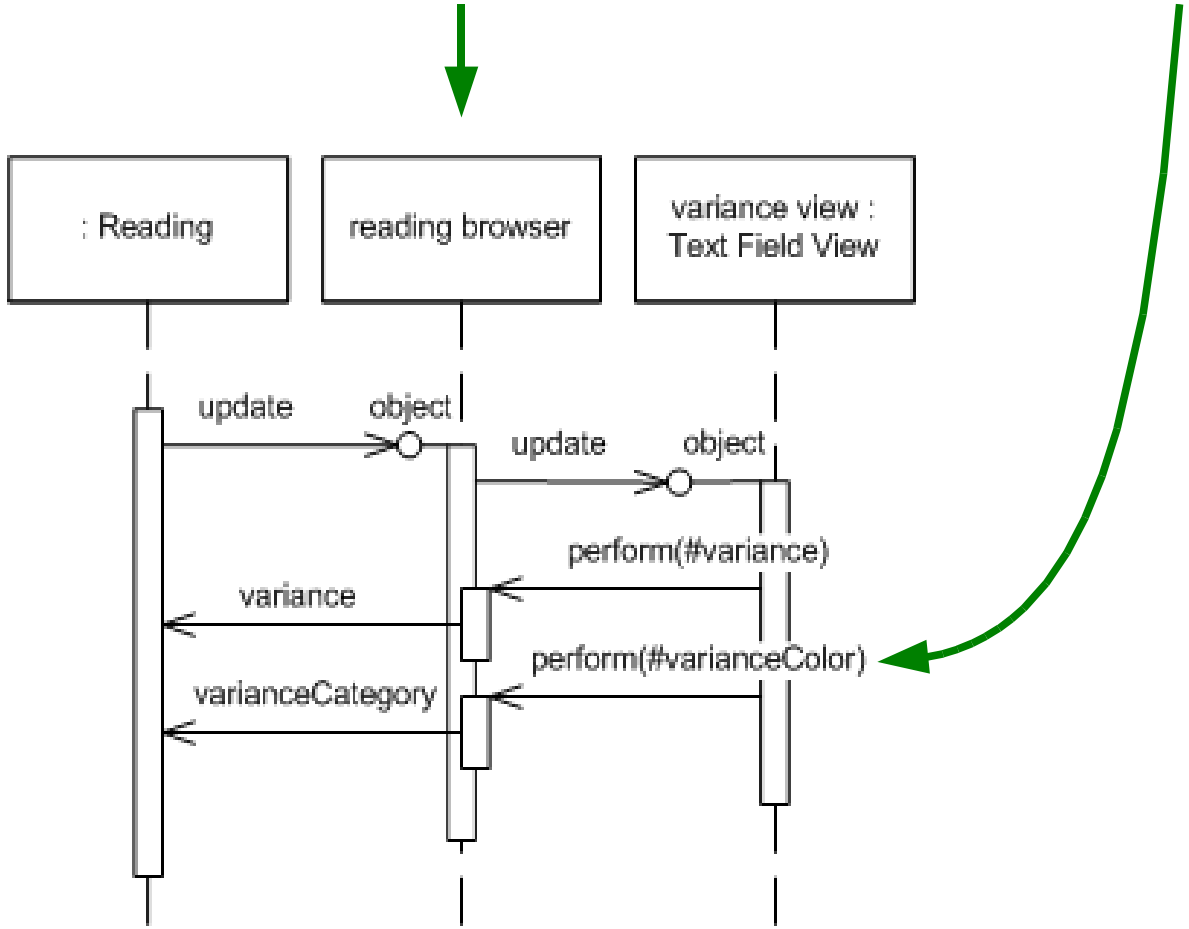
- np. tablicę kolorów fontu, dla pewnej wartości z zakresu 0 do 1.
- w ten sposób model musi udostępniać dodatkową informację o nowym wskaźniku



Pośredni model prezentacji (intermediate presentation model)

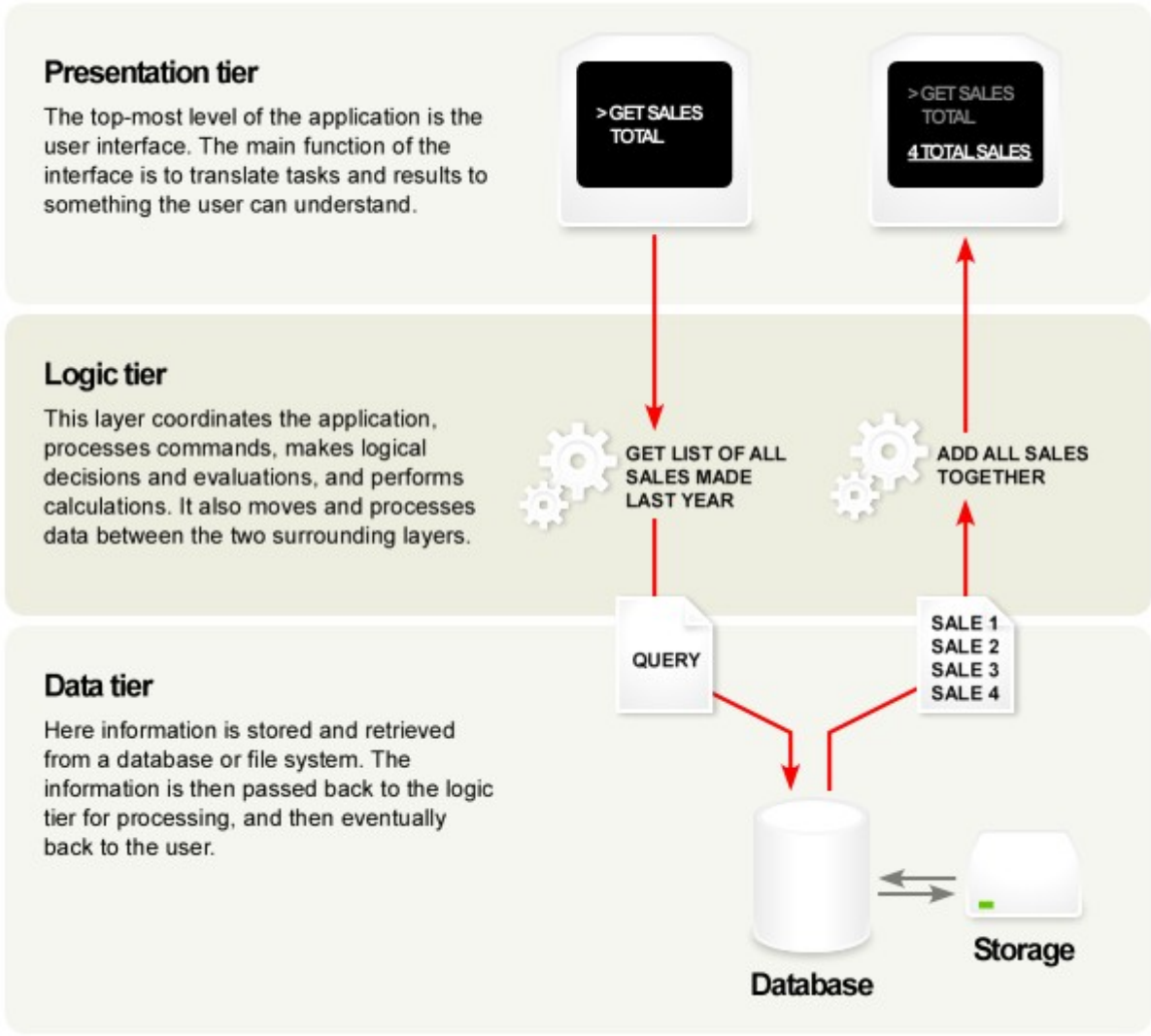
Tworzymy model dla widoku, czyli pewien obiekt tłumaczący wartości z dziedziny aplikacji na dziedzinę interfejsu użytkownika.

Model prezentacji przechowuje i udostępnia konkretne informacje związane bezpośrednio z interfejsem użytkownika: czyli np. kolor.



MVC kontra Architektura Warstwowa

- Porównanie architektury warstwowej z MVC
  - mylona przez wielu autorów z MVC,
  - nowa w stosunku do MVC: koncepcja trzech warstw pochodzi z lat 90:
    - html – prezentacja
    - application server
    - database
  - bardziej można ją porównać z wzorcem model aplikacji lub z wzorcem model prezentacji



Powiadomienie (*Notification*)

Obiekt zbierający informacje o błędach lub innych wiadomościach związanych z dziedziną aplikacji, który następnie jest przekazywany do warstwy (modułu) prezentacji.

Moduł prezentacji następnie może wyświetlić odpowiednie komunikaty zawarte w obiekcie Notification.

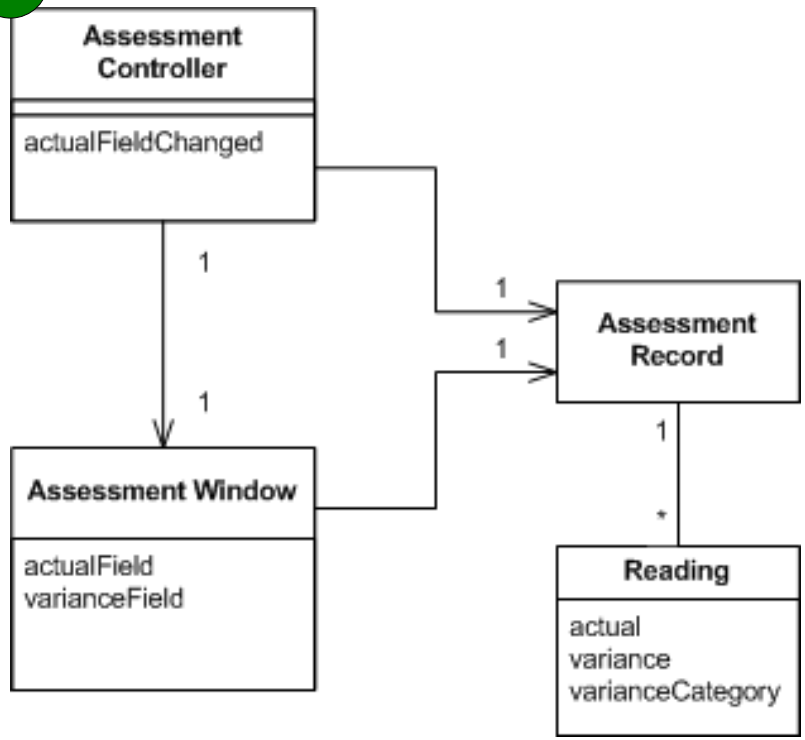
W najprostszej postaci Notification jest zbiorem napisów.

Kiedy używać? Np. w przypadku gdy walidacja wykonywana jest warstwie, która nie ma dostępu do prezentacji.

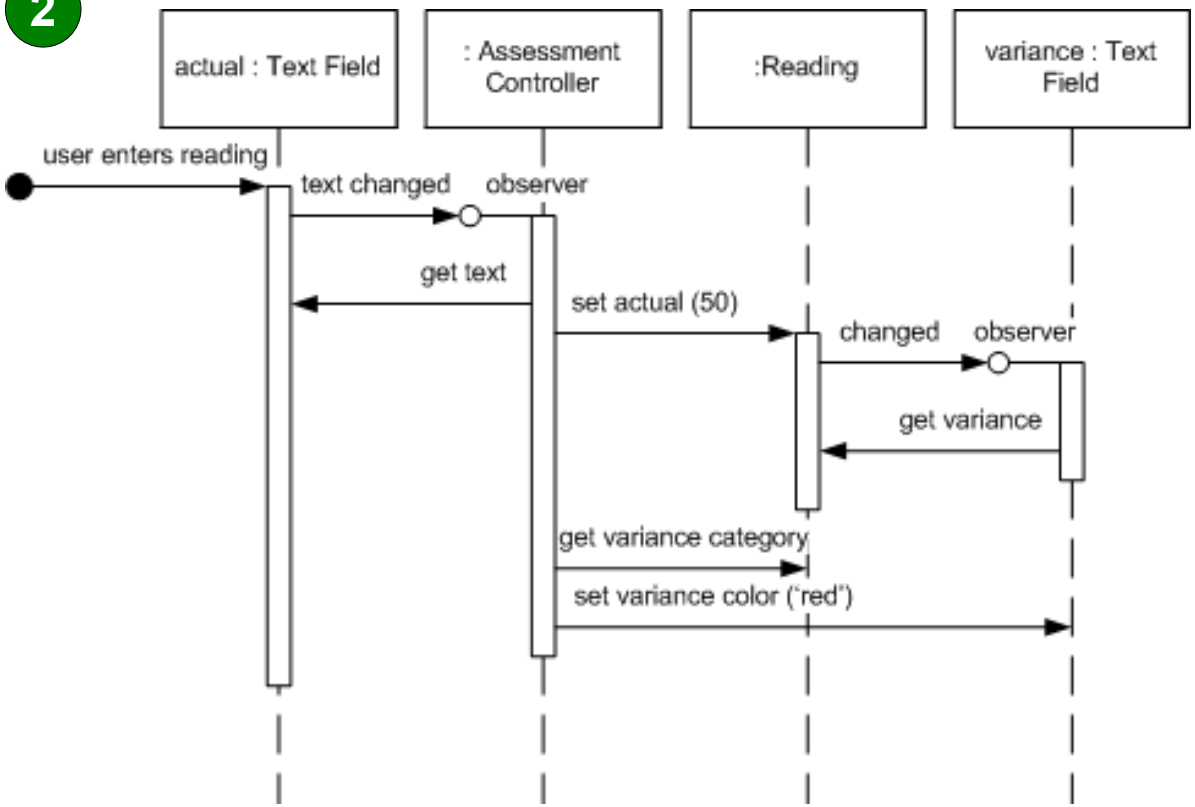
Kontroler nadzorujący (supervising controller)

- Wzorzec wykorzystuje kontroler do obsługi komunikatów pochodzących od użytkownika (koncepcja klasyczna), oraz do obsługi złożonej logiki prezentacji (w naszym przypadku do określenia koloru).
- Niezmiernie ważny w testach automatycznych.

1



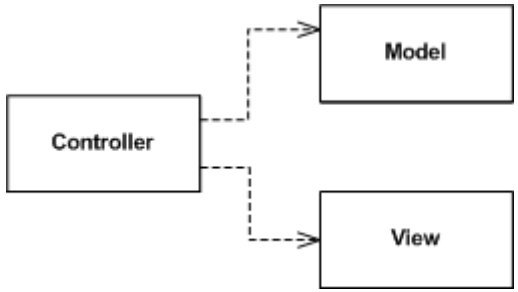
2



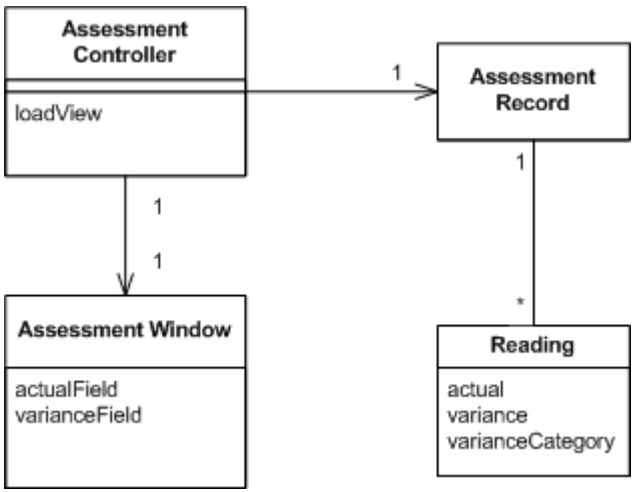
Widok pasywny (passive view)

- Cała funkcjonalność: reakcja na komunikaty użytkownika i zarządzanie prezentacją włącznie z logiką aplikacji specyficzną dla dziedziny jest realizowana przez kontroler.
- Najczęściej stosowany w aplikacjach typu rich client ze względu na możliwość stworzenia automatycznych testów.

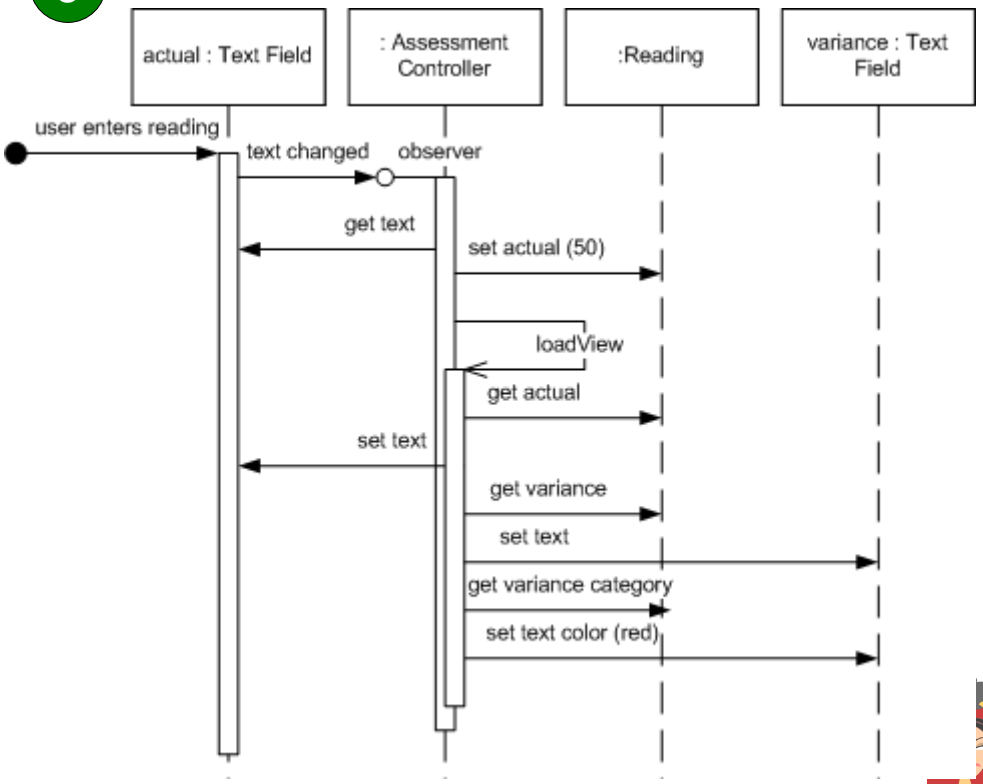
1



2



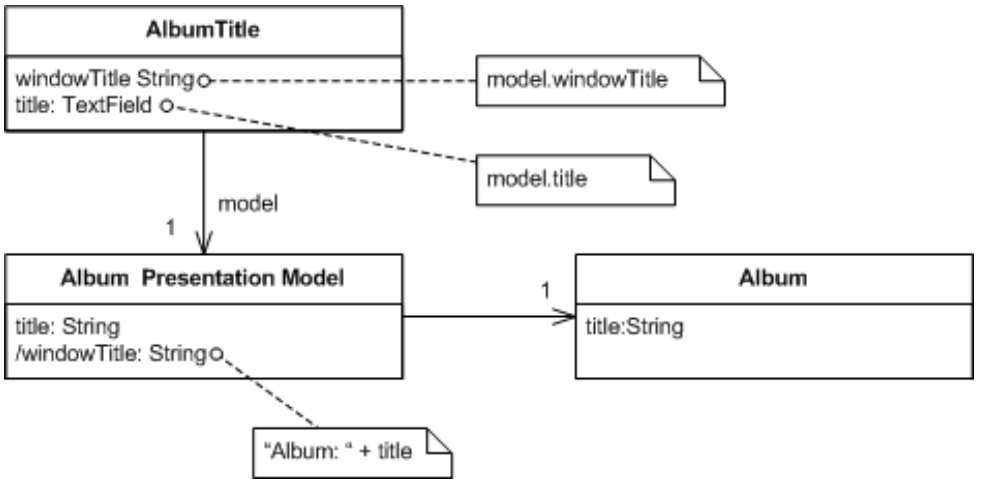
3



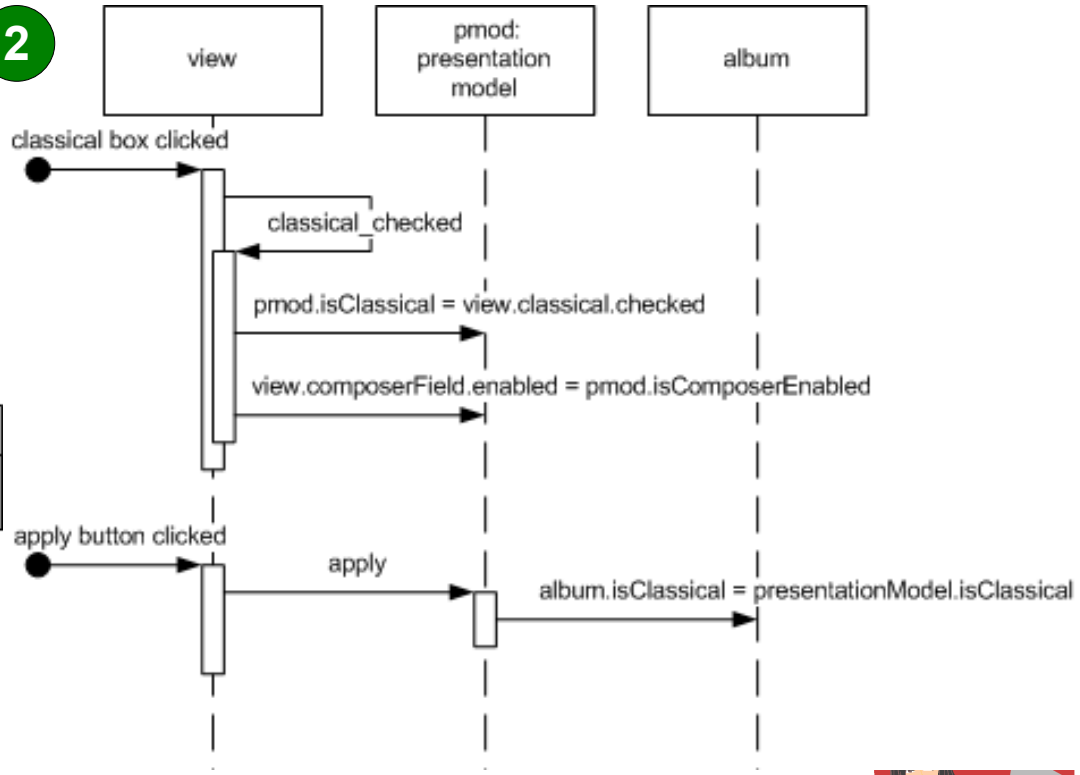
Model prezentacji (presentation model)

- Umożliwia reprezentację stanu i zachowania prezentacji niezależnie od kontrolek GUI używanych na ekranie.
- Oznacza, to że stan ekranu nie znajduje się w kontrolkach lecz w modelu.
- Model prezentacji może odwoływać się do wielu obiektów dziedziny aplikacji. Nie jest zatem fasadą obiektów dziedziny
- Model prezentacji można uznać za pewną 'abstrakcję' niezależną od platformy GUI (*GUI framework*)

1



2



Synchronizacja przepływu (flow synchronization)

Używany w sytuacjach gdy w kilku widokach pokazujemy dane pochodzące z tego samego źródła.

Jeżeli na jednym ekranie użytkownik zmieni wartość jakiegoś pola wówczas formularz uaktualnia kopię sesji (czyli np. Recordset) a następnie **samodzielnie** wymusza pozostałe okna, które pokazują tą samą informację aby się odświeżyły.

Rozwiązanie stosuje się dla małych aplikacji.

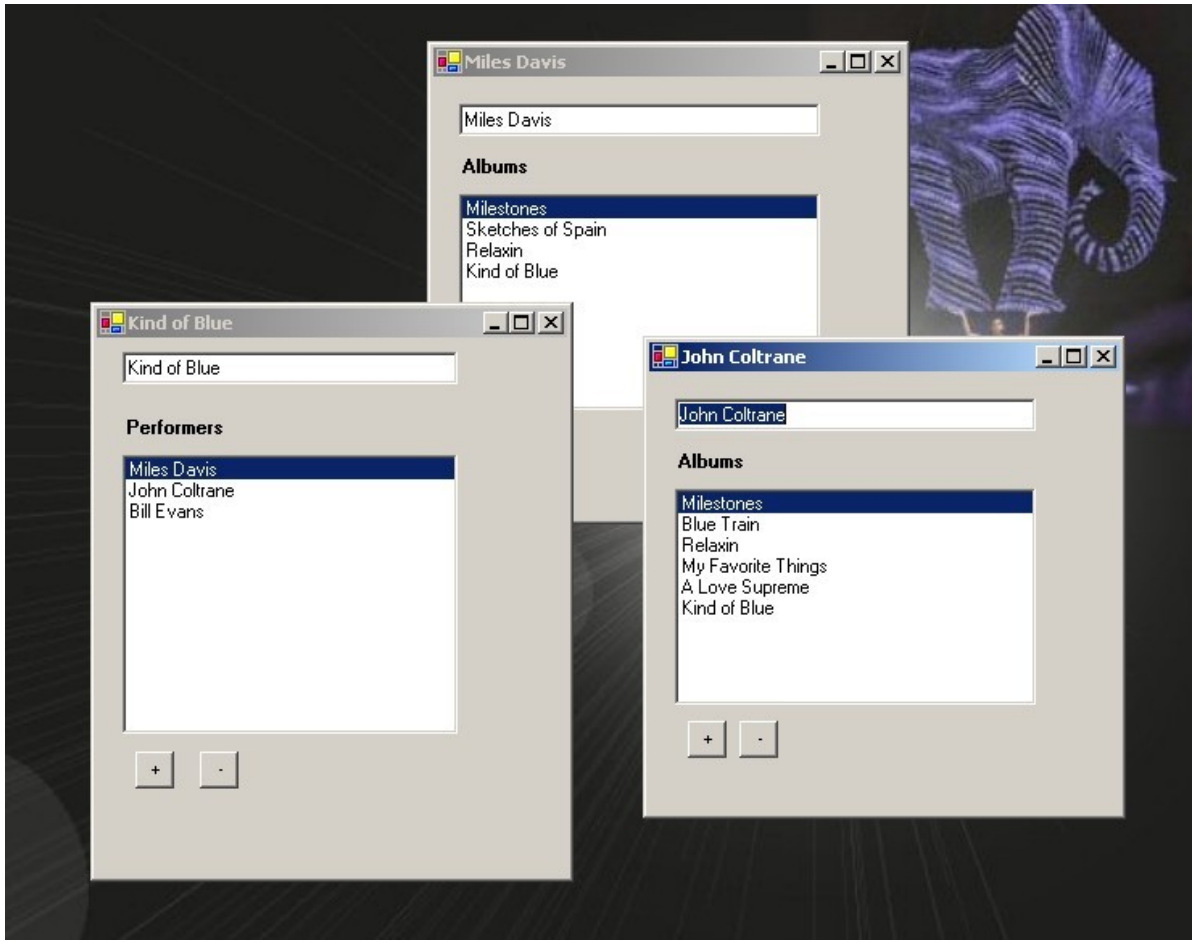
Typowym przykładem jest program z głównym oknem, oraz szeregiem okien modalnych dostępnych z głównego.

Problem z wzorcem tym polega na tym, że wprowadzamy zbyt duże zależności pomiędzy widokami.

Synchronizacja obserwująca (observer synchronization)

Synchronizacja wielu okien. Każde okno obserwuje pewien współdzielony zbiór danych z obszaru dziedziny.

Jeden z głównych wzorców składowych MVC.



Źródło: http://www.martinfowler.com/eaaDev/MediatedSynchronization.html

Synchronizacja obserwatora (kod w C#)

Dziedzina aplikacji

```
1) class DomainObject {
2)     public void SignalChanged()
3)     {
4)         if (Changed != null) Changed(this, null);
5)     }
6)     public event DomainChangeHandler Changed;
7)
8)     public delegate void DomainChangeHandler(
                DomainObject source, EventArgs e);
9) }
10)
11)   class Album : DomainObject {
12)     // ...
13)   public string Title
14)   {
15)       get { return _title; }
16)       set
17)       {
18)             _title = value;
19)             SignalChanged();
20)       }
21)    }
22)    string _title;
23)  }
```

2

1

Prezentacja

```
1) class FrmAlbum {
2)    //...
3)     public FrmAlbum(Album album)
4)     {
5)         InitializeComponent();
6)         this._album = album;
7)        observeDomain();
8)        load();
9)     }
10)   private Album _album;
11)   private void observeDomain()
12)   {
13)         _album.Changed +=
14)         new DomainChangeHandler(Subject_Changed);
15)         foreach (Performer p in _album.Performers)
16)           p.Changed +=
17)                 new DomainChangeHandler(Subject_Changed);
18)    }
19)    private void Subject_Changed(
20)           DomainObject source, EventArgs e)
21)    {
22)         load();
23)    }
24) }
```

3

4

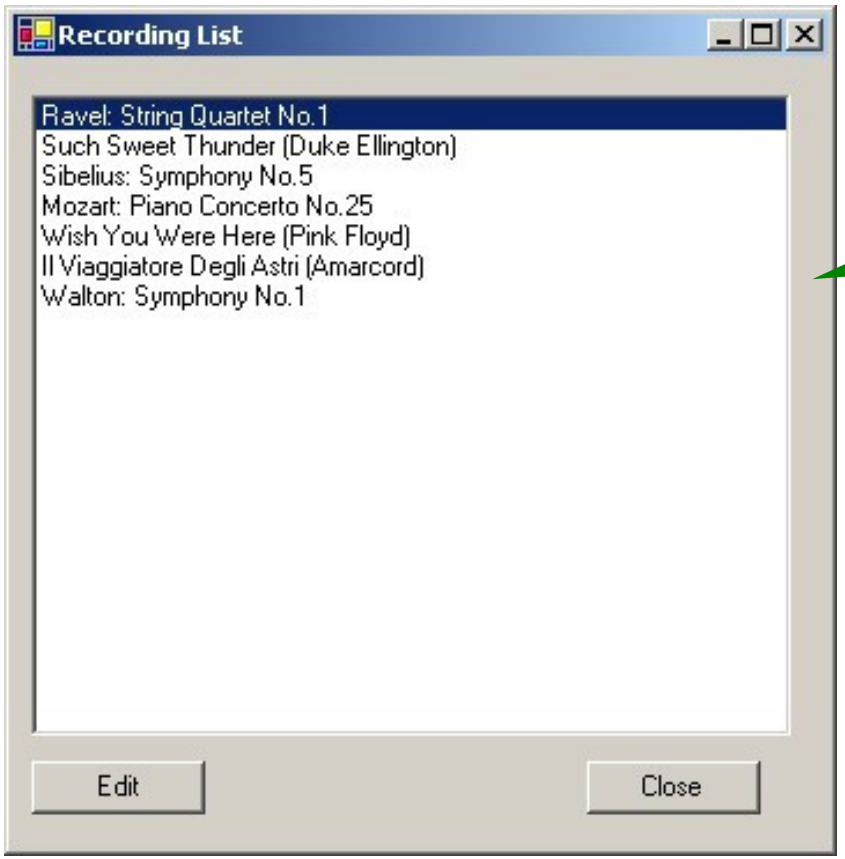
5

6

Selektor prezentacji (presentation chooser)

Wybiera ekran (formularz) odpowiedni dla konkretnego obiektu z dziedziny aplikacji.

Stosowany w sytuacji, gdy w danym momencie musimy wyświetlić różne formularze dla różnych aktualnie wybranych obiektów.

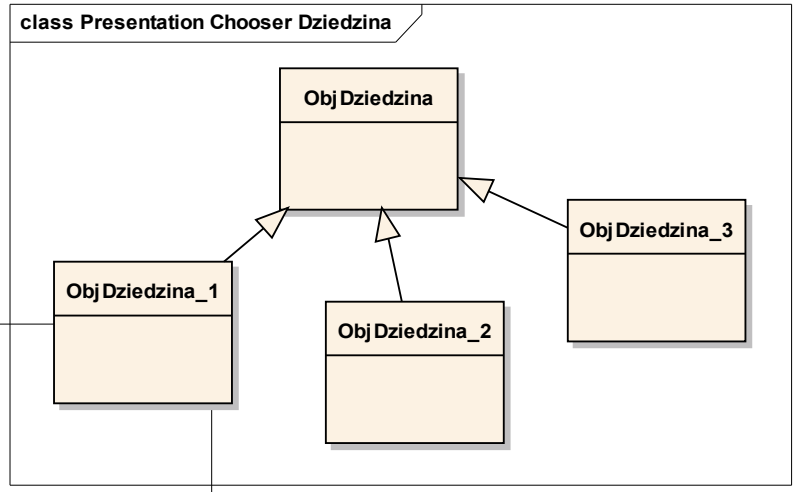


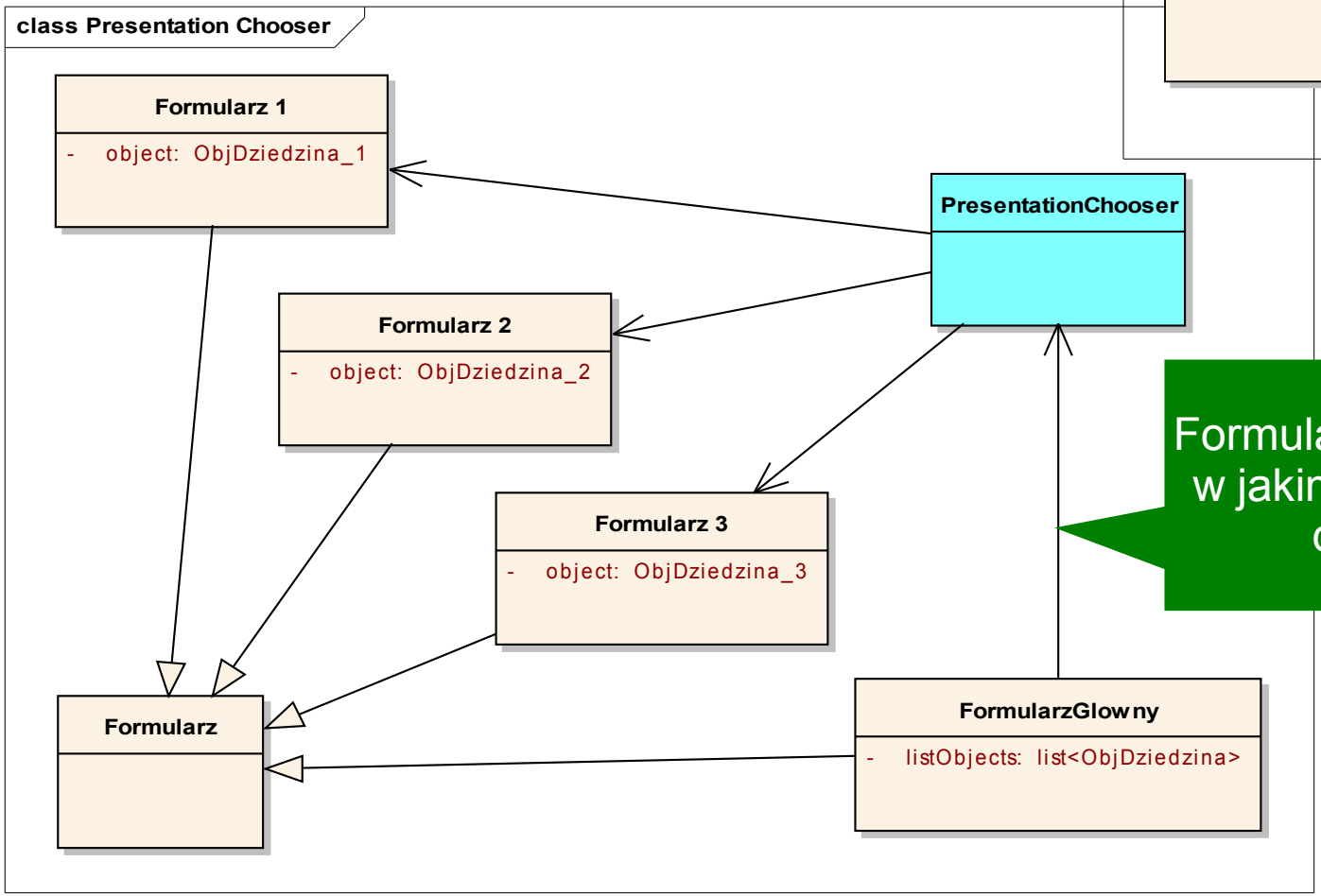
Edycja jest różna dla pozycji muzyki klasycznej a inna dla muzyki popularnej.

Selektor prezentacji (presentation chooser)

Tworzymy osobną klasę, która na podstawie obiektu dziedziny decyduje w którym oknie trzeba go wyświetlić.

Rozwiązanie związane z wzorcem **Fabryka**.





Dziękuję za uwagę.

Chcemy być coraz lepsi!

Jeżeli coś cię zainteresowało napisz e-maila:

- robert@iem.pw.edu.pl

Jeżeli coś cię bardzo znudziło napisz e-maila:

- robert@iem.pw.edu.pl

Jeżeli zauważyłeś błąd napisz e-maila:

- robert@iem.pw.edu.pl