# Komunikacja między luźno powiązanymi komponentami

### Mechanizm komend

1. Rozpoczynamy od SimplePrismApp z poprzednich zajęć
2. Odnajdujemy Delegate command, który leży w class ButtonViewAViewModel
3. Dołączamy potrzebne biblioteki

using Microsoft.Practices.Prism.Mvvm;

using System.Windows.Input;

using System.Windows;

1. Używany przez nas delegate command implementuje interfejs ICommand

dzięki temu można zastąpić: na

public ICommand SwitchViewCommand { get; set; }

1. ICommand wywoływany jest w ButtonViewA.xaml, jak poniżej

<Grid>

<Button Content="Przełącz na widok A" Command="{Binding SwitchViewCommand}" />

</Grid>

1. Przebudujemy to by zastąpić mechanizmem Composite command

Zmieniamy icommand na

public ComponentCommand SwitchViewCommand { get; set; }

1. Odnajdujemy SwitchViewCommand = new DelegateCommand(SwitchView);
2. I zamieniamy na

SwitchViewCommand = new CompositeCommand();

SwitchViewCommand.RegisterCommand(new DelegateCommand(SwitchView));

1. Uzyskany kod powinien działać, już z wykorzystaniem CompositeCommand
2. Tworzymy nową klasę w ModuleA:

public static class GlobalCommands

using Microsoft.Practices.Prism.Mvvm;

using Microsoft.Practices.Prism.Commands;

public static CompositeCommand SwitchViewCommand = new CompositeCommand();

GlobalCommands.SwitchViewCommand.RegisterCommand(new DelegateCommand(SwitchView));

1. Przechodzimy do do ButtonAView – powinniśmy znaleźć:

<Grid>

<Button Content="Przełącz na widok A" Command="{Binding SwitchViewCommand}" />

</Grid>

1. Zamieniamy powyższy kod na

<Grid>

<Button Content="Przełącz na widok A" Command="{x:Static local:GlobalCommands.SwitchViewCommand}" />

</Grid>

1. Koniecznie trzeba pamiętać by dodać:

xmlns:local="clr-namespace:ModuleA"

1. Uzupełniamy EditorCatalogViewModell.cs, żeby zarejestrować komendę

public ModuleAViewViewModel(IAuthority authority)

{

this.authority = authority;

GlobalCommands.SwitchViewCommand.RegisterCommand(new DelegateCommand(kominukat));

}

1. Musimy jeszcze napisać funkcję komunikat

public void kominukat()

{

MessageBox.Show("druga komenda");

}

1. Ewentualne błędy mogą wynikać z braku podłączenia bibliotek:

using Microsoft.Practices.Prism.Commands;

using System.Windows;

### Mechanizm IEventAggregator

1. Dodajemy nową klasę do modułu A, nazywamy ją messageEvent i uzupełniamy:

public class messageEvent : CompositePresentationEvent<string> { }

1. Dodajemy do EditorCatalogViewModel:

IEventAggregator \_eventAggregator;

1. Do konstruktora dodajemy aggregator

public EditorCatalogViewModel(IProductRepository productsRepository, IEventAggregator eventAggregator): base(productsRepository)

1. I na początku deklarujemy event aggregator:

\_eventAggregator = eventAggregator;

1. W add product dodajemy:

\_eventAggregator.GetEvent<messageEvent>().Publish(“dodano element”);

1. W ModuleBViewViewModel dodajemy:

IEventAggregator \_eventAggregator;

1. Aktualizujemy konstruktor:

\_eventAggregator = eventaggregator;

eventaggregator.GetEvent<messageEvent>().Subscribe(komunikat,true);

1. Dopisujemy metodę:

public void komunikat(string tekst) { MessageBox.Show(“ otrzymano wiadomość od innego wątku:”+ tekst); }

1. Otrzymaliśmy wiadomość przesłaną przez mechanizm EventAggregator