Omówienie wzorców wykorzystywanych w Prism

Dominika Różycka, 254130

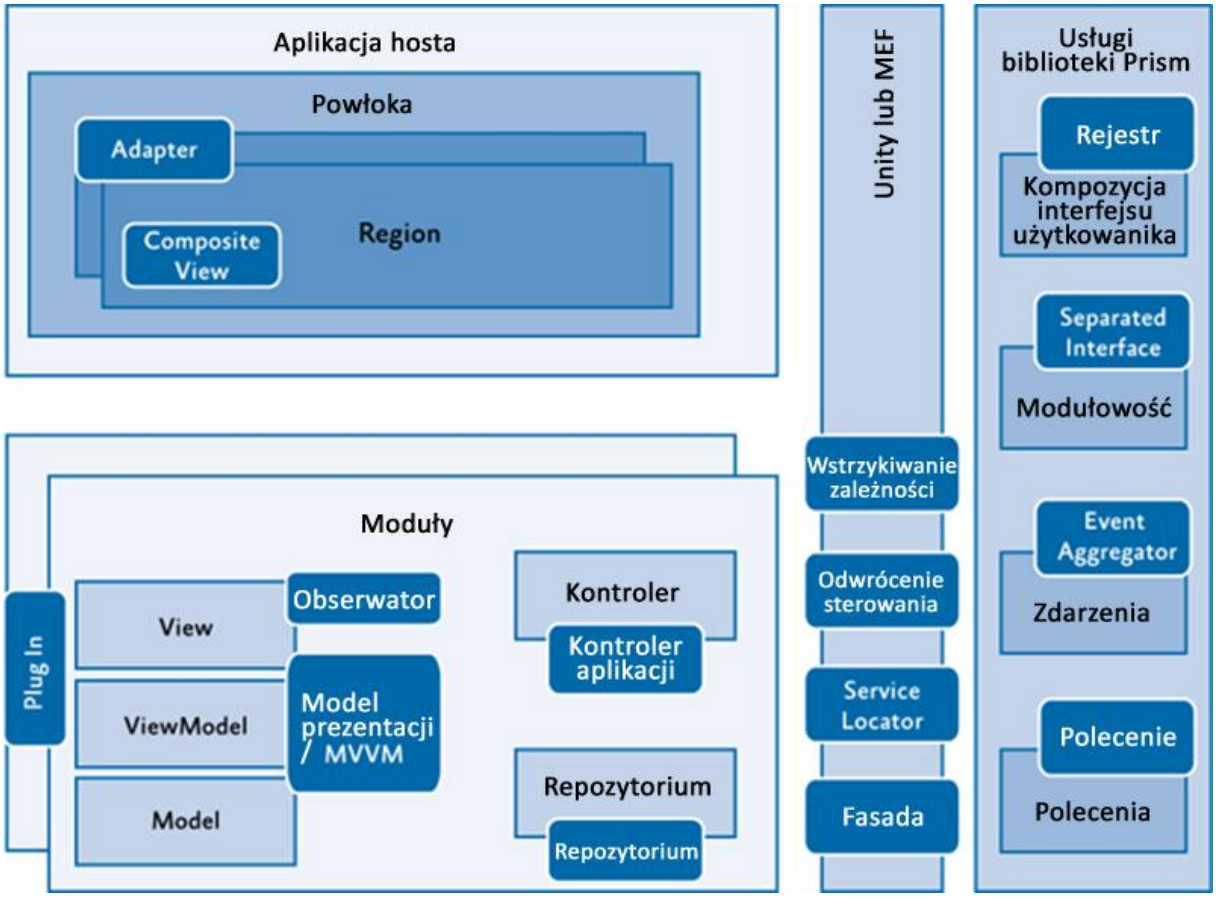
Wzorzec projektowy - w inżynierii oprogramowania jest to uniwersalne i sprawdzone w praktyce rozwiązanie często pojawiających się, powtarzalnych problemów projektowych. Demonstruje powiązania i zależności pomiędzy klasami oraz obiektami i ułatwia tworzenie, modyfikację oraz pielęgnację kodu źródłowego. Jest opisem rozwiązania, a nie jego implementacją.
Wzorce projektowe stosowane są w projektach wykorzystujących programowanie obiektowe.

Tworząc aplikacje, zazwyczaj spotykamy się z wzorcami projektowymi lub też sami je stosujemy.

W przykładowej implementacji przy użyciu Prism, zalecane są wzorce:

- Adapter,
- Application Controller (Kontroler aplikacji),
- Command (Polecenie),
- Composite (Kompozyt) i Composite View,
- Dependency Injection (Wstrzykiwanie zależności),
- Event Aggregator,
- Façade (Fasada),
- Inversion of Control (Odwrócenie sterowania),
- Observer (Obserwator),
- Model-View-ViewModel (MVVM) (,
- Registry (Rejestr),
- Repository (Repozytorium),
- Separated Interface,
- Plug-In,
- Service Locator (Lokalizator Usługi)

Poniższa ilustracja pokazuje przykładową architekturę aplikacji tworzonej przy użyciu biblioteki Prism i niektórych z typowych wzorców. Prostsza aplikacja prawdopodobnie wykorzysta tylko kilka z nich.



Rys 1. Przykładowa architektura aplikacji kompozytowej przy użyciu wymienionych wzorców projektowych.

Adapter

Wzorzec projektowy adapter - jak sama nazwa wskazuje, dopasowuje interfejs jednej klasy do oczekiwanego przez inną klasę.
W bibliotece Prism, wzorzec Adapter jest używany do dostosowywania regionów do Windows Presentation Foundation (WPF) ItemsControl, ContentControl i Selector. Aby zobaczyć zastosowany wzorzec Adapter trzeba zobacztć plik ItemsControlRegionAdapter.cs w bibliotece Prism.

Wzorzec Kontroler Aplikacji

Wzorzec projektowy Kontroler Aplikacji pozwala nam oddzielić zadanie tworzenia i wyświetlania widoków w klasie kontrolera. Ten rodzaj kontrolera jest nieco inny niż kontroler znany z aplikacji MVC. Zadaniem kontrolera aplikacji jest hermetyzacja kontroli prezentacji widoku. Może się zajmować widokami posiadającymi instancję - czyni to poprzez zamieszczenie ich w odpowiednim pojemniku w interfejsie użytkownika (UI), przełączanie się między widokami, które mają ten sam pojemnik i czasami koordynuje komunikację między widokiem a ViewModelem. Chociaż nazwą wzorca jest Kontroler Aplikacji, kontrolery są często zawężane do podzbioru aplikacji, takie jak kontroler modułu w aplikacji Prism lub kontroler, który obejmuje zestaw pokrewnych widoków. W rezultacie, często będziemy mieli więcej niż jeden kontroler w aplikacji Prism. Dla przykładu implementacji tego wzorca zobacz klasę OrdersController w Stock Trader Reference Implementation (Stock Trader RI).

Wzorzec Polecenie

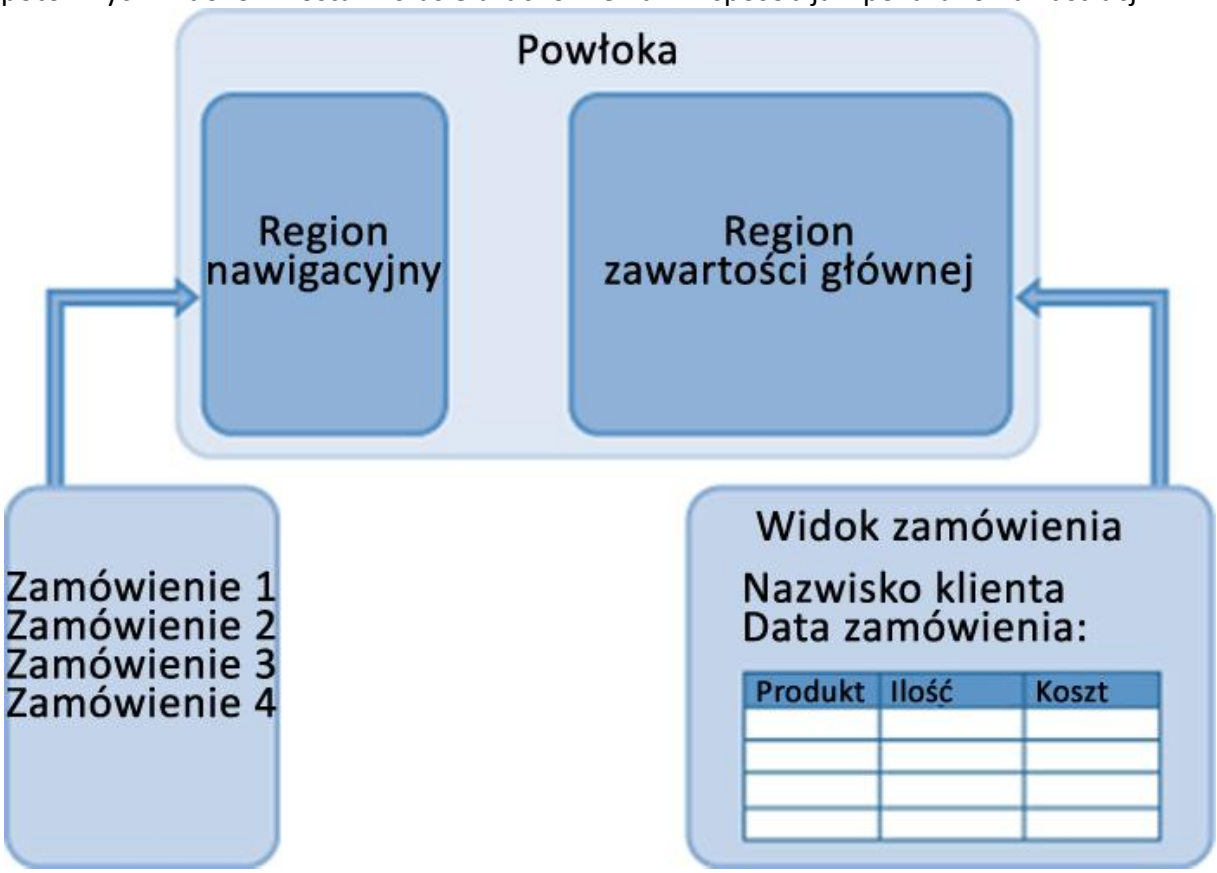
Wzorzec Polecenie jest wzorcem projektowym , w którym obiekty są używane do reprezentowania akcji. Obiekt Command hermetyzuje akcję i jej parametry. Pozwala to na oddzielenie wywołującego polecenia i jego obsługi. Biblioteka Prism pozwala na CompositeCommand - umożliwia ono uruchomienie kilku metod na skutek danej akcji (ICommand i DelegateCommand).

Aby zobaczyć użycie CompositeCommand i DelegateCommand w Stock Trader RI, zobacz pliki StockTraderRICommands.cs i OrderDetailsViewModel.cs.

Polecenia oraz zdarzenia pozwalają na luźne powiązania między widokiem a ViewModel oraz między poszczególnymi ViewModel'ami. PRISM dostarcza kilku mechanizmów, które programiści musieli wcześniej zazwyczaj samodzielnie zaimplementować. Warto zapamiętać różnice pomiędzy zdarzeniami a komendami, programiści często zapominają o nich. Zdarzenia służą do komunikacji, powiadomienia o jakieś akcji i nie mogą być stosowane do wykonywania poleceń na podstawie interakcji użytkownika z interfejsem – do tego służą komendy, które mają dodatkowo specjalną właściwość CanExecute. Innymi słowy, komenda oznacza „zrób coś!", z kolei zdarzenie oznacza to, że „akcja właśnie została wykonana".

Kompozyt i Composite View

Sednem aplikacji kompozytowej jest zdolność do łączenia pojedynczych widoków w widok kompozytowy. Często widok kompozytowy definiuje layout dla widoków potomnych. Na przykład, powłoka aplikacji może definiować obszary: nawigacyjny oraz zawartości dla potomnych widoków hosta w czasie uruchomienia - w sposób jaki pokazano na ilustracji:



Rys 2. Przykładowa kompozycja.

W Stock Trader RI wzorzec można zaobserwować w użyciu regionów w powłoce. Powłoka definiuje regiony, które modulują lokalizowanie i dodawanie widoków podczas procesu inicjalizacji.
Aby zobaczyć przykłady definiowania regionów zobacz Shell.xaml
Widoki kompozytowe nie mogą być dynamicznie składane - tak jak w przypadku kiedy używamy regionów Prism. Widok kompozytowy może również być widokiem , który zbudowany jest z kilku innych widoków potomnych, które są tworzone statycznie przez definicję interfejsu użytkownika.
Przykładem tego są potomne kontrolki użytkownika zadeklarowane w XAML.

Wzorzec Wstrzykiwanie zależności

Wzorzec Wstrzykiwanie zależności jest wyspecjalizowaną wersją wzorca Odwróconego sterowania (Inversion of Control). Kiedy problem jest odwrócony - jest to proces uzyskiwania potrzebnej zależności.

Wstrzykiwanie zależności używany jest w całym Stock Trader RI i Bibliotece Prism. Podczas używania kontenera, odpowiedzialność za konstrukcję spoczywa na kontenerze zamiast używającej go klasie.

Podczas konstruowania obiektu, kontener wstrzykiwania zależności rozwiązuje wszystkie dodatkowe zależności. Dzięki temu, konkretna implementacja zależności może być zmieniana łatwiej kiedy system ewoluuje.

Usprawnia to działanie testów i rozrost systemu w czasie ze względu na luźniejsze połączenie.

Stock Trader RI używa Managed Extensibility Framework (MEF) aby pomóc w zarządzaniu zależnościami między elementami. Jednakże, biblioteka Prism sama w sobie nie jest przywiązana do konktretnego kontenera wstrzykiwania zależności - mamy wolny wybór w doborze kontenera wstrzykiwania zależności, ale musimy zapewnić adapter, który implementuje interfejs IServiceLocator.

Biblioteka Prism zapewnia adaptery dla obu: MEF i Unity Application Block (Unity).
W celu zobaczenia przykładu komponentu z wstrzykiwaniem zależności w Stock Trader RI zobacz konstruktor w NewsController.cs.
Przykłady Unity: klasa Modulent w UI Composition QuickStart.

Wzorzec EventAggregator

Wzorzec Event Aggregator przełącza zdarzenia z wielu obiektów na pojedynczy obiekt dla ułatwienia rejestracji klientów. W bibliotece Prism, odmiana wzorca EventAggregator pozwala wielu obiektom lokalizować i publikować lub zapisać coś do zdarzenia.

Przykład: EventAggregator i PubSubEvent w bibliotece Prism.PubSubEvents.

Przykład użycia Event Aggregatora w Stock Trader RI: WatchListViewModel.cs

Wzorzec Fasada

Wzorzec Fasada upraszcza bardziej złożny interfejs lub zestaw interfejsów w celu ułatwienia ich stosowania lub wyizolowania dostępu do tych interfejsów. Biblioteka Prism zapewnia fasady dla kontenera i usług logowania aby pomóc w izolowaniu biblioteki od zmian w tych usługach.

Pozwala to użytkownikowi biblioteki na dostarczanie własnych usług, które będą działać z biblioteką Prism.

Interfejsy: IServiceLocator i ILoggerFacade definiują interfejsy fasady w bibliotece Prism do oczekiwania komunikacji z kontenerem lub usługą logowania.

Wzorzec odwrócenie sterowania

Często wzorzec odwrócenia sterowania używany jest w celu umożliwienia rozbudowy w klasie lub frameworku. Dla przykładu, klasa zaprojektowana z modelem zdarzeniowym w niektórych punktach egzekucji odwraca sterowanie poprzez zezwolenie detektorom zdarzeń na podjęcie akcji kiedy zdarzenie jest wywołane.

Dwie formy wzorca odwróconego sterowania zademonstrowane w bibliotece Prism i Stock Trader RI dołączają wstrzykiwania zależności i Template Method. We wzorcu Template Method, klasa bazowa dostarcza wzór lub proces, który wywołuje metody wirtualne lub abstrakcyjne.

Z tego powodu dziedziczona klasa może nadpisywać odpowiednie metody aby włączyć wymagane zachowanie.

W bibliotece Prism jest to pokazane w klasie UnityServiceLocatorAdapter, aby zobaczyć kolejny przykład użycia wzorca Template zobaczmy StockTraderRIBootstrapper.cs w StockTraderRi.

Wzorzec Obserwator

Wzorzec Obserwator dąży do uniezależnienia zainteresowanych zmianą stanu obiektu z zmienianego obiektu. W frameworku .NET, jest to często poszukiwane przez zdarzenia.

Prism pokazuje odmianę wzorca Obserwator w celu oddzielenia zapytania o interakcję z użytkownikiem z aktualnie wybranej interakcji. Jest to robione przez obiekt InteractionRequest , który jest często oferowany przez ViewModel w MVVM.

InteractionRequest jest obiektem który hermetyzuje zdarzenie zarządzane przez widok. Kiedy widok otrzymuje zapytanie o interakcję. może wybrać sposób obsługi interakcji. Widok może decydować o wyświetleniu modalnego okna informacji zwrotnej dla użytkownika lub może wyświetlać dyskretne powiadomienia bez przerywania przepływu pracy użytkownika.

Oferując to żądanie jako obiekt zapewnia możliwość data-bindingu w WPF do żądań i precyzowania odpowiedzi bez konieczności stosowania code-behind w widoku.

Wzorzec Model-View-ViewModel

Prezentacja modelu jest jednym z wzorców interfejsu użytkownika, które koncentrują się na utrzymaniu logiki prezentacji z dala od reprezentacji wizualnej. Ma to na celu oddzielenie warstwy prezentacji od reprezentacji wizualnej, co pomaga poprawić komfort wprowadzania zmian i testowalność.

Pokrewne interfejsy użytkownika implementują Model-View-Controller (MVC) i Model-View-Presenter (MVP). Podejście MVVM zademonstrowany w Stock Trader RI jest specjalną implementacją wzorca Presentation Model.

Biblioteka Prism została stworzona do bycia neutralną w stosunku do wyboru oddzielonych wzorców interfejsów użytkownika. Można zaimplementować którykolwiek z wzorców, jednakże biorąc pod uwagę obiekty WPF do wiązania danych (data binding), poleceń i zachowań, wzorzec MVVM jest zalecanym podejściem i doradcy Prism dostarczają dokumentację i sample aby łatwiej zacząć przygodę z MVVM.

Przykłady MVVM: Basic MVVM QuickStart, pliki: QuestionnaireView.xaml, QuestionnaireView.xaml.cs, i QuestionnaireViewModel.cs.

Wzorzec Rejestr

Wzorzec Rejestr określa podejście do lokalizowania jednego lub więcej obiektów z obiektu dobrze znanego.Biblioteka Prism stosuje wzorzec Rejestr kojarząc typy widoku do regionu. Interfejs IRegionViewRegistry i klasa RegionViewRegistry definiują rejestr używany do kojarzenia nazw regionu z typami widoków utworzonymi gdy te regiony są ładowane.

Przykład: Moduleint.cs w UI Composition QuickStart.

Wzorzec Repozytorium

Repozytorium pozwala nam oddzielić sposób w jaki pozyskujemy dane dla aplikacji z kodu, który potrzebuje danych.

Repozytorium reprezentuje zbiór obiektów domeny, które kod aplikacji może używać bez konieczności łączenia ze specjalnym mechanizmem, który pobiera te obiekty. Obiekty domen są częścią modelu aplikacji i poprzez pozyskiwanie tych obiektów przez repozytorium, odzyskiwanie repozytorium i strategia aktualizacji może być zmieniana bez wpływu na pozostałą część aplikacji.

Ponadto interfejs repozytorium stał się łatwą zależnością do zastąpienia dla celów testów jednostkowych.

Separated Interface i Plug-In

Zdolność do lokalizowania i ładowania modułów, w czasie wykonywania aplikacji daje większe możliwości rozwoju równoległego, rozszerza możliwości wyboru i wdrażania modułu i zachęca do luźniejszej architektury. Następujące wzorce włączają tę możliwość:

- **Separated Interface** - ten wzorzec zmniejsza przymus łączenia poprzez zamieszczenie definicji interfejsu w oddzielonej do implementacji paczce. Kiedy używamy Prism z Unity, każdy modłu implementuje interfejs IModule.
  Przykład implementacji modułu w UI Composition QuickStart - Moduleint.cs
- **Plug-In** - wzorzec ten pozwala konkretnej implementacji klasy na bycie zdefiniowaną w czasie pracy, aby uniknąć konieczności rekompilacji, kiedy konkretna implementacja jest w użyciu lub z powodu zmian w niej zachodzących.
  W bibliotece Prism jest to obsługiwane przez DirectoryModuleCatalog, ConfigurationModuleCatalog i ModuleInitialiyer, które pracuj razem w celu lokalizowania i inicjalizacji plug-inów IModule.
  Przykłady wspierania plug-inów: DirectoryModuleCatalog.cs, ConfigurationModuleCatalog.cs i ModuleInitializer.cs w bibliotece Prism.

Ważne: MEF został zaprojektowany do obsługi modelu Plug-In, co pozwala komponentom na deklaratywny eksport i import konkretnych implementacji.

Wzorzec Lokalizator Usługi

Wzorzec Lokalizator Usługi rozwiązuje ten sam problem co Wstrzykiwanie Zależności (Dependency Injection), ale używa odmiennego podejścia. Pozwala on klasom na lokalizowanie konkretnych usług, którymi są zainteresowane bez potrzeby wiedzy kto implementuje usługę.

Najczęściej jest on stosowany jako alternatywa dla Wstrzykiwanie Zależności (Dependency Injection), ale bywa, że niekiedy klasy będą potrzebowały używać lokalizatora usług zamiast wstrzykiwania zależności, jak w przypadku potrzeby rozwiązania wielu realizatorów usługi.

Przykład: W bibliotece Prism w usłudze ModuleInitializer rozwiązuje indywidualne IModule.

Przykład użycia UnityContainer do lokazlizowania usługi - w UI Composition QuickStart - Moduleint.cs.