Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.41
DOI: https://doi.org/10.24867/08BE30Topolic

OPTIMIZACIJA RADA WPF APLIKACIJA UPOTREBOM VIRTUALIZACIJE

OPTIMIZATION OF WPF APPLICATIONS USING VIRTUALIZATION METHODS

Jovan Topolić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu predstavljene su metode koje je moguće primeniti u WPF aplikacijama radi poboljšanja njihovih performansi. Akcenat rada je stavljen na optimizacione metode virtualizacije, koje poboljšavaju performanse prilikom rada sa velikim setovima podataka.. Postoje dve vrste virtualizacije: UI virtualizacija i virtualizacija podataka. Prikazan je način postizanja najboljih performansi aplikacije primenom ovih optimizacionih metoda.*

Ključne reči: *Optimizacija, virtualizacija, velike kolekcije podataka, WPF*

Abstract – *This paper presents the optimization methods that can be utilised in WPF applications for improving application performance. Focus of this paper is on optimization methods called virtualization, which help improve performance when dealing with large data sets. There are two types of virtualization: UI virtualization and Data virtualization. It shows the way of achieving best application performance when using these optimizations methods.*

Keywords: *Optimization, virtualization, large data collections, WPF*

1. UVOD

WPF (*Windows Presentation Foundation*) je *UI* (*User Interface*) radni okvir (*framework*) koji se koristi za izradu desktop aplikacija, a razvijen od strane *Microsoft*-a. Prilikom razvoja *WPF* aplikacije potrebno je poseban osvrt napraviti na način na koji se aplikacija implementira, to jest potrebno je razmišljati o mogućim načinima optimizacije koji bi doveli do boljeg iskorišćenja resursa i samim tim i boljeg rada aplikacije. Postoji veliki broj različitih načina optimizacije, a ovaj rad najveći akcenat stavlja na vid optimizacije koji se naziva virtualizacija.

Postoje dva tipa virtualizacije kada se govori o *WPF* aplikacijama, a to su *UI* virtualizacija i virtualizacija podataka (*data virtualization*). Oba ova vida virtualizacije se odnose na situacije kada je potrebno prikazati veliki set podataka u okviru korisničkog interfejsa.

Osnovna ideja je učitavanje samo onih podataka koji će biti vidljivi na ekranu i tako smanjiti negativan uticaj na performanse učitavanjem celokupne kolekcije podataka.

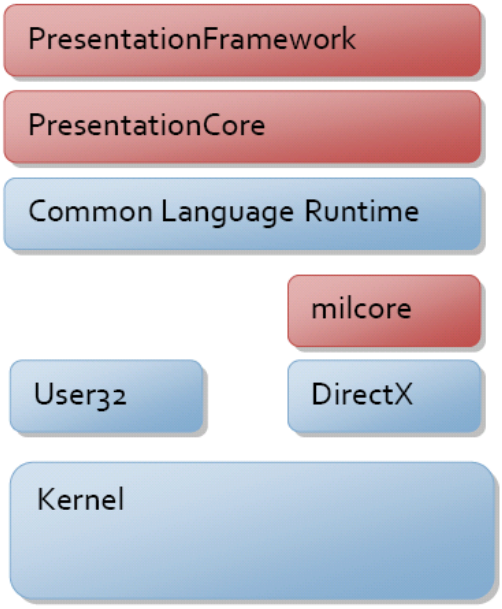
NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Darko Čapko, vanr. prof.

2. TEORIJSKE OSNOVE

2.1. *WPF*

WPF (*Windows Presentation Foundation*) je *Microsoft*-ova tehnologija za kreiranje grafičkih korisničkih interfejsa. Moguće je kreirati širok spektar aplikacija, od onih koje koriste bazične forme pa sve do bogatih 3D okruženja. On je takođe i osnova *Silverlight*-a koji je proširio *WPF* tehnologiju na *Web* i na uređaje kao što su *Windows* mobilni telefoni [1].

Arhitektura *WPF*-a je prikazana na slici 1 gde crvene sekcije na dijagramu predstavljaju *WPF* komponente, dok plave sekcije predstavljaju *Windows* komponente. Sav prikaz u *WPF*-u se vrši pomoću *DirectX engine*-a, koji omogućava efikasno hardversko i softversko iscrtavanje.



Slika 1. *WPF arhitektura* [2]

Korišćenje *WPF*-a ima mnoge prednosti a nabrojane su najvažnije [1]:

- Nezavisnost od rezolucije ekrana – pod ovim se misli na to da će korisnički interfejs izgledati bolje čak i na ekranima sa niskom rezolucijom od korisničkog interfejsa razvijenog u *Windows Forms applications*, a to je zbog toga što *WPF* koristi *DirectX* komponente.
- Kontrola unutar kontrole – *WPF* omogućava da se definiše kontrola kao sadržaj neke druge kontrole kao što je recimo *Button*.
- Dostupnost različitih *layout*-ova – *layout* se koristi da se logički razgraniče kontrole na

korisničkom interfejsu i da se uredno prikažu na prozoru. U *WPF*-u neke od *layout* kontrola su: *StackPanel*, *WrapPanel*, *DockPanel*, *Grid*, *Canvas*.

- 2D, 3D grafika i animacije – u *WPF*-u se mogu koristiti animacije, multimedijalne datoteke i grafika. Podržano je puštanje video ili audio datoteka.
- *Data binding* – prednost *WPF*-a u odnosu na *Windows Forms applications* je ta što se ne mora voditi računa ručno o sinhronizaciji podataka između izvora podataka i *UI* elementa.

2.2. *XAML*

XAML (*Extensible Application Markup Language*) je dizajniran kao efikasan jezik za razvoj korisničkog interfejsa aplikacije [1]. Specifična prednost koju *XAML* donosi *WPF* aplikacijama je to što je on u potpunosti deklarativan jezik, time dopuštajući *developer*-u ili dizajneru da opiše ponašanje i integraciju komponenti bez korišćenja proceduralnog programiranja. *XAML* se ekstenzivno koristi u *.NET Framework* verzijama 3.X i 4.X, posebno u *WPF*-u. *XAML* formira *markup* jezik korisničkog interfejsa za definisanje *UI* elemenata, *data binding*-a, događaja i ostalih mogućnosti.

Sve što se kreira i implementira u *XAML* jeziku takođe je moguće izraziti koristeći tradicionalnije *.NET* jezike, kao što je *C#*. Međutim, ključni aspekt *XAML* tehnologije je smanjena kompleksnost koja je potrebna alatima da procesira *XAML*, jer je baziran na *XML* jeziku [1]. Postoje četiri glavne implementacije *XAML* tehnologije :

- Verzija namenjena za *WPF*, koja se koristi od *.NET Framework* 3.0 i nadalje.
- *Silverlight* 3 verzija.
- *Silverlight* 4 verzija.
- *Windows* 8 *XAML*/*Jupiter* verzija.

3. UI I VIRTUALIZACIJA PODATAKA

Virtualizacija je centralna tema ovog rada, te je potrebno detaljno opisati koncept na kome se ona zasniva. Kao što je napomenuto u uvodu razlikujemo dve vrste virtualizacije: virtualizacija korisničkog interfejsa (*UI* virtualizacija) i virtualizacija podataka. U oba slučaja radi se o optimizaciji prilikom rada sa velikim kolekcijama podataka tako što se vrši učitavanje samo onih podataka koji su vidljivi na ekranu.

3.1 *UI* virtualizacija

WPF kontrole kao što su *ListView* i *ComboBox* se koriste da prikažu listu podataka u aplikaciji. Ukoliko je lista podataka velika, može doći do pada performansi aplikacije. Ovo se dešava zbog toga što standardni *layout* sistem kreira *layout* kontejner za svaki podatak koji je asociran sa *UI* kontrolom [3], i računa njegovu *layout* veličinu i poziciju. Tipično nije potrebno prikazivati sve podatke u isto vreme, već se prikazuje podskup podataka a korisnik se najčešće kroz *scroll* mehanizam kreće kroz listu.

U ovakvim slučajevima ima smisla koristiti *UI* virtualizaciju, što znači da će se generisanje i komputacije za kontejner podataka (*item container*) odložiti dok podatak ne postane vidljiv na ekranu.

Container recycling je optimizacija dodata *UI* virtualizaciji u *.NET Framework* verziji 3.5 za kontrole koje nasleđuju *ItemsControl*, takođe ova optimizacija može poboljšati performanse kod *scroll* mehanizma. Kada se *ItemsControl* koja koristi *UI* virtualizaciju puni podacima, ona kreira *item* kontejner za svaki podatak koji se skroluje u prikaz i uništava *item* kontejner za svaki item koji se skroluje van prikaza [3]. *Container recycling* omogućava *UI* kontrolama da ponovo iskoriste postojeće kontejnere za različite podatke, time se *item* kontejneri ne kreiraju i ne uništavaju konstantno prilikom skrolovanja. Ova metoda optimizacije se omogućava postavljanjem vrednosti *attached property*-a *VirtualizationMode* na *Recycling*. *Deferred scrolling* (odloženo skrolovanje) predstavlja odloženo učitavanje podataka prilikom skrolovanja. Po *default*-u, kada korisnik vuče *thumb* na *scrollbar*-u, prikaz sadržaja se kontinualno ažurira. Ako je skrolovanje sporo u *UI* kontroli, može se iskoristiti *deferred scrolling*. Ovakvim pristupom sadržaj se ažurira samo kada korisnik otpusti *thumb*. Da bi se omogućio ovaj tip skrolovanja potrebno je podesiti *property IsDeferredScrolling* na *true.*

.NET Framework verzija 4.5 je uvela značajne promene koje olakšavaju rad sa velikim setovima podataka kao i poboljšane opcije prilikom *scroll*-a dok je virtualizacija uključena. Jedna od novina je da je od ove verzije omogućena virtualizacija prilikom grupisanja podataka (*grouping*) i to pomoću *IsVirtualizingWhenGrouping attached property*-a. U prošlim verzijama je takođe bio prisutan problem kod skrolovanja u smislu da ukoliko je bilo potrebno da se virtualizacija koristi, moralo se koristiti i logičko skrolovanje koje se podešava postavljanjem *CanContentScroll property*-a na *true*. Ovo je bilo problematično u slučaju kada se u *ListView*-u kao podaci nalaze ekspanderi koji prilikom ekspandovanja zauzimaju veliki vertikalni prostor prikaza. U ovakvom slučaju dolazi do naglih skokova između elemenata što korisniku predstavlja loše iskustvo prilikom korišćenja aplikacije. U *.NET Framework* verziji 4.5 ovaj problem je prevaziđen uvođenjem *ScrollUnit attached property*-a koji je moguće koristiti sa *Virtualizing* panelom. *ScrollUnit* ima dve moguće vrednosti: *Item*(*default*) koji suštinski predstavlja logičko skrolovanje element po element i *Pixel* koji nam upravo omogućava skrolovanje zasnovano na pikselima bez isključivanja virtualizacije.

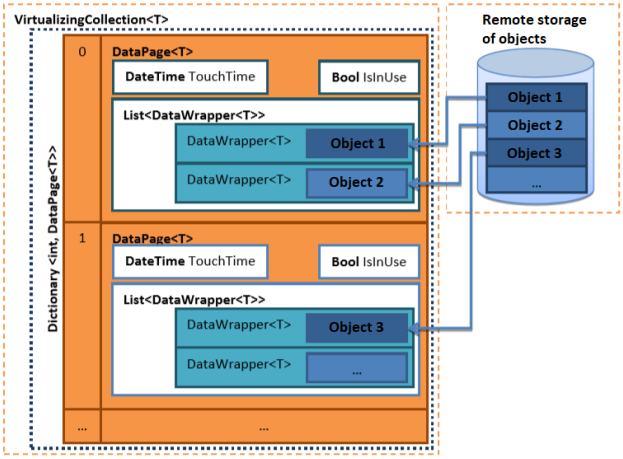
3.2 Virtualizacija podataka

Ukoliko se povećava broj podataka u listi, performanse će sve više opadati zbog toga što aplikaciji postaje teže da manipuliše podacima, sve je više memorije potrebno za njihovo skladištenje i mnogo vremena za učitavanje podataka. Ovaj problem se rešava upotrebom virtualizacije podataka.

Virtualizacija podataka je mehanizam koji omogućava aplikaciji da ne skladišti sve podatke odjednom u memoriji, već samo neophodan deo. Na ovaj način količina memorije koja se koristi za skladištenje podataka u klijentskoj aplikaciji ostaje mala i konstantna za bilo koji broj podataka u kolekciji. *WPF* je dizajniran za pravljenje korisničkog interfejsa i nije vezan za metode skladištenja i procesovanja podataka, te ne postoji ugrađena podrška za virtualizaciju podataka unutar *WPF*-a. Stoga je potrebno osmisliti način implementacije virtualizacije podataka. Virtualizacija podataka se vrši

tako što se kreira specijalna kolekcija objekata, koja sadrži samo deo elemenata i učitava objekte iz izvora podataka samo kada oni postanu potrebni *UI* kontroli. Ovaj mehanizam se može implementirati na različite načine, virtualizacioni mehanizmi se razlikuju u načinu učitavanja podataka. Ideja je da se u memoriji čuva samo deo podataka koji je vidljiv korisniku ali se interakcija nad podacima ponaša kao da su svi podaci učitani. *Scroll bar* indikator će uvek pokazivati poziciju prikazanih podataka u punoj kolekciji elemenata, što dalje omogućava korisniku da može da se pozicionira u bilo kojem delu pune kolekcije. Kolekcija koja virtualizuje podatke treba da simulira skladištenje svih elemenata, na primer kada *UI* kontrola zahteva broj podataka u listi, kolekcija treba da vrati broj objekata u punoj kolekciji [4].

Preporučuje se da virtualizaciona kolekcija koja se dizajnira koristi *IList<T>* interfejs kao bazni. *IList* je interfejs kolekcija objekata kojima se pristupa pomoću indeksa, jer pruža najbolje performanse prilikom *data binding*-a podataka na *WPF* kontrole. Javlja se pitanje skladištenja podataka u virtualnoj kolekciji. Izvor podataka se deli na stranice (*pages*), fragmente fiksirane veličine. Kada eksterni kod traži element kolekcije, učitaće se stranica na kojoj je lociran i vratiće se željeni element. Ove akcije se izvršavaju pomoću indeksera kolekcije. Osim učitavanja stranica, takođe se moraju i brisati one koje se ne koriste više od strane aplikacije. Da bi se ovo postiglo, svaka stranica bi imala informaciju o poslednjem pristupu. U određenom trenutku, proteklo vreme od poslednjeg pristupa stranici bi se računalo. Ukoliko bi ovo proteklo vreme premašilo neki zadati prag, stranica bi se obrisala iz kolekcije [4]. Na slici 2 prikazana je šema za skladištenje podataka u virtualizujućoj kolekciji.



Slika 2. *Šema za skladištenje podataka u virtualizujućoj kolekciji* [4]

4. OPIS PROBLEMA

Problem koji se javlja kao tema ovoga rada je kako optimizovati *UI* kontrolu koja služi za prikaz kolekcije podataka u *WPF*-u da bi negativan uticaj na performanse bio što manji. Potrebno je iskoristiti opcije *UI* virtualizacije na najbolji način, kao i pružiti sopstvenu implementaciju mehanizma virtualizacije podataka, jer u *WPF*-u ne postoji ugrađena podrška za nju. U današnjem vremenu sve veći akcenat se stavlja na brzinu rada aplikacija te je ovaj problem veoma aktuelan.

5. REŠENJE PROBLEMA

Za rešenje prethodno opisanog problema napravljena je test aplikacija u *WPF*-u i *C#* programskom jeziku koja će na primeru *ListView* kontrole prikazati na koje načine upotrebom virtualizacionih metoda možemo doći do optimalnih performansi same kontrole. Metode koje će biti opisane je moguće primeniti na bilo koju *UI* kontrolu koja nasleđuje *ItemsControl* te nije strogo vezana za *ListView*. Sama aplikacija se sastoji od virtualizacionih opcija koje je moguće podesiti preko korisničkog interfejsa, kako onih vezani za *UI* virtualizaciju tako i za virtualizaciju podataka, kao i *ListView* kontrole koja se generiše u zavisnosti od odabranih opcija. Takođe, najbitnije informacije koje dobijamo upotrebom aplikacije su potrošnja memorije u megabajtima kao i vreme koje je bilo potrebno prilikom generisanja *ListView* kontrole kao i prilikom skrolovanja kroz nju.

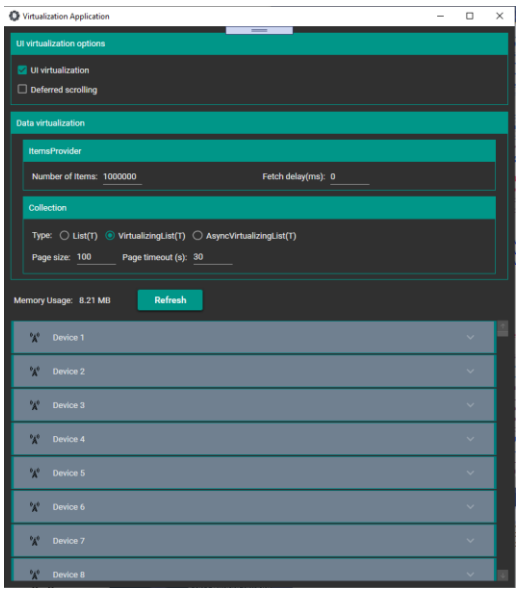
5.1 Implementacija UI virtualizacije

Potrebno je napomenuti da od *UI* virtualizacije nije podržano menjanje *ContainerRecycling* svojstva *ListView* kontrole, zbog toga što se pomenuti *property* ne može menjati nakon *Measure* poziva za sam *ListView*. Ukoliko je potrebna promena vrednosti za *ContainerRecycling* mora se definisati u *XAML*-u jer nije podržana njena promena u *runtime*-u. Dakle od *UI* virtualizacije moguće je konfigurisati da li je *UI* virtualizacija omogućena kao i da li je omogućen *DeferredScrolling* (odloženi *scroll*), koji značajno može pomoći brzini skrolovanja ukoliko nam nije potrebno kontinualno ažuriranje sadržaja prilikom samog skrolovanja. Podrška za *UI* virtualizaciju je ugrađena u *WPF* te nema značajnog napora za njeno omogućavanje. Performanse su najbolje kada je uključena *UI* virtualizacija a *ContainerRecycling* postavljen na *Recycling* vrednost, dok *DeferredScrolling* može pomoći performansama *scroll*-a ukoliko nam nije potrebno kontinualno ažuriranje sadržaja.

5.2 Implementacija virtualizacije podataka

Što se tiče ponuđenih opcija za virtualizaciju podataka prvo što možemo definisati je broj podataka koji će se nalaziti u izvoru podataka, što veći broj podataka to će biti vidljiviji benefiti samih mehanizama virtualizacije. *Fetch delay* poljem možemo simulirati vreme potrebno za dobavljanje podataka u milisekundama, kao recimo iz baze podataka. Proces dobavljanja podataka iz nekog *remote* izvora može biti vremenski zahtevan te ovde dolazi do izražaja mehanizam virtualizacije podataka koji limitira broj podataka koje je potrebno dobaviti i instancirati u memoriji. Dalje opcije se tiču samog izbora kolekcije koji će se vezati za *UI* kontrolu. Odabirom *List<T>* opcije virtualizacija podataka se ne koristi te će se ceo izvor podataka instancirati odjednom u listi. Opcije
- *VirtualizingCollection<T>* i
- *AsyncVirtualizingCollection<T>*

predstavljaju kolekcije koje podržavaju virtualizaciju podataka. Poslednje dve konfigurabilne opcije se tiču definisanja veličine stranice i vremena isteka stranice za virtualizovanu kolekciju. Na slici 3 prikazan je korisnički interfejs test aplikacije.



Slika 3. *Korisnički interfejs test aplikacije*

Da bi se iskoristilo rešenje za virtualizaciju podataka, izvor podataka mora biti u mogućnosti da pruži ukupan broj podataka unutar kolekcije, i mora biti u mogućnosti da pruži fragmente (stranice) od cele kolekcije. Ovi zahtevi su enkapsulirani u *IItemsProvider* interfejsu. *VirtualizingCollection<T>* je *IList* implementacija koja vrši virtualizaciju podataka. Ona deli čitavu kolekciju u određeni broj stranica. Stranice se zatim učitavaju u memoriji po potrebi, i otpuštaju kada više nisu potrebne. *AsyncVirtualizingCollection<T>* klasa je izvedena iz *VirtualizingCollection<T>* klase, i *override*-uje *Load* metode da bi se učitavanje podataka vršilo asinhrono. Ključna stvar kod asinhronog izvora podataka u *WPF*-u je ta da se *UI* mora obavestiti pomoću *data binding*-a kada se podaci dobavljaju. Kod regularnog objekta, ovo se postiže sa *INotifyPropertyChanged* interfejsom. Kod implementacija kolekcije je neophodno koristiti *INotifyCollectionChanged*. Stranice se skladište u *Dictionary*, gde se indeks stranice koristi kao ključ. Još jedan *Dictionary* se koristi da skladišti podatke o poslednjem pristupu stranicama.

5.3 Rezultati testiranja

Prilikom izvršavanja generisanja *ListView* pomoću opisane aplikacije, a pri korišćenju različitih virtualizacionih opcija dobijamo rezultate prikazane na tabeli 1.

Tabela 1. *Rezultati testiranja*

UI virt.	Br. Podataka [10^6]	Tip kolekcije	Veličina stranice [br.el]	Istek stranice [s]	Potrošnja memorije [MB]
NE	0.02	List	/	/	1586
DA	1	List	/	/	2175
DA	1	Virt.List	100	30	12
DA	1	AsyncVirt.List	100	30	10

Primećuje se da upotrebom kombinacije UI virtualizacije i virtualizacije podataka se dolazi do najboljih rezultata potrošnje memorije, te da korišćenje virtualizacije uopšteno može drastično poboljšati performanse aplikacije.

6. ZAKLJUČAK

Opisana je test aplikacija kojom se meri potrošnja memorije prilikom korišćenja raznih opcija *UI* virtualizacije i virtualizacije podataka, kao i vreme potrebno za generisanje *ListView* kontrole i vreme potrebno da se prođe kroz celu listu pomoću *scroll* mehanizma, te je izvršena analiza koje to opcije dovode do najboljih performansi aplikacije. Sama aplikacija nema neku realnu primenu već samo služi kao test scenario za pokazivanje performansi prilikom korišćenja različitih opcija. Ono što je primenjivo je sama implementacija mehanizma virtualizacije podataka koja može značajno pomoći kada aplikaciji nije potrebno da učita celu kolekciju podataka odjednom. Takođe ima i dosta prostora da se sam mehanizam virtualizacije podatka unapredi. Rešenje koje je prezentovano smatra da je izvor podataka *read only* i da se ne menja. Moguće je proširiti rešenje tako da periodično ili na zahtev ponovo učitava *Count* i stranice. Postoji prostor za proširenje funkcionalnosti i u *IItemsProvider* interfejsu u smislu da podržava na primer izmene i sortiranje.

7. LITERATURA

[1] A.Nathan, "*WPF 4.5 Unleashed*", Sams Publishing, August 2013.
[2] https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/wpf-architecture (pristupljeno u novembru 2019.)
[3] S.Yuen, "*Mastering Windows Presentation Foundation*", Packt Publishing, February 2017.
[4] A.S. Chajkovskaja, K. Ya. Kudrjautsev, E. K. Pavlov, "Displaying large sets of dynamically changing remote data in WPF applications", *International Journal of Open Information Technologies*, vol. 5, no. 9, pp.13-20, 2017.

Kratka biografija:

Jovan Topolić rođen je 03.09. 1995. godine u Novom Sadu. Završio je Osnovnu školu "Đorđe Natošević" u Novom Sadu 2010. god. Gimnaziju "Svetozar Marković" završio je 2014. godine i iste upisao Fakultet tehničkih nauka, odsek Računarstvo i automatika. Školske 2016/2017. godine upisao je smer Primenjene računarske nauke i informatika, a u osmom semestru se opredeljuje za modul Internet i elektronsko poslovanje. Nakon toga upisuje master studije školske 2018/2019. godine na Fakultetu tehničkih nauka, smer Primenjeno softversko inženjerstvo, na odseku Računarske nauke i informatike.