Interakcija čovek – računar

Vežbe 7

Fakultet tehničkih nauka

Univerzitet u Novom Sadu

Uvod

Drag-and-Drop predstavlja mehanizam koji se koristi za prenos podataka izmedju struktura podataka u okviru iste aplikacije, ali ga je moguće koristiti i za prenos podataka izmedju struktura koje se nalaze u različitim aplikacijama. Sam postupak se svodi na prevlačenje objekta koji predstavlja podatke sa izvora (*drag source*) na odredište (*drop target*), uz pomoć miša.

Prenos podataka može da se odnosi na stvarni prenos podataka sa izvora na odredište (*move* - podatak se briše iz strukture koja predstavlja izvor i dodaje u strukturu koja predstavlja odredište), ili samo kopiranje podatka sa izvora na odredište (*copy* - sadržaj izvora se ne menja, dok se u odredište dodaje kopija podatka). U postojećim aplikacijama, tip prenosa podatka pri *Drag-and-Drop*-u zavisi od konteksta. WPF takođe podržava oba tipa prenosa podataka putem *Drag-and-Drop*-a.

Drag-and-Drop u WPFu

Većina kontrola u WPFu podržava *Drag-and-Drop* mehanizam. Funkcije za implementaciju samog mehanizma se nalaze u **`DragDrop`** klasi, i biće detaljnije objašnjene u delu dokumenta koji se bavi primerom upotrebe.

Već je ranije pomenuto da se *Drag-and-Drop* vrši tako što podatak (to jest', *UI* element koji ga predstavlja) prevlačimo sa izvora na odredište. Izvor i odredište će biti neke UI komponente kojima su pridružene strukture podataka (*ListView bind*-ovan na *ObservableCollection*, i slično). Za realizaciju *Drag-and-Drop*-a, doduše, neophodna je još jedna komponenta - potrebna nam je struktura za privremeni smeštaj podataka dok se oni prebacuju iz izvora u odredište. Za ovu namenu, WPF pruža klasu **`DataObject`**, koja se može koristiti kao kontejner za podatke koji se prenose. `DataObject`, pored prenosa objekta, dozvoljava nam i specifikaciju *formata*, odnosno *tipa* podataka koji se prenose u DataObjectu-u (putem parametara konstruktora `String` *`format`*, odnosno `Type` *`type`*). Metoda za inicijalizaciju prenosa podataka putem *Drag-and-Drop*-a, **`DoDragDrop`**, implicitno pakuje prosleđene podatke u objekat `DataObject` klase, ali se to može uraditi i ručno, pre poziva `DoDragDrop` metode, ukoliko želimo eksplicitno specificirati *tip* ili *format* podataka koji se prenose. `DoDragDrop` metoda, kao inicijalizator *Drag-and-Drop* mehanizma, poziva se iz *izvora*, i to najčešće u okviru `MouseMove` eventa.

Za ostvarivanje komunikacije između izvora i odredišta, koriste se *Drag-and-Drop efekti*. Pri inicijalizaciji *Drag-and-Drop*-a uz pomoć `DoDragDrop` metode, jedan od parametara koji se prosleđuje jeste i `DragDropEffects` *`allowedEffects`*, koji služi za specifikaciju dozvoljenih efekata, odnosno dozvoljenih tipova prenosa podataka. Na primer, ako preko ovog parametra dozvolimo samo *`Move`* efekte, podatke prevučene iz izvora biće moguće isključivo *preneti* u odredište (ne i kopirati). Treba primetiti da, kao što je pomenuto na početku pasusa, efekti samo omogućavaju *komunikaciju* između izvora i odredišta - sam prenos podataka moramo implementirati ručno, i to se obično radi na odredištu.

Prihvatanje prenetog podatka u odredištu radi se u okviru `Drop` eventa. Podaci se čitaju iz `DataObject` objekta, i potom se realizuje stvarni prenos podataka između struktura podataka koje su pridružene

izvoru i odredištu. Ovo najverovatije znači da se podaci uklanjaju iz jedne strukture podatka (neke liste, na primer) i unose u drugu.

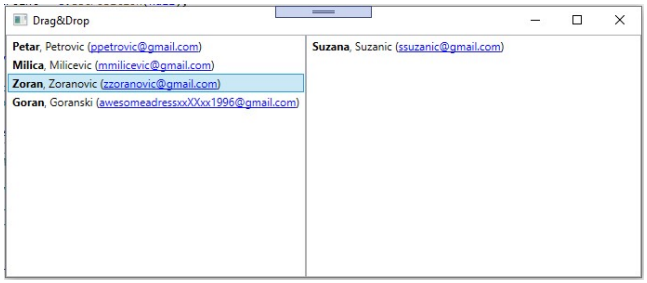
U okviru pomenutog **Drop** eventa, moguće je postaviti vrednost *Effects* polja u DragEventArgs argumentu eventa, i ta vrednost će biti vraćena kao povratna vrednost funkcije DoDragDrop u izvoru, te se ista može koristiti za dodatnu kontrolu celog *Drag-and-Drop* mehanizma (na primer, ukoliko smo neposredno pre poziva DoDragAndDrop metode već uklonili element iz liste, i povratna vrednost DoDragAndDrop metode nije željena vrednost, možemo vratiti uklonjeni element nazad u listu).

Postoji još nekoliko eventova koji mogu biti korisni pri implementaciji *Drag-and-Drop*-a, a to su DragEnter, DragLeave, DragOver i GiveFeedback eventovi. **DragEnter** i **DragLeave** se pozivaju pri prevlačenju podataka unutar granica kontrole koja predstavlja odredište, odnosno pri prevlačenju podataka van granica te kontrole. Pomenuti eventovi se mogu koristiti za pružanje dodatnog *feedback*-a korisniku po pitanju efekata koje će *Drag-and-Drop* operacija da ima po svom završetku. **DragOver** se koristi da, postavljanjem vrednosti *Effects* polja u DragEventArgs argumentu eventa, obavestimo izvor o efektima koje odredište prihvata, pri prevlačenju podataka unutar granica kontrole koja predstavlja odredište. Ovaj event je uparen sa **GiveFeedback** eventom u izvoru, koji u polju *Effects* argumenta DragEventArgs sadrži vrednost postavljenu u DragOver metodi odredišta, i koji se koristi za pružanje *feedback*-a korisniku po pitanju efekata koje odredište prihvata, putem različitih kursora miša. WPF već ima ugrađenu podršku da prikaže različite kursore u zavisnosti od efekata koje odredište prihvata (postavljenih u DragOver eventu), te GiveFeedback event ima smisla koristiti jedino kada želimo koristiti *custom* kursore.

Preduslov za funkcionisanje svih pomenutih komponenti *Drag-and-Drop* mehanizma WPFa jeste i postavljanje vrednosti AllowDrop polja odredišne kontrole na *True*.

Primer

Kod koji je priložen uz ovaj dokument sadrži jednostavnu implementaciju *Drag-and-Drop* mehanizma, na primeru dva *ListView*-a (*bind*-ovana na *ObservableCollection* liste) između kojih je moguće prevlačiti objekte klase *Student*. Zarad lakšeg praćenja primera i demonstracije koncepata *Drag-and-Drop* mehanizma, implementirano je prebacivanje objekata samo u jednom smeru. Za vežbu, možete probati implementirati *Drag-and-Drop* mehanizam nad ova dva *ListView*-a u oba smera.



Slika 1 - Primer Drag-and-Drop mehanizma

Kako je pomenuto u prethodnoj sekciji, *Drag-and-Drop* mehanizam se inicijalizuje pozivom DoDragAndDrop metode DragDrop klase. Iako bi očekivali da ovu metodu treba pozvati pri pritisku levog tastera miša nad elementom liste (event PreviewMouseLeftButtonDown nad *ListView*-om), to zapravo nije slučaj. Ako bi to uradili, *Drag-and-Drop* mehanizam bi se pokretao i u situacijama kada to ne želimo, kao kad, na primer, pokušavamo samo selektovati element liste. Umesto toga, pomenutu metodu treba pozvati u **MouseMove** eventu *ListView*-a, koji se inače poziva pri prevlačenju kursora miša preko kontrole. Iskoristićemo PreviewMouseLeftButtonDown event da sačuvamo *trenutnu* poziciju kursora kada je levi taster miša pritisnut (*listing 1*), a onda ćemo u MouseMove eventu proveriti da li je došlo do promene njegove pozicije (dok je levi taster još uvek pritisnut) (*listing 2*). Za proveru pomeraja kursora miša, razliku u pozicijama kursora poredimo sa promenljivama koje definišu minimalne pomeraje miša da bi se započela *Drag-and-Drop* operacija (i koji su postavljeni na nivou sistema).

```
51      private void ListView_PreviewMouseLeftButtonDown(
            object sender, MouseButtonEventArgs e)
52      {
53          startPoint = e.GetPosition(null);
54      }
```

listing 1

```
56  private void ListView_MouseMove(object sender, MouseEventArgs e)
57  {
58     Point mousePos = e.GetPosition(null);
59     Vector diff = startPoint - mousePos;
60
61     if (e.LeftButton == MouseButtonState.Pressed &&
62         (Math.Abs(diff.X) > SystemParameters.MinimumHorizontalDragDistance ||
63          Math.Abs(diff.Y) > SystemParameters.MinimumVerticalDragDistance))
64     {
            ...
```

listing 2

Nakon što su ovi uslovi ispunjeni, neophodno je pronaći element liste nad kojim se želi sprovesti *Drag-and-Drop* operacija. MouseEventArgs argument MouseMove eventa u sebi sadrži polje *OriginalSource*, i to polje nam može pomoći da pronađemo *ListViewItem* (koji je vezan za odgovarajući objekat klase *Student*) koji prenosimo (*listing 3*). Za razliku od *Source* polja, koje sadrži komponentu koja je pozvala event (u našem slučaju, *ListView*), *OriginalSource* polje sadrži komponentu koja je prva registrovala taj event. Radi se o tome da će event kao što je pritisak tastera miša biti prvo registrovan od strane komponente koja je *list* u stablu komponenata *GUI*-a - u našem slučaju to je mogao biti, na primer, *TextBox* sa imenom studenta. Kako je taj *TextBox* sadržan u *WrapPanel*-u, a *WrapPanel* sadržan u *ListViewItem*-u, vidimo da kretanjem kroz stablo *GUI* elemenata, od *OriginalSource*-a (*TextBox*-a, na primer), možemo doći do *ListViewItem*-a (kao jednog o njegovih naslednika). Za ovu potrebu, implementirana je generička metoda **FindAncestor<T>,** koja pronalazi naslednika tipa T za zadatu kontrolu *current* (*listing 4*).

```
66   ListView listView = sender as ListView;
67   ListViewItem listViewItem =
68      FindAncestor<ListViewItem>((DependencyObject)e.OriginalSource);
```

listing 3

```
82   private static T FindAncestor<T>(DependencyObject current)
         where T : DependencyObject
83   {
84      do
85      {
86           if (current is T)
87           {
88               return (T)current;
89           }
90           current = VisualTreeHelper.GetParent(current);
91       }
92       while (current != null);
93       return null;
94   }
```

listing 4

Treba uzeti u obzir da ova metoda može da vrati `null`, u slučaju da korisnik prevlači miša po *ListView*-u u delu gde nema *ListViewItem*-a, pa moramo rešiti i taj slučaj (*listing 5*).

```
70   if (listViewItem == null) return;
```

listing 5

Kada imamo traženi *ListViewItem*, koristimo **`ItemFromContainer`** iz `ItemContainerGenerator` klase, iz *ListView*-a, da pronađemo odgovarajući objekat klase *Student* za taj *ListViewItem* (*listing 6*).

```
73   Student student = (Student)listView.ItemContainerGenerator.
74        ItemFromContainer(listViewItem);
```

listing 6

Poslednji korak pre inicijalizacije *Drag-and-Drop* operacije jeste pakovanje objekta klase *Student* u **`DataObject`** objekat (*listing 7*). `DataObject` objekat, pored enkapsulacije objekat koji šaljemo, takođe nam dozvoljava i specifikaciju *tipa* objekta koji šaljemo, ili *formata* u obliku proizvoljnog stringa (kako je urađeno i u primeru, sa ciljem da kasnije možemo sprečiti *drop* na odredištu za objekte neodgovarajućeg formata; string `"myFormat"` u ovom primeru koristimo samo kao naziv za objekte tipa *Student* - jednako smo mogli koristiti i `typeof(Student)` umesto tog stringa).

```
77    DataObject dragData = new DataObject("myFormat", student);
```

listing 7

Pomenuti `DataObject` objekat se potom prosleđuje kao jedan od parametara **`DoDragAndDrop`** metode, zajedno sa *izvorom*, i specifikacijom dozvoljenih *efekata Drag-and-Drop* operacije (*listing 8*).

```
78      DragDrop.DoDragDrop(listViewItem, dragData, DragDropEffects.Move);
```

listing 8

Sa strane odredišta, pored postavljanja polja `AllowDrop` na vrednost `True` za odgovarajući *ListBox* (u primeru, to je odrađeno u *.xaml* fajlu na *liniji 32*), treba implementirati i `DragOver` i `Drop` eventove.

`DragOver` event koristimo samo da korisniku naznačimo nemogućnost drop-ovanja objekata koji nisu odgovarajućeg formata, ili potiču sa *odredišta* (jer želimo prihvatati samo objekte koji potiču sa *izvora*) (*listing 9*). Očigledno, *`Data`* polje u `DragEventArgs` argumentu eventa sadrži prenete podatke (*Student* zapakovan u `DataObject`), i koristimo metodu **`GetDataPresent`** da proverimo da li se u `DataObject` objektu nalazi zapakovan objekat traženog formata. Podrazumevana implementacija **`GiveFeedback`** eventa će postaviti odgovarajući *kursor* spram vrednost koju postavimo u `DragOver` eventu. Trenutno, naš primer ima samo jednu kontrolu iz koje je moguće započeti *Drag-and-Drop* operaciju (levi *ListBox*), pa efekti `DragOver` eventa koji smo implementirali verovatno nisu odmah uočljivi. Svakako, sa većim brojem *izvora Drag-and-Drop* operacija, i efekti `DragOver` eventa bi postali primetni, jer bi ga koristili da odlučujemo koje *tipove* i *formate* objekata, i iz kojih *izvora*, prihvatamo u *odredištu*.

```
96      private void ListView_DragOver(object sender, DragEventArgs e)
97      {
98          if (!e.Data.GetDataPresent("myFormat") || e.Source == sender)
99          {
100             e.Effects = DragDropEffects.None;
101         }
102     }
```

listing 9

Sam prenos podataka se realizuje u `Drop` eventu odredišta (*listing 10*). Nakon provere *formata* prenetog objekta, i njegovog *kastovanja* u objekat klase *Student*, ručno moramo ukloniti datog studenta iz prve liste, i uneti ga u drugu.

```
104     private void ListView_Drop(object sender, DragEventArgs e)
105     {
106         if (e.Data.GetDataPresent("myFormat"))
107         {
108             Student student = e.Data.GetData("myFormat") as Student;
109             Studenti.Remove(student);
110             Studenti2.Add(student);
111         }
112     }
```

listing 10

Dodante informacije o *Drag-and-Drop* mehanizmu u WPFu možete pronaći na:

https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/drag-and-drop