Prirodno-matematički fakultet
Institut za matematiku i informatiku
Kragujevac

Diplomski rad

- WPF aplikacija za administriranje studentske oglasne table -

Profesor :
Boban Stojanović

Student:
Tijana Vesović 13/01

Kragujevac, oktobar 2009.

Sadržaj:

Uvod

Brzina kojom se razvijaju nove tehnologije u sferi informatike onemogućava pojedinca da bude upoznat sa svakim od aspekata razvoja. Mnogi koji su već imali priliku da se odluče u kom pravcu će se kretati njihov razvoj i usavršavanje, teško ostavljaju dobro savladane alate i nerado prelaze na nove još uvek nedovoljno ispitane. Iako postoji rizik da neka novoobjavljena tehnologija neće zaživeti, potrebno je ići u korak sa vremenom i rizikovati. Postoji mogućnost da proizvodi koji izađu iz vodećih svetskih kompanija jako brzo prođu kroz upotrebu i završe kao promašena investicija, dok neki opstanu i godinama nakon toga traju, što u originalnoj verziji što u usavršenoj i dorađenoj.

Jedan od klasičnih pristupa u razvoju desktop aplikacija je korišćenje .NET framework-a i klasičnih Windows formi. Ovaj način izrade desktop aplikacija datira još iz 2001. godine kada je u već postojeće osnovne biblioteke .NET platforme uključen API nazvan Windows Forms (predstavljen System.Windows.Forms.dll i System.Drawing.dll asemblijima). Windows Forms paket pruža sve šta je neophodno za razvoj desktop grafičkog korisničkog interfejsa (GUI-Grafical User Interface), i baš zato je još uvek u sklopu osnovnih biblioteka .NET 3.5 framework-a. Međutim, već sa verzijom .NET 3.0 Microsoft pruža potpuno novi paket za razvoj korisničkog interfejsa nazvan Windows Presentation Foundation (WPF), koji omogućava izradu do tada neviđenih korisničkih interfejsa.

Da bi Windows Presentation Foundation zaživeo, Microsoft je udružio svoja dva tima koja su do tada radila na Internet Explorer-u 4 i "Windows core"-u, kako bi napravili novu platformu koja neće biti samo lepog izgleda nego će i povećati produktivnost aplikacija. Ideja za kreiranje nove platfore nije bila samo u tome da se olakša i unapredi dizajnerski posao već da se programerima olakša sam razvoj aplikacije. Glavna zamisao udruživanja dva tima je bila u tome da se razvijanjem jedne aplikacije ujedno dobiju dve: desktop i web aplikacija. Da bi aplikacije prešle iz prezentovanja podataka u kreiranje podataka morala je da nastane nova platforma.

.NET Framework 3.0

.NET Framework 3.0 obuhvata skup upravljivih API-a (Application programming interface) koji čine integralan deo Windows Vista i Windows Server 2008 operativnih sistema. .NET Framework 3.0 koristi CLR .NET Framework verzije 2.0. Najvažnije novine koje je sa sobom doneo .NET Framework 3.0 su sledeća četiri nova paketa:

- **Windows Presentation Foundation (WPF)**, novi podsistem zasnovan na XAML-u namenjen izgradnji aplikacija sa vizuelno razvijenim korisničkim interfejsom.
- **Windows Communication Foundation (WCF)**, je servisno-orijentisan sistem razmene poruka koji omogućava programima da lokalno ili u mreži komuniciraju, slično web servisima.
- **Windows Workflow Foundation (WF)**, je tehnologija za definisanje, izvršavanje, i upravljanje tokovima (workflows). XAML se uobičajeno koristi za deklarisanje strukture tokova. Međutim, tok se takođe može izraziti korišćenjem koda u bilo kom .NET jeziku. Tok se sastoji od aktivnosti. WF omogućava .NET developerima razdvajanje logike od pozadinskog izvršavanja komponenti što doprinosi jasnijem i boljem upravljanju aplikacijom. Ovakav pristup predstavlja metodologiju procesnog

razvoja aplikacije (process-driven application methodology) i teži da razdvoji logički tok aplikacije od njenih komponenata koje se izvršavaju.

- **Windows CardSpace** je softverska komponenta koja bezbedno skladišti digitalni identitet osobe i obezbeđuje jedinstven interfejs za izbor identiteta za konkretnu transakciju, kao što je npr. logovanje na website.

Windows Presentation Foundation

WPF

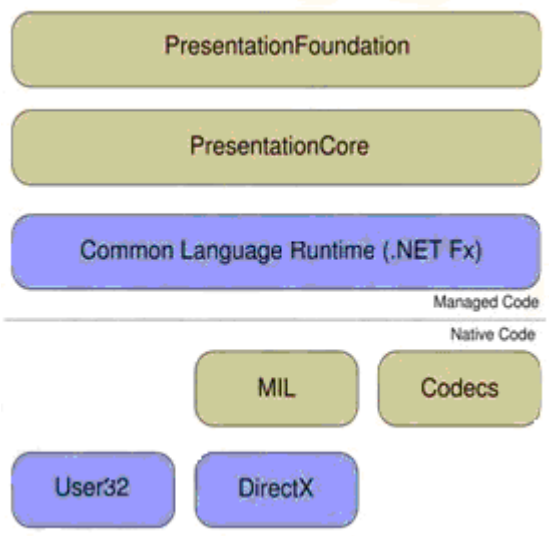
1 Pojam i arhitektura WPF-a i uvod u XAML

Windows Presentation Foundation (WPF) predstavlja grafički podsistem .NET Framework-a 3.0 i direktno je povezan sa XAML-om.

- Omogućava programski model za izgradnju aplikacija i obezbeđuje jasnu razliku između korisničkog interfejsa (UI – User interface) i bussines logike.
- Raspoloživ je na Windows Vista operativnom sistemu. Takođe se može instalirati na Windows XP SP2 i na Windows Serveru 2003.
- WPF aplikacije se mogu razvijati kao desktop i web aplikacije.
- WPF omogućava bogat dizajn, kontrolu i razvoj vizuelnih aspekata Windows programa.
- WPF teži da ujedini: korisnički interfejs, 2D i 3D crtanje, dokumente, razvijenu tipografiju, vektorsko crtanje, raster grafiku, animacije, data binding, audio i video.
- Iako će WinForms programiranje i dalje nastaviti da bude u širokoj upotrebi, WPF će imati prednost pri izboru platforme za razvoj aplikacija, naročito s pojavom .NET Framework-a 3.5, Visual Studia 2008 i Expression Blend-a.

1.1 WPF arhitektura

Arhitektura WPF-a obuhvata komponente koje pripadaju upravljivom i neupravljivom kodu (videti sliku 1-1). Dok veći deo WPF-a predstavlja upravljiv kod, komponenta koja obezbeđuje funkcionisanje WPF aplikacija pripada neupravljivom (prirodnom) kodu. To je **Media Integration Layer (MIL)**.



Slika 1-1

MIL je jedini deo WPF-a koji je neupravljiv i smešten je u **milcore.dll**. On je zadužen za usku integraciju sa DirectX-om. Sva iscrtavanja u WPF-u se vrše kroz DirectX. MIL takođe obezbeđuje osnovnu podršku za 2D i 3D površine, vremenski kontrolisanu manipulaciju sadržaja površine sa pogledom na animacione konstrukcije višeg nivoa i usklađuje individualne elemente WPF aplikacije u finalnu 3D "scenu" koja predstavla UI aplikacije. Media kodeci su takođe implementirani u neupravljivom kodu, i smešteni su u **windowscodecs.dll**. U upravljivom delu, **PresentationCore (presentationcore.dll)**

obezbeđuje "omotač" (wraper) za MIL i implementira core servise za WPF. **PresentationFramework (presentationframework.dll)** implementira prezentacione alate za krajnjeg korisnika kao što su layout, animacije i data binding.

1.2 Uvod u XAML

Extensible Application Markup Language (XAML) je deklarativan jezik zasnovan na XML-u, a koristi se za inicijalizaciju strukturiranih vrednosti i objekata. Specifična prednost koju XAML donosi WPF-u je ta što je XAML u potpunosti deklarativan jezik. U deklarativnom programskom jeziku, programer (ili dizajner) opisuje ponašanje i integraciju komponenti bez upotrebe proceduralnog programiranja. To omogućava nekome ko ima malo ili je bez programerskog iskustva da kreira celokupnu aplikaciju bez programiranja. Iako se retko dešava da se cela aplikacija može izgraditi pomoću XAML-a, uvođenje XAML-a dozvoljava dizajnerima aplikacije da efektivnije učestvuju u životnom ciklusu aplikacije. Zahvaljujući XAML-u, i dizajneri i programeri koriste isti jezik i omogućen je paralelni razvoj dizajna aplikacije (od strane dizajnera) i njenih funkcionalnosti (od strane programera). Na taj način, dizajner sam može da odradi dizajn bez dugotrajnog objašnjavanja svoje ideje programeru. Takođe je izbegnuto preklapanje u poslovima i konfliktimia koji prilikom toga mogu da nastanu. Isti dokument dizajner može da otvori u dizajnerskom alatu i da vrši izmene, a programer taj isti dokument otvara u programerskom alatu i daje objektima funkcionalnost.

Upotreba XAML-a za razvoj korisničkog interfejsa takođe omogućava razdvajanje modela i pogleda (view), što se smatra dobrim arhitektonskim principom (Model View Controler (MVC) pattern).

1.2.1 XAML – pregled

- XAML se široko koristi u .NET Framework 3.0 tehnologijama, posebno u WPF-u i Windows Workflow Foundation (WF). U WPF-u se koristi kao user interface markup language za definisanje UI elemenata, data binding-a, događaja i sl. U WF-u, workflows mogu biti definisani korišćenjem XAML-a.
- XAML elementi se direktno mapiraju u CLR objekte, dok se XAML atributi mapiraju u propertije i događaje tih objekata. XAML fajlovi se mogu kreirati i ažurirati sa visual design alatima kao što su Microsoft Expression Blend, Microsoft Visual Studio i WWF visual dizajner. Oni takođe mogu biti kreirani i ažurirani sa standardnim text editor-om, ili sa razvojnim alatima kao što je XamlPad.
- Sve što se kreira ili implementira u XAML-u može biti izraženo korišćenjem bilo kog .NET jezika, kao što su C# ili Visual Basic.NET.
- Korišćenjem XAML-a postižu se dve važne stvari: smanjenje kompleksnosti projekta i lakši razvoj aplikacija.

1.2.2 XAML – tehnologija

- XAML fajl se kompajlira u .baml (tj. Binary XAML) fajl, koji se može smestiti kao resurs u .NET Framework sklop. U vreme izvršavanja, framework engine ekstrahuje .baml fajl iz resursa sklopa, parsira ga i kreira odgovarajuće WPF vizuelno drvo (visual tree) ili workflow.
- Ako se upotrebljava u okviru WPF-a, XAML se koristi za opis vizuelnog korisničkog interfejsa. WPF omogućava definisanje 2D i 3D objekata, rotacije, animacije i brojne druge efekte.
- Iako je predstavljen kao integralan deo WPF-a, XAML nije njegov zavisan deo (pa čak ni .NET-ov).

Veoma je verovatno da će mnoge aplikacije, kao što su Microsoft PowerPoint i Word, podržati eksportovanje njihovog sadržaja u XAML.

2 WPF koncepti, alati, vrste WPF aplikacija

2.1 WPF koncepti

Osnovni koncepti koje WPF predstavlja su: logičko i vizuelno drvo, Dependency Properties (zavisne osobine), Routed Events (rutirani događaji) i Commands (komande). Neki od njih su potpuno novi (kao što je logičko i vizuelno drvo) dok ostali predstavljaju proširenje već poznatih koncepata (kao što su properties (osobine) i events (događaji)).

- **Logičko i vizuelno drvo**
  XAML na prirodan način reprezentuje UI zbog svoje hijerarhijske prirode. U WPF-u, korisnički interfejs je predstavljen pomoću drveta objekata, poznatog kao Logičko drvo (Logical tree). Logičko drvo predstavlja logičku hijerarhijsku strukturu komponeta. Vizuleno drvo mapira logičko drvo u vizuelne elemente (grafički prikaz logičkog drveta).
- **Dependency property je properti koji zavisi od drugog propertija**
  Odnosno, zavisi od različitih uslova koji determinišu njegovu vrednost u određenom trenutku. Takav uslov može biti animacija koja kontinualno menja vrednost određenog propertija, ili vrednost propertija određenog elementa koji utiče na promenu vrednosti propertija nekog drugog elementa.
- **Rutirani događaji**
  Rutirani događaji su događaji dizajnirani da rade sa drvetom elemenata. Kada se podigne rutirani događaj, on može da "putuje" kroz drvo elemenata. Za razliku od običnih događaja, rutirani događaji mogu imati više event handler-a u zavisnosti od kontrole.
- **Komande**
  Dok su događaji vezani za detalje o specifičnoj akciji korisnika (npr. dugme je pritisnuto ili je selektovan ListBoxItem), komande predstavljaju akcije koje su nezavisne od promena na interfejsu prouzrokovane od strane korisnika. Aplikacije obično izlažu akcije kroz različite mehanizme simultano kao što su na primer MenuItem-i u meniju, MenuItem-i na ContextMeniju, zatim dugmad na ToolBar-u, ili prečice na tastaturi i slično.

2.2 WPF alati

Raspoloživi su mnogobrojni dizajnerski alati za razvoj WPF aplikacija. Navešćemo samo neke od njih:

- **Microsoft Cider** je XAML dizajner u obliku add-in-a za VS 2005 IDE namenjen razvoju WPF aplikacija. Cider je integrisan u Visual Studio 2008.
- **Microsoft Expression Blend** je namenjen dizajnerima korisničkog interfejsa, i odnosi se na vizuelni aspekt aplikacije. To je dizajnersko orijentisan alat za stvaranje WPF aplikacija koje sadrže 2D i 3D grafiku, tekst i forme. Generiše XAML koji se može eksportovati u druge alate.
- **Microsoft Expression Design** je alat za bitmape i 2D-vektorsku grafiku koji omogućava eksport u XAML.

2.3 Vrste WPF aplikacija

WPF nije namenjen razvoju samo tradicionalnih standalone aplikacija, već omogućava razvoj i **XAML Browser aplikacija (XBAP)**.

- **Standalone aplikacije** su one aplikacije koje su lokalno instalirane na računaru korišćenjem softvera kao što je ClickOnce ili Windows Installer i koje se izvršavaju na desktop računaru. Za Standalone aplikacije se kaže da poseduju "full trust" što znači da imaju potpuni pristup resursima računara. One su .exe tipa.
- **XAML Browser aplikacije (XBAPs)** su programi koji se izvršavaju u okviru web browser-a. Ove aplikacije se izvršavaju u "sandbox" okruženju, i nije im omogućen potpuni pristup računarskim resursima, što znači da sve što se izvršava u okviru browsera se smatra bezbednim sa stanovišta korisnika. Aplikacija ima pristup browser komandama. Sa .NET Framework verzijom 3.0, XBAP se jedino izvršava u Internet Exploreru, dok sa .NET Framework 3.5 verzijom je obezbeđena i podrška za Mozillu Firefox.

2.4 Microsoft Silverlight

- Microsoft Silverlight je browser plugin koji omogućava web aplikacijama da budu razvijene sa mogućnostima koje karakterišu bogate internet aplikacije kao što su: animacije, vektorska grafika i audio-video playback. Silverlight se takmiči sa proizvodima kao što su Adobe Flash, Adobe Flex, Adobe Shockwave, Java FX i Apple QuickTime. Verzija 2.0 donosi poboljšanu interaktivnost i omogućava programerima korišćenje .NET jezika i razvojnih alata.
- Kompatibilan je sa web browser-ima koji se koriste na MAC i Windows operativnim sistemima.
- Rad sa Silverlight-om je moguć u Expression Studio alatima i Visual Studiu 2008 sa dodatkom "Silverlight tools for Visual Studio".
- Silverlight ima budućnost u naprednim internet aplikacijama koje zahtevaju više od lepog izgleda interfejsa.
- Ima prednost nad Flashom u tehnološki naprednijim opcijama i u razvojnim zajednicama naviknutim na .NET jezike.

3 Instalacija

U predstojećim poglavljima će biti objašnjen rad sa WPF-om na jednostavnim primerima, dok će se za složenije akcije nad dizajnom koristiti Expression Blend. Projekti urađeni u Expression Blendu mogu jednostavno biti uveženi u WPF aplikaciju i kao takvi korišćeni u okviru Visual Studia.
Prvi problem koji se može javiti jeste redosled instalacija svih potrebnih paketa. Da bi se to izbeglo potrebno je pratiti sledeće uputstvo:

- Kao prvo, potrebno je imati WPF kao sastavni deo .NET Framework-a (najmanje verzija 3.0. WPF nije sastavni deo ranijih verzija .NET Framework-a). Ukoliko je na računaru instaliran Windows Vista operativni sistem, onda je .NET 3.0 već instaliran na računaru, u supotnom, se može preuzeti od Microsofta. Preporučljivo je da se instalira .NET Framework 3.5.

- Nakon toga treba instalirati SP1 za Microsoft .NET Framework 3.5 .

- Sledeća stvar koja će biti potrebna je Visual Studio 2008.

- Zatim treba instalirati i SP1 za Visual Studio 2008.

- Vrlo je bitno da se pre instalacije Expression Blenda instalira Microsoft Silverlight Tools for Visual Studio 2008 SP1.

- Takođe, koristiće se i dodatni alati kao što su XamlPad koji je sastavni deo .NET Framework 3.5 SDK, kao i Microsoft Expression Blend.

- Sledeća stvar koju treba instalirati jeste Microsoft Expression Blend 2.

- Konačno, niz instalacija je završen sa Microsoft Expression Blend SP1.

4 Kreiranje WPF aplikacije u Visual Studiu 2008

Da bi se kasnije prezentovali planirani primere, potrebno je pre svega opisati tok akcija za kreiranje najobičnije WPF aplikacije u Visal Studiu 2008. Prvo će se keirati novi projekat u okviru Visual Studija odabirom opcije File > NewProject. Prilikom kreiranja projekta neophodno je proveriti da li je u levoj koloni "ProjectTypes" selektovan Visual C# i da li je selektovan .NET Framework 3.0 ili 3.5.

- Prvi templejt koji će biti komentarisan je „**WPF Application**". Ovaj projekat kreira standardnu windows aplikaciju tipa .exe. Ovakav tip aplikacije se izvršava sa „full trust", što znači da svemu što korisnik može da pristupi, može i aplikacija - file sistemu ili internetu.
- Sledeća opcija je „**WPF browser application**". Ovo je vrsta aplikacije koja je dizajnirana da se izvršava u web browseru. Uobičajeno se nazivaju XBAP aplikacije prema tipu file-a koji se kreira (tip je .xbap što je skraćenica od XAML Browser application). Zamislite običnu .NET exe aplikaciju koja se izvršava u browseru. Aplikacija ima pristup browser komandama kao što su forward i backward button ili adressbar. Za razliku od windows aplikcije, ove aplikacije se izvršavaju sa "partial trust" u okviru sandbox-a, što znači da sve što se izvršava u okviru browsera se smatra bezbednim sa stanovišta korisnika.

Za početak je dovoljno kreirati običnu windows aplikaciju. U dijalogu za kreiranje novog projekta se može zadati proizvoljno ime i lokacija projekta. Novokreirani projekat se može pokrenuti izborom opcije iz menija Debug > StartDebugging ili samo pritiskom na dugme F5. Iako se nakon kreiranja projekta može videti samo prazan prozor to je prava funkcionalna WPF aplikacija. SolutionExplorer može da sadrži više projekata koji zajedno predstavljaju aplikaciju kao kompaktnu celinu.

Po završetku kreiranja u okviru projekta se nalaze sledeće celine.

Reference

Prilikom otvaranja References može se videti dosta referenci. Reference služe za korišćenje externog koda ili eksternih biblioteka, odnosno dll-ova u okviru projekta. Sve reference koje se mogu videti, odmah nakon kreiranja WPF projekta, su standardne za WPF aplikacije, odnosno dosta njih su standardni .NET sklopovi, a neki od njih su specifični za WPF.

- Prvi od njih je **PresentationCore**. PresentationCore sadrži jezgro WPF servisa i biblioteke kao što su media, oblici i slično.
- Sledeći je **PresentationFramework**. PresentationFramework sadrži koncepte višeg nivoa kao što su kontrole, layout i databinding koje koristimo kada keiramo aplikaciju, ondnosno korisnički interfejs.
- I poslednji je **WindowsBase** koji sadrži osnovne klase i servise koje nudi WPF. Ove reference su dodate pri kreiranju projekta na osnovu templejta koji je izabran.

.xaml fajlovi

Sledeća dva fajla koja se mogu videti su .xaml fajlovi. Može se primetiti da imaju extenziju .xaml. Kao što je već spomenuto, xaml je skraćenica od Extensible Application Markup Language i u WPF-u se koristi za keiranje korisničkog interfejsa. Može se primetiti da je fajl koji je otvoren neposredno po kreiranju projekta - Window1.xaml.
Svaki tag koji se kreira u xaml-u odgovara objektu, ukoliko je postavljen tag Window elementa, to znači da je kreiran objekat tipa Window.
Sledeća važna stvar je da svaki atribut, postavljen kao jedan element, odgovara propertiju tog objekta, tako da u okviru Window elementa postoje i neki propertiji od kojih su interesantni *Title*, *Height* i *Width*. Kada je kreiran window element, postavljene su vrednosti propertija *Title* na ime koje je navedeno za solution, *Height* na 300 piksela i *Width* takođe na 300 piksela.
Xmlns properti, deklarisan na Window klasi, jeste standardan XML atribut koji omogućava da se navede namespace za tagove u dokumentu. Nakon kreiranja projekta deklarisana su dva namespace-a:

- prvi je podrazumevani namespace (ceo xmlns). Podrazumevani namespace je namespace koji sadrži sve standardne WPF tagove, kao što su na primer dugme, window ili grid, elipsa i sl.
- Kada postoji potreba da se koristi kod koji ne postoji u default namespacu, tada se deklariše dodatni namespace kao što je to *xmlns:x* namespace koji je ovde deklarisan.

Već deklarisanim propertijima mogu se dodati još neki propertiji na Window element. Ukoliko se započne sa pisanjem u tagu ili pritiskom SPACE tastera na tastaturi, prikazaće se lista mogućih atributa. Na primer, može se postaviti atribut Background kome se iz liste ponuđenih vrednosti boja može dodeliti konkretna vrednost. Takođe se može postaviti i atribut *WindowStartupLocation* koji će odrediti poziciju prozora nakon njegovog pokretanja. Opcija “CenterScreen” će prilikom pokretaja aplikacije postaviti prozor na sredinu ekrana.
Nakon postavljanja propertija na konkretne vrednosti rezultat se može videti pokretanjem aplikacije.

5 Layout

Pre nego što se objasni pojam Layout, treba da se shvati kako je sastavljen korisnički interfejs u WPF-u. U prethodnom poglavlju je spomenuto da se jedan element može dodati kao dete nekog drugog elementa. XAML na prirodan način reprezentuje korisnički interfejs zbog svoje hijerarhijske prirode. U WPF-u, korisnički interfejs je predstavljen pomoću drveta objekata, poznatog kao Logičko drvo. Ovakav pristup izgradnji UI-a se naziva kompozicija.
U okviru kompozicije, koriste se tri osnovna tipa elementa:

- Najjednostavniji od njih su elementi bez dece i nazivaju se **listovi**. Oni čine osnovne gradivne blokove u WPF-u.

- Sledeći tip elementa u kompoziciji, koji je složeniji od lista, jeste **element sa jednim detetom**. Dete kod ovog tipa elementa obično predstavlja sadržaj tog elementa. U elemente s jednim detetom spadaju Content kontrole (primeri su Button, checkBox, RadioButton, scrollViewer).

- Najsloženiji tip elemenata u kompoziciji je **element sa više dece**. Osnovni oblik ovog tipa su paneli i o njima će se dosta pričati u narednom poglavlju. Postoje i neke kontrole koje imaju više dece (kao što je npr. ListBox). Međutim, u WPF-u, skoro uvek, svaki element koji ima više dece je panel. Paneli se koriste u slučaju kada postoji više elementa i treba razmotriti gde će biti smešteni.

Panel služi za organizaciju rasporeda dece elemenata i taj proces se naziva **Layout**. Layout je proces koji definiše dve stvari: kao prvo veličinu, i kao drugo lokaciju svakog objekta koje predstavlja dete Panela.

6 Vrste Panela

Šta god da je zamišljeno da se uradi u WPF-u, ne može se izbeći rad sa panelima. Paneli su komponente koje kontrolišu iscrtavanje elemenata, njihove veličine i dimenzije, njihov položaj i informacije o tome koji od elemenata su "deca elementi" posmatranog panela. WPF pruža niz predefinisanih panela kao i mogućnost da se kreira standardni panel.

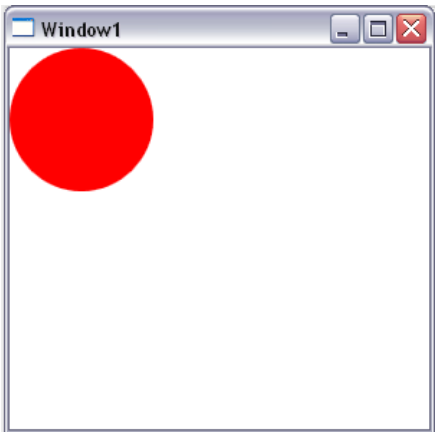
6.1 Canvas

Na samom početku, nakon kreiranja aplikacije, postoji samo prazna strana u okviru koje se može dodati jedan jednostavan panel koji se zove **Canvas**. Canvas je panel koji omogućava standardno x0y koordinatno pozicioniranje elemenata.

```
<Canvas>
</Canvas>
```

Takođe se u okviru Canvas-a može dodati jedno dete element npr. Elipsa:

```
<Ellipse Height="100" Width="100" Fill="Red"/>
```



Dakle, nakon dodavanja elementa elipse u Canvas, sa slike 1-2 se može primetiti da nije postavljen nikakav Layout. Da bi Canvas pozicionirao elipsu, moraju se zadati koodinate, a to se može uraditi primenom koncepta Attached propertija. Attached propertiji se postavljaju direktno na elipsu.

Slika 1-2

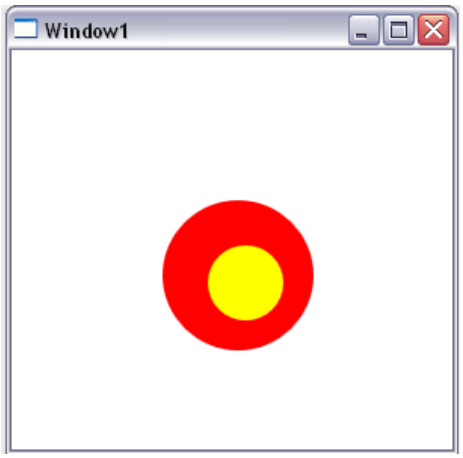
```
<Ellipse Canvas.Top="100" Canvas.Left="100" Height="100" Width="100" Fill="Red" />
```

Znači, postavljajući Canvas.Top="100" i Canvas.Left="200" uticano je na elipsu da se pozicionira u okviru Canvas-a. Ovi propertiji se nazivaju attached propertiji zato što počinju sa „Canvas.„ notacijom. To govori da je properti definisan canvas elementom. Attached propertiji daju mogućnost Canvas-u da definiše propertije koji se koriste na drugim elementima. Elipsa ne mora da zna za Top i Left propertije, oni su definisani od strane Canvas-a, a koriste se za definisanje pozicije elipse. Attached propertiji su uobičajeni kod panela zbog potrebe da panel određuje poziciju elemenata u okviru njega samog. Može se primetiti da nakon dodavanja pozicionih propertija, elipsa je promenila položaj 100 piksela od vrha i 200 piksela od leve ivice.

Nakon dodavanja prve elipse može se dodati još jedno dete u Canvas takođe u vidu elipse, kako bi se prikazao međusoban odnos elipsi u okviru canvas-a.

```
<Ellipse Canvas.Top="130" Canvas.Left="130" Height="50" Width="50" Fill="Yellow"/>
```

Nakon dodavanja druge elipse dobijamo prikaz kao na slici 1-3.



Slika 1-3

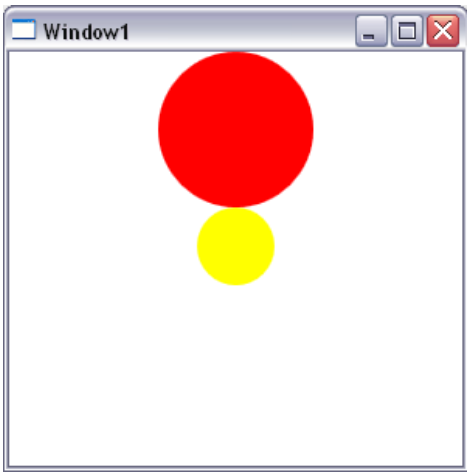
Može se primetiti da su elipse postavljene po redu kako su definisane. Ukoliko se iskopira mala elipsa ispred veće elipse, manja elipsa će nestati. To je podrazumevano ponašanje – elementi seređaju po redu po kom su definisani. Da bi se ovaj problem prevazišao, upotrebljava se još jedan attached properti koji se odnosi na poziciju više elemenata u okviru Panela. To je *Panel.Zindex*. Time će se obezbediti da redosled dodavanja elemenata u panel ne može da utiče na njihovo međusobno prekrivanje (ako je to eksplicitno zadato Zindex-om).

```
<Ellipse Panel.ZIndex="1" Canvas.Top="130" Canvas.Left="130" Height="50"
Width="50" Fill="Yellow" />
<Ellipse Canvas.Top="100" Canvas.Left="100" Height="100" Width="100" Fill="Red" />
```

ZIndex utiče na redosled "pojavljivanja" elemenata u panelu. Veći broj znači da je element ispred elementa sa manjim brojem. Podrazumevana vrednost ZIndex-a je nula.

6.2 StackPanel

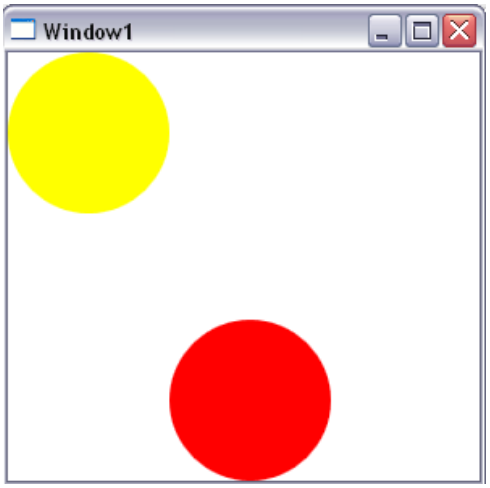
Sledeći panel koji će biti objašnjen je **StackPanel**. Stack Panel može biti horizontalni i vertikalni. Pogledajmo kako StackPanel funkcioniše.
Ukoliko se Canvas iz prethodnog primera zameni StackPanel-om dobiće se prikaz kao na slici 1-4. Dve postojeće elipse su se poređale od vrha jedna posle druge.



Slika 1-4

Ostali propertiji vezani za Canvas više nemaju smisla, pa ih zato treba obrisati.
StackPanel funkcioniše tako što kreira stack elemenata od vrha ka dnu od svih elemenata koji se u njemu nalaze. Takođe, može se primetiti da su svi poređani po sredini. To je zbog jednog propertija koji nije attached već je regularan properti i zove se *HorizontalAlignment*.

HorizontalAlignment je properti koji se može postaviti na bilo koji element. Može imati vrednosti: *Left*, *Right*, *Center* i *Stretch*. Inače, podrazumevana vrednost je stretch (a to se može videti ukoliko se ukloni širina na elementu).



VerticalAlignment je properti koji može imati vrednosti: *Top*, *Center*, *Bottom* i *Stretch*. Podrazumevana vrednost je i u ovom slučaju stretch. Ovaj properti ima smisla ako je StackPanel horizontalno orijentisan.
Kako bi se videle promene nad elementima u okviru panela, postavićemo VerticalAlignment za žutu elipsu na Top, a za crvenu na Bottom (videti sliku 1-5).

Slika 1-5

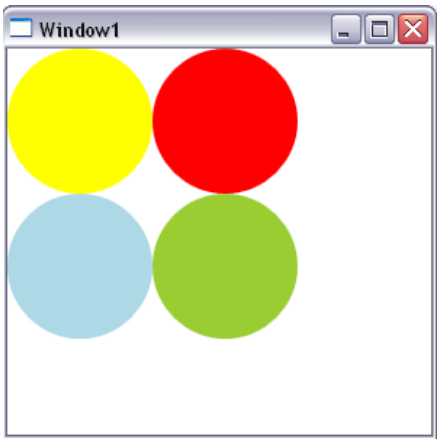
```
<StackPanel Orientation="Horizontal">
      <Ellipse Width="100" Height="100" Fill="Yellow" VerticalAlignment="Top"/>
      <Ellipse Width="100" Height="100" Fill="Red" VerticalAlignment="Bottom"/>
</StackPanel>
```

6.3 WrapPanel

Sledeći panel koji će biti objašnjen je **WrapPanel**. Ovaj panel je naročito koristan ako postoji dosta malih elemenata koje je potrebno rasporediti u ograničenom prostoru, i u osnovi se ponaša kao više stack panela koji su raspoređeni jedan nakon drugog. Panel prvo raspoređuje elemente u jedan red, i kad nestane mesta u prvom redu, on nastavlja da smešta elemente u drugi red i tako dalje, sve dok ne pronađe mesta za sve elemente. Da bi se videlo kako Wrap panel funkcioniše, iskopiraćemo dosta elipsi, a zatim promeniti StackPanel u WrapPanel. Takođe treba obrisati Alignment propertije koji su postavljeni u prethodnom primeru.

```
<WrapPanel>
      <Ellipse Height="100" Width="100" Fill="Yellow" />
      <Ellipse Height="100" Width="100" Fill="Red" />
      <Ellipse Height="100" Width="100" Fill="LightBlue"/>
      <Ellipse Height="100" Width="100" Fill="YellowGreen"/>
</WrapPanel>
```

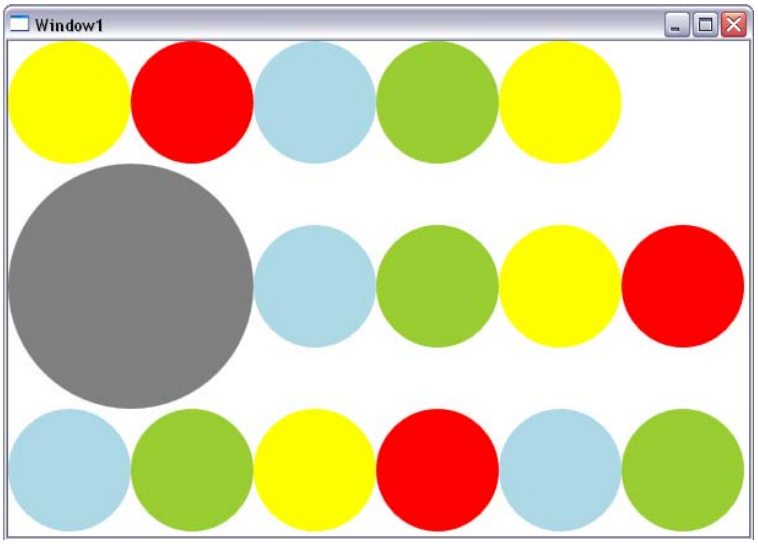
Na slici 1-6 se može videti način na koji su se elipse rasporedile.



Wrap Panel je rasporedio elemente u vrh i sa leve strane tj. smestio ih je u horizontalni stack. Ukoliko već postojeće elipse iskopiramo pet - šest puta i pritisnimo F5, Wrap panel će elipse rasporediti u redove tako da se svaki red popuni dok postoji prostora u njemu, a zatim će se nastaviti raspoređivanje u naredni red. Svaka od ovih elipsi ima istu širinu i visinu.

Slika 1-6

Ukoliko se jednoj od elipsi promeni visina i širina, visina i širina celog reda, u kome se ona nalazi, će se prilagoditi dimenziji te elipse (slika 1-7).



Slika 1-7

Još jedan od propertija na koji treba obratiti pažnju jeste „*Margin*“. Properti „*Margin*” se odnosi na skoro sve vrste Panela. To je placeholder, što znači da dodaje prostor elementu sa njegove spoljne strane. Postavimo margine na neku od elipsi.

```
Margin="30,0,0,0"
```

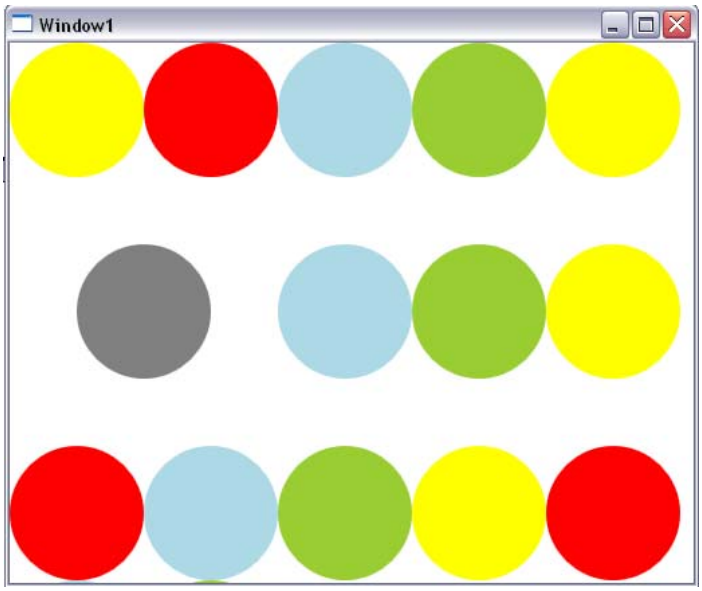
Može se primetiti da je postavljena vrednost leve margine. Svaka od četiri vrednosti između zareza se odnosi redom na vrednosti margina u smeru kazaljke na satu.
Ukoliko je potrebno zadati vrednosti samo leve i desne margine, to se može zapisati na sledeći način:

```
Margin="50,0"
```

Zadavanje margina sa svih strana se može uraditi na sledeći način:

```
Margin="50"
```

Efekat se može videti na slici 1-8.



Slika 1-8

6.4 Grid

Grid Panel omogućava definisanje ćelija, kao u tabelama, i dodavanje sadržaja u te ćelije. Ukoliko Wrap Panel iz prethodnog primera zamenimo Gridom, dobiće se izgled kao da postoji samo jedna elipsa. Ustvari, sve elipse su tu, samo što su sve smeštene direktno u centar jedne ćelije koja postoji u Gridu po njegovom kreiranju (jedan red i jedna kolona). Grid koristi Alignment propertije da odredi položaj elemenata u okviru ćelije. Ukoliko se na neku od elipsi postavi:

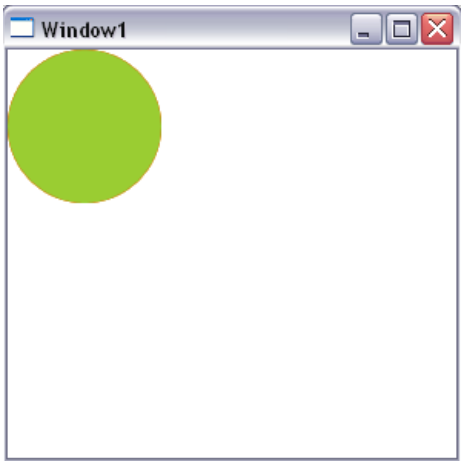
```
HorizontalAlignment="Left" VerticalAlignment="Top"
```

dobiće se izgled kao na slici 1-9.

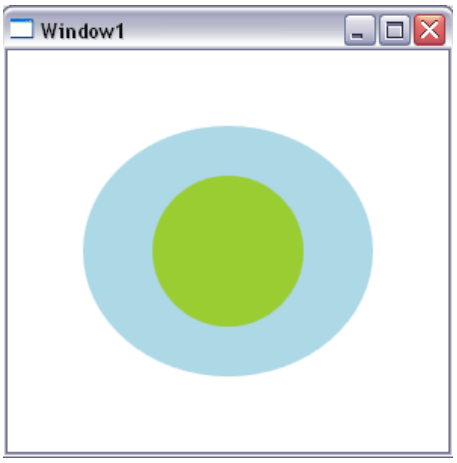
Već je napomenuto da je podrazumevana vrednost za HorizontalAlignment properti – stretch, i da bi se to primetilo ukoliko bi se uklonile širina i visina elementu. Uklonimo širinu i visinu nekoj od elipsi, kao i Alignment propertije. Postavimo sada marginu na ovu elipsu.

```
Margin="50"
```

Sada se može videti na slici 1-10 da elipsa ispunjava čitavu ćeliju uz slobodan prostor oko nje koji je određen marginama.



Slika 1-9



Slika 1-10

6.4.1 Dodavanje redova i kolona u Grid

Da bi se definisali redove i kolone u Gridu, potrebno je upoznati se sa još jednom vrstom popertija a to je *ComplexProperty*. Complex property je properti čija je vrednost složenija od običnog stringa, broja ili enumeracije. Koristi se za postavljanje složenih vrednosti kao što su na primer kolekcija objekata, a u slučaju grida su to redovi i kolone. Pogledajmo kako izgleda dodavanje complex propertija u Grid.

```
<Grid.ColumnDefinitions>
</Grid.ColumnDefinitions>
```

Ovo možda izgleda kao dodavanje deteta elementa u Grid, ali sintaksa koja počinje sa "Grid." govori da se postavlja properti na Grid. Properti koji se postavlja se zove *ColumnDefinitions*. ColumnDefinitions je kolekcija objekata tipa *ColumnDefinition*. Svaka ColumnDefinition koja se dodaje u ovu kolekciju će kreirati novu kolonu u Gridu. Kolona se dodaje jednostavno dodavanjem taga ColumnDefinition. Dodajmo nekoliko kolona i pogledajmo kako one utiču na Grid:

```
<Grid.ColumnDefinitions>
      <ColumnDefinition/>
      <ColumnDefinition/>
      <ColumnDefinition/>
</Grid.ColumnDefinitions>
```

Može se primetiti da se ceo sadržaj premestio u prvu kolonu. Da bi se videle granice kolona, postavlja se properti Grida pod nazivom „*ShowGridLines*" na vrednost true. Sad se mogu videti granice.

```
<Grid ShowGridLines="true">
```

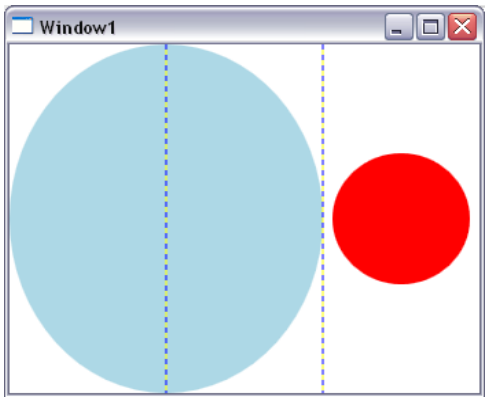
6.4.2 Attached properties Grid.Column i Grid.RowSpan

Premeštanje objekata u okviru kolona se vrši korišćenjem attached propertija. Već je napomenuto da se attached properti koristi za pozicioniranje elementa u okviru panela i da je to properti panela a ne elementa u okviru panela. Attached properti "Grid.Column" se koristi za smeštanje element u određenu kolonu. Postavićemo ovaj properti na jednu od elipsi.

```
<Ellipse Height="100" Width="100" Grid.Column="2" Fill="Red" />
```

Najmanja vrednost ovog propertija je 0 što se odnosi na prvu kolonu i to je ujedno i vrednost koja se podrazumeva, zato su se i sve elipse pojavile u prvoj koloni. Takođe je moguće specificirati da se element prostire na više kolona, a properti koji to omogućava je attached properti „*ColumnSpan*". Podrazumevana vrednost ovog propertija je jedan, što znači da se element prostire u jednoj koloni. Zato se, nakon dodavanja elipsi u okviru grida, može primetiti da je svaka od njih smeštena u jednu kolonu. Ako se promeni vrednost ovog propertija na dva, element će se prostirati na dve kolone. Dodajmo nekoj od elipsi ColumnSpan = 2:

```
<Ellipse Fill="LightBlue" Grid.ColumnSpan="2" />
```



Nakon navedenih izmena i nakon pokretanja aplikacije dobija se izgled kao na slici 1-11.

Slika 1-11

6.4.3 Veličina kolona

Veličina kolone, odnosno, širina, se može definisati na 3 načina: proporcijom, u pikselima i automatski. Podrazumevana vrednost širine je „*" (zvezda) i podrazumeva da širina kolone zauzima „jednu porciju raspoloživog prostora". Ukoliko postoje tri kolone svaka će zauzimati jednu porciju, tj. svaka će zauzimati trećinu raspoloživog prostora. Definišimo npr. za prvu kolonu da zauzima dve porcije raspoloživog prostora:

```
<ColumnDefinition Width="2*"/>
```

Dakle, raspoloživi prostor se deli na četiri dela, pri čemu prva kolona zauzima dve porcije tog prostora, a preostale dve kolone zauzimaju po jednu porciju.

Drugi način da se odredi širina kolone je zadavanje standardne fiksne vrednosti – u pikselima. Ako se promeni širina druge kolone da bude 100 piksela, automatski će se širina postaviti na zadati broj piksela.

```
<ColumnDefinition Width="100"/>
```

Poslednji način da se definiše širina kolone je jednostavno postavljanje vrednosti "*Auto*". Ako postavimo ovako širinu kolone, to znači da će se širina prilagoditi širini sadržaja u njoj. Prvo mora da nešto postoji u toj koloni, inače će se širina kolone smanjiti na nulu. Efekat toga se može videti ako se postavi vrednost širine Auto na treću kolonu, ovim će se širina kolone prilagoditi širini elementa koji se nalazi u njoj:

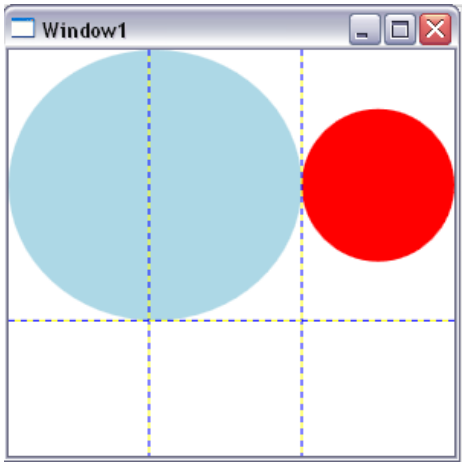
```
<ColumnDefinition Width="Auto"/>
```

Crvena elipsa se trenutno nalazi u trećoj koloni, može se primetiti kako se širina kolone prilagodila širini elipse.

```
<Ellipse Height="100" Width="100" Grid.Column="2" Fill="Red" />
```

I na kraju, poželjno je napomenuti da se redovi ponašaju na isti način kao i kolone samo u suprotnom smeru. Dodajmo dva reda i odredimo im visine tako da prvi zauzima dve porcije prostora:

```
<Grid.RowDefinitions>
      <RowDefinition Height="2*"/>
      <RowDefinition/>
</Grid.RowDefinitions>
```



Može se primetiti da su svi elementi smešteni u prvi red i to je zato što je podrazumevana vrednost za Grid.Row nula.

Konačan izgled je prikazan na slici 1-12.

Slika 1-12

Na kraju je važno napomenuti da je moguće smeštati panele u panele, tj. ugnježdavati ih jedne u druge. Npr. moguće je smestiti StakPanel u Grid ili obrnuto. Ugnježdavanje panela predstavlja snažan Layout mehanizam.

7 WPF kontrole

WPF kontrole su gradivni blokovi aplikacija i imaju veću fleksibilnost u odnosu na tipične Windows kontrole. Čak i obične Windows kontrole koje se obično mogu videti u standardnim windows aplikacijama su poboljšane u WPF aplikacijama. WPF controle se nalaze u System.Windows.Controls paketu, a ne kao i do sada, u System.Windows.Forms paketu.

Kada se prevuče neka od WPF kontrola iz Toolbox-a na neki od panela, ta kontrola liči na standardnu kontrolu iz Windows aplikacija. Međutim, WPF kontrole mogu biti prilagođene, tako što podržavaju stilove i šablone za kreiranje vizuelno privlačnih aplikacija. Izgled kontrole se može promeniti ili direktno pisanjem XAML koda ili korišćenjem nekih od programa koji će sam generisati XAML (npr. Microsoft Expression Blend).

Postoje sledeće vrste kontrola:

- Framework elements
- Controls
- Content controls
- Items controls
- Range bases

Kada kreiramo novu kontrolu, onda pripada nekom od prethodno navedenih tipova.

7.1 Framework elements

7.1.1 Border

Primer prvog elementa je **Border**:

```
<Border
       Width="200"
       Height="200"
       Background="Orange"
       BorderBrush="OrangeRed"
       BorderThickness="5"
       CornerRadius="10">
</Border>
```

gde se mogu primetiti poznati propertije kao što su: *Height*, *Width* i *Background*. Background i Fill funkcionišu na isti način, samo što se Fill properti koristi za oblike (Shapes) a Border element ima properti Background koji se odnosi na kontrole. Ova razlika postoji samo iz istorijskih razloga.

Sledeće propertije koji se mogu videti su: *BorderBrush* i *BorderThickness*.

Promenimo BorderThickness na:

```
BorderThickness="0,0,10,10"
```

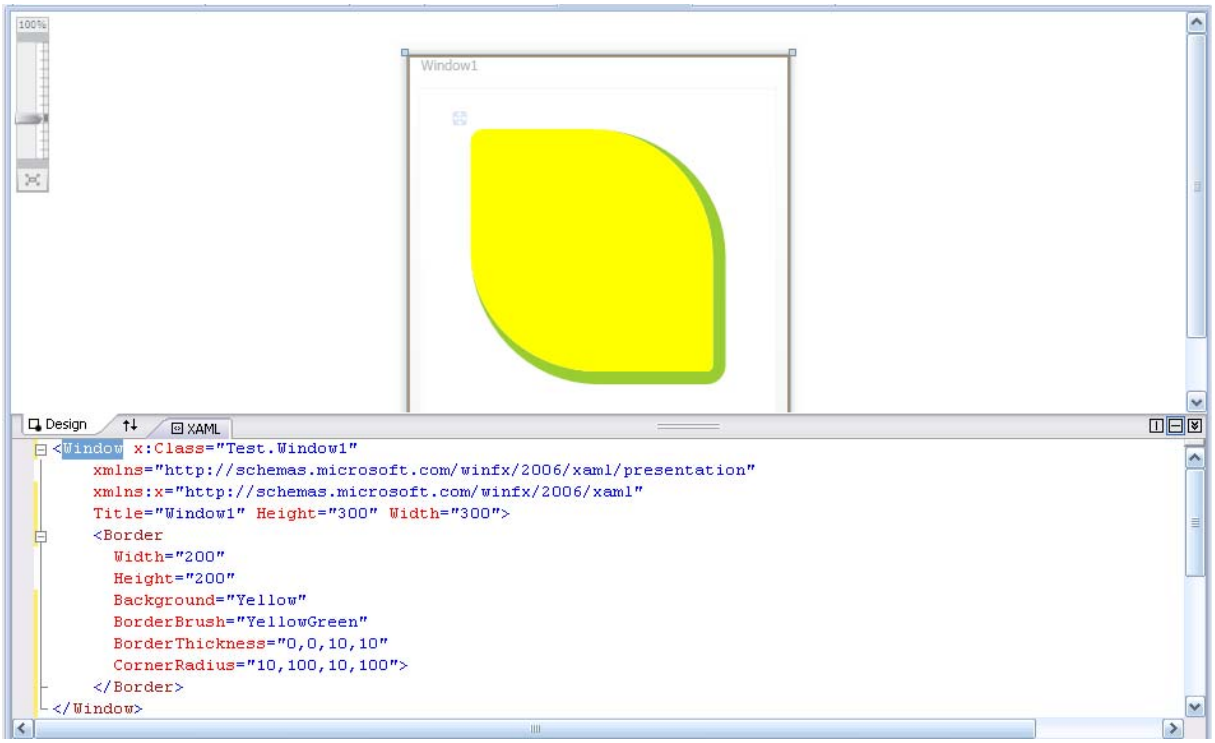
Sledeći properti je *CornerRadius* koji omogućava da se specificira ugao border elementa. On takođe može imati vrednost sa 4 dela. Promenimo vrednost i ovog propertija na:

```
CornerRadius="10,100,10,100"
```

Promenimo i boje da uskladimo sa novim oblikom Border elementa:

```
Background="Yellow" BorderBrush="YellowGreen"
```

Pritiskom na F5 osvežićemo prikaz:



Slika 1-13

7.1.2 Image

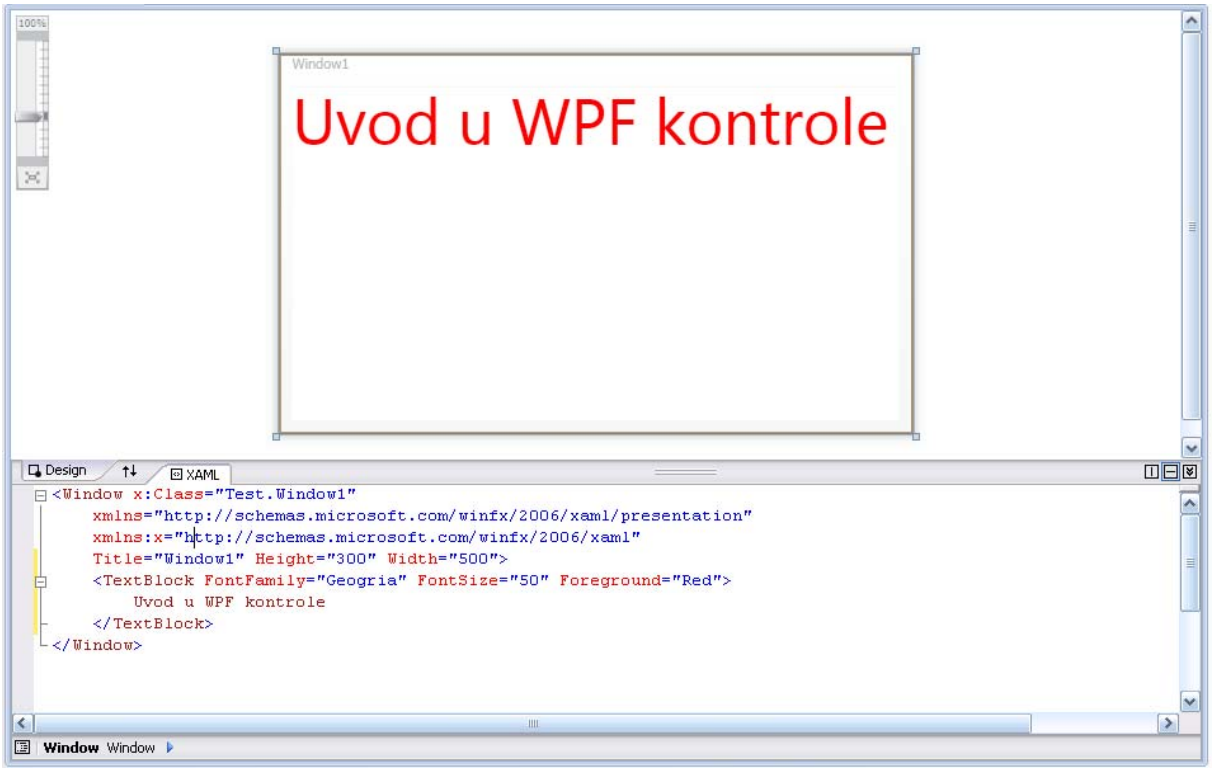
Sledeća stvar koja se može uraditi je ubacivanje slike. U ovom slučaju postoji jedan jednostavan element koji se zove **Image** i dva propertija:

```
<Image Source= "slika.jpg" Stretch="Uniform"/>
```

Prvi od njih je *Source* properti koji kao vrednost sadrži "*Slika.jpg*" što predstavlja relativnu putanju, tj. slika je smeštena u isti direktorijum kao i xaml fajl. *Stretch* properti može imati vrednosti: *Uniform*, *Fill*, *UniformToFill* i *None*. Vrednost koja se podrazumeva je Uniform.

7.1.3 TextBlock

Sledeći element koji će biti predstavljen je **TextBlock**. To je jednostavan element koji omogućava prikaz teksta na ekranu. Pogledajmo neke od karakterističnih propertija koji se odnose na TextBlock. Prvi od njih je *FontFamily*, zatim tu je i *FontSize*. Sledeći properti koji je uobičajen za Text je *Foreground* i njime se menja boja teksta.



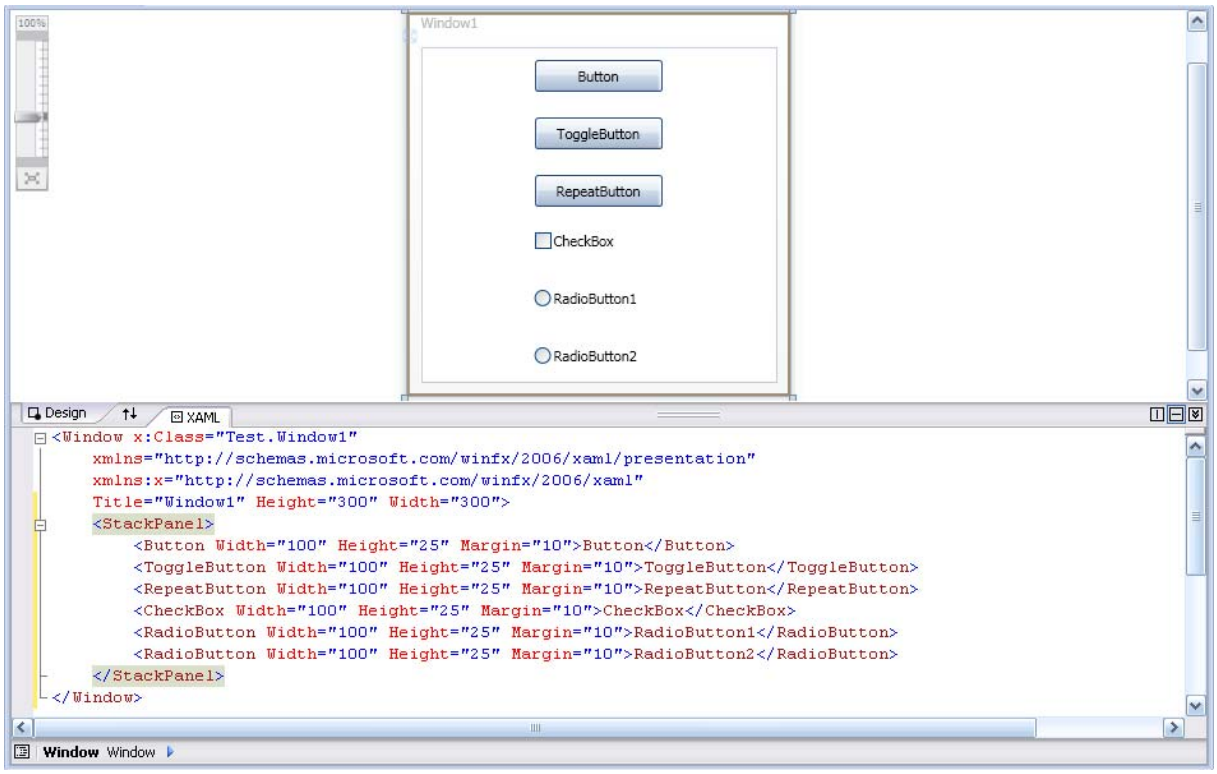
Slika 1-14

7.2 Controls

U hijerarhiji klasa FrameworkElements predstavljaju baznu klasu kontrola, tj. kontrola je izvedena klasa iz klase FrameworkElement.
Ono što kontrolu čini kontrolom je *Template* properti. Osnovna ideja Template-a je da se može tačno specificirati kako ta kontrola izgleda.

7.3 Content controls

Content Controls imaju *Content* property. Karakteristično za Content Controls je to što mogu imati samo jendo dete kao svoj sadržaj.



Slika 1-15

Prvo dugme je obično dugme. Drugo dugme je poseban tip dugmeta koje se naziva **ToogleButton**. Kada se jednom klikne, ToggleButton ostane u kliknutom (pritisnutom, donjem) položaju. Kada se ponovo klikne, ono se vrati u prvobitni položaj. Treće dugme je specijalno dugme koje se naziva **RepeatButton**. Ono što je specifično u vezi sa njim je da ispaljuje click event-e sve dok ga korisnik drži pritisnuto.

Sledeća dva primera su u suštini izvedena iz dugmeta i to su **CheckBox** i **RadioButton**.

Ono što je zajedničko za ove kontrole je da imaju *Content* properti. Nekad se o ovom Content-u može govoriti kao o Rich content-u, to znači da bukvalno sve što se zamisli može

biti postavljeno kao sadržaj content kontrole, npr. to može biti neko drugo dugme, ili neka druga kontrola, pa čak i video zapis.

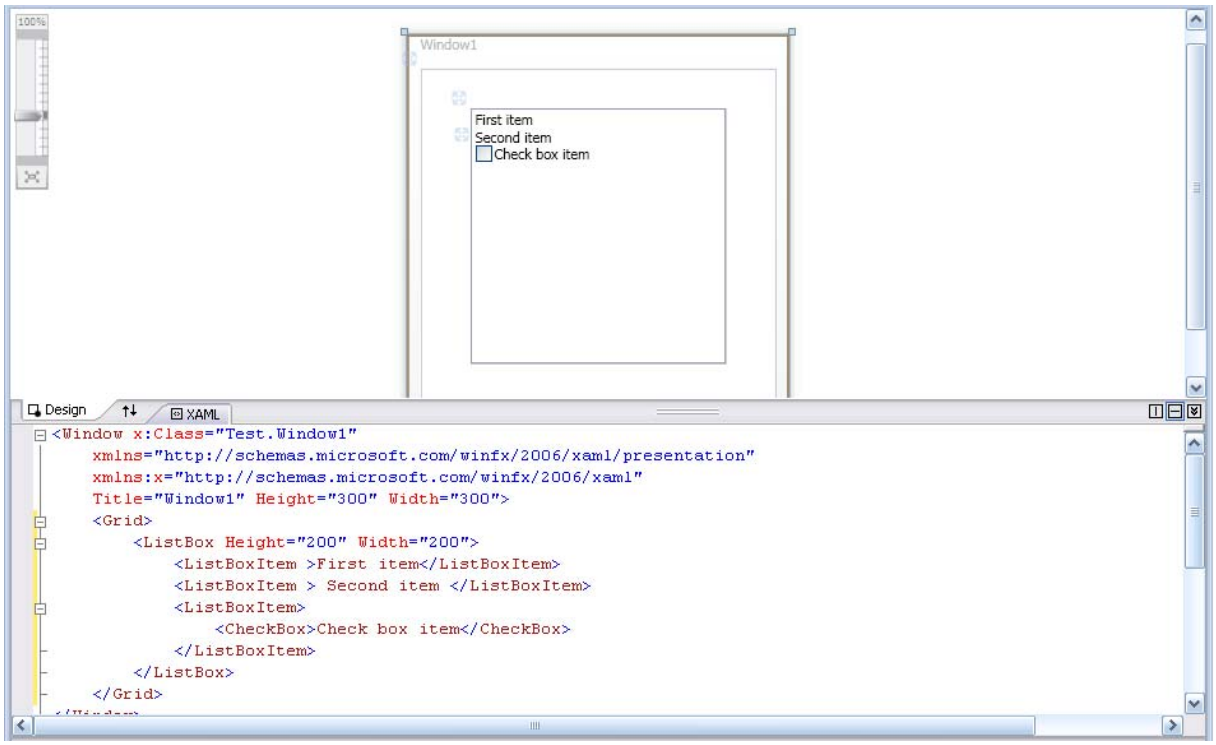
Sledeća zanimljiva Content kontrola je **ScrollViewer**. ScrollViewer omogućava scroll veoma velikih sadržaja, tako što se kao sadržaj ScrollViewer-a postavi panel na kome je potrebno omogućiti scroll. Propertiji koji su karakteristični za ScrollViewer su:

`VerticalScrollBarVisibility` (po default-u je "Auto")

`HorizontalScrollBarVisibility` (po default-u je "None")

7.4 Items controls

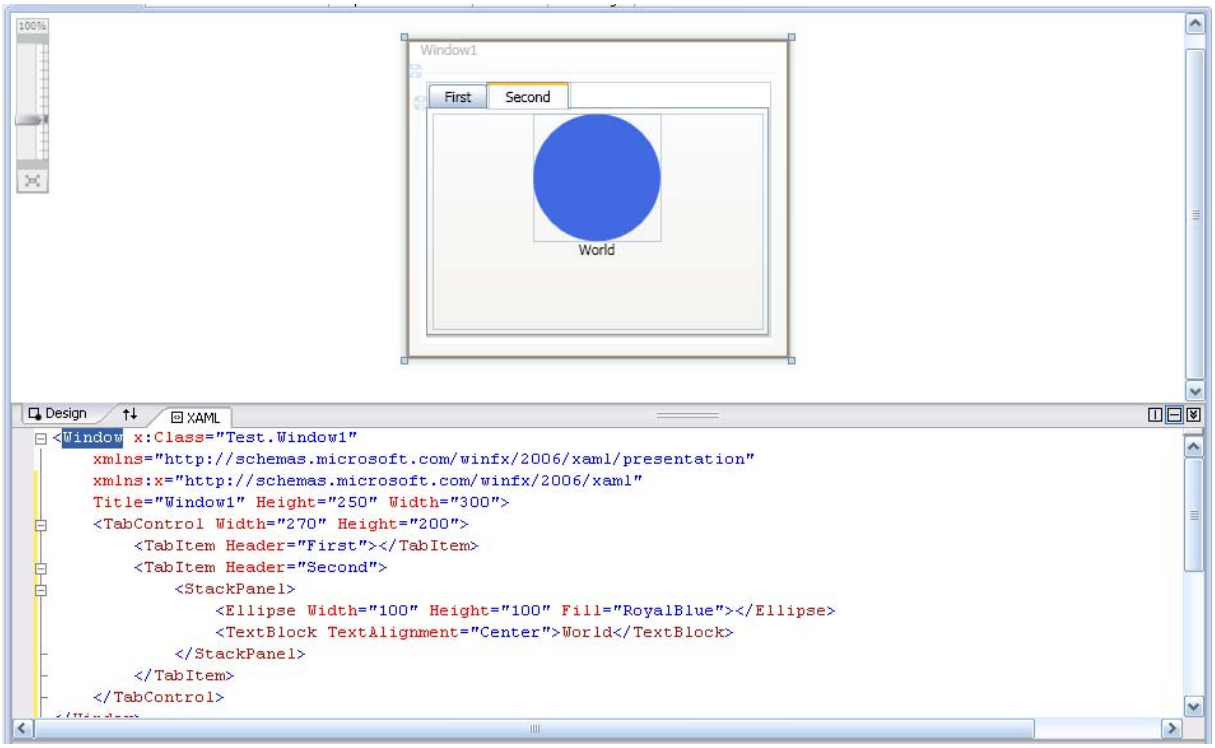
Sledeći tip kontrola su **Items kontrole**. To su *ListBox*, *TabControl* i *Menu*. Razlika između Content kontrola i Items kontrola je ta što prve imaju properti "*Content*" (u jednini, tj. mogu imati jedno dete kao sadržaj) a druge imaju properti "*Items*" (u množini, tj. mogu imati više dece). Najčešće korišćena Items kontrola je **ListBox**.



Slika 1-16

Prve dve stavke, odnosno item-a sadrže string, ali treća sadži check box. ListBoxItem je u stvari content kontrola. Dakle, Item može biti Content kontrola.

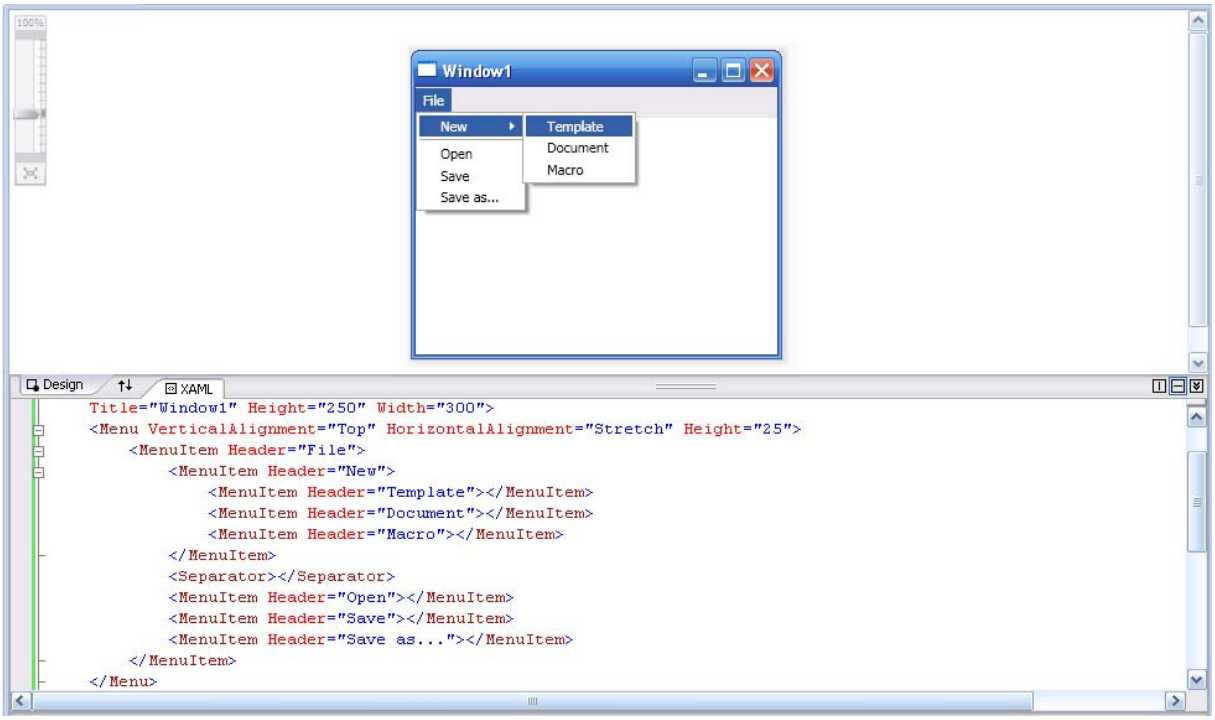
Sledeći primer Items kontrole je **TabControl**.



Slika 1-17

Items u ovoj kontroli su TabItem-i. Za svaki item osim Content propertija, postoji i *Header* properti. Header properti je content properti, što znači da može da sadrži Rich content (u tom slučaju je Complex properti).

I poslednji primer za Items kontrole je **Meni**.



Slika 1-18

Sva fukcionalnost se nalazi u MenuItem-ima. Svaki MenuItem može da sadrži MenuItem, tj. organizovani su hijerarhijski. Sam Meni je samo bar koji se vidi.

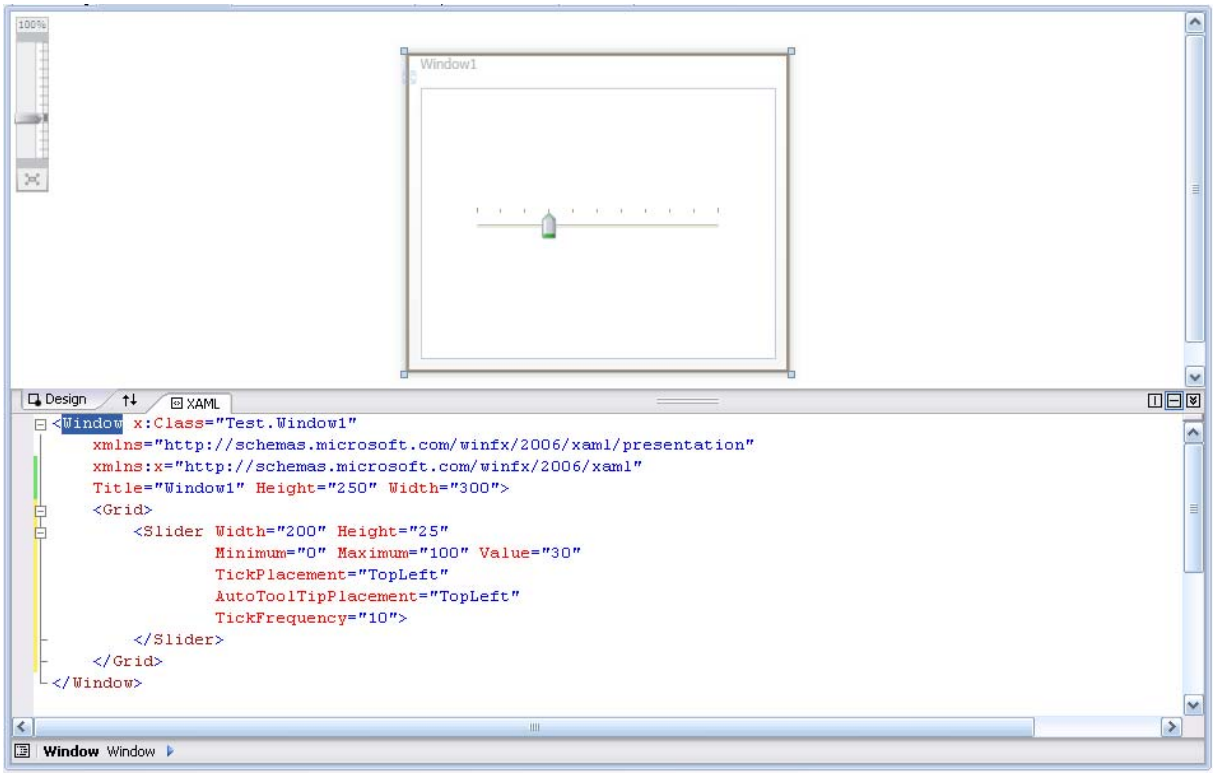
7.5 Range bases

Primeri ovog tipa kontrole su: **Slider**, **ScrollBar** i **ProgressBar**.

Slider je odličan primer Range Base kontrole. Ono što je zajedničko za sve kontrole ovog tipa je da imaju sledeće propertije: *Minimum*, *Maximum* i *Value*. Na sledeći način se Slideru mogu dodati Minimum i Maximum propertiji:

```
Minimum = "0" Maximum = "100" Value="3"
```

Slider ima još nekoliko zanimljivih propertija. Prvi od njih je properti *TickPlacement* koji govori Slideru da prikaže tačkice. Properti TickPlacement može imati vrednosti "*TopLeft*" i "*BottomRight*". Sledeći zanimljiv properti je properti *AutoToolTipPlacement* koji može imati vrednosti takođe "*TopLeft*" i "*BottomRight*". Properti *TickFrequency* govori koliko često se tačkice prikazuju.



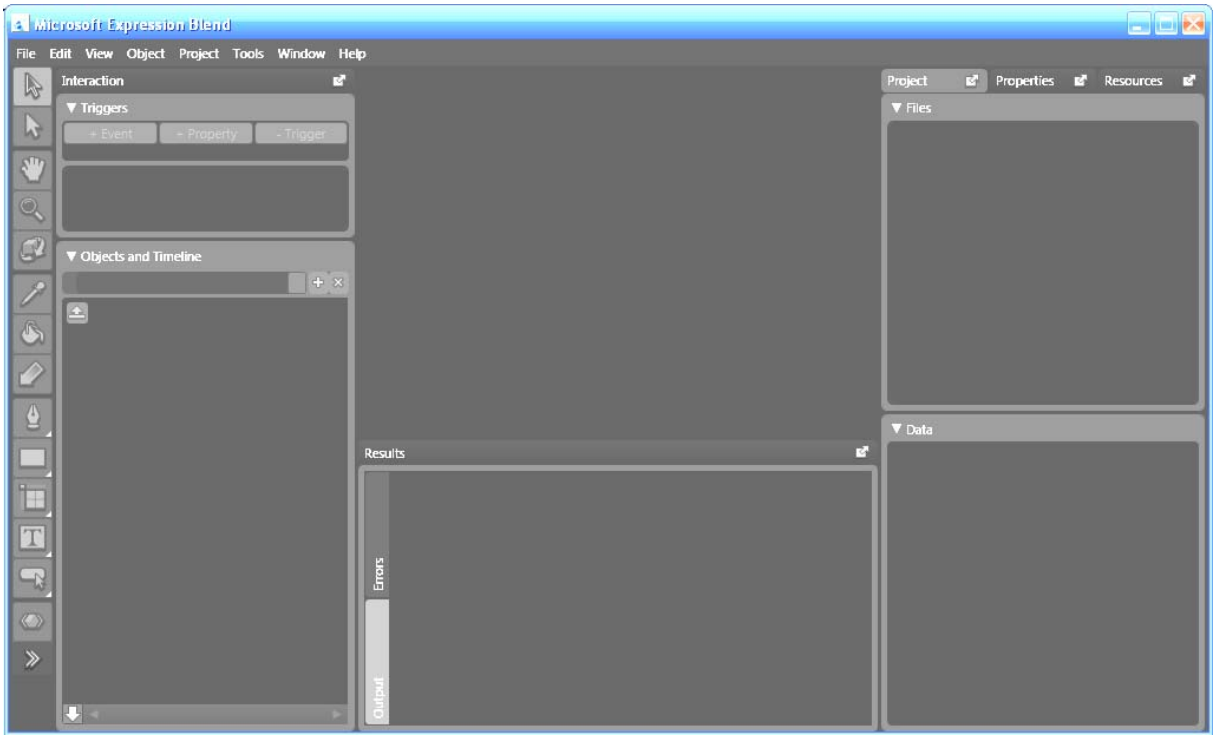
Slika 1-19

Ovim je zaokružena celina o osnovnim kontrolama koje se sreću u WPF-u.

8 Microsoft Expression Blend

Microsoft Expression Blend je dizajnerski alat namenjen izgradnji vizuelno razvijenih grafičkih interfejsa za web i desktop aplikacije koje sadrže 2D i 3D grafiku i animacije. Napisan je pomoću .NET Framework 3.0 i WPF-a. Expression Blend je interaktivan, WYSIWYG (What You See Is What You Get – ''ono što vidite to ćete i dobiti'') alat za dizajniranje interfejsa zasnovanih na XAML-u za WPF i Silverlight.

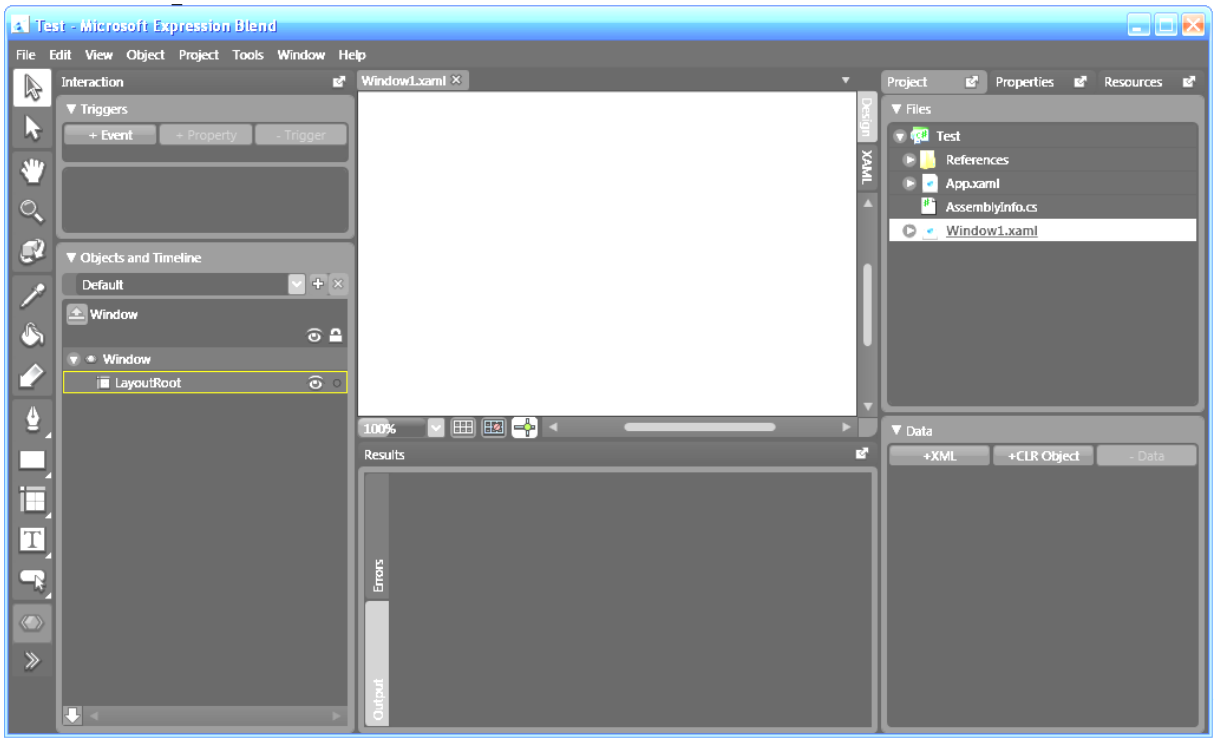
Expression Blend okruženje:



Slika 1-19

Novi projekat se kreira izborom opcije File > New Poject, build-uje se izborom opcije Project > Build Project, a pokreće se (testira) izborom opcije: Project > Test Poject, ili pritiskom na F5.

Kada je novi projekat kreiran, vidi se sadržaj korenog elementa Window1. Kada se dodaju novi elementi, dodaju se u onaj element koji je u odeljku ''*Objects and Timeline*'', na slici je to *LayoutRoot* . LayoutRoot je Grid panel koji se kreira pri kreiranju projekta u okviru Window1 elementa (Slika 1-20).

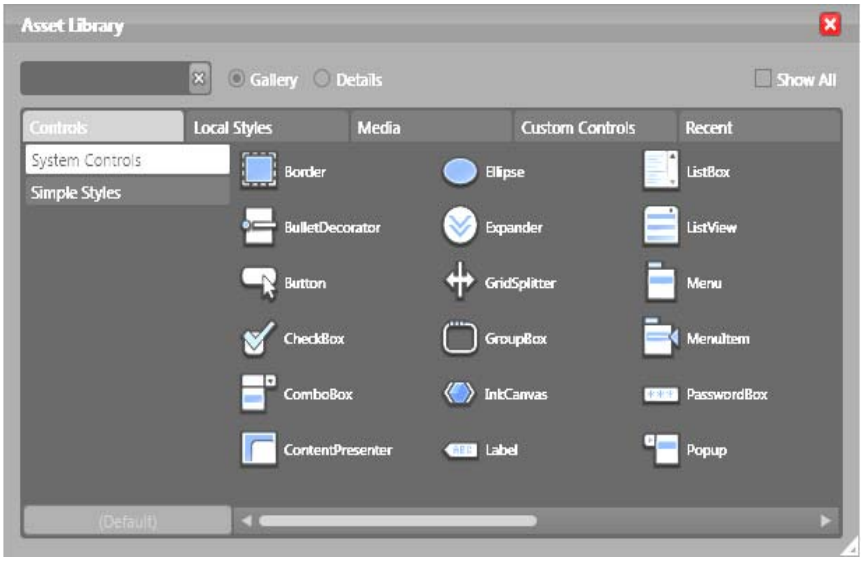


Slika 1-20

Novi elementi se dodaju dvoklikom na željeni element iz ToolBox-a koji se nalazi sa leve strane.

Ako kliknemo jednom na element koji ima malu belu stelicu dole levo i zadržimo, prikazaće se i ostali elementi koji pripadaju toj grupi elemenata.

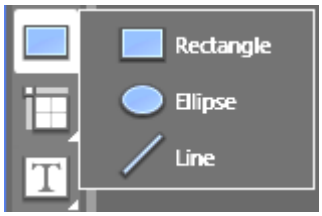
Ako želimo da dodamo element koga trenutno nema u ToolBox-u, odaberemo poslednju stavku u ToolBox-u, pri čemu će se otvoriti Assets Library iz kog možemo odabrati element:



Pogled na projekat može biti iz Dizanjer-a ili iz XAML-a, a menja se izborom opcije View Design (ili XAML), ili pritiskom na F11 (za Dizajner) i F12 (za XAML).

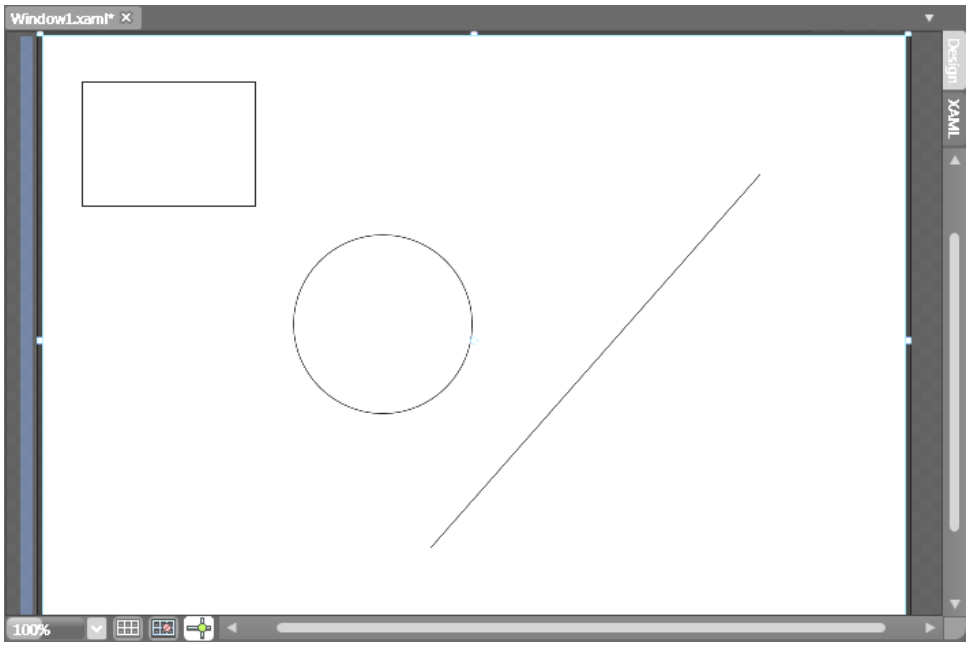
Do sada su primeri uglavnom rađeni u Xaml-u. Međutim, postoje određene vizulene stvari koje ima više smisla raditi u alatu (kao što je Expression Blend).

Pogledajmo oblike u ToolBox-u:



Izaberimo Pravougaonik (Rectangle) i nacrtajmo ga, zatim odaberimo elipsu (Ellipse), i na kraju - liniju (Line).

Dobiće se izgled kao na sledećoj slici:



Pogledajmo Xaml koji se prilikom toga kreirao (pritiskom na taster F12 na tastaturi):

```
<Window
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       x:Class="Test.Window1"
       x:Name="Window"
       Title="Window1"
       Width="640" Height="480">

       <Grid x:Name="LayoutRoot">
              <Rectangle Fill="#FFFFFFFF" Stroke="#FF000000"
                     HorizontalAlignment="Left" Margin="28,33,0,0"
                     VerticalAlignment="Top" Width="128" Height="92"/>
              <Ellipse Fill="#FFFFFFFF" Stroke="#FF000000"
                     HorizontalAlignment="Left" Margin="183,145,0,169" Width="132"/>
              <Path Fill="#FFFFFFFF" Stretch="Fill" Stroke="#FF000000"
                     Margin="283.5,100.5,105.5,70.5" Data="M284,375 L526,101"/>
       </Grid>
</Window>
```

Prvi element je Window, a njegovo dete je Grid, tj. LayoutRoot. U Gridu se mogu videti tri oblika koja su kreirana. Prvo je to pravougaonik, zatim elipsa i na kraju je očekivano da to bude Line element, ali je Blend kreirao *Path* objekat. Ovo su tri osnovna oblika u WPF platformi. Path je višenamenski oblik. Može se koristiti za bilo kakvu geometriju. Važan properti na ovom elementu je *Data* i on definiše tu geometriju.

Ovako kreiran projekat se može jednostavno importovati u Visual Studio ukoliko nije bilo problema pri instalaciji.

9 Resources i Databinding

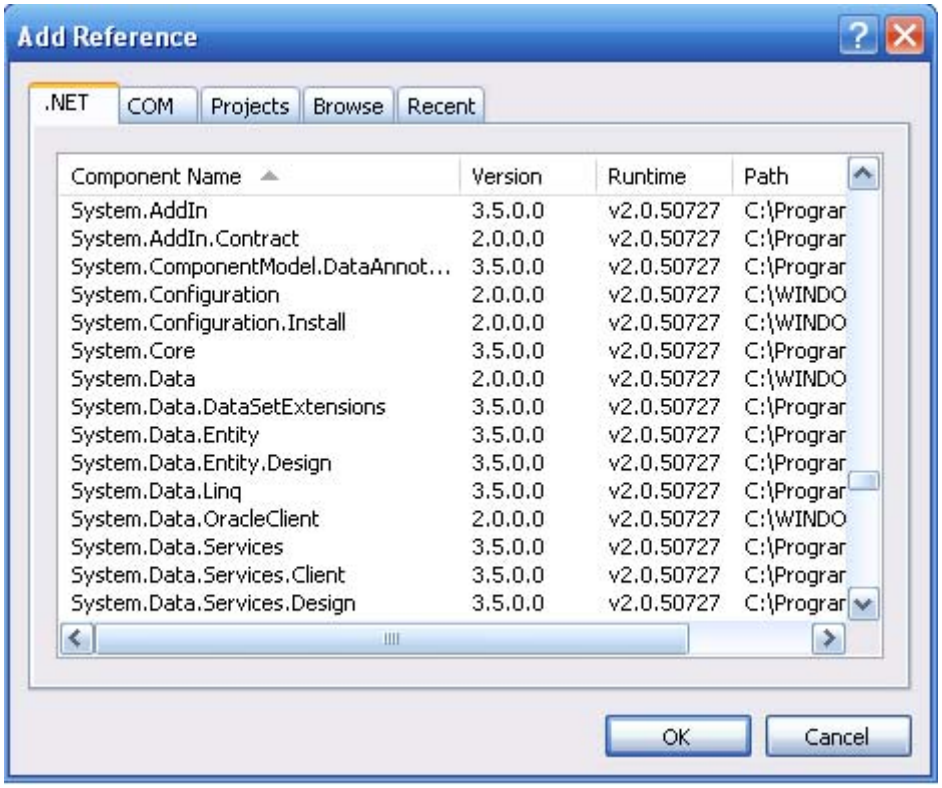
9.1 Dodavanje referenci

Resusi u WPF-u predstavljaju način da se nešto više puta ili naknadno koristi u aplikaciji ili da se deli na više mesta u aplikaciji (npr. određena boja može da predstavlja resurs i da se koristi na više mesta u aplikaciji). Svaki element (objekat) u WPF-u ima resource properti. Prilikom deklarisanja resursa koristi se complex properti sintaksa.

Primer: definisanje resursa na Window elementu

```
<Window.Resources>
      <m: ObjekatKojiPredstavljaResurs
      x:Key="JedinstveniNazivObjekta"
      PropertyObjektaKojiPredstavljaResurs="value"
</Window.Resources>
```

Da bi se koristila klasa iz nekog projekta potrebno je dodati referencu u References aktuelnog projekta (vidi sliku 1-21).



Slika 1-21

Na .NET tabu se mogu izabrati već postojeće .NET-ove klase, dok na Projects se mogu referencirati neki od postojećih projektata u okviru solution-a.

Takođe je potrebno i u Window1.xaml dodati reference na izabrani namespace i istoimeni sklop na sledeći način:

```
xmlns:m="clr-namespace:TestProject;assembly=TestProject"
```

Slovo "m" označava novi namespace koji se dodaje kao properti od Window elementa.

Sada možemo da kreiramo instancu klase "TestProject" kao resurs na Window elementu na sledeći način:

```
<Window.Resources>
      <m:TestProject
      x:Key="TestProject" >
<!--U ovom delu se mogu koristiti propertiji klase TestProject, i njima se mogu
dodeljivati odgovarajuće vrednosti-->
      </m:TestProject>
</Window.Resources>
```

Slovo "m" označava da klasa TestProject pripada "m" namespace-u. Properti "*Key*" nije properti klase TestProject, već pripada "x" namespace-u, što znači da je "Key" iz XAML specifikacije. "Key" predstavlja jedinstveni naziv resursa koji omogućava da referenciramo taj resurs.

Referenciranje resursa se može vršiti:

- statički (**Static Resource Reference**)
- i dinamički (**Dynamic Resource Reference**).

Razlika između ove dve vrste referenciranja je u tome što se statička referenca na resurs kreira prilikom load-a (učitavanja), a dinamička prilikom runtime-a (izvršavanja). Ukoliko nema razloga menjati vrednost resursa kada je on jednom referenciran, primenjuje se statičko referenciranje. Kada vrednost resursa zavisi od uslova koji nisu poznati sve do vremena izvršavanja, koristi se dinamičko referenciranje.

9.2 Markup Extension

Na properti DataSource je potrebno staviti referencu (statičku) na resurs koji smo prethodno kreirali što se može uraditi na sledeći način:

```
DataSource ="{StaticResource TestResource}"
```

Sintaksa sa vitičastim zagradama "{ }" se naziva "markup extension". Prva reč u okviru markup extension-a označava naziv markup extension-a, nakon nje slede parametri.

Reč *StaticResource* u okviru vitičastih zagrada označava naziv markup extension-a a reč TestResource je parametar (u ovom konkretnom slučaju je naziv (Key) resursa). Mark-up extension se može shvatiti kao XAML prečica.

9.3 Data Template

Ukoliko je potrebno da se odvojeno od same kontrole specificira način na koji će se ona prikazati ili da se skup podataka prikaže na identičan način u okviru nekog template-a, potrebno je koristiti **Data Template**.

Sledeći deo koda predstavlja osnovu za DataTemplate:

```
<StackPanel Orientation="Horizontal">
      <Image Width="24" Height="24" Margin="4"
             VerticalAlignment="Center"
             Source="Slika.jpg" />
      <StackPanel VerticalAlignment="Center">
            <TextBlock FontWeight="Bold" Text="Naziv slike"/>
      </StackPanel>
</StackPanel>
```

Nakon napravljene osnove za DataTemplate, može se kreirati sam Data Template kao Resurs Window objekta:

```
<DataTemplate x:Key="NazivTemplate-a">
      <StackPanel Orientation="Horizontal">
            <Image Width="24" Height="24" Margin="4"
                  VerticalAlignment="Center" Source="Slika.jpg" />
           <StackPanel VerticalAlignment="Center">
                  <TextBlock FontWeight="Bold" Text="Naziv slike"/>
           </StackPanel>
      </StackPanel>
</DataTemplate >
```

S obzirom na to da je fiksno zadata vrednosti propertija za lokaciju slike i naziva, potrebno je to izmeniti. U tu svrhu, primeniće se Databinding koji će vrednosti propertija Source i Text da poveže sa odgovarajućim vrednostima propertija neke postojeće klase:

```
<DataTemplate x:Key="NazivTemplate-a">
      <StackPanel Orientation="Horizontal">
            <Image Width="24" Height="24" Margin="4"
                  VerticalAlignment="Center"
                  Source="{Binding Property-PutanjaSlike}" />
            <StackPanel VerticalAlignment="Center">
            <TextBlock FontWeight="Bold" Text="{Binding Property-NazivSlike}"/>
            </StackPanel>
      </StackPanel>
</DataTemplate >
```

U ovom trenutku kreiran je samo DataTemplate. Kao što je već napomenuto, svaka kontrola u WPF-u sadrži Template properti. Osnovna ideja Template-a je da se može tačno specificirati kako ta kontrola izgleda. Napravljeni DataTemplate se može sada povezati sa Template propertijem neke kontrole.

```
TemplateProperty="{StaticResource ListaPesamaTemplate}"
```

Postavljanjem ovog propertija omogućava se željeni izgled podataka koji su definisani u DataTemplate-u. Template propertiji mogu biti ili ItemTemplate neke item kontrole, ContentTemplate koji predstavlja template properti content kontrole itd.

Elektronska oglasna tabla
- WPF aplikacija -

10 Ideja i cilj izrade elektronske oglasne table

U velikim institucijama, kojima svakodnevno cirkuliše veliki broj ljudi, gotovo neprekidno se smenjuju informacije, koje iz trenutka u trenutak menjaju svoju sadržinu. Razlikujemo informacije koje se razmenjuju unutar same institucije i one koje se razmenjuju između zaposlenih i korisnika usluga te institucije. Kako bi informacije brzo stigle do odgovarajuće ciljne grupe, koriste se razne metode. Jedna od njih je korišćenje oglasnih tabli.

Korisnici oglasnih tabli sreću se sa sledećim ograničenjima i problemima:

- Preveliki broj informacija (papiri zalepljeni jedni preko drugih).
- Zastarele vesti (oglasi koje je trebalo neko da ukloni po isteku važenja oglasa).
- Neefikasno korišćenje prostora za oglase (kako bi se postavio novi oglas potrebno je ponekad ručno izvršiti reorganizaciju svih oglasa na tabli i napraviti dovoljno mesta za novi).
- Nemogućnost hronološkog pregleda vesti od najsvežijih ka zastarelim.
- Komplikovano ažuriranje već postojećih oglasa na tabli (ponekad je potrebno napraviti iznova već postojeći oglas zbog promene nekog termina).
- Oglasna tabla je fizički vezana za ustanovu na koju se odnosi (korisnici usluga ustanove moraju da dođu do ustanove kako bi proverili najnovije oglase).

Ideja da se čitav ovaj proces pregledanja i ažuriranja oglasnih tabli automatizuje nije nova. U savremenijim ustanovama, kojima profit zavisi od pouzdanosti i preglednosti informacija, već postoje elektronske oglasne table na kojima se iz minuta u minut stare informacije zamenjuju novim. Tipičan primer su elektronske table na aerodromina širom sveta, koje redovno smenjuju najsvežije informacije o redu letova prikupljene iz baza podataka svetskih avio-kompanija. Za prikazivanje informacija koriste se veliki display-i visoke rezolucije. Ranije su samo najsavremenije kompanije mogle sebi da priušte takav način distibuiranja informacija zbog velikih troškova i skupih komponenti. Međutim, razvoj hardware-a danas je dostigao taj nivo da svaka ustanova sebi može da priušti display odgovarajućeg kvaliteta i informacioni sistem koji će u pozadini obavljati potrebnu logiku. Svaka od ustanova u kojoj su neophodne oglasne table, trudiće se da pruži korisniku upravo one informacije koje su njemu potrebne, tako što će se težiti da oglasne table budu predstavljene u vidu aktivnih ekrana (touchscreen). Time bi se omogućila dvosmerna interakciju između korisnika i oglasne table. Na taj način korisnik neće biti samo pasivan posmatrač, već će aktivno učestvovati u odabiru informacija za prikaz, u zavisnoti od njegovih potreba.

Razvojem u svim oblastima informacionih tehnologija, rešen je i problem statičnosti informacija. Danas gotovo svaki pojedinac poseduje lični računar koji ima pristup internetu. Elektronske oglasne table takođe mogu biti urađene u vidu desktop ili web aplikacije. Na taj način korisnik se može pretplatiti na usluge koje pruža neki web servis i preko svoje desktop aplikacije uvek biti snabdeven najnovijim informacijama iz oblasti njegovog interosovanja. Ili na još jednostavniji način, posetiti neki od web portala koji sve potrebno imaju na jednom mestu.

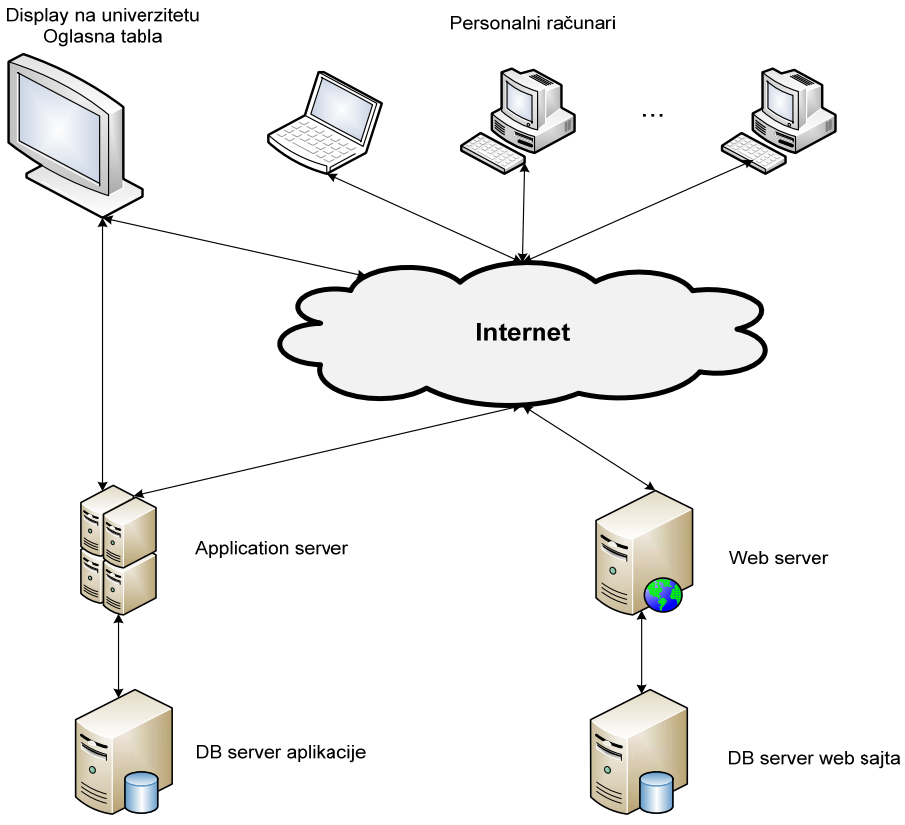
11 Arhitektura studentske elektronske oglasne table

Problemi koji se odnose na standardne oglasne table zastupljeni su i na univerzitetima. Blagovremeno obaveštavanje profesora i studenata o promeni rasporeda zauzetosti učionica, odbranama diplomskih radova, rasporedu polaganja ispita, rezultatima ispita itd. u velikoj meri utiče na efikasnost obavljanja obostranih obaveza. Ukoliko se obaveštenja adekvatno vremenu postavljaju i ažuriraju dolazi do znatne uštede vremena.

Pedantnost nad izmenama oglasne table nije jedina stvar koja studentima i profesorima štedi vreme. Način na koji se skladište informacije, način njihovog dopremanja i prikazivanja na dysplay-u doprinosi lakšem upravljanju velikim brojem informacija, pedantnosti prikaza i uštedi resursa.

11.1 Network diagram

Kako bi se postigli optimalni rezultati u razmeni informacija i njihovom prikazivanju, potrebno je dobro osmisliti arhitekturu koja će biti odgovorna za čitavu logiku, kao i to koji su resursi neophodni za funkcionisanje te arhitekture (videti sliku 2-1).



Slika 2-1

Proces prenosa nekog obaveštenja od trenutka kreiranja do trenutka njegovog prikazivanja bi trebalo da bude podržan arhitekturom prikazanoj na slici 2-1.

Zamisao je da razvijeno rešenje može biti korišćeno od strane više klijenata a da se kompletna logika rešenja nalazi na jednom serveru. Oglasna tabla bi trebalo da može da prikazuje obične textualne oglase ili oglase u vidu HTML strane. Ideja oglasa u vidu Web strane nam omogućava da format oglasa dobijamo kompletno sređen i da oko formatiranja sadržaja olakšamo administrativni posao.

Shodno tome, možemo imati jedan ili više servera baza podataka. Na njima se mogu nalaziti sve informacije neophode za prikaz oglasa na oglasnoj tabli. Prikaz arhitekture je osmišljen sa dva servera baze podataka. Fokusirajmo se na prvi DB server aplikacije. On bi trebalo da sadrži podatke koji se odnose na samu strukturu oglasa (npr. tekst sadržaja oglasa, naslov, sažetak, datum njegovog postavljanja i uklanjanja…). Postoji opcija da se tekst oglasa ne nalazi u samoj bazi, već da se čuva Url koji ukazuje na lokaciju na kojoj se sadržaj oglasa zapravo nalazi. Iz tog razloga nam može zatrebati još jedan DB server koji će sadržati informacije za prikaz na Web sajtu koji bi u ovom slučaju predstavljao sadržaj oglasa.

Application server sadrži windows servis koji obavlja kompletnu logiku potrebnu za upravljanje oglasima smeštenim na DB serveru. Međutim sam windows servis ne obezbeđuje pristup distribuiranim klijentskim aplikacijama. Da bismo razvili distribuirano rešenje, potrebno je da u već postojeće rešenje uvrstimo i Web servis.

Web servis je zaslužan za komunikaciju između više nezavisnih aplikacija, u našem slučaju klijentske aplikacije za prikaz oglasne table i windows servisa zaduženog za upravljanje i dopremanje podataka iz baze. Kao host za Web servis će nam poslužiti windows servis i na taj način smo obezbedili komunikaciju između klijentskih računara i windows servisa.

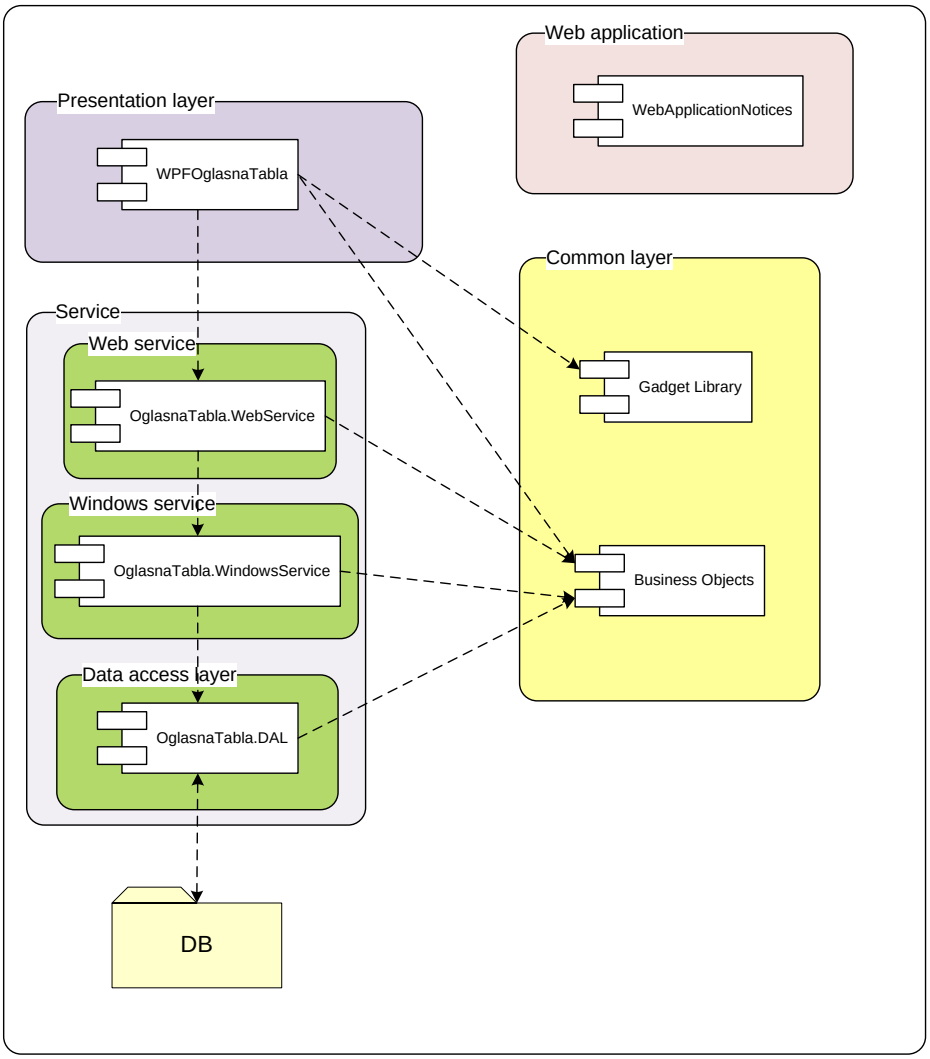
Klijentska aplikacija, u vidu interfejsa za prikaz oglasne table, može pristupati Web servisu putem interneta ili intraneta. Kao primer pristupa putem intraneta možemo navesti prikaz oglasne table na nekom od računara unutar mreže samog univerziteta (oglasne table u hodnicima, unutar kabineta, učionica itd.). Putem interneta dostupnost bi bila moguća sa bilo kog mesta.

Svaki od personalnih računara bi imao instaliranu desktop aplikaciju koja bi preko HTTP protokola komunicirala sa web servisom, lociranom na application serveru, i kao odgovor dobijala XML koji bi sadržao informacije potrebne za prikaz na oglasnoj tabli. Ukoliko je sadržaj oglasa u vidu Url-a, klijentska aplikacija bi putem interneta pokupila sadržaj sa Web sajta u vidu HTML-a i prikazala ga kao sadržaj oglasa.

Na ovaj način kompletirana je neophodna arhitektura za skladištenje, upravljanje i prikaz informacija.

11.2 Component diagram

Detaljan prikaz strukture sistema predstavljen je na slici 2-2.



Slika 2-2

Aplikacija sadrži sledeće slojeve:

- DAL (Data Access Layer) sloj koji omogućava pristup bazi podataka.
- Windows service koji implementira potrebnu logiku za dopremanje podataka iz baze i njihovo prosleđivanje klijentu.
- Web service je posebna komponenta host-ovana na windows servisu, čija je uloga da omogući komunikaciju između klijenta i servisa.
- Bussiness objects sadrži definicije svih entiteta koji se koriste u svim slojevima aplikacije.
- Gadget library predstvalja custom biblioteku potrebnu za prikaz oglasa na klijentu, i nju referencira Presentation layer.

- Na prezentacionom sloju predstavljen je korisnički interfejs koji je implementiran u vidu WPF forme.
- Web application je nezavisna web aplikacija, na čijim su stranama pojedinačno prikazani oglasi. Putanje do tih strana se čuvaju u bazi, na osnovu čega klijentska WPF aplikacija zna kako da pristupi tim stranama i prikaže ih u okviru svog inferfejsa.

Organizacija sistema predstavljena prethodnim dijagramom omogućava nezavisno funkcionisanje fizički odvojenih celina.

12 Razvojno okruženje

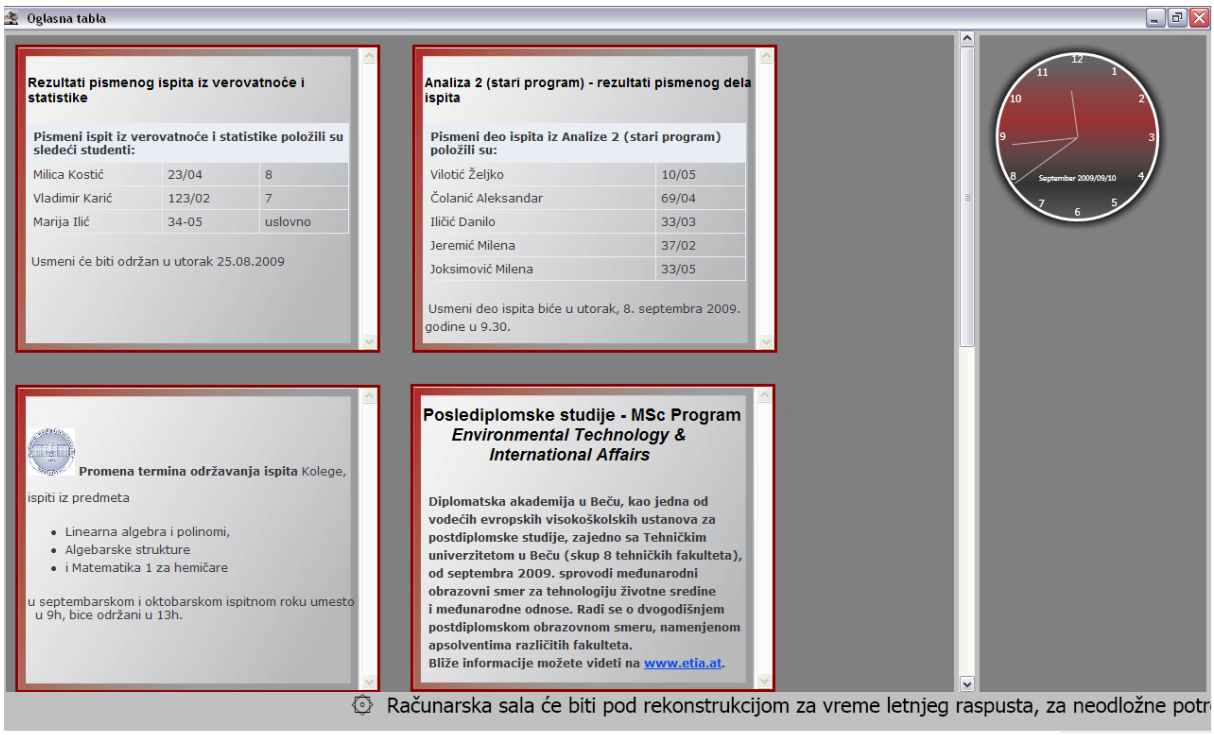
Tokom razvoja rešenja elektronske oglasne table korišćene su sledeće tehnologije i razvojni alati:

- **.NET framework 3.5** – okruženje za razvoj aplikacija namenjenih za rad na Windows operativnim sistemima.
- **Microsoft Visual Studio 2008**– razvojno okruženje za izradu desktop i web aplikacija, kao i windows i web servisa.
- **Microsoft SQL Server 2005** – sistem za upravljanje relacionom bazom podataka koji je omogućio skladištenje podataka vezanih za oglase kao i njihovo dopremanje iz baze podataka.
- **Microsoft SQL Server 2005 Management Studio –** alat za razvoj baze podataka (kreiranje tabela, funkcija i stored procedura).
- **Microsof Expression Blend 2** –dizajnerski alat korišćen za izradu vizuelno složenih grafičkih komponenti klijentskog dela rešenja.
- **WPF –** deo .NET framework-a 3.5 korišćen je za razvoj korisničkog interfejsa.
- **Windows Communication Foundation (WCF)** – je servisno-orijentisan sistem razmene poruka koji je omogućio razvoj web servisa, tj. komunikaciju između klijentske aplikacije i windows servisa.
- **ASP.NET** – deo .NET frameworka korišćen za razvoj web aplikacija.

Kompletno rešenje je razvijeno u programskom jeziku C# i T SQL jeziku za pisanje upita nad bazom podataka.

13 Implementacija klijentske aplikacije

Zamisao je bila da klijentska aplikacija bude urađena u vidu desktop aplikacije, čiji će interfejs sadržati osnovne komponente koje jedna oglasna tabla treba da sadrži. Oglasnu tablu predstavlja WPF forma koja na sebi ima dva posebna panela. Prvi panel služi za prikaz oglasa, dok drugi sadrži pomoćne alate (gadget-e, kao što je sat, vremenska prognoza, slika po izboru itd.). Svaki od oglasa je urađen u vidu gadget-a, tj. posebne korisničke kontrole koje će korisnik moći da raspoređuje po izboru u granicama panela predviđenog za oglase. Takođe je u dnu ekrana oglasne table postavljena pokretna traka (Khyron) kojom se kreću najnovija obaveštenja. Na taj način korisnik aplikacije ima kompletan uvid u sve potrebne informacije koje bi trebalo da mu pruži jedna oglasna tabla (slika 2-3).



Slika 2-3

Na osnovu već opisanog interfejsa vidimo da klijentsku aplikaciju predstavlja jedna WPF forma koja se sastoji iz sledećih komponenti:

- Panela za prikaz oglasa.
- Panela za prikaz pomoćnih alata (gadget-a).
- Sata u vidu dodatne korisničke kontrole.
- Oglasa (takođe u vidu gadget-a).
- Pokretne trake (Khyron-a).

Implementacja svake od navedenih komponenti će biti objašnjena pojedinačno.

13.1 Kreiranje WPF forme

Prvi korak je kreiranje novog solutiona u Visual Studiju i dodavanje projekta tipa WPF Application nazvanog „WPFOglasnaTabla“. Dodati projekat sadrži Window1.xaml fajl u okviru kog se uređuje kompletan izgled interfejsa. Navedeni .xaml fajl sadrži sledeće:

```
<Window x:Class="WPFOglasnaTabla.Window1"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Oglasna tabla"
       Height="600"
       Width="800"
       WindowStartupLocation="CenterScreen"
       Background="Silver" ResizeMode="CanResize"
       MinWidth="800" MinHeight="600" Icon="Logo.jpg">

       <Grid>
       </Grid>
</Window>
```

Properti Title služi za postavljanje naslova same forme, a sa Height, Widtht, MinHeight i MinWidth postavljaju se dimenzije forme u inicijalnom stanju i minimalne ispod kojih neće moći da se promeni veličina prozora. Properti WindowStartupLocation određuje pozicije prikazivanja prozora pri pokretanju aplikacije. Background omogućava postavljanje boje pozadine, dok postavljanje ResizeMode na CanResize omogućava promenu veličine prozora. Naravno, uvek možemo da postavimo da bude onemogućena promena veličine prozora, ali u ovom slučaju je elegantnije rešenje da je moguće promeniti veličinu. Icon postavlja ikonicu pored naslova forme, u vidu slike koja je pre toga importovana u projekat iz kog se poziva.

Pokretanjem aplikacije sa F5, možemo videti prikaz sa slike 2-4.

Slika 2-4

13.2 Postavljanje panela

Kao što se može videti sa slike 2-3, postoje dva panela. Prvi na kome se nalaze oglasi i drugi za dodatne korisničke kontrole. Pošto je deo za pomoćne korisničke kontrole fiksne veličine koja zavisi od veličine tih kontrola, ideja je da panel, planiran za oglase, zauzima sav preostali raspoloživ prostor. To je odlična prilika za korišćenje Grida. Sa interfejsa gotove aplikacije se može videti da ovaj Grid ima dve kolone - jednu za oglase, a drugu za dodatne kontrole. Takođe, ima i dva reda. Prvi red sadrži ova dva panela, a drugi red sadrži pokretnu traku, o kojoj će kasnije biti reči, i koja će se proširiti na obe kolone. Za pokretnu traku je takođe planiran tačno određeni prostor, tako da će paneli zauzimati preostali raspoloživi prostor u odnosu na pokretnu traku.

Kod koji odgovara aplikaciji nakon uvođenja grida izgleda ovako:

```
<Grid Name="GridForPanels">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"
ScrollViewer.CanContentScroll="True"
      ScrollViewer.VerticalScrollBarVisibility="Visible">
             </ColumnDefinition>
           <ColumnDefinition Width="250"></ColumnDefinition>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="*"></RowDefinition>
            <RowDefinition Height="40" ></RowDefinition>
    </Grid.RowDefinitions>
</Grid>
```

SnapCanvas

Nakon uvođenja grida specificiran je prostor za svaku od globalnih komponenti aplikacije. U predodređeni prostor u gridu se postavljaju paneli u oviru kojih će se prikazivati korisničke kontrole i oglasi.

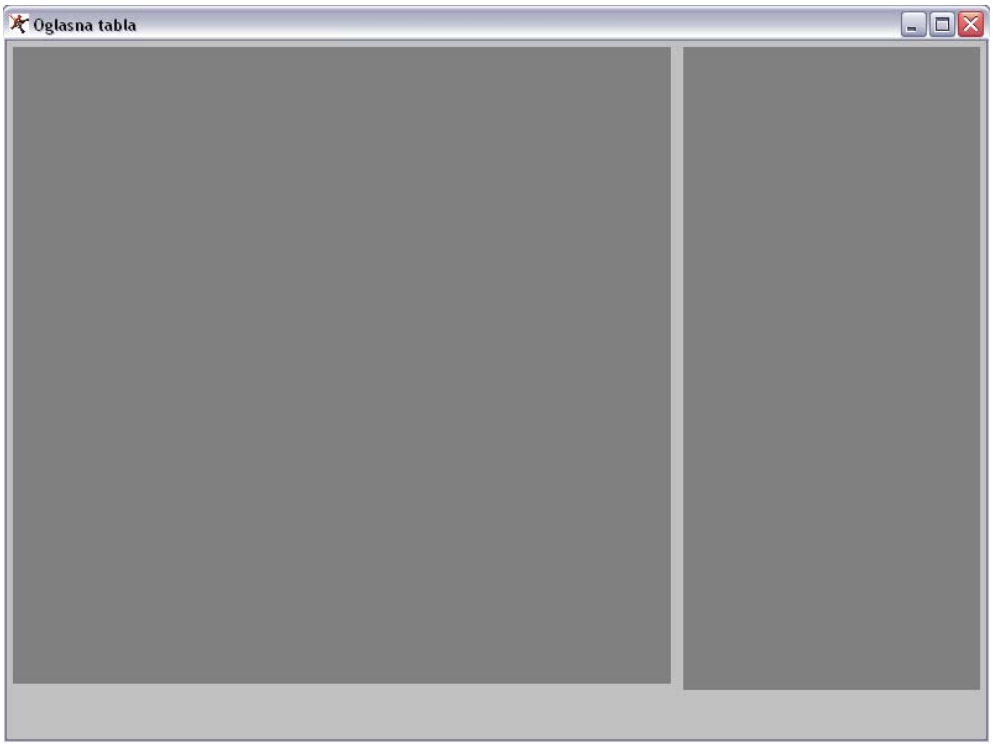
Pošto je ideja da korisnik aplikacije može sam da reorganizuje oglase na tabli kao i raspored pomoćnih kontrola, napravljen je custom canvas koji to omogućava. Zapravo, standardni canvas omogućava samo statički prikaz komponenti koje su prikazane na njemu, bez mogućnosti za njihovim pomeranjem. Da bi dinamički prikaz bio omogućen potrebno je doimplementirati kompletan layout mehanizam. Zbog složenosti i preobimnosti ove implementacije, postupak neće biti tema ovog rada, već se može preuzeti u celosti sa CodeProjecta u vidu gotove biblioteke koja je importovana u vidu novog projekta u već postojeći solution.
Link sa kog se može preuzeti kompletan kod je:
http://www.codeproject.com/KB/WPF/WPFGadget.aspx

Zbog razumevanja dalje implementacije, biće ukratko objašnjeno šta omogućava importovani projekat GadgetLibrary. Ovaj projekat sadrži dve bitne klase.

Prva klasa je SnapCanves koja nasleđuje originalni canvas i predstavlja izmenjenu verziju standardnog canvas-a. Ona omogućava korisniku da prevlači objekte unutar njega kao i mogućnost promene z-indeksa, tj. redosleda prikazivanja objekata. Kao što je već razmatrano u prethodnim poglavljima, većina panela u WPF- u automatski pruža mogućnost layout-a, što može znatno olakšati pozicioniranje kontrola u okviru panela. Međutim, Canvas je jedini panel u WPF – u koji ne omogućava automatski layout, već zadržava objekat na apsolutnoj poziciji kao i kod windows formi. Korisnost te osobine canvas-a se ogleda u tome da elementi canvas-a nikad ne menjaju veličinu ili lokaciju iako npr. sam prozor menja veličinu. Pozicija elemenata unutar canvas-a se određuje pomoću četiri attached propertia: Left, Right, Top i Bottom. Vrednost tih propertija se ogleda u tome gde će element biti pozicioniran u odnosu na dve strane canvas-a. Pošto je za pozicioniranje elementa potrebno navesti samo dve vrednosti u odnosu na sve četiri strane canvas-a, sasvim je dovoljno koristiti samo Left i Top propertije za određivanje pozicije elementa, a ostale ignorisati. Upravo je to ono što omogućava SnapCanvas i na čemu je bazirana cela logika. Nad standardnim canvas-om je implementiran dinamički layout, koji osluškuje akcije korisnika nad elementima SnapCanvas-a i izvršava pomeranje elemenata kao odgovor na te akcije.

Druga klasa GadgetLibrary-a kojom ćemo se kasnije pozabaviti je GadgetContainer.

Nakon definisanja uloge SnapCanvas-a za aplikaciju, vrši se njegovo postavljanje na samom designer-u. Definišu se dva SnapCanvas-a, prvi unutar prve ćelije grida na kome se prikazuju oglasi, a drugi odmah u ćeliji desno do njega, na kome se prikazaju dodatne kontrole. Izgled koji je dobijen nakon ove implementacije se može videti na slici 2-5.



Slika 2-5

Da bi se dobio prikaz kao na slici 2-5 potrebno je dodati u Window.xaml fajlu unutar grida sledeći kod:

```
<GadgetLibrarySnapCanves:SnapCanvas x:Name="_SnapCanvas"
                                    Grid.Column="0"
                                    Grid.Row="0"
                                    SnapThreshold="20"
                                              AllowDragging="True"
                                    Background="Gray"
                                    Margin="5"
                                    AllowDragOutOfView="False">
</GadgetLibrarySnapCanves:SnapCanvas>
<GadgetLibrarySnapCanves:SnapCanvas x:Name="_SnapCanvasForClock"
                                    Grid.Row="0"
                                    Grid.Column="1"
                                    AllowDragging="True"
                                    SnapThreshold="20"
                                    Margin="5,5,5,0"
                                    Background="Gray"
                                    AllowDragOutOfView="False">
</GadgetLibrarySnapCanves:SnapCanvas>
<Canvas Name="CanvasForKhyron"
       Grid.Row="1" Grid.ColumnSpan="2"
       VerticalAlignment="Top" Height="40" Width="800">
</Canvas>
```

Kako bi projekat WPFOglasnaTabla prepoznao SnapCanvas, potrebno je referencirati GadgetLibrary projekat i deklarisati namespace-e koji se koriste unutar Window klase.

```
xmlns:GadgetLibrarySnapCanves=
"clr-namespace:GadgetLibrary;assembly=GadgetLibrary"
```

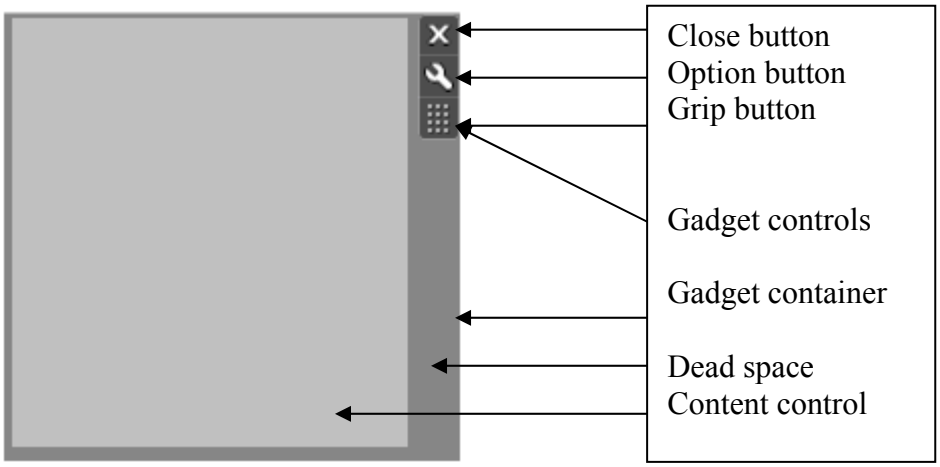
Na osnovu gore navedenog koda vidimo da su korišćeni attached properties za definisanje unutar kojih ćelija grida su pojedini paneli smešteni. Svrha prva dva penela je već objašnjena, dok treći panel *CanvasForKhyron* je planiran za pokretnu traku i poslužiće samo kao podloga za nju. Properti AllowDragging određuje da li će na SnapCanvas-u biti aktiviran dinamički layout mehanizam ili ne, dok properti *AllowDragOutOfView* definiše da li će elementi canvas-a moći da se pomeraju izvan granica panela.

13.3 Kreiranje gadget-a

Pre nego što se pređe na objašnjavanje kako je kreirana sama kontrola koja će predstavljati oglas, treba objasniti šta toj kontroli omogućava karakteristike gadget-a, tj. šta je to što omogućava pomeranje kontrole.

GadgetContainer

Vratimo se na projekat GadgetLibrary. Već je spomenuta klasa GadgetContainer koja je u direktnoj vezi sa SnapCanves-om. Ona obezbeđuje kreiranje generičke user kontrole u vidu kontejnera (container) koja omogućava izgled i definiše ponašanje gadget-a. Takođe pruža mogućnost da se u okviru kontejnera unese proizvoljan sadržaj u vidu user ili custom control-e.



Slika 2-6

Na slici 2-6 se mogu videti sve komponente gadget kontejnera, od kojih glavnu funkciju nosi Content control koja kao svoj sadržaj može da podrži bilo koju funkcionalnu user ili custom kontrolu. Close button omogućava uklanjanje gadget-a sa radne površine ili njegovo zatvaranje, Option button ostavlja prostor da se implementira neko od dodatnih ponašanja gadget-a (promena veličine, otvaranje nekog novog prozora…) dok Grip button obezbeđuje prostor čijim držanjem se pokreće gadget.

Upravo je gadget kontejner kontrola koja se koristi kao osnovau za kreiranje oglasa i dodatnih korisničkih kontrola. Od dodatnih kontrola prikazaćemo kreiranje sata, ali moguće je takođe kreirati i gadget u vidu kalendara, vremenske prognoze itd.

13.3.1 Kreiranje sata

Najlakši način da se napravi vizuelno lepa user kontrola, koja će potom biti importovana u Visual studio, jeste korišćenje nekog od dizajnerskih alata. Već je bilo reči o Expession Blend-u, i u ovom poglavlju će se iskoristiti sve njegove olakšice kako bi se kreirao dizajn za analogni sat.

Konačni izgled sata prikazan je na slici 2-7.

Da bi se kreirao analogni sat u Expression Blendu ili bilo kom drugom dizajnerskom alatu, potrebno je pre svega uočiti koje su to funkcionalne komponente koje neku kontrolu čine analognim satom. Sve ostalo je stvar kozmetike i umeća korišćenja pogodnosti dizajnerskih alata.

Slika 2-7

Analogni sat se sastoji iz sledećih komponenti:

- Elipse – predstavlja telo sata (eventualno dodatnih elipsi koje u kombinaciji sa glavnom elipsom mogu da prestavljaju okvir, senku ili sl.).
- Kazaljki – mogu se predstaviti pomoću pravougaonika u tri različite veličine.
- Centralne tačke sata – u odnosu na nju će se vršiti rotiranje kazaljki.
- Koda u pozadini – zadužen za implementiranje logike za pokretanje kazaljki u odnosu na centralnu tačku.

Kreiranje elipsi i kazaljki u vidu pravaugaonika ne bi trebalo da predstavlja veći problem, međutim postoji jedna stvar na koju bi valjalo skrenuti pažnju. Reč je o određivanju centralne tačke sata (u odnosu na koju će se vršiti rotacija) i kako da kazaljke budu ''svesne'' te tačke. Da bi se uočio centar elipse potrebno ju je selektovati i tada će se pojaviti centralna tačka. Nakon kreiranja pravougaonika, koji će predstavljati kazaljku, može se takođe uočiti njegova centralna tačka transformacije. Da bi kazaljka mogla da se kreće u odnosu na centar elipse potrebno je centralnu tačku kazaljke prevući tako da se poklopi sa centralnom tačkom elipse. Isti postupak se analogno primenjuje i na preostale dve kazaljke.

Nakon dobijanja željenog interfejsa za analogni sat u Expresion Blendu dobijen je izgenerisan XAML kod koji je potrebno importovati u Visual Studio u vidu user kontrole. Međutim, da bi sat proradio potrebno je implementirati pozadinsku metodu koja će na osnovu trenutnog sistemskog vremena računati ugao pod kojim treba da budu postavljene kazaljke kao i brzinu kojom će da se kreću.

Kod koji postavlja pozadinsku logiku je sadržan u sledećoj metodi:

```
void timer_Elapsed(object sender, System.Timers.ElapsedEventArgs e)
        {
            this.Dispatcher.Invoke(DispatcherPriority.Normal, (Action)(() =>
            {
                secondHand.Angle = DateTime.Now.Second * 6;
                minuteHand.Angle = DateTime.Now.Minute * 6;
                hourHand.Angle = (DateTime.Now.Hour * 30) + (DateTime.Now.Minute *
             0.5);
            }));
        }
```

Metoda `timer_Elapsed` predstavlja event handler koji se okida nakon određenog vremenskog perioda, specificiranom u konstruktoru same user kontrole.

13.3.2 Smeštanje user kontrole u gadget kontejner

Da bi se neka user kontrola prikazala u vidu gadget-a potrebno ju je smestiti unutar gadget kontejnera. Postupak će biti prikazan na primeru user kontrole za sat.

```
private void AddClockGadget()
        {
            var gadgetContainer = new GadgetContainer();
            Canvas.SetTop(gadgetContainer, 0);
            Canvas.SetLeft(gadgetContainer, 0);
            gadgetContainer.Close += OnGadgetClose;

            var clockGadget = new ClockGadget();
            gadgetContainer.Gadget = clockGadget;

            _SnapCanvasForClock.Children.Add(gadgetContainer);
        }
```

Pre svega je potrebno kreirati instancu GadgetContainer-a, a nakon toga instancu ClockGadget-a koja se postavlja u Gadget properti GadgetContainer-a. Zatim se tako kreiran gadget dodaje kao element u listu dece canves-a predviđenog za prikaz.

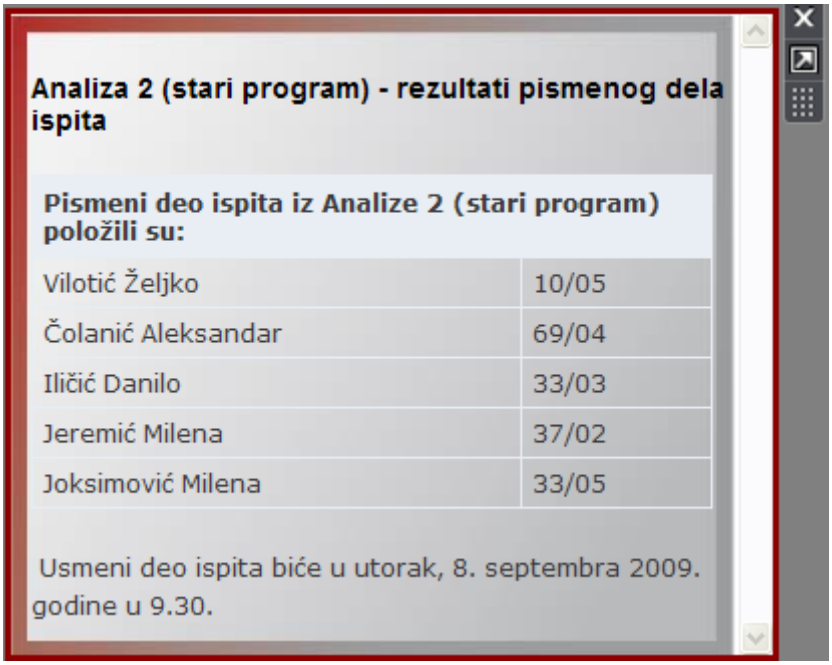
Napomenimo da klasa ClockGadget predstavlja user kontrolu za sat i da implementira interfejs IGadget iz biblioteke GadgetLibrary.

Analogno ovom primeru, kreiran je gadget i za oglas user kontrolu.

13.3.3 Kreiranje oglasa

Može se reći da je oglas centralna tačka ove aplikacije i da je kompletna implementacija usresređena na njega. Oglas je takođe predstavljen u vidu gadget-a, kome se može promeniti pozicija na oglasnoj tabli ili se može ukloniti sa nje.

Krajnji interfejs oglasa prikazan je na slici 2-8.



Slika 2-8

Kao što je već napomenuto, oglasna tabla može da prikazuje obične tekstualne oglase ili oglase u vidu HTML strane. Oglas predstavljen Web stranom omogućava kompletno formatiran izgled oglasa, koji može sadržati sliku, tabelu, link ka nekom drugom oglasu, audio ili video snimak, itd.

XAML kod user kontrole koja predstavlja oglas:

```
<UserControl x:Class="WPFOglasnaTabla.PieceOfPaper.PieceOfPaper"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Height="380" Width="350">
       <Canvas x:Name="CanvasPaper" Width="350" Height="380"
              HorizontalAlignment="Stretch" VerticalAlignment="Stretch">
              <WebBrowser Name="WebBrowserForNotice"
                     Height="380" Width="350" HorizontalAlignment="Stretch"
                     VerticalAlignment="Stretch">
              </WebBrowser>
       </Canvas>
</UserControl>
```

Da bi user kontrola mogla da prikazuje sadržaj Web strane, potrebno je da sadrži Web browser. Web browser je kontrola Windows formi, međutim, sa .NET framework-om 3.5 SP1

uvodi se i u WPF. Iako se sada može koristiti i kao WPF kontrola, Web browser i dalje ima neke od osobina koje su specifične samo za Windows forme i potrebno je dosta umeća prilagoditi ga ostalim kontrolama. Jedan od problema koji se može javiti je taj da je uvek u fokusu i da prekriva sve ostale kontrole koje se nalaze na formi.

Da bi Web browser prikazao neku HTML stranu (u ovom slučaju je to oglas) potrebno je da mu se zada Url Web strane. Url-ovi svih oglasa se čuvaju u jedinstvenoj bazi podataka, kojoj, preko Web servisa, pristupaju svi klijenti. Nakon dopremanja Url-a, na klijentu se može kreirati gagdet za oglas pomoću sledeće metode:

```
private GadgetContainer CreateGadget(Notice notice, int topPosition, int
leftPosition)
        {
            GadgetContainer gadgetContainer = new GadgetContainer();
            Canvas.SetTop(gadgetContainer, topPosition);
            Canvas.SetLeft(gadgetContainer, leftPosition);
            gadgetContainer.Close += OnGadgetClose;

            var paperGadget = new PieceOfPaper();
            paperGadget.WebBrowserForNotice.Navigate(new Uri(notice.Url,
UriKind.Absolute));

            gadgetContainer.Gadget = paperGadget;
            return gadgetContainer;
        }
```

Ključni parametar koji se prosleđuje ovoj metodi je notice, i on predstavlja entitet (business object), koji kao svoje propertije sadrži sve bitne informacije vezane za oglas. Pored notice, ova metoda prima još dva parametra: topPosition i leftPosition.

Već je napomenuto da je canvas jedini panel u WPF-u koji pruža dinamički layout pomoću attached propertia. Upravo to topPosition i leftPosition parametri omogućavaju. Kako bi se optimalno rasporedili oglase na raspoloživoj površini canvas-a, napravljena je metoda, koja na osnovu trenutne veličine panela, rednog broja oglasa i veličine gadget-a određuje pozicije na kojima se postavljaju oglasi.

```
private bool DeterminePosition(int order, ref int topPosition, ref int
leftPosition)
        {
            int gadgetWidth = 420;
            int gadgetHeight = 360;
            double canvesWidth = this._SnapCanvas.ActualWidth == 0 ?
             this._SnapCanvas.Width : this._SnapCanvas.ActualWidth;
            int panelWidth = Convert.ToInt32(Math.Round(canvesWidth, 0));

            int count;
            int maxGadgetsInARow = Math.DivRem(panelWidth, 420, out count);
            if (maxGadgetsInARow == 0)
                return false;

            int row = Math.DivRem(order, maxGadgetsInARow, out count);

            topPosition = row * gadgetHeight + 5;

            leftPosition=(order - row * maxGadgetsInARow) * gadgetWidth +5;
            return true;
        }
```

Metoda DisplayNoticesPanel koristi prethodne dve metode i raspoređuje oglase u okviru panela.

```
private void DisplayNoticesPanel()
        {
            GadgetContainer gadgetToAdd;
            ServiceReference.ServiceClient serviceReference = new
                    ServiceReference.ServiceClient();
            List<GadgetContainer> listGadgetNotice = new List<GadgetContainer>();
            List<Notice> noticeListForPanel =
             serviceReference.SelectNoticeListForWebBrowser();

            int canvasWidth = 600;

            for (int i = 0; i < noticeListForPanel.Count; i++)
            {
                Notice notice = noticeListForPanel[i];
                gadgetToAdd = new GadgetContainer();

                int topPosition = 10;
                int leftPosition = 10;
                bool result =
                          DeterminePosition(i, ref topPosition, ref leftPosition);

                gadgetToAdd = CreateGadget(notice, topPosition, leftPosition);
                listGadgetNotice.Add(gadgetToAdd);

                canvasWidth = topPosition;
            }
            foreach (GadgetContainer gadget in listGadgetNotice)
            {
                _SnapCanvas.Children.Add(gadget);
            }
            canvasWidth += 360 + 10;
            if (canvasWidth > this.GridForPanels.ActualHeight)
                _SnapCanvas.Height = a;
            else
                _SnapCanvas.Height = this.GridForPanels.ActualHeight + 10;
        }
```

Primenom gore navedenih metoda prikazuju se oglasi na SnapCanvas-u. Međutim, može se desiti da broj oglasa bude veći od predviđenog broja oglasa koji mogu da stanu na jednu oglasnu tablu, odnosno da se dinamičkim dodavanjem oglasa deo njih ne postavi na vidljivom delu ekrana. S obzirom na to da canavs podrazumevano nema scroll, svi oglasi koji ne stanu na vidljiv deo ekrana će praktično ostati nevidljivi i nedostupni klijentu. Rešenje ovog problema je uvođenje ScrollViewer-a.

```
<ScrollViewer VerticalScrollBarVisibility="Auto">
    <GadgetLibrarySnapCanves:SnapCanvas x:Name="_SnapCanvas"
              ...
              ...
              ...
</ScrollViewer>
```

Nakon toga pojaviće se scroll i trebalo bi da se canvas raširi u zavisnosti od prostora koji je neophodan za prikaz svih oglasa. Međutim, to se neće desiti samo po sebi, jer canvas je panel koji ne prilagođava svoju veličinu veličini sadržaja. Da bi se to postiglo, mora se

ekspilicitno zadati veličina canvasa u zavisnosti od broja oglasa. Promenljiva koja je zadužena za tu akciju, u metodi DisplayNoticesPanel, jeste canvasWidth, kojoj se menja vrednost nakon svakog prelaska oglasa u novi red.

13.3.4 Kreiranje pokretne trake

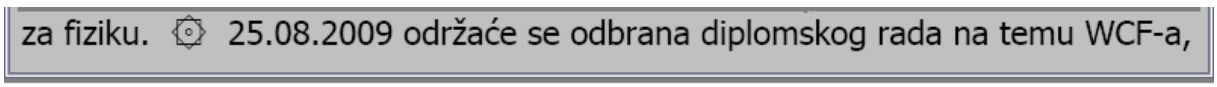
Poslednja komponenta aplikacije je pokretna traka (Khyron) na kojoj se prikazuje tekstualni sadržaj najnovijih oglasa. Za razliku od prethodnih komponenti, koje su kreirane u vidu user kontrola, dizajn pokretne trake je kompletno implementiran u okviru klase Window1.xaml.

```
<!—Pomoćni canvas, neophodan za prikaz pokretne trake-->
<Canvas Name="CanvasForKhyron" Grid.Row="1" Grid.ColumnSpan="2"
      VerticalAlignment="Top"  Height="40" Width="800" ></Canvas>

<Popup x:Name="popupForKhyron" Grid.Row="1" Grid.ColumnSpan="2"
      AllowsTransparency="False" HorizontalAlignment="Left"
      VerticalAlignment="Top"  Height="40" Width="800" Placement="Relative"
      PlacementTarget="{Binding ElementName=CanvasForKhyron}">

<Canvas Background="Silver">
<TextBlock Foreground="Black" Canvas.Left="0" Canvas.Top="0" Height="40"
      Name="textBlockForRotor" Width="219" FontSize="22" TextWrapping="NoWrap"
      VerticalAlignment="Center">
      <TextBlock.RenderTransform>
            <TransformGroup>
                  <ScaleTransform ScaleX="1" ScaleY="1"/>
                  <SkewTransform AngleX="0" AngleY="0"/>
                  <RotateTransform Angle="0"/>
                  <TranslateTransform x:Name="rtTTransform"/>
            </TransformGroup>
      </TextBlock.RenderTransform>
</TextBlock>
</Canvas>
</Popup>
```

Izgled pokretne trake je prikazan na slici 2-9.



Slika 2-9

Za kreiranje pokretne trake, potreban je TextBlock (koji će preuzeti sadržaje svih tekstualnih oglasa), kao i niz specificiranih transformacija neophodnih da bi se tekst kretao u okviru tekstualnog bloka. Međutim, metoda koja omogućava da se tekst kreće je implementirana kao pozadinski kod i u okviru nje je specificiran smer kretanja teksta, kultura, tip animacije (transformacije), brzina kretanja teksta, itd.

```
private void InitializeKhyron()
        {
            double textBoxWidth = 10;

            double pixelXFactor;
            this.popupForKhyron.Width = this.ActualWidth-18;
            double canvaswidth = this.ActualWidth;
            double negXOffSet = 0;
            double fromSecValue = 0;
```

```
        string theText = GetNoticeForRotor();
        double equSlope = 0.022546419;
        double offSetY = 10.96286472;
        double stringSize;

        int textLen = theText.Length;

        System.Globalization.CultureInfo srbCultureInfo;
        Typeface fontTF;
        FormattedText frmmtText;

        if (textLen > 0)
        {
            srbCultureInfo = S
System.Globalization.CultureInfo.GetCultureInfo("sr-Latn-cs");
            fontTF = new Typeface(this.textBlockForRotor.FontFamily,
this.textBlockForRotor.FontStyle, this.textBlockForRotor.FontWeight,
this.textBlockForRotor.FontStretch);
            frmmtText = new FormattedText(theText, srbCultureInfo,
FlowDirection.LeftToRight, fontTF, this.textBlockForRotor.FontSize,
this.textBlockForRotor.Foreground);

            stringSize = frmmtText.Width;

            if (stringSize < 100)
                pixelXFactor = 1.02;
            else
                pixelXFactor = 1.01;

            textBoxWidth = stringSize * pixelXFactor;

            this.textBlockForRotor.Width = textBoxWidth;
            negXOffSet = textBoxWidth * -1;

            fromSecValue = (stringSize * equSlope) + offSetY;

            this.textBlockForRotor.Text = theText;

            Storyboard _sb = new Storyboard();

            Duration durX = new Duration(TimeSpan.FromSeconds(fromSecValue));

            DoubleAnimation daX = new DoubleAnimation(canvaswidth, negXOffSet,
durX);
            daX.RepeatBehavior = RepeatBehavior.Forever;
            Storyboard.SetTargetName(daX, "rtTTransform");
            Storyboard.SetTargetProperty(daX, new
PropertyPath(TranslateTransform.XProperty));
            _sb.Children.Add(daX);
            _sb.Begin(this.textBlockForRotor);
        }
        else
        {
            textBoxWidth = 1;
            stringSize = 0;
        }
    }
```

Ukoliko se pažljivije pogleda XAML kod pokretne trake, može se uočiti Popup kontrola. Postavlja se pitanje, ako je za pokretnu traku neophodan TextBlock i niz transformacija, koja je uloga Popup-a? Pri kreiranju user kontrole za oglas korišćen je Web browser, za koji je rečeno da je uvek u fokusu i da prekriva preostale kontrole na formi. Isti slučaj bi se desio i sa oglasima i pokretnom trakom. Pokretna traka je kontrola implementirana u okviru glavnog prozora dok se oglasi dodaju dinamički pri pokretanju aplikacije. Oglasi sadrže Web browser-e, tako da bi pri skrolovanju njihov sadržaj prelazio preko sadržaja pokretne trake. Da bi se to onemogućilo, pokretna traka je implementirana u vidu Popup-a, za koji se podrazumeva da je uvek u prvom planu i iznad svih preostalih komponenti.

Zaključak

Rešenje obuhvaćeno radom je imalo nameru da pokrije sledeća dva koncepta.

Kao prvo, da se napravi kompleksno rešenje, koje bi integrisalo više različitih tipova aplikacija i tehnologija, na način da se implemetiraju principi skalabilnosti i interoperabilnosti, tako da je omogućeno proširivanje postojećeg rešenja novim komponentama za prikaz informacija. Zatim, da mu se može pristupiti i putem interneta i putem intraneta, kao i da bude dostupno sa više različitih tipova uređaja (display-a koji bi zamenili tradicionalne oglasne table, personalnih računara, mobilnih uređaja). Na taj način je dobijena kompleksna aplikacija, koja je i u svom elementarnom obliku, potpuno funkcionalna i primenljiva.

Drugi cilj rada je bio da se napravi rešenje privlačnog, efektnog i bogatog korisničkog interfejsa. Rad je pokrio osnovne koncepte. Sa jedne strane, obuhvatio je više različitih tipova aplikacija (Windows i Web aplikacija, Web servis). Rešenje je implementirano korišćenjem .NET framework-a nad MS SQL serverom kao bazom podataka, ali na takav način, da se bilo koja od komponenata može izvući i zameniti drugom, implementiranom u drugoj tehnologiji (PHP Web sajt, MySql baza podataka, JAVA Web servis…). Sa druge strane, sama klijentska aplikacija je razvijena na način da prezentuje mogućnosti koje pruža WPF. Implementirano, naravno, ne iscrpljuje sve mogućnosti WPF-a, već samo nagoveštava bogatstvo korisničkog interfejsa koji WPF može pružiti.

Literatura i reference

- Victor Gaudioso, *Foundation Expression Blend 2: Building Applications in WPF and Silverlight*, Friendsoft, 2008.
- Andrew Troelsen, *Pro C# 2008 and the .NET 3.5 Platform*, Springer-Verlag New York Inc Hardcover, Fourth Edition, 2008.
- Matthew MacDonald, *Pro WPF in C# 2008: Windows Presentation Foundation with .NET 3.5*, Apress, 2008.
- Jack Xu, *Practical WPF Graphics Programming*, Unicad, 2007.
- Chris Sells, Ian Griffiths, *Programming WPF* , O'Reilly, 2007.
- www.codeproject.com
- www.link-elearning.com/kurs-WPF-programiranje_244_4
- www.google.com