UNIVERZITET U KRAGUJEVCU
PRIRODNO-MATEMATIČKI FAKULTET
Institut za matematiku i informatiku

DIPLOMSKI RAD

PLAYLIST KONTROLA ZA ELEKTRONSKU OGLASNU TABLU

Student:
Jelena Bojović 04/04

Mentor:
doc. dr Boban Stojanović

Kragujevac
Maj 2011.

Sadržaj

1. Uvod

Oglasne table su uvek bile mesta gde su se ljudi okupljali kako bi dobili najnovije informacije. U svim savremenim institucijama kojima profit u velikoj meri zavisi od pouzdanosti i preglednosti informacija, već postoje elektronske oglasne table na kojima se, za određeni vremenski interval smenjuju stare informacije novim, tako da korisnik kome je potrebna informacija, vrlo brzo i lako može doći do nje.

Pored savremenih kompanija, oglasne table predstavljaju neophodno sredstvo za rad i na univerzitetima. Raspored ispita, predavanja, rezultati ispita kao i ostala potrebna obaveštenja, na vreme moraju dospeti do studenata. Pošto su ove informacije brojne, korišćenjem metoda klasičnih oglasnih tabli papiri bivaju zalepljeni jedni preko drugih, javljaju se potečkoće u pronalaženju prave informacije, zastarele vesti se često nalaze na tablama… Svi ovi razlozi doprineli su uvođenju elektronskih oglasnih tabli i na univerzitetima. Informacije na njima su preglednije, lako uočljive, nakon isteka neke informacije ona se automatski uklanja sa table… Pored toga, ovaj vid oglašavanja doprinosi očuvanju životne sredine jer se ne troši papir.

Kako bi rešio postojeće probleme klasičnih oglasnih tabli i svojim studentima i radnicima pružio savremeni način rada i oglašavanja, Institut za matematiku i informatiku Prirodno-matematičkog fakulteta u Kragujevcu pokrenuo je projekat izrade savremene elektronske oglasne table. Ideja za ovaj projekat nije nova - jedan vid savremenog komuniciranja postoji na aktivnom sajtu fakulteta preko kojeg korisnici redovno dobijaju najsvežije informacije. Međutim, odlukom da se oglasna tabla na fakultetu zameni elektronskom koja će biti prikazana na LCD ekranu, dovela je do nekoliko problema ukoliko bi se koristilo postojeće rešenje. Nemogućnost korisnika da klikne na neki oglas i na taj način ga prikaže, prvi je faktor koji je uticao na unapređenje postojeće table. To je nametnulo ideju da se način prikaza oglasa automatizuje. Ta ideja vodi u smeru novijih tehnologija koje pružaju neograničene mogućnosti za izradu korisničkog interfejsa.

Kao pogodna za izradu animacija i bogatog izgleda, izabrana je Windows Presentation Foundation (WPF) tehnologija. Zamišljeno je da srž oglasne table poseduje deo (Playlist-u) gde će se smenjivati podaci o oglasima koji će biti prikazani, i deo za prikaz sadržaja tih oglasa u odgovarajućem vremenskom trenutku. Kako je ovaj projekat još uvek u izradi, konačan izgled nije tačno definisan, tako da će se pored ovih glavnih funkcionalnosti naći još neke korisne funkcionalnosti i informacije.

Cilj ovog diplomskog rada je izrada Playlist kontrole za elektronsku oglasnu tablu u kojoj će se prikazivati podaci o oglasima i koja će kontrolisati prikaz sadržaja oglasa tj. voditi računa o tome kada će se koji oglas prikazati.

Pre početka priče o ideji i izradi Playlist kontrole, neophodno je upoznati se sa tehnologijom u kojoj je rađena implementacija iste.

2. Windows Presentation Foundation (WPF)

2.1. Uvod u WPF

Windows Presentation Foundation (WPF) je prezentacioni (User Interfaces) podsistem Windows-ovog programskog modela koji se koristi za izradu vizuelno bogatih klijentskih aplikacija.

WPF kombinuje aplikacije korisničkog interfejsa, 2D i 3D grafike, dokumente, animacije i multimediju u jedinstveni paket koji je uveden sa verzijom .NET framework 3.0. Koristi vektorsku grafiku koja sa modernim grafičkim karticama i procesorima čini korisnički interfejs bržim i skalabilnim i pruža nezavisnost od rezolucije monitora.

Slika 1. Šematski prikaz komponenata WPF tehnologije

Jedna od važnijih karakteristika WPF-a je ta što odvaja prikaz korisničkog interfejsa od definisanja njegovog ponašanja. Prikaz je specificiran XAML-om (Extensible Application Markup Language), dok se ponašanje implementira programskim jezicima kao što su C# ili Visual Basic. Ova dva dela su povezana preko databinding-a, događaja (events) i komandi. Ovo razdvajanje prikaza i ponašanja praćeno je nizom pogodnosti:

- prikaz i ponašanje su labavo povezani
- dizajneri i programeri mogu raditi na odvojenim modelima
- alati za grafički dizajn rade sa jednostavnim XML dokumentima umesto kodiranja
- jasno definisana odvojenost prikaza i ponašanja omogućava veoma laku promenu izgleda kontrole.

2.2. XAML – pregled

Extensible Application Markup Language (XAML) je jezik baziran na XML-u i koristi se za kreiranje i inicijalizaciju .NET objekata. Iako je originalno kreiran za WPF, može se koristiti i za kreiranje drugih vrsta objekata. Danas se XAML koristi za kreiranje korisničkog interfejsa u WPF-u, Silverlight-u, definisanje workflows u WF-u (Windows Workflow Foundation) kao i za elektronske papire u XPS standardima.

Sve što se može uraditi u XAML-u, može se uraditi i u kodu. XAML je samo drugi način za inicijalizaciju objekata, pa se WPF može koristiti i bez XAML-a. Međutim, implementacija korisničkog interfejsa u XAML-u ima svojih prednosti:

- XAML kod je kraći i čitkiji
- grafički alati kao što je Expression Blend zahtevaju XAML kao izvorni kod
- XAML omogućava dizajnerima da kreiraju aplikaciju bez poznavanja koncepata programiranja

2.3. WPF alati

Microsoft obezbeđuje dva osnovna razvojna alata za izradu WPF aplikacija.

Visual Studio napravljen za programere. Sa verzijom 2008 uključen je i grafički dizajner za WPF. Instaliranjem add-on-a za verziju 2005 omogućen je razvoj WPF aplikacija i u ovoj verziji.

Expression Blend je namenjen dizajnerima. On je deo Expression Studio, novog Microsoft-ovog alata kreiranog za dizajniranje. Iako je Visual Studio dobar za kodiranje i editovanje XAML-a, on ima slabu podršku za sve grafičke stvari poput gradienta, animacija itd. Ovo je tačka gde nastupa Expression Blend koji pokriva sve nedostatke Visual Studio-a kao sto su gradijenti, animacije, 3D grafika, resursi…

2.4. Osnovni koncepti WPF – a

2.4.1. Logičko i vizuelno drvo

Elementi korisničkog interfejsa u WPF-u su hijerarhijski povezani. Ova relacija se naziva Logičko drvo. Jedan element logičkog drveta sadrži više vizuelnih elemenata. To drvo se naziva Vizuelno drvo.

2.4.2. Dependency Property

Predstavlja osobinu koja zavisi od neke druge osobine, tj. od različitih uslova koji determinišu njenu vrednost u određenom trenutku. Osnovna razlika u odnosu na neki osnovni .NET property je u tome što se osnovni property čita iz nekog privatnog člana klase dok se vrednost *DependencyProperty*-a rešava dinamički pozivom metode *GetValue* koja je izvedena iz *DependencyObject*-a.

2.4.3. Routed events

Predstavljaju događaje koji mogu da se kreću uz ili niz vizualno drvo u zavisnosti od njihove osobine zvane *RoutingStrategy*. Strategija može biti *bubble*, *tunnel* ili *direct*.

- *Tunneling* predstavlja događaj elementa koji se kreće niz drvo dok ne dođe do izvornog elementa ili dok nije stopiran nekim markiranim događajem. Po konvenciji se naziva *Preview*... i pojavljuje se pre odgovarajućeg bubbling događaja.
- *Bubbling*. Podignuti događaj na izvornom elementu navigira uz vizuelno drvo dok ne dodje do korenog elementa ili dok nije prekinut nekim markiranim događajem. Bubbling događaj je podignut posle tunneling događaja.
- *Direct* je događaj koji je podignut i mora biti završen na istom izvornom elementu. Njegovo ponašanje je isto kao kod normalnih .NET događaja.

2.4.4. Komande

Dok su događaji vezani za detalje o specifičnoj akciji korisnika, komande predstavljaju akcije koje su nezavisne od interakcije korisnika.

2.5. Vrste WPF aplikacija

WPF ima dva glavna aplikaciona modela: *standalone* i *browser*. Sa druge strane, poseduje i dva tipa navigacije: *menu-driven*, koji je korišćen u tradicionalnim Windows aplikacijama, i *link-driven* koji je osnova za web aplikacije.

- *Standalone* aplikacije
  Ova vrsta aplikacija je najbliža klasičnim Windows aplikacijama i instalira se lokalno na računaru. One imaju potpuni pristup resursima računara.
- *Navigation* aplikacije
  Ovo je vrsta aplikacije koja imitira web aplikaciju ali je i dalje kreirana kao standalone.
- *Browser* aplikacije
  Predstavljaju aplikacije koje se izvršavaju u okviru web pretrazivača. Ove aplikacije mogu da se objave na web ili intranet serverima. Zbog zaštite, nije im omogućen potpuni pristup računarskim resursima.

2.6. Kreiranje korisničkog interfejsa

WPF tehnologija omogućava alate za izgradnju dinamičkog i elegantnog korisničkog interfejsa. Postoje tri osnovna tipa kontrola: individual i content kontrole, kao sto su dugmići ili tekstualna polja, koja sadrže jedan element; item kontrole, kao što je lista, koje sadrže grupu sličnih kontrola; i layout kontrole, kao što su grid ili paneli, sa kojima se kreira izgled i pozicija elemenata u korisničkom interfejsu.

2.6.1. Content kontrole

Većina kontrola koja se koristi u WPF aplikacijama pripada ovoj grupi. One su izvedene iz *ContentControl* klase i sadrže jedan ugrađeni element kao svoj sadržaj. Taj ugrađeni element može biti bilo kojeg tipa i može biti dodeljen ili dobijen iz koda preko *Content* property. Tip *Content* property je *Object* pa može prihvatiti bilo koji objekat kao sadržaj. Kako se taj sadržaj prikazuje, zavisi od tipa objekta u *Content*-u. Za elemente koji nisu izvedeni iz *UIElement*, poziva se *ToString* metoda i rezultat u vidu stringa predstavlja sadržaj kontrole.

```
<Button Height="23" Margin="36,0,84,15" Name="button2"
        VerticalAlignment="Bottom">This is the content string
</Button>
```

Elementi izvedeni iz *UIElement* su u sadržaju kontrole. Sledeći primer prikazuje kako da se kreira dugme koje sadrži sliku kao svoj sadržaj.

```
<Button Margin="20,20,29,74" Name="button1">
    <Image Source="C:\Pictures\SomeImage.jpg"/>
</Button>
```

Iako content kontrola može imati samo jedan element u svom sadržaju, ne postoji ograničenje u broju elemenata po dubini. Na primer, moguće je da sadržaj kontrole ima layout kontrolu koja u sebi sadrži više UI elementata.

```
<Button Margin="20,20,-12,20" Name="button1">
    <StackPanel>
        <Image Source="C:\Pictures\SomeImage.jpg"></Image>
        <TextBlock>This is a Humpback Whale</TextBlock>
    </StackPanel>
</Button>
```

Neke od osnovnih Content kontrola su:

2.6.1.1. Label

Predstavlja jednu od najjednostavnijih WPF kontrola. Uobičajeno se koristi samo za prikaz sadržaja. Tipično korišćenje *Label* kontrole je:

```
<Label Name="label1">This is a Label</Label>
```

2.6.1.2. Button

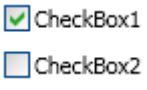
Kontrola je dizajnirana da prihvati klik korisnika koji omogućava korisniku da napravi neki izbor, zatvori neki prozor sa obaveštenjem, ili neku akciju definisanu od strane programera. Hendle-ovanjem *Click* event-a izvršava se akcija specificirana kodom programera.


```
<Button Name="btn" Content="Button" Height="23" Width="50"
        Click="btn_Click"></Button>
```

Button poseduje dve značajne osobine korisne prilikom izrade korisničkog interfejsa: *IsDefault* i *IsCancel*. Kada je *IsDefault* property postavljena na *True, Click* događaj je podignut pritiskom tastera *Enter*. Slično tome, postavljanjem vrednosti *IsCancel* property na *True, Click* događaj se aktivira pritiskom taster *Esc*.

2.6.1.3. Checkbox

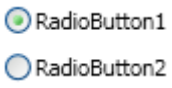
Ova kontrola je izvedena iz *ButtonBase* klase. Omogućava korisniku da selektuje kada je neka od ponuđenih opcija uključena a kada nije. Da li je checkbox selektovan ili ne određuje se preko *IsChecked* property. *IsChecked* je tipa *Boolean?*. Pošto je izveden iz *ButtonBase* klase, *Click* event se aktivira kad god je checkbox selektovan ili deselektovan od strane korisnika.




```
<CheckBox Margin="5" Click="CheckBox_Click"
      IsChecked="True">CheckBox1</CheckBox>
<CheckBox Margin="5" Click="CheckBox_Click">CheckBox2</CheckBox>
```

2.6.1.4. RadioButton

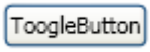
Kao i *Checkbox*, i *RadioButton* je izveden iz *ButtonBase* klase. Ove kontrole se koriste u grupi omogućavajući korisniku da selektuje samo jednu opciju iz grupe. Osobina pojavljivanja u grupi omogućava da se selekcijom jednog radio button-a iz grupe ostale selekcije ponište, čime i dobijamo selekciju samo jedne opcije u datom trenutku. Klik na radio button aktivira *Click* event, koji omogućava da se definše reakcija na korisnikov izbor.




```
<RadioButton Margin="5" Click="RadioButton_Click"
      IsChecked="True">RadioButton1</RadioButton>
<RadioButton Margin="5" Click="RadioButton_Click">RadioButton2
</RadioButton>
```

2.6.1.5. ToogleButton

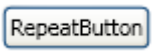
Još jedna specifična vrsta dugmeta koja ima osobinu da nakon klika ostane u kliknutom (pritisnutom, donjem položaju) sve dok korisnik ponovo ne klikne na njega. Za razliku od običnog dugmeta, pored osnovnog *Click* event-a, *ToogleButton* aktivira i dva *Click* event-a: *Checked* kada korisnik klikne na dugme pri cemu dugme ostaje pritisnuto, i *Unchecked* kada korisnik ponovno klikne na dugme vraćajući ga u prvobitni položaj.




```
<ToggleButton Content="ToogleButton" Margin="5"
              Checked="ToggleButton_Checked"
              Unchecked="ToggleButton_Unchecked">
</ToggleButton>
```

2.6.1.6. RepeatButton

Ova vrsta dugmeta poseduje osobinu da se *Click* event aktivira sve dok korisnik drži dugme pritisnuto (kliknuto).




```
<RepeatButton Content="RepeatButton" Margin="5"
      Click="RepeatButton_Click"></RepeatButton>
```

2.6.2. Ostale kontrole

Kontrole o kojima će sada biti reč nemaju *Content* property i ograničene su, u odnosu na content kontrole, u prikazu sadržaja.

2.6.2.1. TextBlock

Ova kontrola je specijalizovana za prikaz teksta.

```
<TextBlock>Here is some text</TextBlock>
```

Da bi ovoj kontroli pristupili iz koda, moramo definisati *Name* property.

```
<TextBlock Name="txtSomeText">Here is some text</TextBlock>
```

Sada možemo u kodu promeniti tekst ili bilo koju drugu osobnu na sledeći način:

```
txtSomeText.Text = "Here is the changed text";
```

Font teksta će biti isti kao i definisani font kontrole u kojoj se nalazi *TextBlock*. Ukoliko želimo različita podešavanja za *TextBlock* možemo definisati odgovarajuće osobine fonta:

```
<TextBlock FontFamily="Batang" FontSize="12"
         FontStyle="Italic" FontWeight="Bold"
         FontStretch="Normal">Here is some text
</TextBlock>
```

2.6.2.2. Image

Image kontrola prikazuje slike. Najvažnija osobina *Image* kontrole je *Source*. Ona je definisana *System.Windows.Media.ImageSource* klasom u kodu, dok je u XAML-u određena *Uniform Resource Identifier* (URI), tj. putanjom sa koje se učitava slika.

```
<Image Source="C:\Pictures\SomeImage.jpg"/>
```

URI može predstavljati lokaciju na hard disku ili na nekoj Web strani.

Još jedna bitna osobina *Image* kontrole je *Stretch* koja određuje kako će slika biti prikazana. Ona može imati sledeće vrednosti:

None Slika je prikazana u svojoj originalnoj veličini. Ako je prostor za prikaz slike manji od veličine slike, ona će biti presečena kako bi popunila dozvoljeni prostor.

Fill Veličina slike je promenjena (uvećana ili umanjena) kako bi cela slika popunila dozvoljeni prostor za njen prikaz.

Uniform Dimenzije slike su promenjene kako bi ona popunila prostor za prikaz zadržavajući proporcije između visine i širine. Ukolliko se dobijene dimenzije slike razlikuju od dozvoljenog prostora, dolazi do pojavljivanja praznina kod neiskorišćenog prostora.

UniformToFill Dimenzije slike su promenjene kako bi ona popunila prostor za prikaz zadržavajući proporcije između visine i širine. Ukoliko se dobijene dimenzije slike razlikuju od dozvoljenog prostora, slika se odseca kako bi popunila prostor.

2.6.2.3. TextBox

Kontrola koja omogućava prikaz i izmenu teksta tako što korisnik može ukucavati tekst sa tastature. Ovaj tekst je dostupan u kodu preko *Text* property. *TextBox* se može koristiti samo za prikaz teksta pomoću osobine *IsReadOnly* postavljanjem njene vrednosti na *True*.

Tekst se prikazuje u jednoj liniji. Da bi se omogućio prikaz teksta u više linija dovoljno je dodeliti vrednost *Wrap* osobini *TextWrapping.*

Ukoliko tekst ne može stati u prostor određen za njegov prikaz, *TextBox* dodaje klizače kako bi se mogao pročitati ceo tekst. Klizači se omogućavaju preko *VerticalScrollBarVisibility* property postavljanjem njene vrednosti na *Auto* ili *Visible.*

```
<TextBox Name="txt" IsReadOnly="True" Text="Here is some text"
         TextWrapping="Wrap" VerticalScrollBarVisibility="Visible"/>
```

2.6.3. Item kontrole

Kreirane su da sadrže više elemanta. Kao i content kontrole, item kontrole nemaju ograničenja na vrste sadržaja koje će prikazati. Samim tim, item kontrola može prikazati jednostavnu listu string-ova, ili listu koja uključuje kompleksnije elemente, na primer listu *Checkbox* kontrola.

2.6.3.1. ListBox

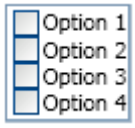
Predstavlja najjednostavniju item kontrolu. Kao što i samo ime kontrole kaže, kreirana je za prikaz liste elemenata. Ona prikazuje listu *ListBoxItem* kontrola koje su content kontrole. Najjednostavniji način za popunjavanje *ListBox* kontrole je dodavanje njenih elemenata u XAML-u.


```
<ListBox>
     <ListBoxItem>Item1</ListBoxItem>
     <ListBoxItem>Item2</ListBoxItem>
     <ListBoxItem>Item3</ListBoxItem>
</ListBox>
```

Lista automatski smešta sadržaj u *Stack* panel i dodaje vertikalne klizače ukoliko je lista duža od raspoloživog prostora na kontroli.

Osnovno ponašanje liste dozvoljava selekciju jednog njenog elementa. Indeks selektovanog elementa se dobija preko *SelectedIndex* dok se ceo element dobija preko *SelectedItem* osobine. Da bi se omogućilo selektovanje više elemata koristi se *SelectionMode* property.

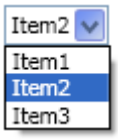
Iako se najčešće koristi za prikaz *ListBoxItem* kontrola, ona može prikazati listu elemanata bilo kog tipa. Na primer, možemo kreirati listu *Checkbox* kontrola.




```
<ListBox>
     <CheckBox Name="chb1">Option 1</CheckBox>
     <CheckBox Name="chb2">Option 2</CheckBox>
     <CheckBox Name="chb3">Option 3</CheckBox>
     <CheckBox Name="chb4">Option 4</CheckBox>
</ListBox>
```

2.6.3.2. ComboBox

Kontrola čije je ponašanje slično listi. Sadrži listu kontrola od kojih svaka može biti bilo kojeg tipa. Razlika u odnosu na *ListBox* kontrolu je u načinu prezentovanja elemanta. *ComboBox* kontrola se pojavljuje u vidu drop-down liste. Kao i kod liste, može se dobiti referenca na selektovani element preko *SelectedItem* ili indeks selektovanog elementa preko *SelectedIndex.* Kada je element selektovan, njegova tekstualna reprezentacija je prikazana u *ComboBox* kontroli.




```
<ComboBox>
      <ComboBoxItem>Item1</ComboBoxItem>
      <ComboBoxItem>Item2</ComboBoxItem>
      <ComboBoxItem>Item3</ComboBoxItem>
</ComboBox>
```

2.6.4. Layout kontrole

WPF nudi podršku za nekoliko stilova prilikom kreiranja izgleda aplikacije. Nekoliko specijalizovanih kontrola omogućava kreiranje izgleda i panela koji mogu biti ugnježđeni jedan u drugi kako bi kreirali kompleksni korisnički interfejs sa kompleksnim ponašanjem.

2.6.4.1. Osobine Layout kontrola

U sledećoj tabeli su prikazane osnovne zajedničke osobine layout kontrola na osnovu kojih se određuje konačna pozicija kontrole u zavisnosti od njene konkretne pozicije i pozicije u njenom sadržaocu.

Osobina	Opis
FlowDirection	Pravac teksta ili drugog UI elementa u okviru bilo kojeg elementa roditelj koji definiše njegovo ponašanje
Height	Visina kontrole
HorizontalAlignment	Horizontalno poravnanje u odnosu na element roditelj
HorizonalContentAlignment	Horizontalno poravnanje sadržaja kontrole
Margin	Udaljenost od svake ivice kontrole do ivica kontrole roditelja
MaxHeight	Maksimalna moguća visina kontrole
MaxWidth	Maksimalna moguća širina kontrole
MinHeight	Minimalna moguća visina kontrole
MinWidth	Minimalna moguća širina kontrole
VerticalAlignment	Vertikalno poravnanje u odnosu na element roditelj
VerticalContentAlignment	Vertikalno poravnanje sadržaja kontrole
Width	Širina kontrole

2.6.4.2. Layout paneli

Grid

Grid je najčešće korišćen panel za kreiranje korisničkog interfejsa u WPF-u. On omogućava definisanje kolona i redova čime se elementi grida prikazuju nalik ćelijama u tabeli. Kolone i redovi se definišu pomoću kompleksnih property-ja *ColumnDefinitions* i *RowDefinitions*:

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="125"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="80"/>
        <ColumnDefinition Width="*"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
</Grid>
```

Kolone i redovi mogu biti fiksne veličine kao što je definisano u prvoj koloni i prvom redu, ili se njihova veličina (kako širina tako i visina) mogu menjati u zavisnosti od prostora koji im je raspoloživ za prikaz. Ovo se postiže korišćenjem * koja označava da će red ili kolona zauzeti sav raspoloživi prostor. U navedenom primeru, prilikom iscrtavanja redova, prvo se odvaja fiksna veličina za prvi red a zatim se preostali slobodan prostor dodeljuje drugom redu. Što se kolona tiče, prvo se odvaja prostor za prvu kolonu, a zatim se preostali slobodan prostor deli na dva dela i svako od njih dodeljuje drugoj, odnosno trećoj koloni.

Grid Attached properties

Grid kontrola obezbeđuje attached properties svim kontrolama u svom sadržaju. Kontrola se može pozicionirati u određenu kolonu ili red definisanjem attached property-a *Grid.Column* i *Grid.Row*:

```
<Button Name="btn" Grid.Row="0" Grid.Column="1">Button</Button>
```

Ponekad je povoljno imati kontrolu koja zauzima više od jednog reda ili kolone. Ovo se postiže definisanjem *Grid.ColumnSpan* ili *Grid.RowSpan* osobina:

```
<Button Name="btn" Grid.Row="0" Grid.Column="1"
    Grid.ColumnSpan="3" Grid.RowSpan="2">Button</Button>
```

StackPanel

Pruža jednostavni model izgleda. Kontrole u *StackPanel*-u su smeštene u redu kojim su dodavane, ispod ili desno od prethodne kontrole. Da li će se nova kontrola dodati ispod poslednje dodate ili desno od nje, definiše *Orientation* property. Može imati vrednosti *Horizontal* ili *Vertical.* Kao što je napomenuto, prilikom horizontalnog orijentisanja kontrola u *StackPanel*-u, kontrole seređaju s leva na desno. Postavljanjem vrednosti *FlowDirection* na *RightToLeft*, dobija se suprotan smer prikaza kontrola.

WrapPanel

Ovaj panel ređa kontrole u jednom redu sve dok ima raspoloživog prostora a zatim kreira novi red u kojem nastavlja smeštanje preostalih kontrola. Ovaj postupak se nastavlja sve dok se ne smeste sve kontrole koje se nalaze u *WrapPanel*-u.

DockPanel

Vrši postavljanje svojih kontrola uz ivice *DockPanel*-a. Ovo se vrši pomoću attached property *Dock*:

```
<Button DockPanel.Dock="Top">Button</Button>
```

DockPanel.Dock property može imati jednu od četiri vrednosti: *Top*, *Bottom*, *Left* i *Right* koje ukazuju da li će kontrola biti postavljena uz gornju, donju, levu ili desnu ivicu dock panela.

DockPanel poseduje *LastChildFill* property koja može imati vrednosti *True* ili *False.* Kada je postavljena na *True*, poslednja dodata kontrola zauzuima sav preostali prostor.

Redosled kojim se kontrole dodaju u dock panel je odlučujući u definisanju izgleda. Prva dodata kontrola zauzima sav slobodan prostor kod ivice kod koje je pozicionirana. Sledeća dodata kontrola zauzima preostali prostor uz ivicu koja joj je određena.

Canvas panel

Ovaj panel omogućava apsolutno pozicioniranje svojih elemenata. Kontrole se pozicioniraju sa četiri osnovne attach properties: *Canvas.Top, Canvas.Bottom, Canvas.Right* i *Canvas.Left.* Vrednost svake osobine ukazuje na udaljenost od određene ivice *CanvasPanel*-a do odgovarajuće ivice kontrole.

```
<Canvas>
    <Button Canvas.Top="20" Canvas.Left="30">Button</Button>
</Canvas>
```

Može se definisati samo jedna horizontalna ili samo jedna vertikalna attached property. Dakle, nije dozvoljeno definisanje i *Canvas.Top* i *Canvas.Bottom.*

2.7. Rad sa grafikom

WPF obezbeđuje izuzetnu podršku za kreiranje i prikazivanje grafičkih elemenata.

2.7.1. Brushes

Brushes su osnovni objeki za iscrtavanje korisničkog interfejsa. Svaka kontrola sadrži neke property-je koje prihvataju *Brushes* objekte. Neke od ovih osobina su prikazane u tabeli:

Osobina	Opis
Background	Boji pozadinu kontrole.
BorderBrush	Boji okvir kontrole
Fill	Boji unutrašnjost nekog grafičkog oblika
Foreground	Boji sadržaj kontrole
Stroke	Boji ivice nekog grafičkog oblika

2.7.1.1. SolidColorBrush

SolidColorBrush je najjednostavnija od svih *Brush* klasa. *SolidColorBrush* objekat, kao što i ime kaže, iscrtava samo jednu boju. Nekoliko boja je dostupno u *Brushes* klasi. Njima se može pristupiti na sledeći način:

```
Brush brush;
brush = Brushes.LightBlue;
```

U XAML-u boja se dodeljuje navođenjem njenog imena:

```
<Button Background="Green"></Button>
```

Za definisanje boje u XAML-u može se koristiti i heksadecimalna notacija. Korišćenjem ove notacije definiše se 8 heksadecimalnih brojeva koji definišu boju. Prvi par određuje propuštanje (*Opacity*), drugi par definiše crveni kanal, treći definiše jačinu zelenog kanala, dok se poslednji par odnosi na plavi kanal. Prethodnik heksadecimalnom broju je #.

```
<Button Background="#FFF7112A"></Button>
```

SolidColorBrush objekat se može kreirati i definisanjem svakog kanala posebno:

```
<Button>
      <Button.Background>
          <SolidColorBrush>
              <SolidColorBrush.Color>
                  <Color A="255" R="0" G="0" B="255"/>
              </SolidColorBrush.Color>
          </SolidColorBrush>
      </Button.Background>
</Button>
```

U kodu, može se koristiti *Color.FromArgb* metoda za definisanje individualnih kanala:

```
SolidColorBrush brush;
brush = new SolidColorBrush(Color.FromArgb(255, 0, 255, 0));
```

2.7.1.2. LinearGradientBrush

Omogućava kreiranje objekata sa dve ili više boja, koristeći prelivanja između njih i omogućavajući nekoliko efekata.

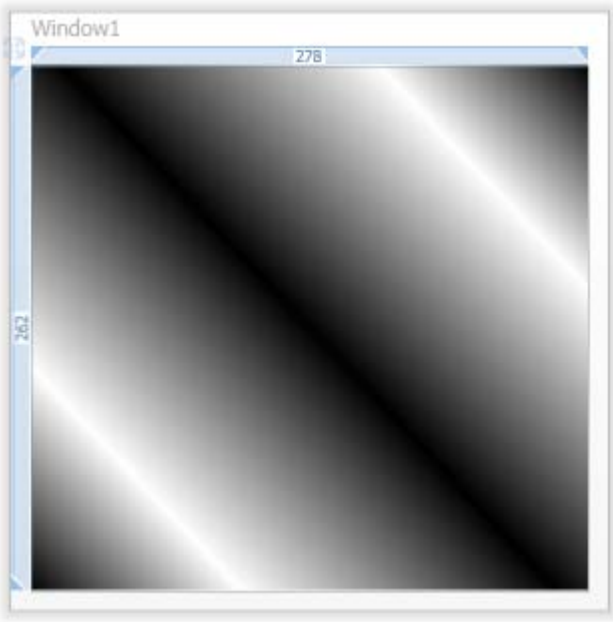

```
<Grid>
   <Grid.Background>
       <LinearGradientBrush>
           <GradientStop Color="Black"
                Offset="0"/>
           <GradientStop Color="White"
                Offset="1"/>
       </LinearGradientBrush>
   </Grid.Background>
</Grid>
```

Slika 2: Primena *LinearGradientBrush*

LinearGradientBrush objekat koristi sistem koordinata da odredi kako će se iscrtavati prelivi. Koordinatni sistem je zasnovan na pravougaoniku koji označava površinu koja će biti obojena. Gornji levi ugao pravougaonika označen je koordonatama (0, 0), a donji desni ugao koordinatama (1, 1). Ove koordinate su relativne u odnosu na veličinu površine za bojenje i ne predstavljaju piksele.

Svaki *LinearGradientBrush* sadrži kolekciju *GradientStop* objekata. *GradientStop* objekat obezbeđuje dve značajne osobine: *Color* i *Offset. Color* property definiše boju dok *Offset* predstavlja broj koji se odnosi na tačku u koordinatnom sistemu gde je definisana boja čista i ne meša se sa ostalim bojama. U prethodnom primeru, gornji levi ugao je crn, donji desni ugao je beo, a ostatak pravougaonika predstavlja mešanje crne i bele boje.

Gradient niz kojim su boje pomešane dešava se na liniji koja počinje u početnoj a završava se u krajnjoj tački. Ukoliko se ne navede drugačije, početna tačka ima koordinate (0, 0) a krajnja (1, 1), koje kreiraju dijagonalni gradijent. Početna i krajnja tačka definišu se preko *StartPoint* i *EndPoint.*



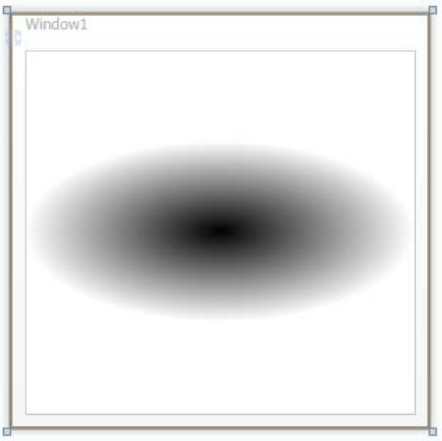
Slika 3: Primena *LinearGradientBrush* koji sadrži više *GradientStop* objekata

```
<Grid>
   <Grid.Background>
      <LinearGradientBrush StartPoint="0,1" EndPoint="1,0">
          <GradientStop Color="Black" Offset="0"/>
          <GradientStop Color="White" Offset=".2"/>
          <GradientStop Color="Black" Offset=".5"/>
          <GradientStop Color="White" Offset=".8"/>
          <GradientStop Color="Black" Offset="1"/>
      </LinearGradientBrush>
   </Grid.Background>
</Grid>
```

Linija koja definiše gradijent ne mora početi i završiti se u uglovima koordinatnog sistema. Može se, na primer, kreirati *LinearGradientBrush* sa *StartPoint* (.3, .3) i *EndPoint* (.7, .7), ili sa bilo kojim vrednostima između 0 i 1.

2.7.1.3. RadialGradientBrush

RadialGradientBrush objekat je vrlo sličan *LinearGradientBrush* objektu. Razlika je u tome što gradijent ne predstavlja pravu liniju već definiše centar koncentričnih krugova kojim se boje prelivaju. Takođe sadrži kolekciju *GradientStop* objekata. Centar gradijenta je definisan pomoću *RadialGradientBrush.GradientOrigin* property (.5, .5 je početna vrednost). Krajnji krugovi su karakterisani osobinama *RadiusX* i *RadiusY. RadiusX* definiše udaljenost poslednjeg kruga u vertikalnoj dimenziji, dok *RadiusY* definiše udaljenost poslednjeg kruga u horizontalnoj dimenziji. *GradientStop* objekti, kao i kod *LinearGradientBrush,* određuju u kojim tačkama je boja čista a gde se meša sa ostalim bojama.




```
<Grid>
   <Grid.Background>
       <RadialGradientBrush Center=".5, .5"
             RadiusX=".5" RadiusY=".25">
           <GradientStop Color="Black"
                   Offset="0"/>
           <GradientStop Color="White"
                   Offset="1"/>
       </RadialGradientBrush>
   </Grid.Background>
</Grid>
```

Slika 4: Primena *RadialGradientBrush*

2.7.1.4. ImageBrush

ImageBrush objekat omogućava bojenje objekata korišćenjem slike kao izvora boje.

```
<Grid>
   <Grid.Background>
      <ImageBrush
          ImageSource="C:\SomeImage.jpg">
      </ImageBrush>
   </Grid.Background>
</Grid>
```

ImageBrush poseduje *Strech* property koja može imati iste vrednosti kao i *Image* kontrola.

2.7.1.5. VisualBrush

Sličan je *ImageBrush* objektu, ali umesto slike kao izvora koristi neki vizuelni element, kao što je WPF kontrola ili bilo koji element izveden iz klase *Visual.*

```
<VisualBrush Visual="{Binding ElementName=btn}">
</VisualBrush>
```

2.7.2. Opacity Mask

U WPF-u većina kontrola poseduje property *Opacity* koja celu kontrolu čini providnom ili delimično providnom. *OpacityMask* pruža mnogo veće mogućnosti. Ona omogućava kreiranje providnih regiona kontrole ili delimično providnih. Delovi kontrole koji nisu prekriveni maskom biće providni.

OpacityMask property prihvata objekat tipa *Brush.* Alfa kanal boje određuje gde će biti ta providnost. Korišćenje *OpacityMask* sa *SolidColorBrush* nema mnogo smisla. Ako postavimo providnu boju u kanalu za transparentnost heksadecimalne notacije boje, cela kontrola će nestati (prva definicija dugmeta), dok će kontrola biti prikazana ako koristimo neku boju koja nije providna (druga definicija dugmeta). Ostali kanali boje (crveni, zeleni i plavi) nisu bitni za definisanje transparentnosti pa se ignorišu, čime se uloga *OpacityMask* osobine gubi. Korišćenje *Opacity* dovodi do istih rezultata.

```
<Button FontSize="14" FontWeight="Bold" Height="100" Margin="5">
    <Button.OpacityMask>
        <SolidColorBrush Color="#00000000"></SolidColorBrush>
    </Button.OpacityMask>
    <Button.Content>A Partially Transparent Button</Button.Content>
</Button>

<Button FontSize="14" FontWeight="Bold" Height="100" Margin="5">
    <Button.OpacityMask>
        <SolidColorBrush Color="#FF000000"></SolidColorBrush>
    </Button.OpacityMask>
    <Button.Content>A Partially Transparent Button</Button.Content>
</Button>
```

Međutim, korišćenje *OpacityMask* sa *LinearGradientBrush* ili *RadialGradientBrush* postaje veoma korisno. Korišćenje preliva koji se pomeraju od pune do providne boje, može se kreirati efekat providnosti i učiniti da jedan deo kontrole nestaje.

```
<Button FontSize="14" FontWeight="Bold" Height="100" Margin="5">
    <Button.OpacityMask>
        <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">
            <GradientStop Offset="0" Color="Black">
            </GradientStop>
            <GradientStop Offset="1" Color="Transparent">
            </GradientStop>
        </LinearGradientBrush>
    </Button.OpacityMask>
<Button.Content>A Partially Transparent Button</Button.Content>
</Button>
```

Slika 5: Primena *OpacityMask* na *Button* kontrolu

2.8. Animacije

Jedno od glavnih unapređenja koje dolazi sa pojavom WPF programskog modela je dobro iskorišćenje vizuelnih sposobnosti sistema kao što su animacije. One omogućavaju promenu osobina u vremenskim trenucima što može biti korisno pri kreiranju efekata. Ovo svojstvo omogućava da se oseti puna snaga WPF prezentacionog sloja.

Animation klase predstavljaju veliku grupu klasa koje su dizajnirane da implementiraju automatsku promenu vrednosti property-ja. Postoje 42 klase animacija u S*ystem.Windows.Media.Animation* namespace-u i svaka od njih ima specifični tip podataka koji animira. Dele se u tri grupe klasa: linearne animacije, key frame-based animacije i path-based animacije.

Linearne animacije, koje linearno menjaju vrednost property-ja, imaju format imena *<ImeTipa>Animation,* gde je *<ImeTipa>* naziv tipa koji animira. *DoubleAnimation* je jedan primer linearne animacije i jedana od najčešće korišćenih animacija.

Tok *key frame-based* animacije počinje u početnom, a zatim se nastavlja kroz svaki key frame do svog završetka. Taj progres je obično linearan. Format imena key frame animacija je *<ImeTipa>AnimationUsingKeyFrames* gde *<ImeTipa>* predstavlja ime tipa koji animiraju. Kada je reč o ovoj vrsti animacija, obično se spominje *StringAnimationUsingKeyFrames.*

Path-based animacije vode animaciju kroz *Path* objekat. Najčešće se koriste za animiranje osobina koje se odnose na pomeranje objekata po nekoj definisanoj putanji. Njihov format imena je *<ImeTipa>AnimationUsingPath* gde se *<ImeTipa>* odnosi na tip koji animira. Trenutno postoje samo tri path-based animacije: *PointAnimationUsingPath, DoubleAnimationUsingPath* i *MatrixAnimationUsingPath.*

2.8.1. Važne osobine animacija

Iako postoji mnogo različitih klasa animacija, sve one rade na isti način – menjaju vrednost property-ja u određenom vremenskom periodu. Samim tim one dele neke zajedničke osobine. Mnoge od ovih osobina pripadaju i *Storyboard* klasi koja se koristi za organizaciju animiranih objekata. Najvažnije zajedničke osobine prikazane su sledećom tabelom.

Osobina	**Opis**
AutoReverse	Ukazuje na to kada će se nakon završetka definisane animacije odratiti i inverzna animacija
BeginTime	Vreme kada animacija treba da počne sa izvršavanjem u odnosu na početak izvršavanja. Na primer ako *BeginTime* ima vrednost 0:0:5, pokretanjem animacije ona će čekati 5 sekundi a zatim početi svoje izvršavanje
Duration	Vreme dužine trajanja animacije
FillBehavior	Vrednost koja ukazuje na to kako će se animacija ponašati nakon svog završetka
RepeatBehavior	Vrednost koja definiše kako će se animacija ponavljati

Klase linearnih animacija implementiraju još nekoliko značajnih osobina:

Osobina	**Opis**
From	Startna vrednost animacije. Ako je ova vrednost izostavljena posmatraće se trenutna vrednost property-ja
To	Završna vrednost animacije
By	Veličina za koju će se uvećavati vrednost property-ja za vreme trajanja animacije. Ukoliko su postavljene obe vrednosti *To* i *By,* vrednost *By* će se ignorisati

2.8.2. *DoubleAnimation*

Sledeći primer pokazuje jednu jednostavnu animaciju. Ona menja vrednost property-ja koji je tipa *Double* od 1 do 200 u toku 10 sekundi:

```
<DoubleAnimation Duration="0:0:10" From="1" To="200" />
```

Može se primetiti da nešto nedostaje u ovom primeru. Koji property ova animacija animira? Kao odgovor ne možemo dobiti o kojoj osobini se radi. Animacija zapravo ne vodi računa o osobinama koje animira, ona se primenjuje na property specificiran pomoću *Storyboard* objekta.

2.8.3. *Storyboard* objekti

Storyboard predstavlja objekat koji kontroliše i organizuje animacije u korisničkom interfejsu. *Storyboard* klasa poseduje *Children* kolekciju koja organizuje kolekciju *Timeline* objekata koji uključuju i *Animation* objekte. Kada se deklarišu u XAML-u, svi *Animation* objekti moraju biti u okviru *Storyboard* objekata.

```
<Storyboard>
    <DoubleAnimation Duration="0:0:10" From="1" To="200" />
</Storyboard>
```

Najvažniji dodatak *Storyboard* objekata je što sadrže property-je kojima se može definisati ciljani element i njegov property *Animation* objekata.

```
<Storyboard TargetName="Button1" TargetProperty="Height">
    <DoubleAnimation Duration="0:0:10" From="1" To="200" />
</Storyboard>
```

Ovaj primer se, za razliku od prethodnih, može koristiti za kreiranje animacije. U intervalima od 10 sekundi, *Height* property dugmeta sa imenom *Button1* menja se od vrednosti 1 do vrednosti 200. *TargetName* i *TargetProperty* su attached properties, pa se, umesto definisanja u samom *Storyboard* objektu, mogu definisati u unutrašnjoj animaciji.

```
<Storyboard>
    <DoubleAnimation Duration="0:0:10" From="1" To="200"
        Storyboard.TargetName="Button1"
        Storyboard.TargetProperty="Height" />
</Storyboard>
```

Pošto *Storyboard* može držati više *Animation* objekata u isto vreme, ovaj vid korišćenja omogućava postavljanje različitih elemenata i njihovih property-ja za svaku animaciju, pa je mnogo pogodnije koristiti attach propery-je.

Kada se *Storyboard* aktivira, sve animacije sadržane u njemu počinju svoje izvršavnje u isto vreme, i izvršavaju se paralelno.

2.8.4. Korišćenje animacija sa *Trigger*-ima

Nakon definisanja animacije i *Storyboard* objekta za organizaciju animacija i postavljanje elementa i njegove osobine na koju deluju animacije, nedostaje još jedan deo da bi se upotpunila slika o kreiranju animacija. Postavlja se pitanje kako se započinje a kako stopira animacija?

Sve deklarativno kreirane animacije moraju biti smeštene unutar *Trigger* objekta. To može biti ili deo nekog definisanog stila, ili deo *Triggers* kolekcije nekog elementa koja prihvata jedino *EventTrigger* objekte. *Trigger* objekti definišu kolekciju *Action* objekata koja kontroliše kada se animacija startuje a kada zaustavlja. Sledećim primerom je prikazana upotreba *EventTrigger* objekta.

```
<EventTrigger RoutedEvent="Button.Click">
   <EventTrigger.Actions>
      <BeginStoryboard>
          <Storyboard>
              <DoubleAnimation Duration="0:0:5"
                  Storyboard.TargetProperty="Height" To="200" />
          </Storyboard>
      </BeginStoryboard>
   </EventTrigger.Actions>
</EventTrigger>
```

Storyboard objekat je uokviren tagom *BeginStoryboard,* koji je uokviren *EventTrigger.Actions* tagom. *BeginStoryboard* predstavlja akciju – označava kada će započeti *Storyboard* koji sadrži. *EventTrigger* klasa sadrži kolekciju akcija koje će biti inicijalizovane kada je *Trigger* aktiviran. U ovom primeru, kada se klikne na dugme izvršiće se definisana animacija.

Neke od akcija koje se koriste za upravljanje animacijama su:

Akcija	Osobina
BeginStoryboard	Započinje *Storyboard* objekat
PauseStoryboard	Pauzira *Storyboard* objekat u određenom trenutku
ResumeStoryboard	Nastavlja prekinut *Storyboard*
SkipStoryboardToFill	Pomera *Storyboard* na kraj vremenskog intervala definisanog za izvršavanje animacije
StopStoryboard	Zaustavlja animaciju i vraća je u početnu poziciju

2.8.5. *ColourAnimation*

Kao što je napomenuto, klase animacija postoje za svaki tip koji je podložan animacijama. Pored *DoubleAnimation* često se koriste i *ColourAnimation* koje omogućavaju promenu boje definisanog property-ja.

```
<Button Height="23" Width="100" Name="Button1">
  <Button.Background>
      <SolidColorBrush x:Name="myBrush" />
  </Button.Background>
  <Button.Triggers>
      <EventTrigger RoutedEvent="Button.Click">
          <BeginStoryboard>
              <Storyboard>
                   <ColorAnimation
                        Storyboard.TargetName="myBrush"
                        Storyboard.TargetProperty="Color"
                        From="Red" To="Green"
                        Duration="0:0:5" />
              </Storyboard>
          </BeginStoryboard>
      </EventTrigger>
  </Button.Triggers>
</Button>
```

U ovom primeru, nakon klika na dugme njegova pozadina se menja od crvene do zelene u toku 5 sekundi.

Ovim se uokviruje priča o upoznavanju sa malim delom WPF tehnologije koja je bila potrbna za izradu ove Playlist kontrole za elektronsku oglasnu tablu.

3. Playlist kontrola za elektronsku oglasnu tablu

3.1. Ideja izrade Playlist kontrole za elektronsku oglasnu tablu

Prilikom razrade ideje o projektu elektronske oglasne table uvidelo se da je neophodan mehanizam koji će kontrolisati prikaz oglasa. Ideja je da se taj mehanizam predstavi kroz Playlist kontrolu koja će, pored automatizacije prikaza, posmatraču pružiti korisne informacije o oglasima koji su, ili će biti, prikazani.

Playlist kontrola će biti podeljena na dva dela. Prvi i najvažniji deo, će sadržati listu oglasa koji će biti prikazani na tabli. Svaki oglas koji je prikazan u Playlist kontroli prikazivaće neke osnovne podatke o sebi kao što su naziv oglasa, autor oglasa, datum njegovog kreiranja i vreme za koje će on biti prikazan na tabli. Kada oglasu istekne vreme do prikaza, njegov sadržaj će biti prikazan na predviđenom prostoru elektronske table. Ovim je rešen prikaz oglasa bez interakcije korisnika.

Još jedan problem koji se može javiti je da lista sadrži veći broj oglasa od raspoloživog prostora na LCD ekranu, naročito u vreme ispitnih rokova kada se broj obaveštenja naglo povećava. Kako bi posmatrač za kratko vreme imao uvid u sve oglase, i na taj način dobio obaveštenje da li se oglas koji očekuje nalazi u listi i kada će biti prikazan, moralo se pristupiti kreiranju još jednog mehanizma automatizacije. Playlist-a prikazuje grupu oglasa koja se može prikazati u raspoloživom prostoru. Nakon određenog vremena Playlist kontrola smenjuje tu grupu novom.

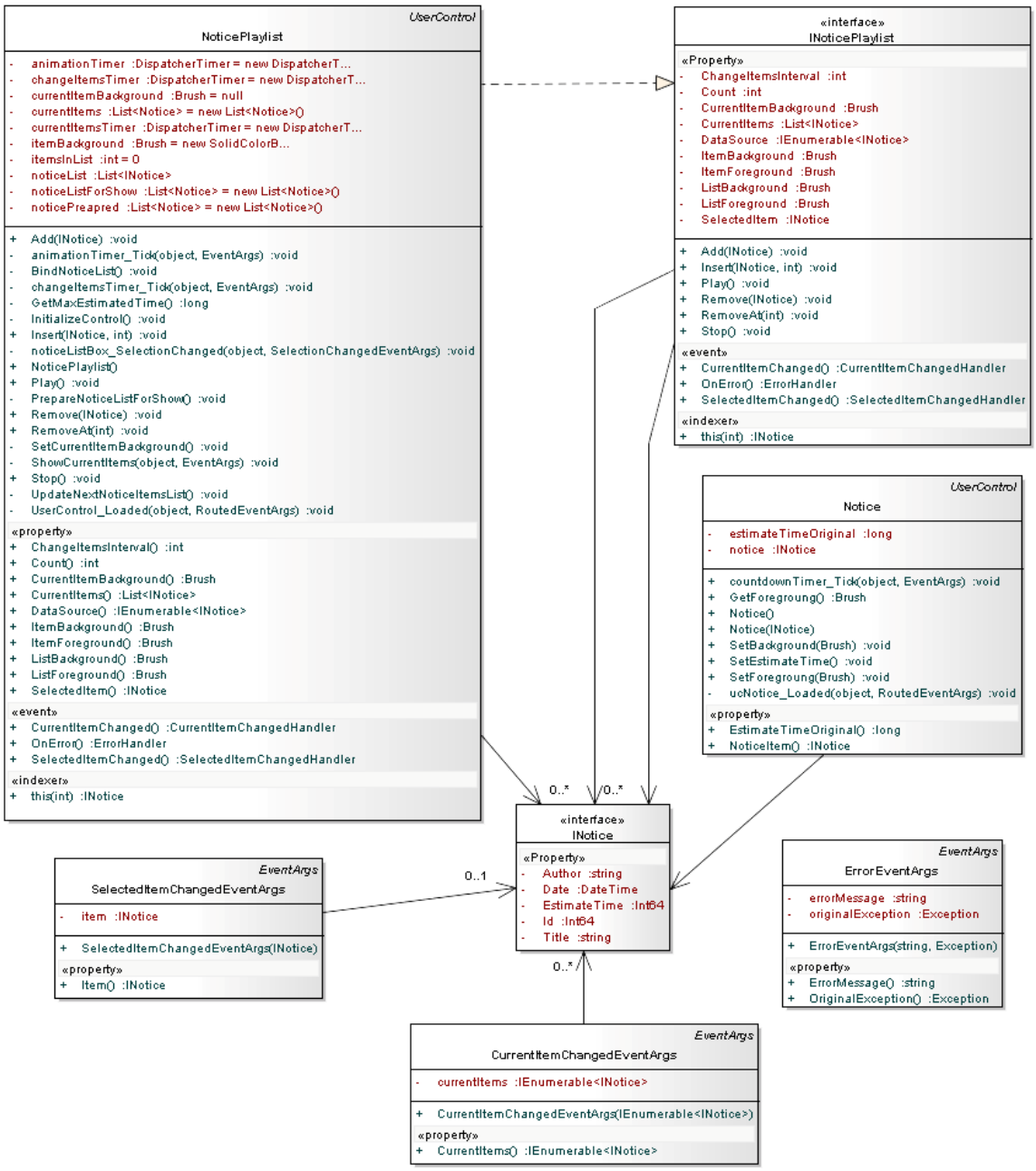
Pošto su ove promene relativno brze, posmatrač lako može izgubiti predstavu o tome koji se oglas trenutno prikazuje. Zato se došlo na ideju da se oglasi, čiji je sadržaj trenutno prikazan, oboje bojom koja se razlikuje od boje ostalih oglasa. Kako bi kontrola dobila na univerzalnosti, neophodno je omogućiti i promenu njenog celokupnog izgleda.

Drugi deo Playlist kontrole će sadrži neke korisne informacije za posmatrača. Prva od njih je lista oglasa koji su sledeći u redu za prikaz, zatim datum poslednjeg postavljenog oglasa (na taj način će posmatrač znati da li su dodati novi oglasi u listu) i na kraju ukupan broj oglasa u listi.

3.2. Model klasa

3.2.1. Dijagram klasa

Model klasa prikazan je dijagramom klasa koji opisuje strukturu projekta objašnjavajući klase unutar njega, njihove atribute i odnose.



Slika 6: Dijagram klasa Playlist kontrole za elektronsku oglasnu tablu

3.2.2. Opis klasa

Radi boljeg razumevanja klasa i njihovih međusobnih odnosa, kao i atributa i metoda, sledi njihov kratak opis:

INotice – interfejs na osnovu koga korisnik definiše podatke o oglasima koji će biti prikazani. Implementacijom interfejsa *INotice,* on dobija spisak potrebnih podataka koje oglas treba da sadrži, tj. koje korisnik treba da definiše:

- *ID (Int64)* – Id oglasa, koji je jedinstven na nivou oglasa i preko koga se vrši njegova identifikacija
- *Title (string)* – Naslov oglasa
- *Author (string)* – Naziv autora oglasa
- *Date (DateTime)* – Datum postavljanja oglasa
- *EstimateTime (Int64)*– Vreme do prikaza definisanog oglasa na tabli

Notice – kontrola koja definiše izgled oglasa koji će biti prikazani. Glavni property ove klase je *NoticeItem* tipa *INotice* koji sadrži podatke o oglasu.

INoticePlaylist – interfejs preko koga se definišu podaci koji su vezani za celokupnu Playlist kontrolu. Kreiranjem liste oglasa u navedenom formatu dobijamo *DataSource* sa kojim kontrola zna da manipuliše. Pored definisanja i prosleđivanja *DataSource*-a kontroli, pre njenog pokretanja mogu se definisati još neke osobine:

- *ItemBackgroung (Brush)* – pozadinska boja oglasa
- *ItemForeground (Brush)* – boja slova koja se nalaze na oglasu
- *ListBackground (Brush)* – pozadinska boja kontrole
- *ListForeground (Brush)* – boja slova koja se nalaze na kontroli
- *CurrentItemBackground (Brush)* – pozadinska boja koja će se kroistiti za oglas koji je trenutno prikazan na tabli kako bi korisnik imao vizuelnu predstavu (a i oznaku) koji oglasi se trenutno prikazuju
- *ChangeItemsInterval (Int64)*– interval vremena (u sekundama) za koji će se smenjivati grupe oglasa na tabli

Da bi Playlist kontrola krenula sa radom neophodno je pozvati definisanu metodu *Play.* Kao što i samo ime metode govori, ona služi za pokretanje kontrole tj. pokretanje svih brojača koji su u njoj definisani. Pored ove metode, neizostavna metoda je *Stop* koja stopira rad liste, tj. stopira rad brojača.

Tokom rada kontrole, korisnik ima uvid u sledeće osobine:

- *Count (int)* – ukupan broj elemenata (oglasa) u listi

- *CurrentItems (List<INotice>)* – lista oglasa koji su trenutno prikazani na tabli
- *SelectedItem (INotice)* – oglas koji je selektovan
- Može dobiti oglas sa određene pozicije u listi

Dodavanje novog oglasa u postojeću listu može se izvršiti na dva načina:

- Dodavanje oglasa na kraj liste, korišćenjem metode *Add* i
- Dodavanje oglasa na određenu poziciju u listi korišćenjem metode *Insert* gde se korisniku pruža mogućnost definisanja pozicije u listi na koju će se ubaciti oglas

Uklanjanje nekog oglasa iz liste takođe se može uraditi na dva načina:

- Uklanjanje korišćenjem metode *Remove* gde korisnik definiše sve podatke o oglasu koji želi ukloniti i
- Uklanjanje oglasa sa određene pozicije u listi korišćenjem metode *RemoveAt.* U ovom slučaju potrebno je definisati samo indeks elementa u listi

NoticePlaylist – kontrola koja definiše izgled cele Playlist kontrole. Implementira *INoticePlaylist* interfejs.

SelectedItemChangedEventArgs – klasa izvedena iz *EventArgs* klase. Čuva podatke o selektovanom oglasu u listi.

SelectedItemChangedHandler – pokazivač na metodu koja se izvršava prilikom seletovanja nekog oglasa u listi. Playlist-a dozvoljava selektovanje jednog oglasa u listi. Događaj *SelectedItemChanged* obaveštava korisnika da je neki oglas selektovan i prosleđuje ga.

CurrentItemChangedEventArgs – klasa izvedena iz *EventArgs* klase. Čuva podatke o trenutnim oglasima u listi (oglasi koji se trenutno prikazuju).

CurrentItemChangedHandler – pokazivač na metodu koja se izvršava prilikom zamene oglasa koji su prikazani na tabli sa onima koji će biti prikazani. Prikupljanjem oglasa kojima je vreme do prikaza isteklo, kontrola prosleđuje korisniku tu listu kako bi imao uvid koje oglase treba prikazati i na taj način adekvatno ažurirao oglasnu tablu. Događaj zadužen za obaveštavanje korisnika o oglasima koje treba prikazati je *CurrentItemChanged*.

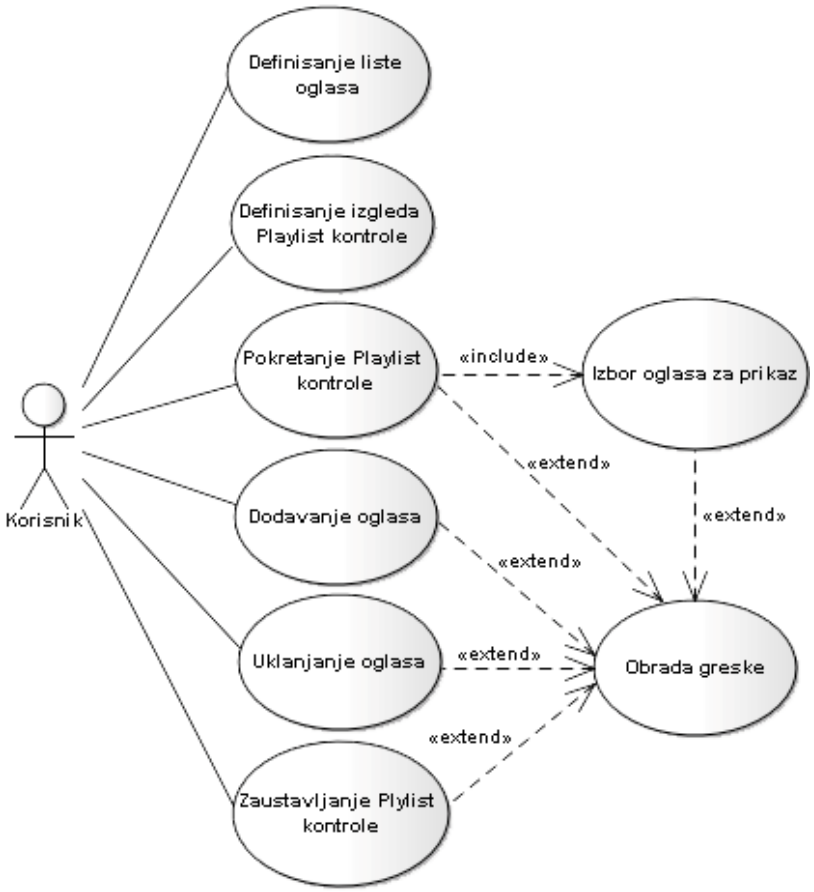
ErrorEventArgs – klasa izvedena iz *EventArgs* klase. Čuva podatke o greški koja se desila tokom rada kontrole.

ErrorHandler – pokazivač na metodu koja se izvršava ukoliko dođe do greške u radu kontrole.

3.3. Model slučajeva korišćenja

UseCase dijagram u softverskom inženjeringu predstavlja opis koraka ili akcija koje korisnik treba da sprovede kako bi došao do nekog cilja. Korisnik aplikacije može biti osoba ili nešto apstraktnije, na primer softverski sistem što je slučaj sa ovom Playlist kontrolom.

UseCase model Playlist kontrole predstavljen je sledećim dijagramom:



Slika 7: UseCase dijagram Playlist kontrole za elektronsku oglasnu tablu

UC 001 - Definisanje liste oglasa:

Opis:
Definiše se lista oglasa koja će biti prikazana u Playlist kontroli.

Glavni tok događaja:
Korisnik implementira *INotice* interfejs kao format za definisanje oglasa.
Korisnik definiše listu oglasa koja će predstavljati *DataSource* za Playlist kontrolu.
Slučaj korisćenja se završava prosleđivanjem definisane liste Playlist kontroli.

Definisanje uslova:
Pred-uslovi: Uključivanje kontrole u projekat. Implementacija *INotice* interfejsa kao formata za definisanje oglasa.
Post-uslov: Regularan rad Playlist kontrole.

Alternativni tokovi događaja:
Greška: Sistemska greška.

UC 002 - Definisanje izgleda Playlist kontrole:

Opis: Definisanje izgleda Playlist kontrole.

Glavni tok događaja:
Korisnik definiše pozadinsku boju za Playlist kontrolu.
Korisnik definiše pozadinsku boju za oglase.
Korisnik definiše boju slova kontrole.
Korisnik definiše boju slova oglasa.
Korisnik definiše vremenski interval nakon kojeg će se vršiti promena grupe oglasa koji su trenutno prikazani u Playlist kontroli.
Slučaj korišćenja se završava definisanjem izgleda Playlist kontrole.

Definisanje uslova:
Pred-uslov: Uključivanje kontrole u projekat.
Post-uslov: Regularan rad Playlist kontrole.

Alternativni tokovi događaja:
Izuzeci: Ako se ne definiše izgled, Playlist kontrola će se prikazati u njenom osnovnom izgledu.
Greška: Sistemska greška.

UC 003 - Pokretanje Playlist kontrole:

Opis:
Pokretanje i prikaz Playlist kontrole.

Glavni tok događaja:
Korisnik pokreće rad Playlist kontrole.
Slučaj korišćenja se završava prikazom Playlist kontrole.

Definisanje uslova:
Pred-uslovi: Definisana lista oglasa (*DataSource*) je prosleđena Playlist kontroli. Kontrola nije pokrenuta.
Post-uslov: Regularan rad Playlist kontrole.

Alternativni tokovi događaja:
Izuzeci: Ako korisnik prosledi praznu listu kao *DataSource* primiće informaciju o tome.
Greška: Pokušaj narušavanja rada Playlist kontrole.

UC 004 - Izbor oglasa za prikaz:

Opis:
Korisniku se prosleđuje lista oglasa koji treba da se prikažu na tabli.

Glavni tok događaja:
Playlist kontrola prlazi kroz listu oglasa i proverava kojim oglasima je isteklo vreme do prikaza.
Ukoliko takvi oglasi postoje, prosleđuje ih korisniku.
Slučaj korišćenja se završava prosleđivanjem oglasa koji treba da se prikažu na tabli.

Definisanje uslova:
Pred-uslov: Playlist kontrola je pokrenuta.
Post-uslov: Regularan rad Playlist kontrole.

Alternativni tokovi događaja:
Greška: Pokušaj narušavanja rada Playlist kontrole.

UC 005 - Dodavanje oglasa:

Opis:
Dodavanje novog oglasa u listu.

Glavni tok događaja:
Korisnik definiše oglas koji želi dodati.
Korisnik definiše poziciju na koju želi dodati oglas.
Slučaj korišćenja se završava dodavanjem novog oglasa u postojeću listu oglasa.

Definisanje uslova:
Pred-uslovi: Playlist kontrola je pokrenuta. Definisan je oglas koji se dodaje ili (u zavisnosti od izabranog načina dodavanja) pozicija oglasa u listi.
Post-uslov: Regularan rad Playlist kontrole. Usled dodavanja novog oglasa ne sme biti narušen rad Playlist kontrole.

Alternativni tokovi događaja:
Izuzeci: Pozicija u listi ne postoji (lista je prazna, pozicija je negativan broj ili broj koji je veći od ukupnog broja elemenata u listi).
Greška: Pokušaj narušavanja rada Playlist kontrole.

UC 006 - Uklanjanje oglasa:

Opis:
Uklanjanje postojećeg oglasa iz liste.

Glavni tok događaja:
Korisnik defiše oglas koji želi ukloniti.
Korisnik definiše poziciju sa koje želi ukloniti oglas.
Slučaj korišćenja se završava uklanjanjem oglasa iz liste.

Definisanje uslova:
Pred-uslovi: Playlist kontrola je pokrenuta. Definisan je oglas koji se uklanja ili (u zavisnosti od izabranog načina uklanjanja) pozicija oglasa u listi.
Post-uslov: Regularan rad Playlist kontrole. Usled uklanjanja postojećeg oglasa ne sme biti narušen rad Playlist kontrole.

Alternativni tokovi događaja:
Izuzeci: Oglas ne postoji u listi. Pozicija u listi ne postoji (lista je prazna, pozicija je negativan broj ili broj koji je veći od ukupnog broja elemenata u listi).
Greška: Pokušaj narušavanja rada Playlist kontrole.

UC 007 - Zaustavljanje Playlist kontrole:

Opis:

Zaustavlja se rad Playlist kontrole.

Glavni tok događaja:

Korisnik zaustavlja rad Playlist kontrole.
Slučaj korišćenja se završava zaustavljanjem Playlist kontrole.

Definisanje uslova:

Pred-uslov: Playlist kontrola je pokrenuta.

Alternativni tokovi događaja:

Greška: Sistemska greška.

UC 008 – Obrada greške:

Opis:

Obrada greške koja se dogodila prilikom rada Playlist kontrole.

Glavni tok događaja:

Playlist kontrola definiše grešku koja se dogodila.
Playlist kontrola prosleđuje grešku korisniku.
Playlist kontrola se vraća u stanje pre greške.
Slučaj korišćenja se završava prosleđivanjem nastale greške korisniku.

Definisanje uslova:

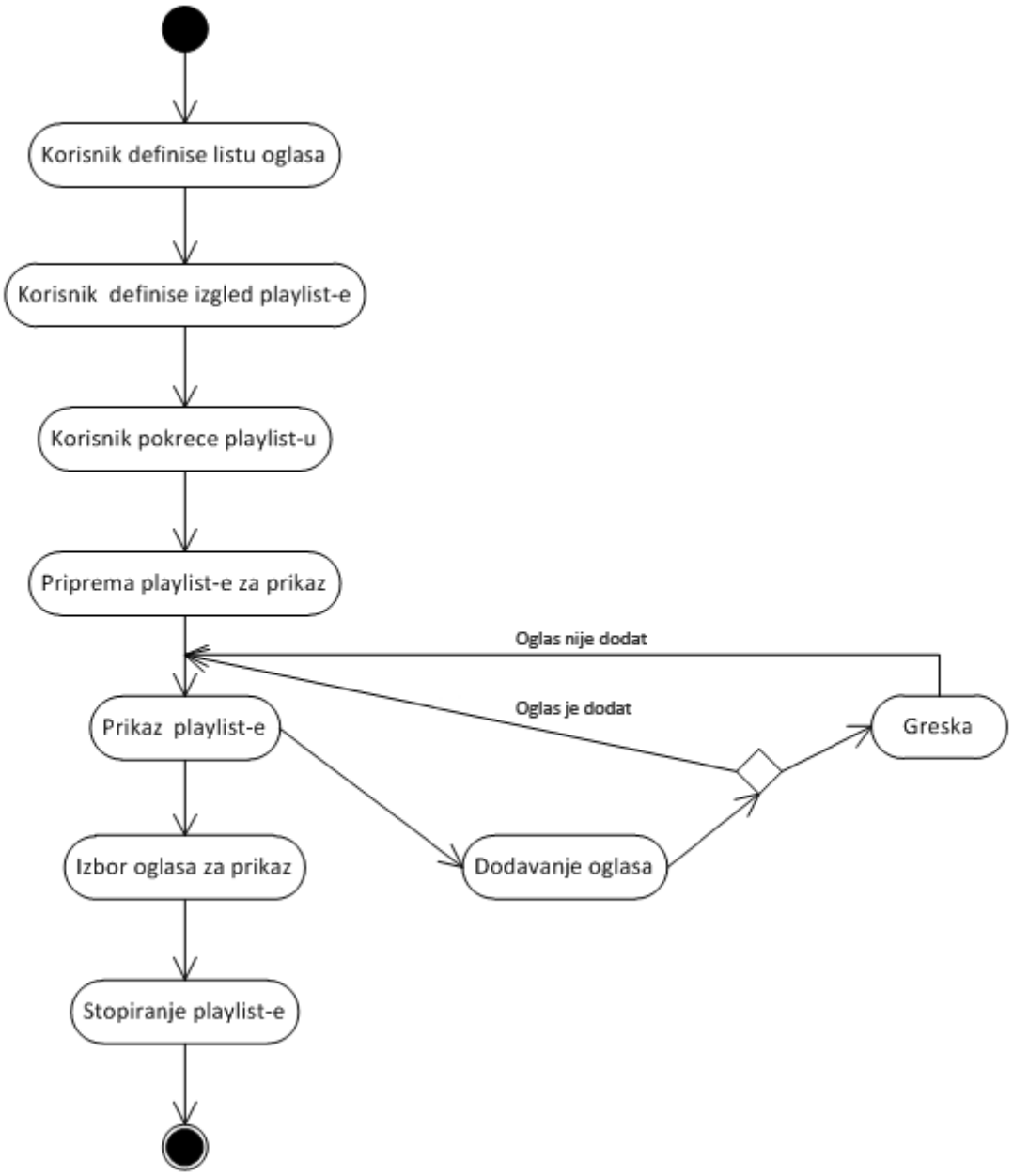
Pred-uslov: Playlist kontrola je pokrenuta.
Post-uslov: Regularan rad Playlist kontrole.

3.4. Dijagram aktivnosti

Inicijalizacija i pokretanje Playlist kontrole

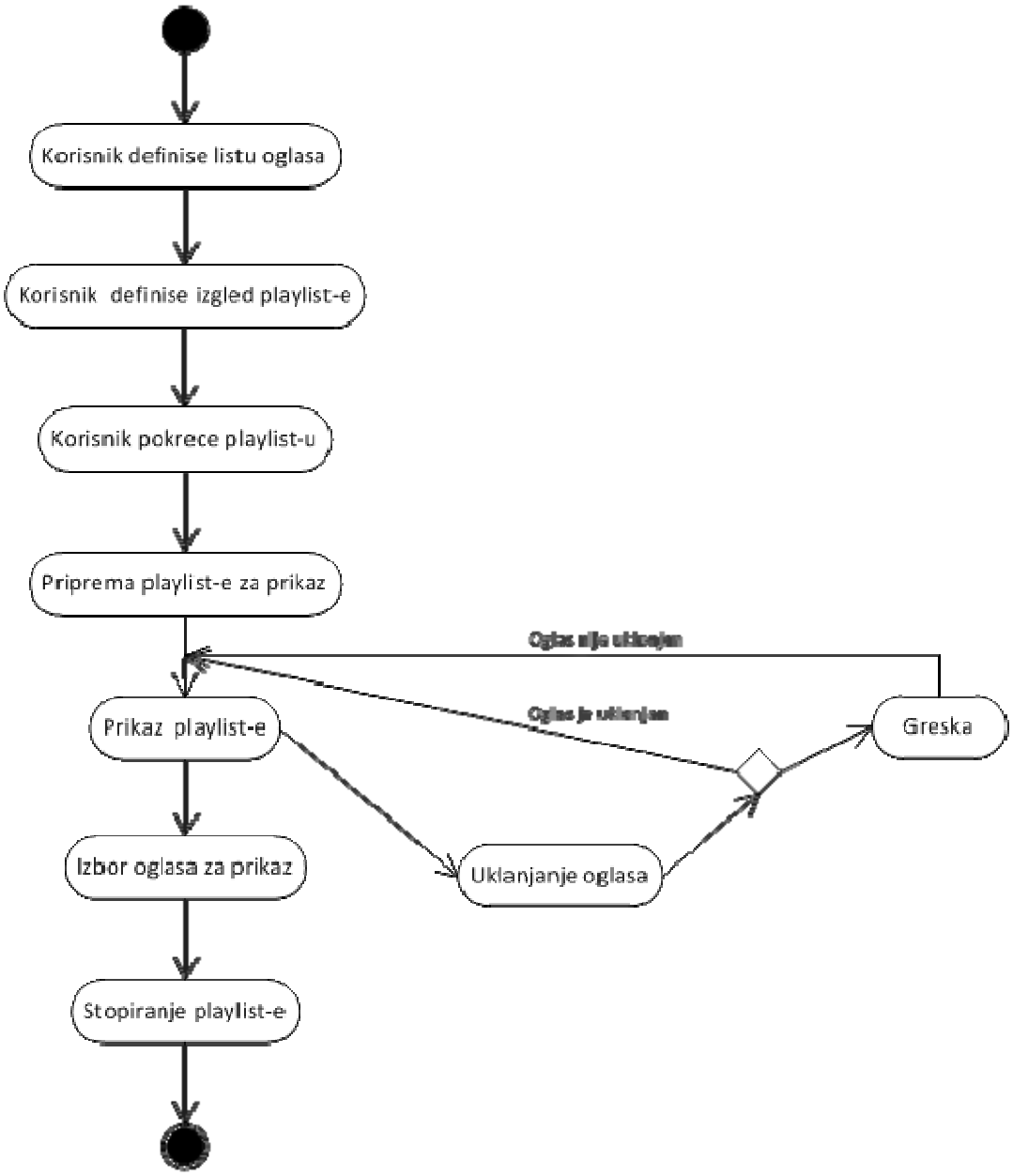
Slika 8: Dijagram aktivnosti za inicijalizaciju i pokretanje Playlist kontrole

Dodavanje oglasa



Slika 9: Dijagram aktivnosti za dodavanje oglasa

Uklanjanje oglasa



Slika 10: Dijagram aktivnosti za uklanjanje oglasa

3.5. Implementacija

3.5.1. Razvojno okruženje

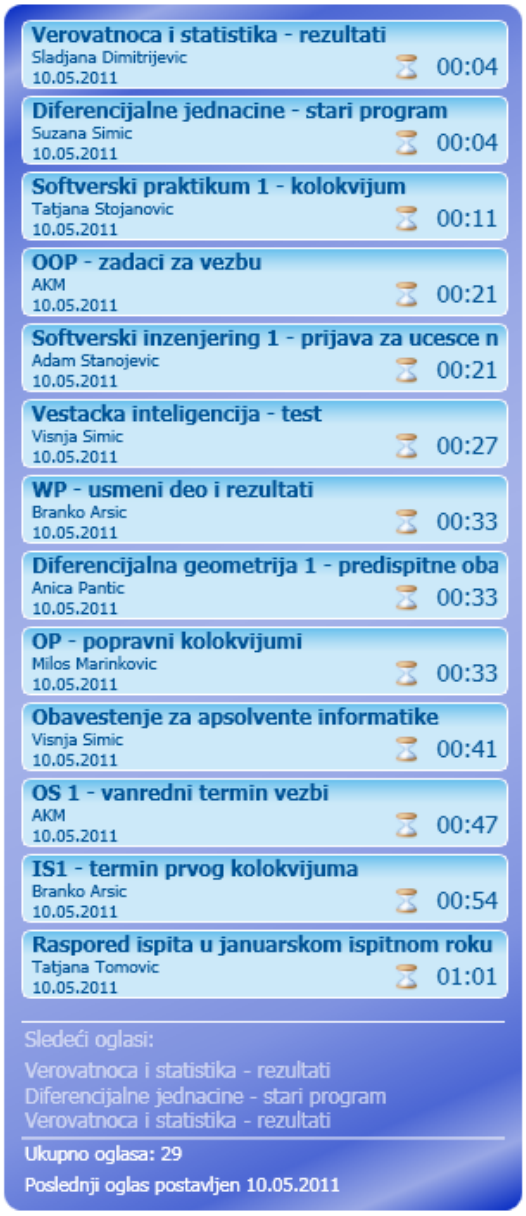
Tokom razvoja Playlist kontrole korišćene su sledeće tehnologije i razvojni alati:

- **.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 Expression Blend 4** – dizajnerski alat korišćen za izradu vizuelno složenih grafičkih komponenti.
- **WPF** – deo .NET framework-a korišćen za razvoj korisničkog interfejsa.
- **Microsoft Office Visio 2010** – alat korišćen za izradu dijagrama
- **Enterprise Architect 9.0** – alat za projektovanje informacionih sistema

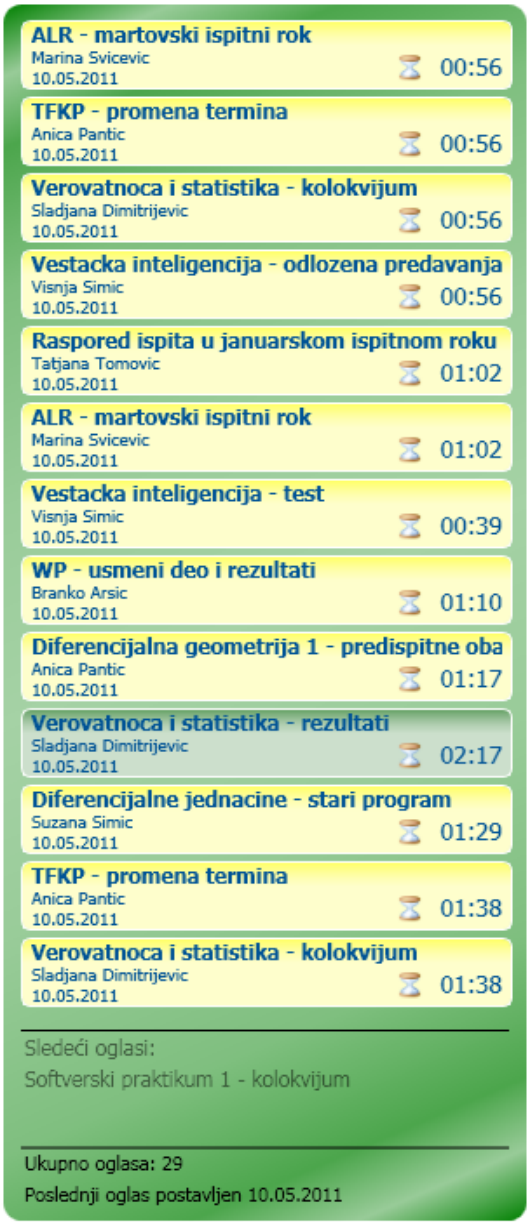
Funkcionalnosti rešenja razvijene su u programskom jeziku C# dok je za izradu korisničkog interfejsa korišćen XAML jezik.

3.5.2. Izgled Playlist kontrole

Slika 11. predstavlja njen početno definisani izgled koji korisnik dobija sa celokupnom kontrolom, dok druga Slika 12. predstavlja jedan primer promene njenog izgleda.



Slika 11: Početno definisani izgled Playlist kontrole definisan



Slika 12: Izgled Playlist kontrole od strane korisnika

3.5.2.1. Definisanje preliva

Kao što se sa slika može primetiti, prilikom promene izgleda kontrole menja se samo osnovna boja dok definisani prelivi ostaju. Ovaj zahtev vodio je ka upotrebi *OpacityMask* u kombinaciji sa *LinearGradientBrush*. Međutim, jedna osobina WPF elemenata onemogućila je jednostavno korišćenje ovih objekata. Naime, kada se na neki WPF element primeni *OpacityMask,* ona se odražava i na elemente koje sadrži. U ovom slučaju to je otežavajuća osobina jer Playlist kontrola ima različito definisane prelive od onih na samom oglasu kao njenom elementu. Ako bi se definisala *OpacityMask* na kontroli, ona bi se prenela i na oglase. Definisanjem druge *OpacityMask* na oglasima dovelo bi do mešanja tih definicija.

Da bi se prevazišao ovaj problem, korišćena je osobina WPF *Grid* layout kontrole da ukoliko su pozicije i margine elemenata iste, *Grid* kontrola ih smešta jedne preko drugih u redosledu kako su dodavani. Kombinacija ovih osobina radi dobijanja željenih osobina celokupne kontrole počinje definisanjem pomenute layout *Grid* kontrole kao panela u koji će se smestiti lista. Zbog osobine prenošenja *OpacityMask* na child elemente ona se ne može primeniti na ovaj panel. Njegova pozadinska boja će biti osnovna boja liste (nijansa plava kao izabrana boja za početni izgled liste). *Grid* će sadržati dva elementa:

- *Rectangle* pravougaonik koji će nositi *OpacityMask*
- *Grid* panel u kome će se smeštati svi elementi

Dimenzije ova dva child elementa u svakom trenutku moraju biti iste kao i dimenzije glavnog *Grid* panela kako bi njihova preklapanja bila potpuna i u okviru dimenzija cele kontrole. Postavljanjem *Rectangle* kao prvog child elementa a *Grid* panela kao drugog dobili smo da se grid pikazuje prego *Rectangle* elementa. Sada možemo definisati *OpacityMask* na *Rectangle* element bez njenog prenošenja na ostale elemente (jer su svi oni definisani van njega). Sve potrebne elemente u listi smeštamo u child *Grid* panel. Njegova pozadinska boja mora biti definisana kao *Transparent* (providna) kako bi se video *Rectangle* ispod njega. Prilikom promene izgleda kontrole, menja se samo pozadina prvog *Grid* elementa.

Za kreiranje *OpacityMask* korščen je *ExpressionBlend* zbog već pomenutih prednosti u izradi korisničkog interfejsa u odnosu na *VisualStudio*.

Deo koda koji sledi predstavlja implementaciju opisanih koraka. Ista logika primenjena je i na definisanje izgleda kontrole koja predstavlja jedan oglas u listi.

```
<Grid Background="#FF0027C2">

     <Rectangle Name="recStyle" Fill="White" Height="700" Width="300">
         <Rectangle.OpacityMask>
             <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
                        <GradientStop Color="#00000000" />
                        <GradientStop Color="#99DADADA" Offset="0.05"/>
                        <GradientStop Color="#4C282828" Offset="0.1"/>
                        <GradientStop Color="#66333333" Offset="0.15"/>
                        <GradientStop Color="#7FA2A2A2" Offset="0.2"/>
                        <GradientStop Color="#99656565" Offset="0.35"/>
                        <GradientStop Color="#A5808080" Offset="0.5"/>
                        <GradientStop Color="#99656565" Offset="0.65"/>
                        <GradientStop Color="#7FA2A2A2" Offset="0.80"/>
                        <GradientStop Color="#66333333" Offset="0.85"/>
                        <GradientStop Color="#4C282828" Offset="0.90"/>
                        <GradientStop Color="#99DADADA" Offset="0.95"/>
                        <GradientStop Offset="1" />
             </LinearGradientBrush>
         </Rectangle.OpacityMask>
     </Rectangle>

     <Grid Name="gridNotice" Width="Auto" Height="Auto"
           Background="Transparent">

      <!--Add elements-->

     </Grid>

</Grid>
```

3.5.2.2. Izrada animacija

Da bi lista sledećih oglasa ukazivala na važan podatak, osmišljeno je da ona bude animirana. Korišćena je *DoubleAnimation* klasa za definisanje animacije koja će se odnositi na vidljivost ove liste kako bi se dobio utisak da ona treperi. Samim tim *TargetProperty* je *Opacity*, a vidljivost ide od 1.0 (vidljivo) do 0.0 (nevidljivo). *AutoReverse* ima vrednost *True* kako bi se posle nestajanja lista ponovo pojavila, dok *RepeatBehavior* sa vrednošću *Forever* govori da će se animacija ponavljati dok je lista pokrenuta. Kao *RoutedEvent* uzet je *Load* event *TextBlock* kontrole u kojoj se nalazi lista. Na ovaj način je obezbeđeno da se animacija pokreće sa pokretanjem cele kontrole jer se u tom trenutku učitavaju svi njeni elementi.

```
<TextBlock Name="txtNextNoticeList" Foreground="White" Margin="2,2,2,2"
           Background="Transparent" HorizontalAlignment="Left"
      FontSize="12"
           Height="44">
    <TextBlock.Triggers>
        <EventTrigger RoutedEvent="TextBlock.Loaded">
             <BeginStoryboard>
                  <Storyboard>
                       <DoubleAnimation
                           Storyboard.TargetName="txtNextNoticeList"
                           Storyboard.TargetProperty="Opacity"
                           From="1.0" To="0" Duration="0:0:0.7"
                           AutoReverse="True" RepeatBehavior="Forever"/>
                  </Storyboard>
             </BeginStoryboard>
        </EventTrigger>
    </TextBlock.Triggers>
</TextBlock>
```

Ista vrsta animacije korišćena je i prilikom promene grupe elemenata u listi.

3.5.3. Implementacija funkcionalnosti Playlist kontrole

Nakon definisanja izgleda svih elemenata neophodno je dodati zahtevane funkcionalnosti liste kako bi se dobila kompletno upotrebljiva kontrola. Zbog obimnosti koda sledi implementacija nekih važnijih funkcionalnosti.

3.5.3.1. Pokretanje Playlist kontrole

Nakon prosleđivanja liste oglasa koja će biti prikazana, pozivanjem metode *Play* pokrenut je mehanizam automatizacije liste.

```
public void Play()
{
   try
   {
       InitializeControl();
       PrepareNoticeListForShow();

       //Start timer to change first n items in the list for show.
       changeItemsTimer.Start();
       //Start timer to activate animation while group is changing.
       animationTimer.Interval = new TimeSpan(0, 0,
                                    changeItemsTimer.Interval.Seconds -
                                   1);
       animationTimer.Start();

       //Start countdown timer for estimate time.
       currentItemsTimer.Interval = new TimeSpan(0, 0, 1);
       //Invoke event to decrease estimate time for every notice item in
       //list.
       noticeListForShow.ForEach(f => currentItemsTimer.Tick += new
                                 EventHandler(f.countdownTimer_Tick));
       //Invoke event to show current items.
       currentItemsTimer.Tick += new EventHandler(ShowCurrentItems);
       currentItemsTimer.Start();
    }
    catch (Exception ex)
    {
       string errorMessage = "Error while attempting to play list!";
       //Invoke event to show error.
       OnError.Invoke(new ErrorEventArgs(errorMessage, ex));
    }
}
```

Prva metoda koja se poziva u okviru *Play* metode je *InitializeControl* kojom se postvalja datum poslednjeg oglasa i ukupan broj oglasa u listi.

Metoda *PrepareNoticeListForShow* vrši pripremu liste za prikaz. Tu se pre svega računa koliko oglasa može biti prikazano u zavisnosti od raspoloživog prostora za to. Od visine cele kontrole oduzima se visina panela u kom su smeštene dodatne informacije o sledećim oglasima, ukupnom broju oglasa i datumu poslednjeg oglasa, kao i margine definisane na kontroli. Pošto su svi elementi liste istih dimenzija, dovoljno je uzeti dimenzije jednog od njih za dalja preračunavanja. Zato se iz liste oglasa *noticeListForShow* uzima visina prvog oglasa sa odgovarajućim marginama. Deljenjem broja koji označava visinu dozvoljenog prostora sa brojem koji ukazuje na visinu prostora koji treba obezbediti za jedan element, dobija se broj oglasa koji se mogu prikazati u jednom trenutku. Sledi deo koda kojim je ovo implementirano.

```
double controlHeight = this.Height;
double infoHeight = stackInfo.Height;
double listHeight = controlHeight - infoHeight;
listHeight -= noticeListBox.Margin.Top + noticeListBox.Margin.Top;

double itemHeight = (noticeListForShow.First() as Notice).Height +
                    (noticeListForShow.First() as Notice).Margin.Bottom +
                    (noticeListForShow.First() as Notice).Margin.Top;

itemsInList = (int)(listHeight / itemHeight);
```

Na sličan način se računa i širina oglasa. Od širine kontrole oduzimaju se ogovarajuće margine čime se dobija nova širina oglasa. Na ovaj način je omogućeno da se širina oglasa koriguje sa promenom širine cele kontrole, dok je preračun broja oglasa u listi obezbedio da se popuni sav raspoloživ prostor.

Nakon određivanja ovih podataka, iz liste svih oglasa uzima se izračunati broj oglasa za prikaz i prosleđuje se *ListBox* kontroli.

```
noticePreapred = noticeListForShow.Take(itemsInList).ToList();
noticeListBox.ItemsSource = noticePreapred.ToList();
```

Kada je određena veličina grupe oglasa koja će biti u listi, i prikazana prva od njih, mogu se pokrenuti brojači. Definisana su tri brojača:

- *changeItemsTimer* koji kontroliše promenu grupe elemenata u listi.
- *animationTimer* koji kontroliše animaciju koja se dešava prilikom promene te grupe elemenata. Interval vremena nakon koga se aktivira animacija je za jednu sekundu kraći od intervala vremena koji koristi *changeItemsTimer* kako bi se dobio utisak da jedna grupa polako nestaje a zatim se nova prikazuje.
- *currentItemsTimer* koji kontroliše da li postoje oglasi koje treba proslediti korisniku kako bi ih prikazao.

countdownTimer_Tick metoda odbrojava vreme do prikaza oglasa. Svake sekunde *currentItemsTimer* za svaki oglas u listi poziva metodu *countdownTimer_Tick* koja umanjuje trenutno vreme do prikaza za jednu sekundu.

3.5.3.2. Izbor oglasa za prikaz

Prilikom pokretanja Playlist kontrole pokreće se i mehanizam koji korisniku prosleđuje oglase za prikaz. U prethodnoj metodi *Play*, metoda *ShowCurrentItems* predstavlja event handler za *currentItemsTimer* event.

```
private void ShowCurrentItems(object sender, EventArgs e)
{
     try
     {
          //Get all items with etimate time equals zero.
          currentItems = noticeListForShow.FindAll(n =>
                                        n.NoticeItem.EstimateTime == 0);
          //If exist items with estimate time zero, prepare them for show.
          if (currentItems != null && currentItems.Count > 0)
          {
               //Update estimate time for every item in list.
               foreach (Notice item in currentItems)
               {
                    item.NoticeItem.EstimateTime = GetMaxEstimatedTime() +

                                        item.EstimateTimeOriginal;
               }
               //Invoke event to show prepared items.
               CurrentItemChanged.Invoke(new
                         CurrentItemChangedEventArgs(currentItems.Select(i
                        =>
                                                    i.NoticeItem)));
               //Change colour for displayed items
               SetCurrentItemBackground();
               //Show next items.
               UpdateNextNoticeItemsList();
          }
     }
     catch (Exception ex)
     {
          string errorMessage = "Error while finding current items!";
          //Invoke event to show error.
          OnError.Invoke(new ErrorEventArgs(errorMessage, ex));
     }
}
```

Ova metoda se poziva svake sekunde i proverava kojim oglasima je isteklo vreme. Ako pronađe da je nekim oglasima brojač vremena stigao do nule dogodiće se sledeće akcije:

- Njegovo vreme do prikaza će se promeniti tako da ono bude najveće i na taj način je obezbeđeno da se on pomeri na kraj liste i njegov ponovni prikaz bude tek posle prikaza ostalih oglasa. Ta promena se vrši tako što se pronađe originalno najveće vreme svih oglasa i njemu doda originalno vreme oglasa. Originalna vremena svih oglasa su sačuvana prilikom inicijalizacije kontrole upravo zbog podešavanja novog vremena nakon isteka prethodnog (svake sekunde preostalo vreme se umanjuje i korišćenjem tog broja ne bi se dobili relevantni rezultati).

- Aktivira se *CurrentItemChanged* event koji korisniku prosleđuje listu oglasa za prikaz.
- Menja se boja oglasa koji su trenutno prikazani.
- Postavlja se lista oglasa koji su sledeći za prikaz. Metoda koja je zadužena za to je *UpdateNextNoticeItemsList*. Ona pronalazi minimalno vreme do prikaza a zatim i sve oglase koji imaju to vreme i prikazije ih.

```
private void UpdateNextNoticeItemsList()
{
    txtNextNoticeList.Text = string.Empty;
    if (noticeListForShow != null && noticeListForShow.Count > 0)
    {
        //Get minimum of all estimated times
        long min = noticeListForShow.Select(n =>
                                    n.NoticeItem.EstimateTime).Min();
        //Find all elements with that time.
        List<Notice> minItems = noticeListForShow.FindAll(n =>
                                    n.NoticeItem.EstimateTime ==
                            min);

        //Show them
        if (minItems != null && minItems.Count > 0)
        {
            minItems = minItems.Take(3).ToList();

            minItems.ForEach(item =>
            {
                if (txtNextNoticeList.Text.Equals(string.Empty))
                {
                    txtNextNoticeList.Text += item.NoticeItem.Title;
                }
                else
                {
                    txtNextNoticeList.Text += Environment.NewLine +
                                              item.NoticeItem.Title;
                }
            });
        }
    }
}
```

3.5.3.3. Dodavanje novog oglasa

Novi oglas može biti dodat na kraj ili na definisanu poziciju u listi. Prvi način dodavanja oglasa obezbeđuje metoda *Add* dok je za drugi način zadužena metoda *Insert.* Da bi se oglas dodao na definisanu poziciju u listi, potrebno je da lista ne bude prazna i da definisana pozicija postoji u listi. Ukoliko su zadovoljena oba uslova, pokreće se brojač koji kontroliše vreme do prikaza i oglas se dodaje u listu, dok se neispunjenjem nekog uslova korisniku prosleđuje odgovarajuća greška. U slučaju greške, oglas neće biti dodat.

```
public void Insert(INotice notice, int index)
{
    try
    {
        if (noticeListForShow != null && noticeListForShow.Count > 0)
        {
            if (index >= 0 && index < noticeListForShow.Count)
            {
                Notice n = new Notice(notice);
                //Start timer
                currentItemsTimer.Tick += new
                                    EventHandler(n.countdownTimer_Tick);
                noticeListForShow.Insert(index, n);
            }
            else
            {
                string errorMessage = "Index was out of range!";
                //Invoke event to show error.
                OnError.Invoke(new ErrorEventArgs(errorMessage, null));
            }
         }
         else
         {
             string errorMessage = "List is empty!";
             //Invoke event to show error.
             OnError.Invoke(new ErrorEventArgs(errorMessage, null));
         }
     }
     catch (Exception ex)
     {
         string errorMessage = "Error while adding notice!";
         //Invoke event to show error.
         OnError.Invoke(new ErrorEventArgs(errorMessage, ex));
     }
}
```

3.5.3.4. Uklanjanje oglasa

Playlist kontrola nudi dve metode za uklanjanje oglasa iz liste. Upotrbom metode *RemoveAt* uklanja se oglas sa definisane pozicije u listi, dok se pomoću metode *Remove* uklanja definisani oglas. Metoda *RemoveAt* radi slično prethodno objašnjenoj metodi *InsertAt.* Prvo se proverava da li je lista prazna i da li postoji definisana pozicija a zatim uklanja oglas. Metodom *Remove* se, pre uklanjanja oglasa, proveravaju uslovi o postojanju liste i oglasa u listi.

```
public void Remove(INotice notice)
{
   try
   {
      if (noticeListForShow != null && noticeListForShow.Count > 0)
      {
         Notice nToRemove = noticeListForShow.FirstOrDefault(n =>
                                          (n.NoticeItem.Id == notice.Id));

         if (nToRemove != null)
         {
            noticeListForShow.Remove(nToRemove);
         }
         else
         {
            string errorMessage = "Notice doesn't exist in list!";
            //Invoke event to show error.
            OnError.Invoke(new ErrorEventArgs(errorMessage, null));
         }
      }
      else
      {
         string errorMessage = "List is empty!";
         //Invoke event to show error.
         OnError.Invoke(new ErrorEventArgs(errorMessage, null));
      }
   }
   catch (Exception ex)
   {
      string errorMessage = "Error while removing notice!";
      //Invoke event to show error.
      OnError.Invoke(new ErrorEventArgs(errorMessage, ex));
   }
}
```

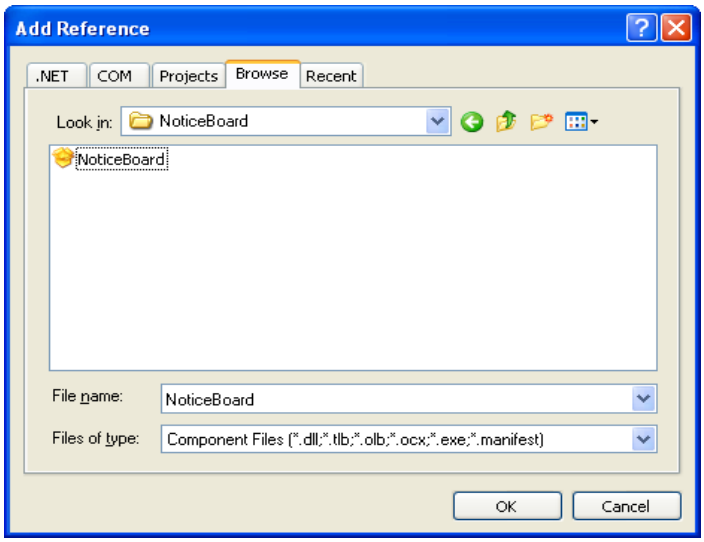
3.6. Instalacija

1. Preduslovi instalacije Playlist kontrole

Slika 13: Preduslovi instalacije Playlist kontrole

2. Dodavanje reference

Korisnik dobija Playlist kontrolu u vidu .dll fajla, *NoticeBoard.dll*. Da bi mogao da koristi atribute i metode definisane kontrolom, neophodno je da dobijeni .dll fajl uključi u svoj projekat. Desnim klikom na *References* u aktivnom projektu otvara se prozor za izbor reference. U okviru *Browse* taba pronaći lokaciju na kojoj se nalazi .dll fajl, selektovati ga i kliknuti *OK*.



Slika 14: Dodavanje reference na Playlist kontrolu

3. Postavljanje kontrole na WPF stranu

Dodati namespace kontrole:

```
xmlns:NoticePlaylist="clr-
namespace:NoticeBoard.UserControls;assembly=NoticeBoard"
```

Dodati kontrolu:

```
<NoticePlaylist:NoticePlaylist x:Name="noticePlaylist"
      Height="700" Width="300"
      SelectedItemChanged="OnSelectedItemChanged"
      CurrentItemChanged="OnCurrentItemChanged"
      OnError="NoticePlaylist_OnError">
</NoticePlaylist:NoticePlaylist>
```

Prilikom dodavanja kontrole korisnik se "pretplatio" na događaje koji ga obaveštavaju o:

- Selektovanju nekog oglasa *SelectedItemChanged*
- Oglasima koji trebaju da se prikažu na tabli *CurrentItemChanged*
- Greški koja se dogodila u aplikaciji *OnError*

4. Korišćenje kontrole u programskom jeziku C#

- Definisanje liste oglasa:

```
List<INotice> showList = new List<INotice>();
```

- Definisanje pojedinačnog oglasa i njegovo dodavanje u listu:

```
Notification n1 = new Notification();

n1.Id = 1;
n1.Title = "Softverski inzenjering 1 - prijava za ucesce na
projektu";
n1.Author = "Adam Stanojevic";
n1.Date = DateTime.Today;
n1.EstimateTime = 10;
showList.Add(n1);
```

gde *Notification* predstavlja klasu koja implementira *INotice* interfejs.

- Postavljanje *DataSource*-a i osnovnih podešavanja liste:

```
noticePlaylist.DataSource = showList;

noticePlaylist.ChangeItemsInterval = 9;
noticePlaylist.ListBackground = Brushes.Red;
noticePlaylist.ItemBackground = Brushes.Green;
noticePlaylist.ListForeground = Brushes.Black;
noticePlaylist.ItemForeground = Brushes.Yellow;
noticePlaylist.CurrentItemBackground = Brushes.DarkBlue;
```

- Pokretanje i stopiranje Playlist kontrole

```
noticePlaylist.Play(); noticePlaylist.Stop();
```

- Prikaz oglasa

```
void OnCurrentItemChanged(CurrentItemChangedEventArgs e)
{
    txtShow.Text = string.Empty;
    e.CurrentItems.ToList().ForEach(item =>
                                    txtShow.Text += " " +
                               item.Title);
}
```

- Dodavanje elementa u listu

```
noticePlaylist.Add(n2); noticePlaylist.Insert(n2, 3);
```

- Uklanjanje elementa iz liste

```
noticePlaylist.Remove(n2); noticePlaylist.RemoveAt(3);
```

- Obrada greške

```
private void NoticePlaylist_OnError(ErrorEventArgs e)
{
   MessageBox.Show(e.ErrorMessage);
}
```

Pored *ErrorMessage* korisnik može dobiti i originalnu grešku koja se dogodila upotrebom *e.OriginalException*;

3.7. Literatura

- Matthew A. Stoecker, *Microsoft .NET Framework 3.5 - Windows Presentation Foundation Training Kit 70-502,* Microsoft Press, 2008
- Matthew MacDonald, *Pro WPF in C# 2010 - Windows Presentation Foundation in .NET 4,* Apress, 2010
- Andrew Troelsen, *Pro Expression Blend 4*, Apress, 2011
- Dušan Stefanović, *Predavanja iz projektovanja programa i informacionih sistema*, Prirodno – matematički fakultet u Kragujevcu, 2008
- www.wpftutorial.net
- msdn.microsoft.com/en-us/library
- www.wikipedia.org
- www.google.com