UDC: 004.2 INFO M: str. 38-45

IZRADA MODULARNE LOB SILVERLIGHT APLIKACIJE
BUILDING OF MODULAR LOB SILVERLIGHT APPLICATION

Igor Pantelić, Slobodan Jovanović
igorpantelic@gmail.com , slobodan.jovanovic@metropolitan.ac.rs
Metropolitan univerzitet, Beograd, www.metropolitan.edu.rs

REZIME: Silverlight tehnologija se koristi za izradu LOB (Line of bussiness) aplikacija. Kod Silverlight tehnologije, LOB aplikacija se izvršava u Web pretraživaču, pri čemu se cela aplikacija nalazi na radnoj stanici (radna stanica „klijenta“). U ovom radu se opisuje primena Silverlight tehnologije za izradu jedne složene LOB aplikacije, aplikacije koja se koristi u poslovnicama banaka, gde su ove poslovnice rasporedjene na mnogo različitih i udaljenih lokacija. U nastavku teksta će biti prvo opisana Silverlight tehnologija, kao i problemi koji se javljaju prilikom razmeštanja Silverlight aplikacije, a takođe će biti opisana i rešenja za probleme koji se tiču razmeštanja. Zatim će biti opisana izrada jedne Silverlight LOB aplikacije koja se, iz ugla razmeštanja sistema, odlikuje: modularnošću, brzim startovanjem.
KLJUČNE REČI: Poslovnice banaka, Razmeštanje aplikacije, Dijagram klasa
ABSTRACT: Silverlight technology is used for building of LOB (Line of bussiness) applications. In this technology, an LOB application is executed in Web browser, where the application is deployed on client work station. This paper describes an application of Silverlight technology for building of an complex LOB application, which is used in bank branches, distributed on manu distant locations. After describing Silverlight technology, and the problems associated with deployment of this technology, the solutions of these problems of the application deployment will be discussed. And the constrction of an Silverlight LOB application will be described, characterized by: modularity, and fast starting.
KEY WORDS: Bank branches, Application deployment, Class diagram

UVOD

LOB (Line-of-business) aplikacijLOB applications.e predstavljaju aplikacije koje su ključne za rad organizacije. U slučaju banke to može biti aplikacija za vođenje knjigovodstva, ili aplikacija koja se koristi na šalteru. U slučaju organizacije koje se bavi trgovinom, CRM aplikacija (Customer Relationship Management) bi predstavljala LOB aplikaciju. U svakom slučaju, LOB aplikacija predstavlja softver koji igra ključnu ulogu u informacionom sistemu organizacije. Osnovne karakteristike LOB aplikacija su [1]:

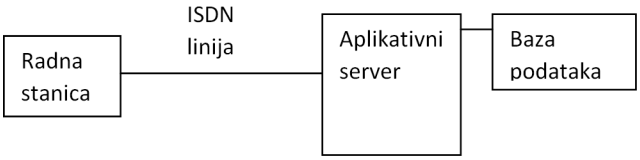
- Interaktivnost (Interactivity) – u direktnom su kontaktu sa korisnicima. Korisnici unose odgovarajuće informacije u aplikaciju i dobijaju određene povratne informacije iz aplikacije
- Sastoje se iz “delova” (Composable) – LOB aplikacije često imaju više delova koji sačinjavaju korisnički interfejs. Ovi delovi mogu međusobno komunicirati.
- Usko vezana za podatke (Data-Driven) – imaju visok stepen razmene informacija za bazama podataka
- Integrisanost (Integrated) – LOB aplikacija je često povezana sa drugim sistemima (unutar organizacije ili van nje)
- Proširivost (Extensible) – podrška plug-in modulima u cilju proširenja funkcionalnosti ili lakše distribucije aplikacije

LOB aplikacija ima troslojnu arhitekturu: baza podataka, aplikativni server, i radna stanica, kao što je prikazano na slici 1. LOB aplikacija predstavlja softversko rešenje koje će biti korišćeno od strane nekoliko stotina korisnika (radnih stanica) unutar organizacije. Radne stanice se nalaze na nekoliko stotina različitih fizičkih lokacija, a mrežna struktura je ostvarena preko ISDN linije. Razmeštanje LOB aplikacije podrazumeva spuštanje instalacione verzije softvera na radnu stanicu i njeno izvršenje. Pri tome, mogući problemi su: neuspešno ili sporo kopiranje softvera na odredjenu lokaciju, pad mrežne veze, itd. Problem razmeštanja aplikacije je od ogromne važnosti, jer ako se ne reši ovaj problem na adekvatan način, može se doći u situaciju da se ne može LOB aplikacija razmestiti kod svih radnih stanica, čime ona postaje neupotrebčliva. Razmeštanje LOB aplikacije mora se obavljati brzo, inače njena funkcionalnost se dovodi u pitanje.

Uzimajući u obzir prirodu LOB aplikacija, ove aplikacije najčešće sadrže veliki broj funkcionalnosti, što ima za posledicu da je veličina ovih aplikacija velika. LOB aplikacije danas, ne retko, imaju veličinu od nekoliko desetina megabajta. Razmeštanje (deployment) LOB aplikacije na sve radne stanice unutar organizacije predstavlja poseban izazov i zahteva pažljivo planiranje već u fazi dizajna. Pri tome, uvođenje modularnosti se pokazuje kao imperativ u razvoju ovog tipa aplikacija.

Silverlight tehnologija se koristi za izradu LOB aplikacija [1]. Kod Silverlight tehnologije, LOB aplikacija se izvršava u Web pretraživaču, pri čemu se cela aplikacija nalazi na radnoj stanici (radna stanica „klijenta“). Aplikacija se spušta na radnu stanicu klijenta po pokretanju (tj. pristupanjem odredjenoj url-lokaciji). Silverlight tehnologija se koristi za tzv. RIA aplikacije (Rich Internet Application) [2,3]. RIA aplikacija je aplikacija kojoj se pristupa pomoću web-pretraživača, ona ima znatno veće mogućnosti u izradi grafičkog korisničkog interfejsa u odnosu na standardne web-aplikacije (znatno veća interaktivnost). Pri tome, potrebno e da postoji instaliran Silverlight plug-in. Kod RIA aplikacije se izvršni fajl se ne nalazi na računaru korisnika već se on prevlači sa centralnog web-servera u tzv. „sandbox“ web-pretraživača po iniciranju aplikacije.

U ovom radu se opsuje primena Silverlight tehnologije za izradu jedne složene LOB aplikacije, aplikacije koja se koristi u poslovnicama banaka, gde su ove poslovnice rasporedjene na mnogo različitih i udaljenih lokacija. U nastavku teksta će biti prvo opisana Silverlight tehnologija, kao i problemi koji se javljaju prilikom razmeštanja Silverlight aplikacije, a takođe će biti opisana i rešenja za probleme koji se tiču razmeštanja. Zatim će biti opisana izrada jedne Silverlight LOB aplikacije koja se, iz ugla razmeštanja sistema, odlikuje: modularnošću, brzim startovanjem aplikacije i uvek najnovijom verzijom aplikacije na radnim stanicama klijenata.



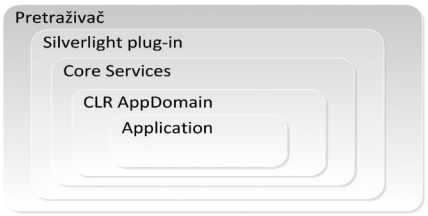
Slika 1: *Arhitektura LOB aplikacije*

SILVERLIGHT I MEF TEHNOLOGIJA

Silverlight, u osnovi, predstavlja plug-in za web pretraživač koji omogućava prikazivanje (renderovanje) korisničkog interfejsa (UI) unutar web strane. Podržan je od strane svih poznatijih web pretraživača (Google Chrome, Internet Explorer, FireFox, Safari) i većine operativnih sistema (Windows, MacOS, Linux u vidu mono projekta).

Definisanje korisničkog interfejsa aplikacije se ostvaruje pomoću XAML-a (Extensible Application Markup Language) koji je u osnovi XML, dok je programiranje u .net jezicima (c#, Visual Basic) [4,5]. Programiranje Silverlight aplikacije u velikoj meri podseća na programiranje Rich Client aplikacija (RCA), stim što je krajnji rezultat drugačiji. Kreirana RCA aplikacija će biti instalirana na računaru korisnika. Sa druge strane kreirana Silverlight aplikacija će biti postavljena na web server a na zahtev klijenta prevlačiće se u web pretraživač korisnika.

Preduslov da bi kreirana Silverlight aplikacija radila u web pretraživaču je instaliran Silverlight plug-in. Plug-in je veličine od par megabajta i moguće ga je instalirati online ili offline (u slučaju većih organizacija to može biti adekvatniji pristup). Na slici 2 prikazuje odnos pretraživača, plugin-a i aplikacije.

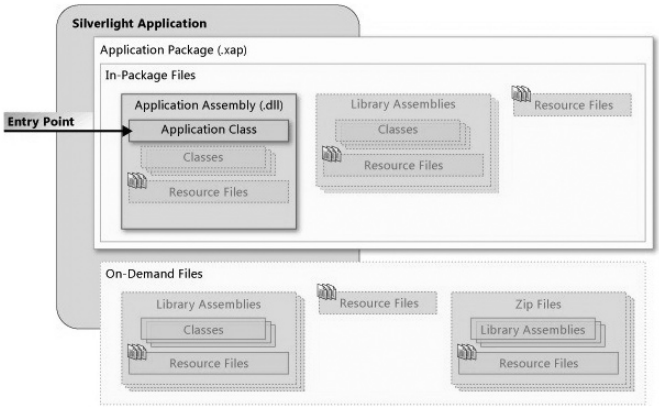


Slika 2:. *Pozicija Silverlight aplikacije u odnosu na plug-in i web pretraživač*

Pre nego što pretraživač pokuša da preuzme Silverlight aplikaciju sa web sajta, vrši se provera da li postoji instaliran Silverlight plug-in. Pretraživač zatim aktivira plug-in i započinje prevlačenje Silverlight aplikacije. U međuvremenu plug-in učitava određene komponente koje su neophodne za rad aplikacije (Core Services, CLR AppDomain). Po uspešnom prevlačenju aplikacije plug-in pokreće aplikaciju. Aplikacija se izvršava u takozvanom "sandbox-u", odnosno u okruženju sa ograničenim pravima pristupa resursima računara. Ovo je sigurnosna mera koja ima za cilj sprečavanje zloupotrebe Silverlight plugin-a.

Već je rečeno da se Silverlight aplikacija prevlači sa web servera. Ona se nalazi na web serveru upakovana u zip arhivu, stim što ova arhiva ima ekstenziju „.XAP". Web pretraživač po preuzetom xap fajlu, vrši njegovo raspakivanje, a zatim sledi pokretanje aplikacije. Struktura XAP fajla (arhive) je data na slici 3. Arhiva u sebi sadrži:

- Izvršnu datoteku aplikacije (Application Assembly dll)
- AppManifest.xaml – xml fajl koji sadrži informacije o izvršnoj datoteci aplikacije kao i o dodatnim datotekama koje se nalaze u zip fajlu.
- Dodatne datoteke koje koristi aplikacija (Library Assemblies)
- Resurs fajlove – sadrže slike, prevode itd.



Slika 3: *struktura Silverlight aplikacije*

Sa kreiranjem dodatnih funkcionalnosti aplikacije povećava se veličina XAP fajla. Veći XAP fajl znači duže vreme prevlačenja (download time) aplikacije sa servera, a samim tim i sporije pokretanje aplikacije. U situaciji LOB aplikacija, vrlo lako se može doći do situacije da je XAP fajl veličine 30MB, a da korisnik ima relativno slabu mrežnu konekciju sa web serverom. Kao rezultat, korisnik aplikacije će čekati nekoliko desetina sekundi (ako ne i minuta) samo da pokrene aplikaciju.

Managed Extensibility Framework (MEF) predstavlja komponentu .net framework-a koji omogućava "proširivost" (extensibility) aplikaciji. Takođe, MEF predstavlja komponentu koja povezuje module aplikacije u celinu, čineći time aplikaciju fleksibilnom, održivom i testabilnom.

LOB aplikacije često sadrže veliki broj komponenti. Iz ugla proširivosti aplikacije, problem koji je nepremostiv za aplikaciju je da sama utvrdi koje su joj komponente na raspolaganju, bez prethodnog konfigurisanja aplikacije ili instanciranja komponenti u kodu. Uz pomoć MEF komponente, aplikacija dobija "sposobnost" da sama uoči koje su

joj komponente na raspolaganju kroz proces kompozicije (composition). Osnovna komponenta MEF-a se naziv "deo" (part) i on oslikava:

- Zavisnosti (imports) i
- Sposobnosti (exports)

Kad god se kreira objekat koji u sebi sadrži zavisnosti, MEF automatski popunjava ove zavisnosti sa odgovarajućim sposobnostima. Ovaj proces "uparivanja" se naziva kompozicija (Composition). Na taj način se izbegava potreba za instanciranjem određenih komponenti u kodu ili njihovim konfigurisanjem. MEF, analizirajući metapodatke komponenti, sam donosi odluku da li postoje odgovarajuće sposobnosti i da li postoji potreba za njima u vidu zavisnosti.

Ukoliko bi uprostili princip MEF-a, moglo bi se reći da MEF dinamički uparuje zavisnosti i sposobnosti. Prilikom učitavanja novog modula aplikacije, MEF će automatski izvršiti analizu svih klasa unutar modula i utvrditi koje su sposobnosti iz modula na raspolaganju i dodati ih katalogu sposobnosti. Dalje, prilikom instanciranja objekata, ovaj katalog će se koristiti ukoliko kreirani objekat ima potrebu za određenim sposobnostima. Baš zbog ove sposobnosti automatskog uparivanja, MEF se smatra da ima ulogu lepka u povezivanju različitih komponenti unutar aplikacije.

Jedna od karakteristika LOB aplikacija je da se aplikacija sastoji iz delova (Composable parts) koji u osnovi predstavljaju nezavisne celine. Ove nezavisne celine se nazivaju moduli, a sposobnost aplikacije da učita module se naziva modularnost. Pored nezavisnih celina, modularnost takođe uvodi mogućnost nezavisnog učitavanja modula kao i učitavanja modula na zahtev. Modul ne mora biti učitan pre nego što se pojavi potreba za njim. Uzimajući u obzir da LOB aplikacije sadrže ogroman broj funkcionalnosti, kao i da korisnici najčešće koriste samo određene funkcionalnosti (retko kad sve funkcionalnosti), moguće je adekvatno grupisati funkcionalnosti u module. Korisnik bi, prilikom korišćenja aplikacije, učitao samo module koji su mu potrebni za izvršenje zadatka, a ne sve module.

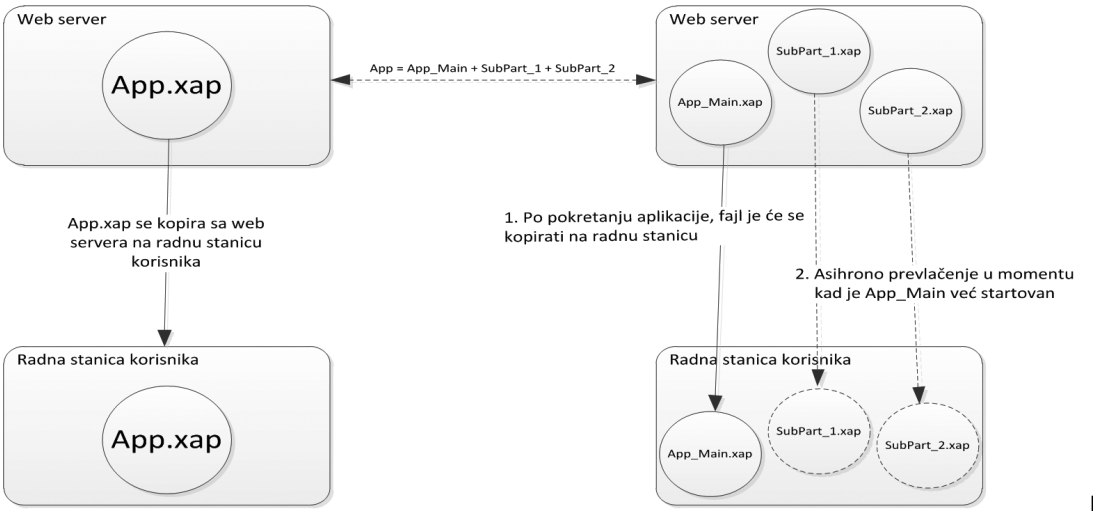
Modularnost u Silverlight aplikacijama se postiže korišćenjem:

- PRISM biblioteke – PRISM predstavlja Microsoft preporuku za izradu RIA aplikacija. U pitanju je biblioteka koja donosi dodatne funkcionalnosti prilikom izrade WPF (Windows Presentation Foundation) i Silverlight aplikacija, kao što su: MVVM Pattern, navigacija, modularnost itd.
- Managed Extensibility Framework (MEF) biblioteke – biblioteka koja omogućava "proširivost" aplikacije

PRISM u značajnoj meri olakšava izradu LOB aplikacija [4]. Ipak korišćenje PRISM-a zahteva izvesni stepen discipline prilikom kodiranja. Dalje, mnoge funkcionalnosti koje nudi PRISM imaju alternative, koje su ne retko jednostavnije za implementaciju nego PRISM. Sa druge strane, MEF je znatno jednostavniji za korišćenje i implementaciju. MEF nema većinu funkcionalnosti koje sadrži PRISM, ali iz ugla implementacije modularnosti, MEF predstavlja brži i jednostavniji pristup. U daljem tekstu će biti opisani principi MEF-a kao i njegova primena u izradi modularnih Silverlight aplikacija.

MEF biblioteku je moguće koristiti i u Silverlight-u. Mada se njena primena može svesti na InvertOfControl (IOC) komponentu na nivou objekata, MEF funkcionalnost može pružiti modularnost Silverlight aplikaciji. Ranije je već opisana struktura Silverlight aplikacije iz ugla njenog raspoređivanja (deployment), pri čemu smo naveli da se svi fajlovi nalaze unutar XAP arhive. XAP arhiva će se uvek prevlačiti sa web servera na radnu stanicu klijenta. Uz primenu MEF-a moguće je jedan XAP fajl zameniti sa više manjih XAP fajlova, koji se mogu "prevlačiti" sa web servera na radnu stanicu korisnika po potrebi (on demand). Na slici 4 je dat uporedni pregled "standardne" i "modularne" Silverlight aplikacije. Scenario modularne aplikacije prikazuje osnovni modul App_main.xap i dva dodatna modula SubPart_1.xap i SubPart_2.xap.

Prilikom pokretanja aplikacije, samo će se jedan XAP fajl prevući na radnu stanicu korisnika. Po startovanju aplikacije, preostali xap fajlovi se mogu asinhrono prevlačiti a da korisnik toga ne budu svestan. Ipak, ukoliko su XAP faj-



Slika 4: *uporedni pregled nemodularne I modularne Silverlight aplikacije*

lovi "razumne" veličine, znatno bolja strategija je prevlačiti dodatne XAP fajlove po potrebi. Velika je verovatnoća da će korisniku trebati samo određene funkcionalnosti aplikacije, a ne sve. Samim tim prevlačiti sve XAP fajlove bi predstavljalo bespotrebni utrošak resursa (prvenstveno mrežnih).

Komponenta MEF-a koja omogućava navedeni scenarijo se naziva DeploymentCatalog. DeploymentCatalog omogućava asinhrono prevlačenje XAP fajlova sa servera, pružajući ujedno i povratne informacije o progresu prevlačenja. Još jedna bitna karakteristika DeploymentCatalog-a je rekompozicija. Rekompozicija omogućava da se ponovi proces "uparivanja" sposobnosti (exports) i zavisnosti (imports) nad komponentama aplikacije koje su već učestvovale u inicijalnom procesu uparivanja.

Naredno poglavlje daje detaljan opis kreiranog rešenja LOB aplikacije, sa akcentom na komunikaciju sistema za navigaciju (menija), modula i pokretanja funkcionalnosti unutar modula.

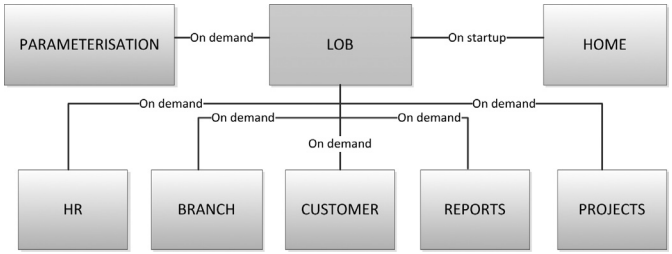
IZRADA MODULARNE LOB SILVERLIGHT APLIKACIJE

U ovom poglavlju će biti detaljno opisana primena MEF-a u izradi jedne modularne LOB aplikacije. U pitanju je aplikacija koja je trebala da pokrije širok dijapazon poslovnih aktivnosti unutar finansijskih institucija. Krajnji rezultat predstavlja rešenje koje se sastoji iz nekoliko modula, pri čemu se određeni modul učitava na radnoj stanici korisnika samo ukoliko postoji potreba za njim. U suprotnom modul se neće ni prevući sa web servera. Uzimajući u obzir da su profili korisnika uglavnom orjentisani ka jednom do dva modula, prilikom korišćenja aplikacije, veći broj modula se neće ni učitati (a samim tim ni prevlačiti sa servera). Krajnji rezultat predstavlja aplikaciju koja se brzo startuje, uvek je u pitanju poslednja verzija softvera a aplikacija prevlači samo one module koji su potrebni.

Aplikacija se sastoji iz sledećih modula:

- Home – osnovni modul (HUB) – sadrži funkcionalnosti za manipulisanje korisničkim profilom, kao i funkcionalnost navigacije (recently used, favorites, navigacija ka opcijama svih modula). Ovaj modul predstavlja „put" do ostalih modula. Prilikom pokretanja aplikacije ovaj modul se automatski učitava. Izborom odgovarajućih opcija (stavki menija) korisnik će pristupati funkcionalnostima drugih modula
- HR – sadrži funkcionalnosti namenjena kadrovskoj službi kao i ostalim korisnicima za podnošenje raznih zahteva ka kadrovskoj službi
- Branch – funkcionalnosti namenjene korisnicima u filijalama
- Customer – razne opcije koje se tiču klijenta
- Parameterization – parametrizacija sistema
- Project Management – sadrži funkcionalnosti namenjena PMO odeljenju za praćenje projekata (planiranje, troškovi, testiranja itd.)
- Reports – segment izveštavanja

Svaki modul sadrži strane (maske) koje predstavljaju različite funkcionalnosti modula. Izbor odgovarajuće stavke u navigaciji osnovnog modula (HUBa) inicira odgovarajuću masku iz odgovarajućeg modula. Na sledećem dijagramu (slika 5) su prikazani nabrojani moduli kao i način njihovog učitavanja:



Slika 5: *Struktura modularne LOB aplikacije*

Osnovne karakteristike učitavanja aplikacije:

- Inicijalno će samo biti učitan modul HOME
- Module HOME sadrži navigacioni mehanizam ka svim funkcionalnostima (iz svih modula)
- Ostali moduli se učitavaju "na zahtev" – kada korisnik zahteva funkcionalnost koja se nalazi u modulu koji nije učitan, inicira se prevlačenje modula na računar korisnika a zatim se pokreće željena funkcionalnost

Uloga MEFa je da:

- Asinhrono prevuče modul sa web servera
- Primenom kompozicije instancira/kreira odgovarajuću stranu/funkcionalnost iz prevučenog modula

Drugim rečima, MEF je tu da prevuče potreban modul sa web servera na radnu stanicu korisnika i da ga učita u aplikaciju. Pri tome, MEF će imati i zadatak da obavesti aplikaciju koje su nove maske na raspolaganju u aplikaciji. Na osnovu ovih informacija, aplikacija će biti u mogućnosti da pristupi bilo kojoj funkcionalnosti modula. Da bi MEF ispunio navedene zadatke, neophodno je uvođenje odgovarajućih entiteta koji će biti korišćeni u procesu kompozicije i to:

- **IView** – svaka maska koja učestvuju u procesu kompozicije mora razvijati ovaj interfejs. Ovaj interfejs na neki način predstavlja potpis maske, odnosno znak prepoznavanja prilikom kompozicije. IView interfejs praktično nema implementaciju (nema ni jednog člana).
- **IPageMetadata** – sposobnost MEF-a (export) može sadržati i metapodatke koji se mogu iskoristiti u procesu kompozicije. IPageMetadata predstavlja interfejs koji označava metapodatke svake maske. Svaka maska koja učestvuje u kompoziciji (i razvija interfejs IView) je u obavezi da obezbedi i metapodatke. Aplikacija će prilikom pokušaja da otvori masku, pretragu raspoloživih maski vršiti upravo oslanjajući se na metadata informacije.
- **ExportPageAttribute** – sposobnost MEF-a se definiše korišćenjem „Export" atributa (njegovim postavljanjem nad odgovarajućim entitetom). Za potrebe LOB aplikacije kreiran je proširen „Export" atribut. Ovaj prošireni

atribut dodaje još jedno svojstvo, NavigateUri, odnosno relativnu adresu maske unutar modula. Relativna adresa maske će se koristiti prilikom pretrage svih funkcionalnosti unutar modula kako bi se locirala željena funkcionalnost. Takođe svaka maska koja bude koristila ovaj atribut, smatraće se da implementira i interfejs IView, ranije objašnjen.

- **UIProviderBase** – u pitanju je klasa koja sadrži sve maske određenog modula sa metapodacima. Svaki modul sadrži po jednu implementaciju ove apstraktne klase. Nakon prevlačenja modula i njegovom "priključivanju" aplikaciji, iz ugla MEF-a pojaviće se dodatna sposobnost tipa UIProviderBase koja sadrži sve maske/funkcionalnosti modula. Sve maske koje su dostupne u modulu će biti definisane i nabrojane u instanci klase UIProviderBase. Aplikacija će se uvek obraćati ovom objektu u pokušaju da nađe određenu masku.

Primenu navedenih klasa je najlakše razumeti na konkretnom primeru. Pretpostavimo da u okviru projekta TestProject, imamo dve maske PageA i PageB. Obe klase su ukrašene atributom ExportPage koji definiše „adresu" strane. Prvi parametar ukazuje na modul, a drugi na relativnu putanju do maske unutar modula.

Dalje, projekat će imati klasu izvedenu iz UIProviderBase klase čiji je zadatak da sadrži kolekciju svih maski tekućeg modula, tačnije maski koje su ukrašene atributom Export i implementiraju interfejs IView. Nova klasa se naziva TestProjectUIProvider. Prilikom instanciranja klase, MEF će automatski popuniti svojstvo EntryPage, tako što će analizirati sve klase unutar projekta koje imaju atribut `Export` (ne zaboravite MEF samo analizira klase koje imaju atribute `Import` ili `Export`) i koje razvijaju interfejs `IView`. U našem slučaju kolekcija EntryPage će imati dva člana, po jedan za obe maske (PageA i PageB).

Obratite pažnju da i sama klasa TestProjectUIProvider ima Export atribut. Kada centralna (host) aplikacija pokuša da učita modul, MEF će pronaći klasu i izvršiti njeno instanciranje. Centralna aplikacija će dalje do bilo koje maske unutar TestProject modula, dolaziti upravo preko TestProjectUIProvider klase i njenog svojstva EntryPage.

Vratimo se inicijalnoj LOB aplikaciji. Dakle, svaki modul sadrži set maski koje su napisane uz poštovanje određenih pravila (korišćenjem ExportPage atributa kao što smo upravo opisali). Dalje, svaki modul sadrži implementaciju klase UIProviderBase. Prilikom učitavanja modula u centralni modul (HOME), aplikacija će imati na raspolaganju spisak maski koje se nalaze u modulu. Sledeći zadatak je povezivanje mehanizma navigacije centralnog modula (HOME) sa konkretnim maskama.

Home modul sadrži navigaciju pomoću koje korisnik dolazi do određene funkcionalnosti. Standardni mehanizam navigacije Silverright-a omogućava učitavanje maski (klase tipa Page) unutar kontejnera koji se naziva frejm (klasa tipa Frame). Kad korisnik izabere određenu opciju, na osnovu informacija izabrane opcije(u vidu relativnog uri-ja) aplikacija kreira masku i smešta je u frejm. Standardna funkcionalnost učitavanja maski pretpostavlja da se sve maske nalaze unutar osnovnog, inicijalno učitanog, modula. Međutim, u našem slučaju to nije situacija. Maske se nalaze u modulima koji će tek biti učitani. U ovakvoj situaciji neophodno je zameniti standardno ponašanje sistema za navigaciju.

Centralnu ulogu u sistemu navigacije Silverilght aplikacije ima klasa PageResourceContentLoader koja razvija interfejs INavigationContentLoader. Svrha ove klase, je da na osnovu relativne uri adrese maske, izvrši njeno instanciranje (otvaranje). Kako bi unapredili standardno ponašanje, potrebno je kreirati klasu koja razvija pomenuti interfejs, i sa njom zameniti standardnu komponentu PageResourceContentLoader.

Šta donosi nova klasa MefContentLoader? Donosi drugačiji način razrešavanja, odnosno kreiranja maske u odnosu na standardnu klasu. Nova klasa će izvršiti inspekciju kroz sve učitane module I pokušati da nađe masku čiji uri (uri se nalazi u metapodacima maske) odgovara zahtevanom uri-ju. Moduli aplikacije će egzistirati u novoj klasi u vidu kolekcije `UIProviderBase[] Plugins`. `Plugins` svojstvo je obeleženo atributom `[ImportMany(AllowRecomposition = true)]`, koji omogućava da se po učitavanju novog modula, nov modul automatski pojavi i u kolekciji Plugins. Drugim rečima svaki modul ima svog PlugIn predstavnika (u vidu nasleđene klase UIProviderBase) u pomenutoj kolekciji, koji sadrži spisak svih maski iz modula kog predstavlja. Zahvaljujući MEF mehanizmu rekompozicije, po učitavanju svakog novog xap fajla, MefContentLoader komponenta će povećati svoju kolekciju UIProviderBase sa novim modulom, čime će biti omogućeno učitavanje bilo koje maske iz novog modula.

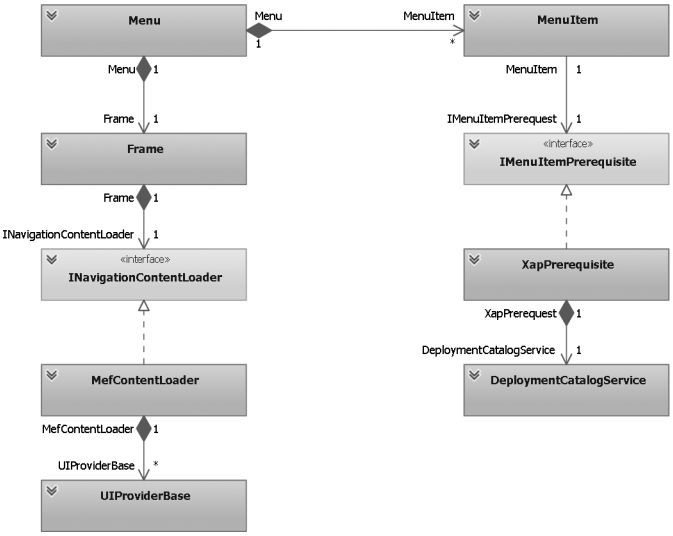
MefContentLoader će pretražiti učitane module kako bi našao odgovarajuću masku, ali ova klasa neće odraditi zadatak prevlačenja modula na zahtev. Ovaj zadatak obavljaju druge dve klase:

- DeploymentCatalogService – prevlači xap fajl na zahtev. DeploymentCatalogService će najpre izvršiti proveru da li je fajl već prevučen (proverom interne kolekcije ranije prevučenih xap fajlova), a zatim će, ukoliko fajl nije prevučen, započeti asihrono prevlačenje fajla. Po završenom prevlačenju fajla, ažuriraće se interna kolekcija prevučenih fajlova. Klasa takođe pruža mogućnost praćenja progresa prevlačenja fajla, kao i mogućnost obaveštavanja kada je prevlačenje fajla završeno.
- XapPrerequisite – koordinira proces prevlačenja xap fajla. Ova komponenta ima zadatak da nakon primljene instrukcije da se učita određena maska, proveri da li je odgovarajući xap fajl već učitan. Ukoliko xap fajl nije učitan, komponenta inicira prevlačenje fajla sa web servera(uz pomoć DeploymentCatalogService komponente) i ujedno manipuliše korisničkim interfejsom u toku prevlačenja fajla.

Mehanizam navigacije možemo uprošćeno posmatrati kao meni u klasičnoj desktop aplikaciji. Svaka stavka menija inicira otvaranje određene maske u aplikaciji. Tehnički, svaka stavka menija, sadrži instancu objekta tipa XapPrerequisite, koji pak sadrži instancu DeploymentCatalogService-a. Nakon što korisnik pritisne određenu stavku menija, kontrola se predaje XapPrerequisite komponenti. Ona proverava da li je odgovarajući Xap fajl već prevučen. Ukoliko nije, XapPrerequisite daje instrukciju DeploymentServiceCatalog komponenti da započne proces prevlačenja xap fajla. Ujedno XapPrerequisite, "zamrzava" delove korisničkog interfejsa (u vidu „blur" efekta, sve stavke menija koje zahtevaju modul koji se prevlači će biti nedostupne), kako bi sprečio korisnika da izvršava nove akcije koje zahtevaju modul koji se upravo prevlači. Po "prevučenom" xap fajlu, vrši se njegovo učitavanje, pronalaženje određene maske unutar modula i njeno instanciranje (zadatak UIProviderBase komponente).

Na slici 6 je at klasni dijagram (class-diagram) koji predstavlja sve komponente navigacije i njihove odnose. Dodatne klase koje do sad nisu objašnjavane su:

- Menu - komponenta koja predstavlja meni i sadrži kolekciju objekata tipa MenuItem.
- MenuItem - predstavlja stavku menija, sadrži svojstvo tipa interfejs IMenuItemPrerequisite
- Frame – Silverlight komponenta, kontejner za prikaz maske (Page komponente)



Slika 6: *Klasni dijagram navigacije LOB aplikacije*

Komponente sa leve strane dijagrama su većim delom već objašnjene. Novina je klasa Menu, koja predstavlja meni aplikacije i zadužena je za manipulaciju korisničkog interfejsa prilikom kretanja kroz stavke menija, korišćenjem opcija „Recently Used", „Favorites" itd. Menu sadrži objekat tipa Frame u kom će se prikazivati svaka otvorena maska. Već smo ranije pojasnili mehanizam koji se koristi od strane Frame-a za učitavanje maski, a koji se oslanja na sledeća tri entiteta sa class dijagrama: INavigationContentLoader, MefContentLoader, UIProviderBase.

Desna strana dijagrama se tiče pojedinačne stavke menija. Klasa MenuItem predstavlja jednu stavku menija. Jedno od svojstava koje sadrži MenuItem je i svojstvo IMenuItemPrerequisite.

Svrha interfejsa IMenuItemPrerequisite je da izvrši proveru da li su ispunjeni uslovi za otvaranje maske na koju se MenuItem odnosi. Interfejs takođe omogućava dodavanje novih preduslova kao i njihovo „ispunjenje". Prilikom „click" događaja (event), MenuItem proverava da li su ispunjeni preduslovi za otvaranje maske, korišćenjem metode PrerequisiteSatisfied. Ukoliko uslovi nisu ispunjeni, poziva se metoda SatisfyPrerequisite koja ima za zadatak da „ispuni uslove". U kontekstu otvaranja svake menija, preduslov može biti bilo šta, na primer prava pristupa, dostupnost nekog resursa i sl. Ali u našem slučaju, preduslov predstavlja učitan xap fajl. Drugim rečima, maska se može otvoriti tek pošto je prevučen xap fajl u kojoj se maska nalazi. Za ovu svrhu je razvijena klasa XapPrerequisite koja razvija interfejs IMenuItemPrerequisite.

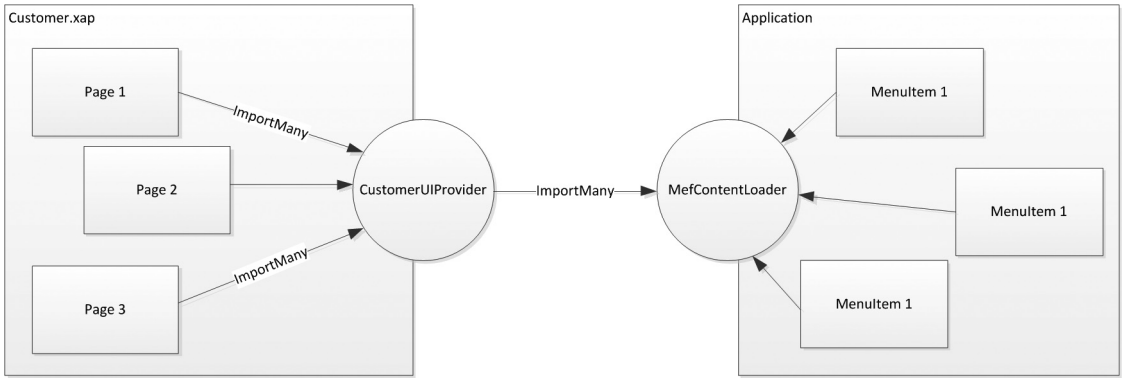
XapPrerequisite klasa predstavlja komponentu koja korisničkom interfejsu (komponenta menija aplikacije) donosi sposobnost da inicira prevlačenje fajla. Takođe komponenta kreira vizuelni efekat ("blur") nad korisničkim interfejsom tokom prevlačenja fajla, tačnije sve stavke menija koje zahtevaju modul koji se upravo prevlači će biti nedostupne. Primena klase se bazira na pretpostavci da svaka stavka menija sadrži informacije u kom modulu se nalazi maska koju je potrebno pozvati. Ova informacija se koristi prilikom analize da li je potrebno prevući novi xap fajl ili ne.

Kada korisnik izabere odgovarajuću stavku menija (MenuItem), kontrola se predaje XapPrerequisite komponenti koja treba da proveri da li je odgovarajući xap fajl već prevučen sa web servera. Ukoliko je xap fajl već prevučen, uloga XapPrerequisite komponente se završava. U suprotnom, ova komponenta inicira prevlačenje xap fajla (uz pomoć DeploymentCatalogService komponente). Tek po završenom prevlačenju xap fajla, komponenta završava svoj rad. Tokom prevlačenja xap fajla, sve meni stavke koje "zavise" od xap fajla koji se prevlači, će biti nedostupne i "zamagljene". Sa povećanjem "download" progresa "zamagljenost" će biti sve manja. Po završenom prevlačenju xap fajla, MenuItem daje instrukciju frejmu da učita stranicu, koji pak ovaj zadatak prosleđuje MefContentLoader komponenti.

Krajnji rezultat opisanog rešenja predstavlja jednostavno „ubacivanje" nove funkcionalnosti i novog modula u LOB aplikaciju. Možemo definisati dva koraka

- Napraviti modul i funkcionalnosti unutar modula
- Povezati modul sa LOB aplikacijom

Novi modul. Svaki modul predstavlja odvojenu Silverlight aplikaciju. Međutim da bi aplikacija postala modul, neophodno je da sadrži klasu koja je izvedena iz klase UIProviderBase. Instanca izvedena klase će sadržati definicije svih maski koje se nalaze u modulu putem svojstva EntryPage. U pitanju je kolekcija koja će se, uz pomoć MEF-a, dinamički popuniti sa definicijama svih maski modula koje ispunjavaju određene preduslove. Dalje, instanca izvedene klase će nakon prevlačenja



Slika 7: *Učitavanje modula LOB aplikacije*

XAP fajla, automatski završiti u kolekciji MefContentLoader. Plugins koja se koristi prilikom učitavanja maske od strane sistema za navigaciju. Kao što je već naglašeno, ključno svojstvo nove klase je EntryPage, koji sadrži definicije svih maski modula (koje razvijaju interfejs IView).

Da bi maska (strana, page) završila u kolekciji EntryPage klase CustomerUIProvider, neophodno je da razvijan interfejs IView i da je „ukrašena" atributom ExportPage. Atribut ExportPage definiše relativnu adresu maske koja će biti korišćena od strane sistema za navigaciju LOB aplikacije (koji se nalazi u HOME modulu). Interfejs IView ima ulogu „markiranja" komponente da je u pitanju maska koja se može učitati iz HOME modula.

Na slici 7 se prikazuje odnos novog modula koji se učitava i same aplikacije. Krajnji rezultat je da su definicije svih strana (koje implementiraju interfejs IView) iz modula pohranjene u MefContentLoader komponentu, i koriste se prilikom iniciranja akcije od strane korisnika putem menija.

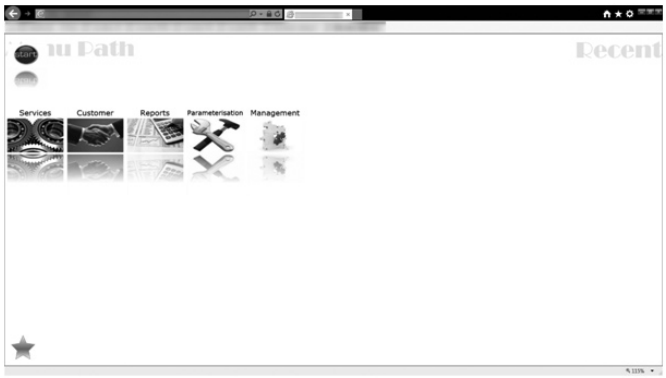
Povezivanje. Da bi iskoristili funkcionalnosti novog modula, prvo je neophodno postaviti novi modul na web server gde se nalazi i sama LOB aplikacija (folder ClientBin). Zatim je potrebno napraviti stavke menija i povezati ih sa maskama modula. Ovde u osnovi daju se sledeće instrukcije aplikaciji prilikom iniciranja akcije („click" događaj nad stavkom menija):

- Učitaj masku definisanu svojstvom PagePath
- Pre nego što pokušaš da učitaš masku, proveri da li je ispunjen preduslov definisan svojstvima MenuItemPrerequisite i MenuItemPrerequisiteParameter
- Ukoliko nije, ispuni preduslov, a zatim učitaj masku

MenuItemPrerequisiteParameter definiše parametar koji će biti prosleđen MenuItemPrerequisite objektu, u ovom slučaju to je xap fajl u kom se nalazi maska koju je potrebno prikazati. **MenuItemPrerequisite** definiše „mehanizam" koji će se koristiti za ispunjenje preduslova a to je komponenta XapPrerequisite. **PagePath** definiše relativnu adresu maske unutar modula (xap fajla). XapPrerequisite će izvršiti proveru da li je prisutan modul NewWay.Branch.xap i ukoliko ga ne pronađe iniciraće njegovo prevlačenje. Za vreme prevlačenja ovog modula, sve stavke menija koje se odnose na modul NewWay.Branch.xap će biti nedostupne (ali korisnik može, ukoliko to želi aktivirati neku od opcija već učitanih modula). Po učitavanju modula, a na osnovu PagePath svojstva, izvršiće se instanciranje nove maske.

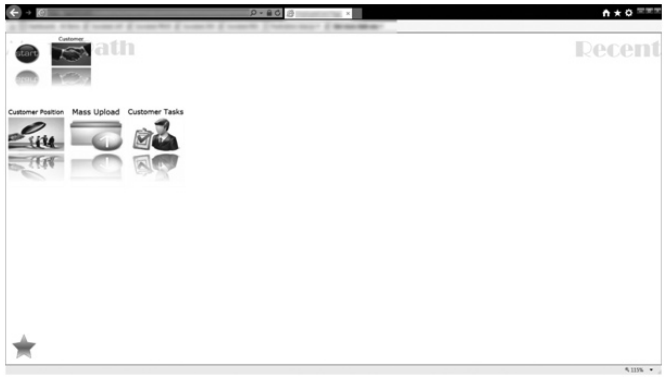
UPOTREBA MODULA LOB APLIKACIJE

Sledećih nekoliko slika predstavljaju konačan rezultat prethodnog izlaganja. Na slici 8 je dat prikaz osnovnog HOME modula, koji je ujedno i nosilac navigacije aplikacije. Prikaz ujedno predstavlja i početnu tačku za korisnika aplikacije. Aplikacija trenutno ima učitan samo HOME modul (ostali moduli se nalaze na web serveru).



Slika 8: *HOME modul*

Korisnik izborom odgovarajuće stavke u meniju (centralni deo maske) se kreće kroz meni. U ovom trenutku je učitan samo osnovni modul aplikacije. Sve dok korisnik ne izabere opciju koja bi trebala da prikaže masku (listovi stabla menija), ne dešava se učitavanje modula. Sledeće slike prikazuju opcije podmenija aplikacije (još uvek se ne vrši učitavanje modula).



Slika 9: *Podmeni 1 osnovnog modula aplikacije*

Slika 10: *Podmeni 2 osnovnog modula aplikacije*

Nakon što korisnik izabere stavku menija koja bi trebala da otvori odgovarajuću masku, dešava se prevlačenje modula sa web servera i pokretanje odgovarajuće maske iz modula. Na sledećoj slici je dan primer maske iz modula Branch. Aplikacija vrši proveru da li je Branch modul već učitan. Ukoliko nije, modul se prevlači sa web servera na radnu stanicu korisnika i vrši se njegovo učitavanje u aplikaciju. Nakon izvršenog učitavanja, aplikacija instancira željenu masku i prikazuje je korisniku.

Slika 11: *Modul BRANCH*

ZAKLJUČAK

Savremene LOB aplikacije, usled velikog broja funkcionalnosti koje nose sa sobom, predstavljaju velike aplikacije. Razmeštanje (deployment) ovih aplikacija zahteva analizu i pažljivo planiranje I dizajn. Jedan od koncepata koji u velikoj meri može doprineti efikasnom razmeštanju aplikacije je koncept modularnosti. Podeliti aplikaciju na module, a zatim učitavati module „po potrebi“ (on demand) ili asinhrono, predstavlja pristup koje slobodno možemo svrstati u grupu „najboljih praksi“ (best practices). Drugim rečima, kvalitetna LOB aplikacija mora biti modularna.

Silverlight tehnologija nudi nekoliko mehanizama za postizanje modularnosti. Managed Extensibility Framework je jedna od njih. Korišćenjem MEF-a za izradu modula postiže se visok stepen interaktivnosti i eliminišu negativni efekti učitavanja velikih aplikacija. Izradom dodatnih, pomoćnih komponenti može se postići integracija sistema navigacije Silverlight aplikacije sa nezavisnim modulima aplikacije. Krajnji rezultat predstavlja aplikacija koja se brzo učitava/startuje, koja učitava dodatne module samo kad postoji potreba za njima, a na osnovu akcija korisnika aplikacije (izborom odgovarajućih opcija u sistemu navigacije).

U ovom radu se opsuje primena Silverlight tehnologije za izradu jedne složene LOB aplikacije, aplikacije koja se koristi u poslovnicama banaka, gde su ove poslovnice rasporedjene na mnogo različitih i udaljenih lokacija. Detaljno je opisana izrada ove Silverlight LOB aplikacije koja se, iz ugla razmeštanja sistema, odlikuje: modularnošću, brzim startovanjem aplikacije i uvek najnovijom verzijom aplikacije na radnim stanicama klijenata.

LITERATURA

[1] Pro Business Applications with Silverlight 4, Apress, Chris Anderson, 2010
[2] Beginning Silverlight 4 in C Sharp, Apress, Robert Lair, 2010
[3] MEF Community Site, http://mef.codeplex.com/
[4] Developer Guide to Microsoft Prism – Microsoft Patterns & Practices, 2010
[5] Microsoft Developer Network (MSDN) Library, http://msdn.microsoft.com/library/default.aspx

Igor Pantelić, Metropolitan univerzitet, Beograd
Kontakt: igorpantelic@gmail.com
Oblasti interesovanja: veliki softverski sistemi, C# programiranje, Microsoft tehnologije

Slobodan Jovanović, Metropolitan univerzitet, Beograd
Kontakt: slobodan.jovanovic@metropolitan.ac.rs
Oblasti interesovanja: veliki softverski sistemi, C++ programiranje, inteligentni sistemi