ISSN 1846-6168
UDK 004.43

WPF i MVVM

WPF AND MVVM

Željko Knok, Mark Marčec

Stručni članak

Sažetak: *Današnje tržište softvera ima dosta razvijenu tehnologiju s kojom se bitno olakšava izrada i održavanje kompleksnih informacijskih sustava. Jedna od tih tehnologija je WPF (Windows Presentation Foundation) s primjenom MVVM (Model View ViewModel) obrasca. Ovaj rad se bavi analiziranjem MVVM obrasca u WPF-u uz opis značenje samog obrasca.*

Ključne riječi: *Model, View, ViewModel, Command, Converter, Binding, C#, XAML*

Professional paper

Abstract: *Today's software market has really advanced technology to benefit the process of designing and maintaining really complex information systems. One of the technologies is WPF (Windows Presentation Foundation) with the MVVM (Model View ViewModel) pattern. This work analyses the MVVM pattern in WPF and will describe the meaning of the pattern.*

Key words: *Model, View, ViewModel, Command, Converter, Binding, C#, XAML*

1. UVOD

Model View ViewModel je pattern ili obrazac koji omogućava odvajanje prezentacijskog dijela aplikacije (GUI-a) od implementacije same aplikacije. WPF tehnologija je od samog početka osmišljena sa ciljem da ide ruku pod rukom sa MVVM obrascem. Ovaj obrazac je podijeljen u tri dijela. Model, gdje se definiraju podaci i njihovi tipovi. View ili pogled definira izgled korisničkog sučelja programa i prikazuje konačan izgled podataka te ViewModel koji je posrednik Modela i View-a te je zadužen za prezentaciju podataka i za navigaciju kroz korisničko sučelje. Binder je XAML koji oslobađa programera da ne mora pisati logiku koja služi za sinkronizaciju Viewa i ViewModela. Ovakvim načinom rada se bitno olakšava održavanje aplikacije te dodavanje novih mogućnosti.

2. ELEMENT BINDING I CONVERTERS

U ovom djelu predstavljena je ideja MVVM arhitekture kroz neke praktične primjere. Prvi korak je Binding koji će biti podijeljen u dva dijela Element Binding i Data Binding.

2.1. Element Binding

Element binding povezuje podatke između dvije kontrole na korisničkom sučelju. Binding se definira u XAML-u. Jednostavan primjer Element Binding-a bi bio povezivanje metode za ispis vrijednosti u TextBox i slidera.

Ako bi se radilo na stari način bez povezivanja ili bindinga, dohvatio bi se event kada se promijeni vrijednost slajdera, pa bi promjenu se vrijednost konvertirala u niz znakova, da bi se onda taj niz znakova dodijelilo kao vrijednost svojstva tekst ili Text željenog TextBlock-a. Taj kod bi izgledao ovako [2]:

```
private void slider_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
{
   lblSliderValue.Text = slider.Value.ToString();
}
```

Kod korisničkog sučelja sa mnogo kontrola ovakav kod može se nagomilati i smanjiti čitljivost koda koji je bitan za rad aplikacije. Primjenom bindig-a izbjegavamo gomilanje koda u „code-behind" datoteci (.xamlcs) nekog prozora. U XAML-u povezivanje gumba sa metodom za zatvaranje prozora izgleda kao na slijedećem primjeru.

```
<Button Command="{Binding CloseCommand}" Content="Zatvori" Style="{StaticResource ActionButton}"
ctrl:ControlExtensions.IsHighlighted="True"/>
```

2.1.1. TwoWay mod Binding-a

Dvije nove ključne riječi u XAML-u su Mode i UpdateSourceTrigger. Mode=TwoWay govori da meta bindinga može ažurirati izvor, dakle ovaj put je binding

obostran. Binding se vrši u suprotnom smjeru uz pomoć ConvertBack. Druga ključna riječ UpdateSourceTrigger=PropertyChanged govori da se poziva binding čim se promijeni tekst polja. UpdateSourceTrigger je enumeracija koja može primiti i neke druge vrijednosti.

2.1.2. Multibinding

Multibinding omogućava povezivanje jedne kontrole na više izvora. Za razliku od običnog Element binding-a, multibinding mora koristiti konverter. Konverteri kod multibindinga nasljeđuju IMultiValueConverter interface i imaju dvije metode Convert i ConvertBack.

2.2. Converters

Konverteri omogućavaju pretvaranje vrijednosti iz jednog oblika u drugi, tj. promjenu vrijednosti iz jednog tipa promjenjive u drugi tip, kao i oblikovanje vrijednosti.

Svaki konverter mora imati dvije metode koje se zovu Convert i ConvertBack. Convert metoda je trenutno zanimljiva jer je ona zadužena za pretvaranje vrijednost npr. nekog TextBoxa-a u niz znakova te vraćanje oblikovanog niza znakova. Druga metoda ConvertBack je zadužena za „TwoWay“ binding. O TwoWay bindingu će biti riječ kasnije. Obje metode primaju četiri parametra: vrijdenost ili value, željena tip varijable ili targetType, parametar ili parameter i kulturu ili culture. Value je vrijednost koju će proslijediti (u ovom slučaju) textbox. TargerType je tip promjenjiva varijabla u koju konvertira našu vrijednost u ovom slučaju je to niz znakova. Culture služi za određivanje „kulture“ u koju se konvertira vrijednost, ovaj parametar je značajan ako aplikacija podržava više jezika, jer se u jednim jezicima na primjer koristi decimalna točka, a u drugim decimalni zarez. U ovom primjeru će se koristiti samo prvi parametar.

```
public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        int number = (int)value;
        return number.ToString(SettingsBll.InvoiceNumberFormat);
    }
    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
```

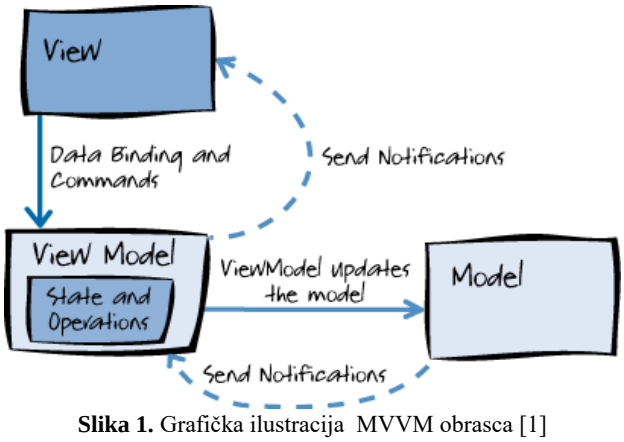
Konverter se primjenjuje uz pomoć StaticResource ključne reči i ključa resursa. Primjena konvertera se može vidjeti na slijedećem primjeru.

```
<Grid Margin="10" Name="grid1" Grid.Row="1"
        Visibility="{Binding Path=User.IsAdmin, Converter={StaticResource BoolToVis}}">
```

3. DATA BINDING

U prethodnom poglavlju je bio opisan Element Binding koji nije bio direktno vezan za Model View ViewModel pattern. Detaljnim pregledom koristeći Data Binding ulazi se u „dubinu“ MVVM obrasca koji ima centralno mjesto u arhitekturi MVVM aplikacija.

3.1. Model View ViewModel



Slika 1. Grafička ilustracija MVVM obrasca [1]

Kao što se vidi sa slike, MVVM aplikacije bi trebalo podijeliti u tri dijela:

- Model
- ViewModel i
- View

Pogled tj. view je jedan prozor ili češće UserControl. View se definira koristeći XAML kao što se definira bilo koji prozor. ViewModel sadrži model ili kolekciju modela, koji sadrže informacije za korisnika. Pokraj Modela, ViewModel može sadržati i komande (Commands) preko kojih korisnik vrši interakciju sa aplikacijom. Komande će biti objašnjene u slijedećem poglavlju. Svaki View je vezan za samo jedan ViewModel, dok recimo jedan prozor može sadržati više pogleda (Views). ViewModels i Views se definiraju u C# ili nekom drugom .NET jeziku.

Svaki Model i ViewModel trebaju se implementirati INotifyPropertyChanged interface iz System.ComponentModelnamespace-a. Implementacija ovog korisničkog sučelja se sastoji od samo jednog pristupa tipa PropertyChangedEventHandler (iz istog namespace-a) koji nosi naziv PropertyChanged. Prilikom promijene vrijednosti nekog svojstva potrebno je podići ovaj event da bi se obavijestio View da je došlo do promjene, da bi View (putem Data Binding-a) ažurirao kontrole sa novim podacima.

Svaka aplikacija koja prati MVVM obrazac bi trebala da ima tri direktorija, jedan za Views, jedan za ViewModels i jedan za Models:

- Model
- ViewModel i
- View

Svaki Model se treba nalaziti u Models direktoriju, ViewModel u ViewModels direktoriju kao i svaki View u Views direktoriju.

3.2. Data binding

Da bi View "znao" koji ViewModel da koristi za Data Binding potrebno je primjerak ContactsViewModel klase postaviti kao DataContext tog pogleda. DataContext je svojstvo koji ima svaka kontrola i svaki prozor. Sustavno svaki Data Binding se "veže" na ovo svojstvo.

4. KOMANDE I NAVIGACIJA

Komande su jedan od glavnih aspekata MVVM obrasca kada su u pitanju WPF aplikacije. Komande omogućuju interakciju korisnika sa aplikacijom. Povezivanje kontrola sa komandama se vrši najčešće Binding-om, ali često i putem resursa.

4.1. Komande

Svaka komanda (engl. Command) implementira ICommand interface iz System.Windows.Input namespace-a. Ovo sučelje objedinjuje dvije metode i jedan događaj:

- CanExecute
- Execute
- CanExecuteChanged

Metoda CanExecute provjerava da li se komanda može izvršiti, Execute je metoda koju poziva kontrola kada je potrebno izvršiti komandu, npr. Gumb prilikom klika. CanExecuteChanged je događaj koji bi se trebao aktivirati prilikom promijene "izvršivosti" komande. CanExecute metoda utiče na stanje kontrole. Ako se komanda ne može izvršiti, onda je npr. gumb za koju se veže komanda u stanju Disabled.

4.2. Navigacija

Navigacija sa stranicama je vrlo praktičan način navigacije u MVVM aplikacijama, kao što je to slučaj kod Windows 8 Metro aplikacija, Windows Phone te nekih Desktop aplikacija. Cilj je da se u glavnom prozoru prikazuju pogledi (Views) te "go back" gumb osim ako se ne prikazuje prva stranica. Navigacija se implementira koristeći komande. Da bi ViewModel-i znali kad dođe do navigacije na pogled ili se vrši navigacija sa pogleda dodaju se virtual metode u BaseViewModel koje će pozivati servis za navigaciju. Također glavni prozor treba imati panel koji će prikazivati trenutni pogled i gumb za navigaciju nazad ako je ona moguća. Ako nije moguće napraviti navigaciju na prethodnu stranicu, gumb "go back" treba sakriti. Pored ove dvije osobine BaseViewModel treba sadržavati i naslov pogleda koji se prikazuje. Naziv pogleda će se prikazati kao naslov prozora. Uzimajući u obzir navedene osobine dodaju se metode i svojstva u BaseViewModel klasu.

5. ZAKLJUČAK

Data Binding nam omogućava jednostavnu implementaciju bez pozadinskog koda, što povećava lakoću održavanja aplikacije kao i jasan put koji treba pratiti prilikom kreiranja CRUD aplikacija. Komandama je moguća interakcija sa aplikacijom i kreiranje novog kontakta. Navigacija se vrši koristeći ovaj servis, a koji se može koristi i u ostalim aplikacijama. WPF već imaju ugrađeni navigacijski servis sličan servisima koje koriste Windows Phone i Windows 8 aplikacije. Ovaj servis omogućava i navigaciju naprijed-nazad te prikaz aplikacije u Internet pretraživaču. Kompletna ideja MVVM obrasca je da razdvoji implementaciju korisničkog sučelja od poslovne logike u čemu je MVVM veoma uspješan. Prateći MVVM obrazac osigurava se konzistentan način implementacije aplikacija čime je olakšan način održavanja aplikacije i dodavanje novih mogućnosti.

5. LITERATURA

[1] https://msdn.microsoft.com/en-us/library/hh848246.aspx - (Dostupno: 28.9.2015)
[2] http://147.91.14.147/Blog.aspx?id=324 - (Dostupno: 28.9.2015)
[3] http://147.91.14.147/Blog.aspx?id=327 - (Dostupno: 28.9.2015)
[4] http://147.91.14.147/Blog.aspx?id=328 - (Dostupno: 28.9.2015)
[5] https://en.wikipedia.org/wiki/Model_View_ViewModel - (Dostupno: 28.9.2015)

Kontakt autora:

Željko Knok, mr.sc.
Međimursko veleučilište u Čakovcu
B. J. Jelačića 22a
040/396-987, zknok@mev.hr

Mark Marčec, student 3. godine Računarstva
Međimursko veleučilište u Čakovcu
B. J. Jelačića 22a
mmarcec1@student.mev.hr