Primljen: 20.05.2012. Stručni rad

Prihvaćen: 04.06.2012. UDK 004.42

**Uvod u razvoj višeslojnih poslovnih aplikacija u Microsoft .NET okolini**

**Introduction to Development of Multi-layered Business Applications in Microsoft.NET Environment**

1 Krešimir Kolac, 2 Velimir Sanjković, 3 Željko Knok  
1 Međimurje IPC d.d. Mihovljanska 72, Čakovec 2 S.Soft Software Development, N. Tesle 28, Pribislavec  
3 Međimursko veleučilište u Čakovcu, Bana Josipa Jelačića 22a, Čakovec  
e-mail: 1 [kresimir.kolac@gmail.com](mailto:kresimir.kolac@gmail.com), 2 [velimir.sanjkovic@s-soft.hr](mailto:velimir.sanjkovic@s-soft.hr), 3 [zknok@mev.hr](mailto:zknok@mev.hr)

**Sažetak**: *U praksi se često susrećemo s vrlo raznolikim pristupima izgradnje novih aplikacija. Programeri kreću u izradu aplikacija bez dovoljne tehničke i tehnološke pripreme. Arhitektura aplikacije se mijenja „ad-hoc“ za vrijeme izrade same aplikacije. Često se u istoj aplikaciji može naći više različitih pristupa rješavanja određenog problema, ovisno u kojoj fazi projekta je nastao koji dio koda. Objektno orijentirana paradigma (OOP) u dizajnu aplikacije se nedovoljno koristi. Dijelovi koda se „copy-paste“ metodom multipliciraju po aplikaciji. Korisničko sučelje često ima na sebi previše pozadinskog koda i poslovne logike (code behind). Za razliku od takvog pristupa u ovom radu objasnit će se sustavan pristup dizajnu arhitekture velikih poslovnih aplikacija s orijentacijom na Microsoft .NET razvojnu okolinu. Kod sustavnog pristupa faza pripreme traje duže, ali se ukupno vrijeme potrebno za izradu aplikacije skraćuje. Svaka funkcionalnost ima svoje točno određeno mjesto u aplikaciji gdje se implementira. Kod je pregledan, redundancija u kodu se svodi na minimum a dokumentiranost projekta je kvalitetnija.*

**Ključne riječi**: *višeslojne poslovne aplikacije, Microsoft, .NET, Prism, MVVM, WPF, Silverlight, NHibernate, CSLA*

**Abstract**: *We are often faced with very diverse approaches to building new applications in practice. Developers engage in application development without sufficient technical and technological training. Application architecture is changing "ad-hoc" during the application development itself. We can find many different approaches to solving a particular problem depending on in which stage of project the code was created. Parts of the code are multiplied by „copy-paste“ method throughout the application. The user interface usually contains too much background code and business logic (code behind). In contrary to this approach, in this paper we'll explain the systematic way to designing the big business applications architecture using Microsoft .NET development environment. In a systematic approach preparation phase lasts longer, but the total time required to develop an application shortens. Every functionality has its specific spot in application where it is implemented. The code is clear, its redundancy is minimised and project documentation is better.*

**Key words**: *multi-layered business applications, Microsoft, .NET, Prism, MVVM, WPF, Silverlight, NHibernate, CSLA*

1. **Uvod**

Prilikom dizajna i razvoja složenih poslovnih aplikacija projektanti i razvojni programeri danas imaju na raspolaganju veliki broj različitih baza podataka, tehnologija za razvoj softverskih rješenja i različite dobavljače razvojnih alata i rješenja kao i rješenja otvorenog koda (*open source*). Također postoji velik broj različitih koncepata, paradigmi i smjernica za programiranje koje možemo koristiti prilikom razvoja rješenja. Brzi razvoj tehnologije i velika paleta dostupnih tehnologija i koncepata mnogima umjesto prednosti može zadati velike nedoumice i probleme. Ukoliko odaberemo krivi smjer, krivu tehnologiju ili se orijentiramo na krivog dobavljača, to nas može koštati puno uloženog truda, vremena i novaca. Složeni poslovni sustavi danas zahtijevaju velike investicije u infrastrukturu, tehnologiju (licence) i ljude. Kada se kreće u novi razvoj ili implementaciju nekog rješenja, treba kalkulirati da će se jedno takvo rješenje koristiti u produkciji minimalno desetak, petnaestak godina, a po mogućnosti i puno duže.

Navodimo neka od pitanja na koja treba odgovoriti prije nego što se krene u dizajn i razvoj aplikacije:

1. Da li se želimo strateški vezati na jednog ili više dobavljača razvojnih i poslovnih rješenja (Microsoft, Oracle, SAP, IBM itd.) ili će orijentacija biti na rješenja otvorenog koda?
2. Da li ćemo arhitekturu podrediti nekoj konkretnoj relacijskog bazi podataka ili ćemo raditi rješenje koje će biti neovisno o izvorima podataka.
3. Da li će arhitektura aplikacije biti dvoslojna (klijent-server) ili neki oblik troslojne arhitekture (bazni server - aplikacijski server – klijent)
4. Da li ćemo poslovnu logiku smjestiti na bazu podataka, na aplikacijski sloj ili na samog klijenta, odnosno koje dijelove logike aplikacije ćemo smjestiti u koje dijelove aplikacije.
5. Kakve klijente će koristiti aplikacija? Desktop forme, web forme, pametni telefoni?
6. Koje tehnologije ćemo koristiti za razvoj aplikacije kada se odlučimo za neku okolinu?
7. Na koji način ćemo riješiti sigurnost u aplikaciji?
8. Kako ćemo riješiti zahtjeve za više konkurentnih korisnika koji istovremeno mogu raditi nad određenim podacima u aplikaciji?
9. Potrebno je i definirati načine testiranja koda.
10. Na koji način i s kojim alatima će se implementirati izvještajni podsustav?
11. Kako ćemo distribuirati nove inačice aplikacija, izvještaja i modula?

Tematika razvoja složenih poslovnih aplikacija je dosta široka i o svakome od navedenih pitanja bi se mogao napisati samostalan rad. Zbog toga je u ovom radu stavljen naglasak na unaprijed zadane okvire prilikom izgradnje sustava, a to su:

* orijentacija na Microsoft razvojne alate;
* korištenje neke od modernih relacijskih baza podataka poput Oracle baza podataka ili Microsoft SQL Servera;
* Primarno korisničko sučelje će biti „bogate“ Windows desktop forme;
* potencijalno prebacivanje određenih funkcionalnosti u Internet preglednik ili na pametne telefone;
* istovremeni rad nekoliko razvojnih timova i programera.

1. **Analiza zahtjeva**

Apriori orijentacija na Microsoft rješenja i tehnologije ili primarno korištenje Windows desktop formi ima svoje prednosti i mane, ali samim tim što je strategija unaprijed definirana riješeno je jako puno nedoumica za arhitekta aplikacije oko odabira razvojne okoline i tehnologija koje će se koristiti prilikom razvoja aplikacije.

Odabir dobavljača razvojnih alata i rješenja je najčešće unaprijed zadan sa strane naručitelja sustava s obzirom da se tu radi o godinama ulaganja u strateška partnerstva, licence i školovanje i certifikaciju razvojnog tima. Naručitelj može biti ili vanjski kupac ili uprava firme koja naručuje softver za tržište. Slično je i s Microsoftom. Microsoft nudi široku lepezu alata i tehnologija potrebnih za razvoj složenih sustava, relativno pristupačnih i dobavljivih za proizvođača rješenja kao i za kupca budućeg rješenja. Windows operativni sustav danas prevladava na desktop računalima a na tržištu rada ne bi trebalo biti problema u pronalasku iskusnih .NET programera.

Odabir moderne, napredne relacijske baze podataka je bitan ukoliko želimo raditi s velikim transakcijskim tablicama, spremati dijelove poslovne logike u bazu podataka ili uvesti određene mehanizme skladištenja podataka (*data warehouse*) u samu relacijsku shemu.

Primarna orijentacija na desktop klijente znači da želimo bogato korisničko sučelje, dobru interakciju s korisnikom aplikacije te prirodnu integraciju aplikacije u operativni sustav klijentskog računala što će znatno olakšati rad u samoj aplikaciji i dati veći izbor mogućnosti projektantima aplikacije.

Zahtjev da na aplikaciji mora raditi istovremeno više razvojnih timova podrazumijeva obavezno centralizirano spremanje koda i ostalih datoteka aplikacije te korištenje nekog alata za kontrolu koda. Paralelni razvoj različitih dijelova aplikacije i korisničkog sučelja zahtijeva kompozitni pristup izgradnji korisničkog sučelja što znači da se dijelovi korisničkog sučelja odnosno aplikacije razvijaju, testiraju i isporučuju naručitelju neovisno jedan o drugome.

1. **Arhitektura aplikacije**

Danas kad se govori o arhitekturi velikih poslovnih aplikacija moramo razlikovati fizičku i logičku arhitekturu aplikacije. Fizički se aplikacija najčešće isporučuje dvoslojno, bazni server – debeli klijent, ili troslojno, bazni server – aplikacijski server – tanki klijent. Broj fizičkih slojeva ovisi o veličini sustava i funkcijama koje mora sustav obavljati. Prilikom fizičke izgradnje sustava trebamo razmišljati o komunikaciji između dijelova sustava te o distribuciji novih verzija aplikacija, izvještaja i modula što je direktno ovisno o fizičkoj implementaciji rješenja.

Za razliku od fizičke arhitekture nas puno više zanima logička arhitektura sustava. Logički se danas svaka aplikacija sastoji najčešće od 3 glavna sloja:

* Sloja za pristup podacima (DAL - *Data access layer*)
* Sloja poslovne logike (BLL - *Business logic layer*)
* Sloja korisničkog sučelja – sloj prezentacije (UIL, GUI, UI – *User interface*, *presentation layer*).

  
*Slika 1. – Logička arhitektura aplikacije.*

Ti slojevi se dalje mogu dijeliti svaki na više podslojeva. Postoji mnogo različitih varijacija osnovnog pristupa u praksi. Na primjer, sloj korisničkog sučelja možemo podijeliti na samo sučelje, ulazno-izlazne kontrole, aplikacije i sloj koji upravlja sučeljem, npr. Controller klase u Model–View–Controller (MVC) i ViewModel klase u Model–View–ViewModel (MVVM) obrascu za implementaciju sučelja. Ukoliko aplikacija ima više tipova korisničkog sučelja, vjerojatno će imati i više različitih slojeva korisničkog sučelja i slojeva za kontrolu sučelja. Ukoliko želimo aplikaciju koja je neovisna o izvorima podataka, vjerojatno ćemo i sloj za pristup podacima podijeliti na više podslojeva koji će se neovisno brinuti za dohvat i spremanje podataka iz različitih izvora podataka (relacijske baze podataka, Web servisi, jednostavne tekstualne datoteke, XML datoteke, itd.).

* 1. **Microsoft Prism**

Microsoft s jedne strane nudi dobre alate i tehnologije za izradu aplikacija, no s druge strane odabirom Microsoft tehnologija na početku dobijemo vrlo malo znanja (*know-how*) kako napraviti i postaviti arhitekturu dobre poslovne aplikacije. Primjeri na Internetu koji prikazuju izradu formi za ažuriranje podataka u bazi podataka su najčešće elementarni i neupotrebljivi, ukoliko želimo sustavno riješiti tehnološku vertikalu ažuriranja podataka od korisničkog sučelja do spremanja u bazu podataka kroz više aplikacijskih slojeva.

U posljednje vrijeme se situacija ipak znatno popravila i Microsoft sve bolje surađuje sa zajednicom što rezultira sve većim brojem smjernica za izradu aplikacija namijenjenih razvojnim inženjerima, projektantima sustava i programerima. Te smjernice i preporuke možemo pronaći dobro dokumentirane i objavljene na „Microsoft patterns & practices“ (<http://msdn.microsoft.com/en-us/practices>) Web portalu.

Prilikom razvoja višeslojnih poslovnih aplikacija preporuka je koristiti Microsoft Prism smjernice za razvoj Model-View-ViewModel (MVVM) modularnih aplikacija s Windows Presentation Foundation (WPF) desktop sučeljem, bogatih Internet aplikacija (RIA - Rich Internet Applications) sa Silverlight sučeljem i Windows Phone 7 aplikacijom. Microsoft Prism definira obrasce za dizajn i arhitekturu aplikacija s ciljem kreiranja aplikacija iz modula koji se mogu razvijati neovisno jedni o drugima od različitih razvojnih timova, smanjenja međuovisnosti razvojnih timova, razdvajanja poslovne logike od dizajna sučelja i povećanje kvalitete aplikacija apstrakcijom općih servisa koji su raspoloživi svim timovima.

* 1. **Model-View-ViewModel (MVVM)**

MVVM je obrazac za dizajn arhitekture WPF, Silverlight i Windows Phone 7 aplikacija definiran u sklopu Microsoft Prism. MVVM se sastoji od tri komponente:

* **View –** **Sučelje** - XAML (*eXtensible Application Markup Language*) je sučelje aplikacije sa svojim pozadinskim kodom. XAML je programski jezik razvijen za dizajn i razvoj sučelja aplikacija. Ideja je odvojiti upravljanje sučeljem od dizajna sučelja. Elementi sučelja se povezuju (*binding*) na klase sloja za kontrolu sučelja.
* **ViewModel – Kontrola sučelja** - klase koje sadrže podatke potrebne za renderiranje sučelja, kao i metode za operacije nad modelom. Povezivanje mora biti dvosmjerno, što znači da se promjenom podataka na sučelju mijenja i sadržaj u ViewModel objektima, a promjenom stanja ViewModel objekta se osvježava sučelje.
* **Model – Podaci** - objektni model podataka domene aplikacije. Domena zna komunicirati s bazom podataka i promjene nad domenom spremiti u bazu podataka.

  
*Slika 2. – Shema MVVM obrasca  
Izvor: Microsoft Prism.*

* 1. **Pristup podacima**

U MVVM obrascu najniži sloj naziva se Model. Model su složeni poslovni objekti u kojima je implementiran najveći dio poslovne logike aplikacije. Model u klasičnoj arhitekturi aplikacije nazivamo još i sloj poslovne logike ili *Business logic layer* (BLL). BLL ne smije sadržavati nikakav kod koji se odnosi na funkcioniranje sučelja aplikacije te treba biti neovisan o izvorima podataka. Da bi mogli implementirati BLL moramo riješiti komunikaciju s bazom podataka, sloj za pristup podacima (DAL- *Data access layer*). O svim specifičnostima različitih baza podataka i različitih izvora podataka treba se brinuti sloj za pristup podacima.

Pristup podacima u bazi podataka može se implementirati na klasičan način preko ADO.NET (*ActiveX Data Objects*) podsustava Microsoft.NET platforme ili pomoću neke od ORM (*Object-relational mapping*) tehnika za mapiranje baza podataka kao što je ADO.NET Entity Framework ili NHibernate. Funkcija ORM-a je mapiranje relacijske baze podataka te generiranje upita prema bazi podataka. U samoj aplikaciji nije potrebno pisati SQL (*Structured Query Language*) upite za dohvat i ažuriranje podataka. Umjesto toga manipulira se s objektima domene aplikacije. Time se znatno smanjuje količina koda i ubrzava razvoj aplikacije. Korištenje ORM tehnike kod aplikacije ne ovisi o bazi podataka i aplikacija može raditi na više različitih baza podataka. Jedina specifičnost je konfiguracija samog ORM-a za rad s određenom bazom podataka.

* + 1. **NHibernate**

NHibernate je Microsoft .NET inačica vrlo popularnog Hibernate ORM-a razvijenog za Javu. NHibernate je besplatan i otvorenog koda. Ima vrlo mnogo mogućnosti, dokazan je na velikim (*enterprise*) projektima, kvalitetan je i pouzdan. Za NHibernate je dostupan LINQ (*Language Integrated Query*). Dodatak Fluent NHibernate omogućava mapiranje relacijskih tablica s domenskim klasama direktno iz C# koda bez korištenja XML (*EXtensible Markup Language*) datoteka. Detaljan popis osobina i karakteristika NHibernate-a može se pronaći na NHibernate Forge- službenim Web stranicama NHibernate zajednice za .NET ([http://nhforge.org](http://nhforge.org/)).

* 1. **Poslovna logika - Component-based, scalable, logical architecture (CSLA)**

Sloj poslovne logike sadržava klase u kojima je implementirano ponašanje i logika aplikacije i poslovnih procesa. Klase poslovnog sloja su često vrlo složeni objekti koji implementiraju cijele procese unutar aplikacije nad više entiteta baze podataka. Sloj poslovne logike možemo razviti sami tako da pišemo poslovne klase ili sustavno koristeći neki od aplikacijskih okvira namijenjenih razvoju poslovne logike. U posljednje vrijeme je CSLA.NET aplikacijski okvir (*framework*) postao jedan od najkorištenijih aplikacijskih okvira za razvoj poslovnih objekata na Microsoft .NET platformi. CSLA.NET je implementacija arhitekture zvane CSLA (engl. *Component-based*, *scalable*, *logical architecture*).

Najvažnije mogućnosti CSLA aplikacijskog okvira su sljedeće:

* Implementacija poslovnih validacijskih pravila i održavanje liste prekršenih pravila.
* Praćenje stanja objekata, da li je objekt validan ili ne.
* Implementacija autorizacijskih pravila na razini objekata i svojstva objekata.
* N razina poništenja (*Undo*).
* Podrška za povezivanje (*binding*) objekata s korisničkim sučeljima: WPF, Windows Forms i Web Forms (ASP.NET).
* Spremanje objekata u bazu podataka i vraćanje natrag.
* Podrška za transakcije.
  1. **Sučelje aplikacije**

Sučelje aplikacije se izrađuje u raznim .NET tehnologijama, ovisno o zahtjevima i potrebama koje se očekuju od aplikacije. Aplikacija se može izvršavati na klijentskim desktop računalima, terenskim ili mobilnim uređajima ili im se može pristupati kroz Web preglednik.

* + 1. **WPF (*Windows Presentation Foundation*), Silverlight**

Ukoliko se odlučimo na izgradnju modernog desktop sučelja, tada je WPF prvi izbor zbog brojnih prednosti u usporedbi s klasičnim Windows formama. Ovdje se unaprijed ograđujemo od Windows RT (*Runtime*) i Metro sučelja koji su namijenjeni za razvoj asinkronih, prstima upravljanih aplikacija za novi Windows 8 operativni sustav.

WPF je novi grafički podsustav .NET platforme koji se temelji na XAML (*eXtensible Application Markup Language*) jeziku za označavanje. Neke od karakteristika WPF-a i XAML-a su sljedeće:

* Aplikacije su neovisne o rezoluciji, u potpunosti prilagodljive rezoluciji i veličini korisničkog zaslona.
* Donose visoku razinu podijeljenosti između korisničkog sučelja i poslovne logike.
* Može se ponašati kao desktop aplikacija ili aplikacija za Internet preglednik (XBAP - XAML *Browser Application*).
* Podržava napredne grafičke mogućnosti, 3D grafiku, fiksne ili prilagodljive elemente, vektorsku i rastersku grafiku, animacije, naprednu tipografiju, audio, video i povezivanje (*binding*) elemenata sučelja na objekte koji upravljaju sučeljem.
* Microsoft ga preporuča za izgradnju poslovnih aplikacija.
* Rasterećuje procesor i prenosi grafičke operacije na grafičku karticu.
* Kompatibilan je sa Silverlight tehnologijom, Web i Windows Phone inačicom WPF-a.
* Podržava predloške za definiranje dizajna.

1. **Zaključak**

U ovom radu uveli smo čitatelje vrlo površno u problematiku razvoja složenih poslovnih sustava u Microsoft .NET okolini. Broj dostupnih tehnologija i besplatnih aplikacijskih okvira (*framework*) koje možemo i trebamo što više koristiti u izgradnji aplikacije je vrlo velik. Svi nam oni omogućavaju apstrakciju određenih dijelova koda i sustavno rješavanje određenih problema s kojima se susrećemo prilikom izgradnje aplikacije. Kako smo već napomenuli na početku, tehnologiju nam daje Microsoft, a znanje (*know-how*) kako napraviti dobru aplikaciju nam daje zajednica Microsoft stručnjaka u obliku aplikacijskih okvira koji su najčešće besplatni i mogu se zajedno s originalnim kodom pronaći na Internetu.

Komunikacija između sučelja i baze podataka preko „ViewModel“ klasa za kontrolu sučelja, CSLA poslovnih objekata i na kraju NHibernate domenskih objekata, funkcionira u praksi vrlo dobro. Dobrim objektnim dizajnom i apstrakcijom dijelova koda uštede u fazi razvoja aplikacije mogu biti vrlo velike. Uz pomoć generatora koda i drugih pomoćnih alata razvoj se još više može ubrzati.



*Slika 3. – Logička arhitektura aplikacije u Microsoft .NET okolini.*

Nadam se da smo čitatelje barem malo zainteresirali da problemima arhitekture i dizajna aplikacije daju veću težinu prilikom razvoja novih aplikacija. Prikazali smo jedno vertikalno rješenje koje funkcionira u praksi. Najboljeg rješenja nema tako da treba tražiti prednosti i mane određenih pristupa i pronaći sredinu između jednostavnosti rješenja i apstrakcije koju postižemo slojevitošću i objektno orijentiranim pristupom kod izgradnje aplikacije.

**Literatura**

1. CSLA.NET framework, <http://www.lhotka.net/cslanet/> (10.5.2012).
2. Lhotka, Rockford (2009). Expert C# 2008 Business Objects. Apress, Berkeley.
3. Liberty, Jesse (2005). Programiranje C#. Dobar Plan, Zagreb.
4. MacDonald, Matthew (2008). Pro WPF in C# 2008 Windows Presentation Foundation with .NET 3.5. Second Edition, Apress, Berkeley.
5. Microsoft, 2012, Developer's Guide to Microsoft Prism 4.1: Building MVVM and Modular Applications with WPF and Silverlight, <http://msdn.microsoft.com/en-us/library/gg406140> (10.05.2012).
6. NHibernate - Relational Persistence for Idiomatic .NET: NHibernate Reference Documentation 3.3. <http://nhforge.org/doc/nh/en/index.html> (10.5.2012).
7. Smith, Josh (2009). WPF Apps With The Model-View-ViewModel Design Pattern. <http://msdn.microsoft.com/en-us/magazine/dd419663.aspx> (10.5.2012).