UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Luka Kralj

PODPORA RAZVOJU POSLOVNEGA NIVOJA APLIKACIJ Z OGRODJEM CSLA.NET

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

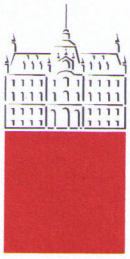
Mentor: doc. dr. Zoran Bosnić

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si



Št. naloge: 00465/2009
Datum: 01.09.2009

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **LUKA KRALJ**

Naslov: **PODPORA RAZVOJU POSLOVNEGA NIVOJA APLIKACIJ Z OGRODJEM CSLA.NET**

SUPPORT FOR BUSINESS LAYER APPLICATION DEVELOPMENT USING THE CSLA.NET FRAMEWORK

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

Tematika naloge:

Pri tronivojski delitvi razvoja aplikacij na podatkovni, predstavitveni in poslovni nivo v praksi primanjkuje standardov in tehnologij za implementacijo poslovnega nivoja. Ena od rešitev, ki naslavlja to problematiko, je ogrodje CSLA.NET (Component-based Scalable Logical Architecture), ki z objektno usmerjeno implementacijo poslovnih objektov predstavlja orodje za hiter in standardiziran razvoj velikih aplikacij.

Kandidat naj v diplomskem delu poda pregled nad ogrodjem in zgradbo CSLA.NET. Predstavi naj strukturo in organizacijo razredov v tem ogrodju in poda napotke, kako naj se programer loti razvoja aplikacij z uporabo tega ogrodja. V praktičnem delu diplomske naloge naj primerja razvoj aplikacije z uporabo tega ogrodja z razvojem aplikacije po klasični poti. Primerjavo med pristopi naj v zaključnem delu tudi ovrednoti.

Mentor:

doc. dr. Zoran Bosnić

Dekan:

prof. dr. Franc Solina

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a Luka Kralj,

z vpisno številko 63050274,

sem avtor/-ica diplomskega dela z naslovom:

Podpora razvoju poslovnega nivoja aplikacij z ogrodjem CSLA.NET

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  doc. dr. Zoran Bosnić

  in somentorstvom (naziv, ime in priimek)

  /

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 11.2.2010 Podpis avtorja/-ice: _____

Zahvala

Zahvaljujem se mentorju doc. dr. Zoranu Bosniću za vodenje, pregledovanje in nasvete pri izdelavi diplomske naloge.

Zahvala gre tudi podjetju RRC, ki mi je omogočilo strokovno gradivo, sodelavcu Juriju Leskovcu za seznanitev z ogrodjem CSLA.NET in strokovno pomoč.

Prav tako gre zahvala moji družini, ki mi je študij omogočila, Dunji Elikan za prevod povzetka in Petri Cimerman za vzpodbudo skozi celotni študij.

Kazalo

Seznam uporabljenih kratic in simbolov

CSLA.NET	Component-based Scalable Logical Architecture
SQL	Structured Query Language
WCF	Windows Communication Foundation
WPF	Windows Presentation Foundation
CRUD	Create, Read, Update and Delete

Povzetek

Z razvojem računalniških sistemov so se pojavile tudi večnivojske aplikacije oziroma sistemi. Te večnivojske aplikacije imajo ponavadi tronivojsko arhitekturo, ki je sestavljena iz podatkovne baze, uporabniškega vmesnika in poslovne logike. Za prva dva nivoja imamo na voljo uveljavljene tehnologije, zato se tu večinoma ne pojavljajo problemi pri razvoju. Pri poslovnem nivoju pa ne obstaja tehnologija, ki bi zadostno podpirala ta del aplikacije. Tako se zna zgoditi, da pričnemo izdelovati lastno ogrodje, ki bi nam olajšalo razvoj poslovnega nivoja. Tu moramo poskrbeti za nekaj kompleksnih funkcionanosti kot so transportne tehnologije, serializacija, varnost, sledenje stanju objektov itd.

Osrednja tema v diplomski nalogi je predstavitev ogrodja CSLA.NET za razvoj porazdeljenih poslovnih aplikacij. To ogrodje se osredotoča predvsem na poslovni nivo aplikacije. Omogoča hiter razvoj poslovne aplikacije, saj večina tehničnih stvari na tem nivoju odpade, tako da se lahko posvetimo poslovni logiki in poslovnim razredom. V zaključku diplomske naloge primerjamo aplikacijo, razvito z ogrodjem CSLA.NET, s klasično razvito enakovredno aplikacijo ter ovrednotimo prednosti in slabosti obeh.

Ključne besede:

CSLA.NET, ogrodje .NET, logična arhitektura, fizična arhitektura, poslovni objekt

Abstract

The development of computer systems has caused the emergence of so called n-tier applications or systems which usually have a three-level structure consisting of a database, user interface and business logic. Since the first two levels can operate through established technologies, there are usually no difficulties in their process of development. On the business logic level, however, there are no technologies which would sufficiently support this part of the application. That is why we may start building our own framework to facilitate the development of the business level. We have to provide several complex functionalities, such as transport technologies, serialization, security, object-state tracking etc.

The thesis aims primarily to present the CSLA.NET framework for development of distributed business applications. This framework mainly focuses on the business level of the application. It enables a rapid development of a business application because the majority of technical matters do not take place on this level; we can then concentrate on the business logic and business classes. In the final section of the thesis we compare a sample application developed with the CSLA.NET framework with an equivalent, conventionally developed application, and assess advantages and disadvantages of both applications.

Key words:

CSLA.NET, .NET Framework, logical arhitecture, physical arhitecture, business object

1. Uvod

Pri razvoju poslovnih aplikacij se pogosto pojavlja problem neproduktivnega programiranja, ki se odraža predvsem v razvoju lastnih ogrodij za pomoč pri izdelavi aplikacije. Takšna ogrodja kmalu postanejo prezahtevna za vzdrževanje ali pa jih lahko razvijalci preveč prilagodijo določeni aplikaciji in si tako otežijo njihovo nadaljnjo splošno uporabnost. Dodatni problem predstavlja tudi dokumentacija, saj je znano, da v današnjem svetu razvoja aplikacij ne ostane kaj dosti časa za dokumentiranje lastnih ogrodij oz. nam ostane čas le za osnovno dokumentacijo, ki sodi k projektu. Zato ima vsak novi član razvojne ekipe, ki mora začeti delati z takšnim ogrodjem, na začetku težave, saj se nima iz česa učiti in tako kmalu dobi odpor do dela z ogrodjem. Veliko razvijalcev in podjetij se tako obrača k že razvitim in uveljavljenim razvojnim ogrodjem.

Primer takega ogrodja je odprtokodno ogrodje CSLA.NET avtorja Rockforda Lhotke, ki je tudi najbolj razširjeno med razvijalci v Microsoftovi tehnologiji .NET. Ogrodje se uporablja za razvoj porazdeljenih poslovnih aplikacij z uporabo objektnih pristopov. Predstavlja dobro alternativo klasičnim pristopom, saj se razvijalec osredotoča le še na poslovno vsebino aplikacije. CSLA.NET namreč reši razvijalca tistega, čemur pravimo »drobno tehnično programiranje« (angl. *plumbing*), tako da nam veliko tehničnih podrobnosti odpade. CSLA.NET objekti so samostojne poslovne entitete, ki poleg podatkov vsebujejo tudi poslovno logiko, preverbo podatkov in avtorizacijo. So avtomatično mobilni, imajo dobro podporo za podatkovno povezovanje (angl. *data binding*) v različnih predstavitvenih tehnologijah in so popolnoma neodvisni od predstavitvene plasti.

Cilj diplomskega dela je raziskati in predstaviti ogrodje CSLA.NET, njegov namen, funkcionalnosti ter na kaj je potrebno biti pozoren pri njegovi uporabi. Pred samim pregledom ogrodja bomo nekaj poudarka namenili računalniški arhitekturi, saj je za razumevanje predmetnega ogrodja potrebno dobro razlikovati med logično in fizično arhitekturo. Pregled ogrodja bomo pričeli z predstavitvijo pojma »poslovni objekt«, nadaljevali z smernicami, ki jih je zastavil avtor ogrodja, nato z osnovnimi cilji, na koncu pa še s teoretično realizacijo ogrodja. Po tem pregledu bomo še na praktičnem primeru prikazali in ocenili uporabnost ogrodja in smotrnost uporabe.

Diploma je razdeljena na tri vsebinske dele, in sicer na teoretičen del, analizo ogrodja ter praktični del. V poglavju 2 nekaj besed namenimo logični in fizični arhitekturi aplikacij, saj v splošni praksi razlikovanje med tema dvema arhitekturama ni v celoti jasno. Sledi še

predstavitev petplastne logične arhitekture ter njene preslikave v različne fizične konfiguracije. V poglavju 3 si bomo ogledali načrtovalske cilje in smernice ogrodja ter nato samo oblikovanje le-tega. V poglavju 4 bomo pogledali praktično uporabo ogrodja CSLA.NET, ki ga bomo primerjali z razvojem aplikacije po klasični poti. V zadnjem, 5. poglavju, sledijo sklepne ugotovitve.

2. Logična in fizična arhitektura

Objektno usmerjena aplikacija mora biti izdelana tako, da lahko deluje na več različnih fizičnih računalniških konfiguracijah [1]. Da dosežemo ta učinek, pa moramo dobro razlikovati med logičnim in fizičnim modelom aplikacije. V razvoju porazdeljenih objektno usmerjenih aplikacij se pogosto srečujemo z pojmi kot sta *plast* (angl. *layer*) in *nivo* (angl. *tier*), za katera si na začetku poglejmo, kaj predstavljata. Nivo predstavlja fizično reprezentacijo, definirano s fizičnim strežnikom ali računalnikom. Plast pa je del logičnega sistema, ki je vsebovan v enem ali v več nivojih. Za dobro razumevanje ogrodja CSLA.NET moramo poznati tudi petplastno logično arhitekturo, ki si jo bomo ogledali v razdelku 2.2. Pri samem opisu ogrodja v poglavju 3 bomo namreč prikazali vmestitev CSLA.NET v to petplastno logično arhitekturo.

V nadaljevanju bomo v razdelku 2.1 primerjali arhitekturi ter pojasnili kako zasnovati obe. V razdelku 2.2 pa si bomo podrobneje pogledali petplastno logično arhitekturo.

2.1. Primerjava arhitektur

Velikokrat, kadar govorimo o večnivojski arhitekturi, govorimo o fizičnem modelu, torej o tem, kako je aplikacija razporejena na več računalnikov z različnimi funkcionalnostmi (odjemalec, aplikacijski strežnik, spletni strežnik, podatkovni strežnik). Čeprav so vse to večnivojski sistemi, večina misli, da je med fizičnim in logičnim modelom povezava ena proti ena, kar pa ni vedno res. Fizični model je namreč lahko drugačen od logičnega večplastnega modela. Logični model nima nobene povezave s tem, na koliko strežnikih bo aplikacija tekla, temveč predstavlja porazdelitev različnih tipov funkcionalnosti v aplikaciji. Najbolj pogosta porazdelitev le-teh je na predstavitveno (angl. *presentation*), poslovno (angl. *business*) in podatkovno (angl. *data*) plast. Vse te plasti lahko tečejo na enem, dveh ali treh računalnikih. Potrebno se je zavedati, da obstaja povezava med tema dvema arhitekturama - logična arhitektura mora imeti vsaj toliko plasti, kolikor je fizičnih nivojev, ali več. Žal se pogosto pojavlja, da veliko aplikacij nima dobro zasnovanega logičnega modela, saj logični model predstavlja zgolj preslikavo fizčnega. V primeru, da je sistem zasnovan tako, da teče na dveh fizičnih nivojih, in kasneje število nivojev spremenimo, lahko pridemo do težav. Če pa že na začetku smotrno zastavimo logično arhitekturo treh nivojev, lahko kasneje enostavno preklapljamo med eno-, dvo- in tronivojsko fizično arhitekturo. S tem dosežemo logično organizirano kodo, lažje vzdrževanje, boljšo ponovno uporabnost kode, boljše razvijalske

izkušnje in čistost kodiranja. Po drugi strani pravilno izbrana fizična arhitektura prinaša tudi naslednje prednosti: hitrost, skalabilnost, robustnost in varnost.

2.1.1. Zasnova logičnega in fizičnega modela

Ko začnemo razvijati aplikacijo, je pomembno, da se na začetku načrtuje logična arhitektura, ki definira vloge vseh komponent in razdeli funkcionalnosti. Vključevati mora zadostno število plasti, da smo lahko fleksibilni pri izbiri fizične arhitekture. Ponavadi imamo troplastno arhitekturo (predstavitvena, poslovna, podatkovna), ki pa čedalje pogosteje ne zadošča, saj je lahko fizično razdeljena na dva dela tako predstavitvena (brskalnik in spletni strežnik), kot tudi poslovna plast (odjemalec ali spletni strežnik ali aplikacijski strežnik). Tako preidemo na štiri- do šestplastno logično arhitekturo. Obravnavano ogrodje CSLA.NET je zasnovano na petplastni logični arhitekturi.

Pri dobro zasnovanem logičnem modelu imamo pri konfiguraciji fizičnega modela več fleksibilnosti, ki bo odvisna predvsem od učinkovitosti.

2.1.2. Medplastna komunikacija

Dejstvo, da smo razdelili aplikacijo na več plasti, še ne pomeni, da je ne moremo razdeliti tudi preko več fizičnih nivojev. Koda v eni plasti komunicira s plastjo, ki je v arhitekturi nad ali pod njo. Primer medplastne komunikacije je npr. meja med poslovno in podatkovno plastjo, ki je močno optimizirana, saj imajo aplikacije običajno med njima še mrežno plast. Meja med predstavitveno in poslovno plastjo pa zaradi podatkovnega povezovanja pogosto ni optimizirana in je zato med te dve plasti neprimerno vriniti še mrežno plast.

2.2. Petplastna logična arhitektura

Slika 1 prikazuje petplastno logično arhitekturo, za katero bomo zaradi boljšega razumevanja v tem razdelku podrobneje razčlenili funkcionalnosti.

Slika 1. Petplastna logična arhitektura

Predstavitvena plast (angl. *presentation layer*): ni enaka pojmu *uporabniškega vmesnika* (UV) [2], razliko pa opisujemo v nadaljevanju. V smislu okenskih sistemov sta predstavitvena plast in UV združeni v eno, ki se imenuje GUV (*Grafični Uporabniški Vmesnik*). S perspektive spletnih aplikacij pa je ta razdelitev bolj razumljiva. Spletni brskalnik zgolj prikazuje podatke uporabniku in sprejema njegove vnose (angl. *user input*), vsa interakcijska koda pa se izvaja ločeno na spletnem strežniku.

Uporabniški vmesnik (angl. *user interface*): v tej plasti se nahaja koda, ki določa, kaj uporabnik vidi in kako se odzvati na njegove vnose. V aplikacijah vrste *Windows Forms* in znotraj *Windows Presentation Foundation* [3] (grafični podsistem za upodabljanje grafičnih vmesnikov v okenskih aplikacijah), to predstavlja kodo v ozadju, ki se odziva na vnose uporabnika. Ta plast služi kot vmesni sloj med predstavitveno plastjo in poslovno logiko.

Poslovna plast (angl. *business layer*): ta plast vključuje vsa poslovna pravila, preverbe vnosa pravilnosti podatov, manipulacijo, procesiranje in avtorizacijo za aplikacijo. Poslovna logika mora biti ločena od plasti vmesnika, čeprav se lahko nekaj poslovne kode podvoji tudi na plasti vmesnika za bogatejšo uporabniško izkušnjo, vendar mora poslovna plast vsebovati vso poslovno logiko, saj je to osnovna točka za kontrolo in vzdrževanje. Ta razdelitev plasti kontrole vmesnika in poslovne plasti je ključnega pomena, če želimo pridobiti na vzdrževanju in ponovni uporabnosti kode.

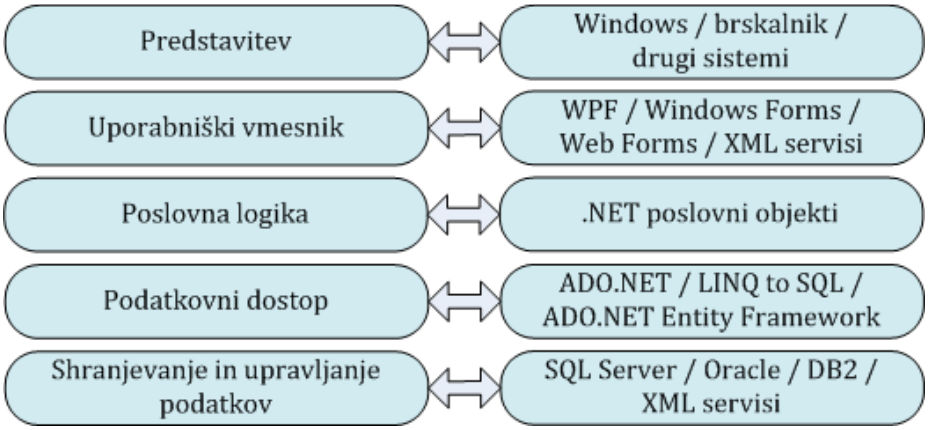
Podatkovni dostop (angl. *data access*): ta plast komunicira s plastjo za pridobivanje, vnašanje, posodabljanje in brisanje podatkov. Dejansko ne upravlja in shranjuje informacij, temveč zgolj zagotavlja vmesnik med poslovno logiko in podatkovno bazo.

Shranjevanje in upravljanje podatkov (angl. *data storage and management*): ta plast predstavlja podatkovne strežnike, kot so SQL Server in Oracle, ki že sami upravljajo z nalogami shranjevanja in upravljanja s podatki. Skrbi za fizično ustvarjanje, pridobivanje, posodabljanje in brisanje podatkov.

3. CSLA.NET

V 2. poglavju smo si pogledali osnovne koncepte fizične in logične večplastne arhitekture, vključno z opisom sistema z petplastnim logičnim modelom. Ta model nam služi za razumevanje osnovne ideje ogrodja CSLA.NET, ki ga predstavljamo v nadaljevanju tega poglavja.

Ogrodje CSLA.NET se ostredotoča na poslovno logiko in na dostopanje do podatkovne plasti. Na to se osredotoča predvsem zaradi dejstva, da že obstajajo napredne tehnologije za izgradnjo okenskih, spletnih in mobilnih vmesnikov. Prav tako že obstajajo napredne tehnologije za hranjenje podatkov, kot so podatkovne baze MS SQL Server, Oracle, DB2 itd. Povezavo med logičnimi plastmi petplastne logične arhitekture (razdelek 2.2) in omenjenimi tehnologijami ponazarja slika 2. Z uporabo teh ogrodij za izdelavo uporabniškega vmesnika in upravljanje z podatki, se lahko torej bolj posvetimo poslovni logiki, kar je tudi namen ogrodja CSLA.NET.



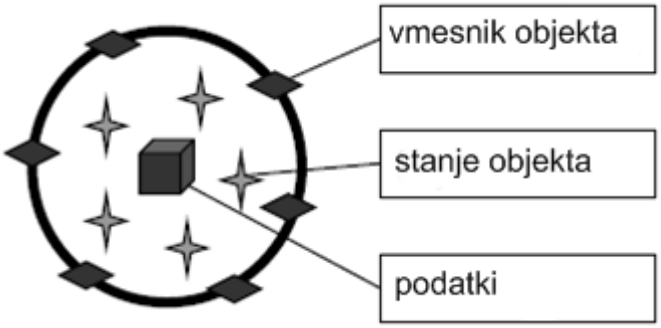
Slika 2. Povezave med logičnimi plastmi in tehnologijami

V tem delu bomo pregledali glavne smernice ogrodja, (razdelek 3.1), načrtovalske cilje (razdelek 3.2) in samo implementacijo ogrodja (razdelek 3.3).

Preden nadaljujemo s pregledom ogrodja, je potrebno razjasniti pojem poslovnega objekta (angl. *business object*). Glavni cilj, kadar načrtujemo programski objekt, je abstraktna predstavitev entitete ali koncepta [6]. Glavni ključ do uspeha je ovijanje ali inkapsulacija (angl. *encapsulation*). To pomeni, da igra objekt vlogo črne škatle (angl. »black box«), ki vsebuje podatke in logiko. Uporabnik tega objekta torej ne ve, s kakšnimi podatki in po kakšni logiki ta sistem deluje, temveč samo interagira z objektom. Razlika med navadnim in

poslovnim objektom je le v tem, kaj poslovni objekt predstavlja. V primeru poslovnega objekta je ta del poslovne ali problemske domene.

Podatki v poslovnem objektu so »pametni«, kadar objekt vsebuje vsa poslovna pravila, ki spremljajo te podatke. S takim pristopom zagotovimo, da bodo poslovna pravila, manipulacije, kalkulacije in ostali procesi enako izvedeni kjerkoli v aplikaciji. V nekem smislu postanejo podatki torej samozavedni in se zavarujejo pred nepravilno uporabo. Struktura poslovnega objekta je prikazana na sliki 3.



Slika 3. Poslovni objekt sestavljen iz vmesnika, stanja objekta in podatkov

3.1. Glavne smernice ogrodja CSLA.NET

Glavne smernice pri zasnovi ogrodja CSLA.NET so:

- poenostavljena izdelava objektno usmerjenih porazdeljenih aplikacij v okolju .NET,
- razvijalec vmesnikov (windows, spletne storitve) ne sme vedeti, kakšne tehnologije skrbijo za podatke, temveč se mora opirati na objektno usmerjen model problemske domene,
- razvijalec poslovne logike naj bi uporabljal enostavne tehnike kodiranja z uporabo spremenljivk, lastnosti in metod. Veliko dodatnega znanja naj ne bi bilo potrebno,
- poslovni razredi morajo zagotoviti popolno ovijanje poslovne logike, vključno s preverjanjem pravilnosti, manipulacijo, računanjem in avtorizacijo. Vse, kar se nanaša na entiteto problemske domene, moramo imeti v posameznem razredu,
- ločevanje poslovne in podatkovne kode,
- enostavna izdelava generatorjev kode ali osnutkov za obstoječe generatorje za pomoč pri izdelavi poslovnih razredov,
- večplastna logična arhitektura, ki se jo da enostavno preurediti za delovanje na enem do štirih fizičnih nivojih,
- če uporabljamo kompleksne posebnosti v .NET kot so WCF (*Windows Communication Foundation*), serializacija, varnost in distribucija, jih je potrebno skriti in avtomatizirati.

3.2. Osnovni načrtovalski cilji

Kadar se lotimo ustvarjanja objektno usmerjene aplikacije, je zaželeno, da so vsi neposlovni objekti, kot so grafične kontrole in podatkovni dostop, že na voljo. V tem primeru se lahko razvijalci osredotočijo na izdelavo in testiranje poslovnih objektov in tako zagotovijo, da vsak objekt ovije podatke in poslovno logiko, potrebno za delovanje aplikacije.

Čeprav je ogrodje .NET močno, ne vsebuje vseh neposlovnih objektov, potrebnih za izdelavo večine poslovnih aplikacij. Tako moramo vložiti veliko dela, preden lahko začnemo s pisanjem poslovne logike, saj obstaja veliko visokonivojskih funkcij in zmogljivosti, ki jih potrebujemo, .NET pa jih ne vsebuje. Te so:

- preverbe podatkov in vdrževanje seznama pokvarjenih poslovnih pravil,
- standard implementiranja poslovnih in validacijskih pravil,
- sledenje stanju podatkov v objektu (ali so podatki spremenjeni),
- integrirana avtorizacijska pravila na objektnem in lastnostnem nivoju,
- tipizirani seznami (t.i. *child*) objektov,
- sposobnost večnivojske razveljavitve sprememb,
- preprost in abstrakten model za razvijalce vmesnikov,
- popolna podpora za podatkovno povezovanje v WPF, Windows Forms in Web Forms,
- shranjevanje in pridobivanje objektov iz podatkovne baze,
- ročna prijava (avtentikacija).

V vseh teh primerih ogrodje .NET nudi vse dele sestavljanke, vendar pa jih moramo sestaviti, da ustrezajo specifičnim zahtevam. Ne želimo pa jih sestavljati za vsak objekt posebej, temveč želimo to narediti le enkrat ter s tem zagotoviti dodatne funkcianalnosti vsem objektom. Ob vsem tem pa morajo le-ti ohraniti:

- abstrakcijo,
- enkapsulacijo,
- polimorfizem,
- dedovanje.

Rezultat tega je poslovno ogrodje, ki je sestavljeno iz številnih razredov.

3.2.1. Preverbe podatkov in poslovna pravila

Veliko poslovne logike vključuje izvajanje pravil za preverbo podatkov. Ta pravila preverjajo, ali je npr. vnos kakšnega podatka obvezen ali pa če mora biti določen datum pred nekim drugim datumom. Pravilo je lahko veljavno ali pa ne. V primeru, da je kakšno pravilo neveljavno, je tudi objekt neveljaven. Podoben princip uporabljajo poslovna pravila, kot so preverjanje, če mora biti tekst v velikih črkah, izračun ene spremeljivke na podlagi drugih spremenljivk itd.

Ker vse preverbe kot rezultat vračajo logično vrednost (boolean), lahko abstrahiramo koncept pravil. Pravilo je torej lahko izraženo kot metoda, ki vrne logično vrednost. Poslovna pravila tipično spremenijo stanje objekta in običajno ne vključujejo istočasno tudi preverbe.

Ogrodje .NET ponuja koncept delegatov. Z uporabo delegatov je v ogrodju CSLA.NET formalno definirana metoda za vse preverbe in poslovna pravila. To dovoljuje ogrodju, da vzdržuje seznam pravil za vsak objekt. Drugače povedano, vsak objekt enostavno vzdržuje seznam neveljavnih pravil in ima standardiziran način implementacije poslovnih pravil. To, da lahko pravila izdelamo kot metode, pomeni, da lahko naredimo knjižnico skupnih pravil, ki jih nato uporabljamo v aplikacijah in jih nam ni potrebno znova in znova implementirati.

Za sledenje, katera pravila so neveljavna, skrbi ogrodje. Uporabniku omogoča, da kadarkoli preveri vsa poslovna pravila. To preverjanje vrne seznam neveljavnih pravil, katerega lahko razvijalec vmesnikov uporabi in prikaže.

3.2.2. Sledenje stanju podatkov v objektu

Pri tem konceptu se objekt zaveda, v kakšnem stanju so njegovi podatki. To je pomembno za učinkovitost posodabljanja zapisov v podatkovni bazi, da ne bi po nepotrebnem posodabljali zapisa, ki se v resnici ni spremenil. Takšno preverjanje bi lahko načeloma pregledovali tudi že na samem vmesniku, vendar je bolje, če za to skrbi kar objekt sam. Tako dobimo generičen mehanizem, pri katerem lahko poslovna logika sporoča ogrodju, kateri objekti so se spremenili.

3.2.3. Integrirana avtorizacija

Aplikacija mora avtorizirati uporabnika, da lahko ta opravlja (ali ne sme opravljati) določene operacije ali vidi določene podatke. Takšne avtorizacije so tipično del uporabniških nastavitev ali vlog. Vsaka vloga pa ima določene pravice v aplikaciji in nad podatki.

Ogrodje CSLA.NET podpira dve integrirani avtentikaciji (windows in domenska), prav tako pa tudi ročno avtentikacijo. Rezultat vseh teh avtentikacij je seznam vlog, ki pripadajo uporabniku. Čeprav je lahko avtorizacijska koda napisana v poslovni kodi aplikacije, pa poslovno ogrodje v nekaterih primerih pomaga formalizirati proces. Objekti morajo imeti informacije o uporabnikovih vlogah, da lahko omejijo dostop do podatkov glede na uporabnika. Na objektnem nivoju pa omejujejo nalaganje, bisanje in shranjevanje.

Poleg poslovnega objekta mora tudi uporabniški vmesnik dostopati do avtorizacijske logike. Dober uporabniški vmesnik spreminja svoj izgled glede na pravice, ki jih ima trenutni uporabnik na objektu. Za podporu temu konceptu pomaga CSLA.NET poslovnim objektom izpostaviti avtorizacijska pravila tako, da so dostopna na predstavitveni plasti brez podvajanja pravil.

3.2.4. Tipizirani seznami otrok

Ogrodje .NET ponuja veliko tipiziranih podatkovnih struktur za delo s seznami. Takim seznamom pravimo generični seznami, ki so bili predstavljeni v ogrodju .NET Framework 2.0. Žal pa osnovne funkcionalnosti, ki jih ponujajo generični seznami, niso dovolj za popolno integracijo z ogrodjem CSLA.NET. Ogrodje CSLA.NET podpira veliko naprednih funkcionalnosti, kot so: preverbe pravil, razveljavitev sprememb itd. Za podporo teh funkcionalnosti se zahteva, da seznam podobjektov komunicira s starševskim objektom in da objekti v seznamu niso implementirani z navadnimi .NET razredi. Kot primer vzemimo seznam podobjektov, ki mora signalizirati, če je bil katerikoli objekt spremenjen. To bi lahko programer naredil tako, da bi šel skozi seznam in za vsak objekt posebej pogledal, če je označen kot spremenjen (angl. *IsDirty*). Bolj smiselno je, če je to implementirano že kar v ogrodju samem. Enako je s preverbami, če je eden od objektov v seznamu neveljaven - takrat mora seznam to tudi sporočiti in obratno.

Cilj ogrodja je, da je izdelava tipiziranih seznamov podobna običajnim .NET vzorcem.

3.2.5. Možnost večnivojske razveljavitve

Veliko aplikacij ponuja uporabniku vmesnik, ki vključuje gumbe »V redu« in »Prekliči«. Kadar uporabnik pritisne gumb »V redu«, se pričakuje, da se vse spremembe, ki jih je uporabnik naredil na uporabniškem vmesniku, shranijo. Enako velja za gumb »Prekliči«, le da se tu spremembe razveljavijo.

3.2.6. Preprost in abstrakten model za razvijalce grafičnega vmesnika

Eden od ciljev ogrodja je tudi to, da bi imeli malo interakcijske kode na vmesniku. Na voljo imamo tri modele:

- *UI-in-charge*
  Ta model uporablja .NET lastnost prenašanja objektov po vrednosti, vendar zahteva da razvijalec uporabniškega vmesnika pozna način, kako interagirati z aplikacijskim strežnikom.

- *Object-in-charge*
  Druga možnost je, da znanje interakcije z aplikacijskim strežnikom premaknemo v objekte same. S tem dobimo enostavno kodo uporabniškega vmesnika, a kompleksnejšo kodo objekta, kar se odraža v tem, da se objekta ne da prenašati po vrednosti.

- *Class-in-charge (Factory Pattern)*
  Ta model prinaša dober kompromis z relativno enostavno kodo na uporabniškem vmesniku, ki se ne zaveda aplikacijskega strežnika in uporablja pošiljanje objekta po vrednosti. Ker s tem skrijemo veliko informacij pred uporabniškim vmesnikom, se povečata abstrakcija in fleksibilnost.

Za upoštevanje *class in charge* pristopa, uporabimo t. i. statične tovarniške metode (angl. *Factory methods*), ki skrbijo za kreiranje in pridobivanje objektov, kot je prikazano na sliki 4.

```
// Constructor (private)
private User()
{
}


// New
public static User New()
{
    return DataPortal.Create<User>();
}
```

Slika 4. Programska koda, ki prikazuje primer tovarniške metode. Na sliki sta prikazana konstruktor in statična metoda *New*, ki pokliče metodo *Create* na podatkovnem portalu, ta pa vrne objekt ustreznaga tipa.

Koda uporabniškega vmesnika obsega samo klic te metode, ki pokliče aplikacijski strežnik in mu naloži, naj naredi nov objekt ter ga napolni z privzetimi vrednostmi. Ko strežnik naredi objekt, ga vrne metodi, ki teče na odjemalcu. Sedaj, ko je bil objekt vrnjen po vrednosti, ga enostavno vrnemo uporabniškemu vmesniku.

Po tem principu moramo narediti še statične metode za shranjevanje in za pridobivanje objekta. Tak pristop izkorišča lastosti ogrodja .NET in je najboljši način za minimizacijo kode.

3.2.7. Podpiranje podatkovnega povezovanja

Že pred več kot desetimi leti je Microsoft v .NET vključil možnost podatkovnega povezovanja (angl. *data binding*). To omogoča razvijalcem, da razvijejo okna in jih napolnijo s podatki s skoraj nič dodatne kode. Posamezna kontrola je povezana z lastostjo ali poljem v objektu ali v podatkovni zbirki. Ta lastnost je vključena v WPF, Win Forms in Web Forms, prinaša pa naslednje prednosti:

- dobra učinkovitost, kontrola in fleksibilnost,
- lahko povežemo kontrolo na posamezno polje v poslovnem objektu,
- zmanjšanje kode v uporabniškem vmesniku,
- hitrejše kot ročno kodiranje, še posebej kadar nalagamo podatke v bolj kompleksne kontrole.

Med vsemi prednostmi najbolj izstopa zmanjšanje kode v uporabniškem vmesniku, kar omogoča lažje vzdrževanje. V WPF, Win Forms in Web Forms je podatkovno povezovanje obojesmerno (bralno-pisalno), kar pomeni, da vežemo podatek na tekstovno polje in ko se ta podatek spremeni (bodisi v objektu bodisi na vmesniku), se spremembe izvedejo tudi v obratni smeri.

3.2.8. Ročna avtentikacija

Varnost v aplikaciji je ponavadi obsežno poglavje. Aplikacija mora avtenticirati uporabnika, kar pomeni, da mora preveriti njegovo identiteto in pridobiti uporabnikove podatke, ki jih uporabi za avtorizacijo. CSLA.NET direktno podpira integrirano varnost. To pomeni, da lahko uporabimo objekte znotraj ogrodja za določanje uporabnikove identitete znotraj okolja Windows ali domene, ki ji pripada.

Ogrodje CSLA.NET vključuje podporo za ročno avtentikacijo, v kateri razvijalec definira, kako bo uporabnikove pravice preverjal in kako bo te vloge nalagal. Avtentikacija po meri je model, s katerim lahko prilagodimo aplikacijo, da uporablja že obstoječe varnostne tabele ali storitve.

3.3. Oblikovanje ogrodja

Po pregledu glavnih načrtovalskih ciljev ogrodja CSLA.NET lahko pogledamo, kako so le-ti implementirani v ogrodju samem. Samo ogrodje vključuje veliko razredov in podatkovnih tipov, kar zna biti prekompleksno, če bi ga želeli obravnavati do potankosti. Za boljše razumevanje ga lahko razdelimo na manjše funkcionalne enote, ki so:

- kreiranje poslovnih objektov,
- večnivojska razveljavitev,
- podatkovno povezovanje,
- validacija in poslovna pravila,
- podatkovni portal,
- preklop med različnimi fizičnimi konfiguracijami,
- avtentikacija in avtorizacija.

V naslednjih razdelkih bomo podrobneje opisali vsako od teh funkcionalnih enot.

3.3.1. Kreiranje poslovnih objektov

Ključni razredi v ogrodju so tisti, ki jih razvijalci uporabljajo za kreiranje poslovnih objektov (prikazani so na sliki 5). Na sliki je prikazan le del razredov, ki so na voljo. Veliko je tudi razredov, ki jih pri uporabi ne vidimo.

Slika 5. Razredi ogrodja, ki jih uporabljajo razvijalci poslovnih razredov

Če želimo uporabljati ogrodje CSLA.NET, morajo vsi naši poslovni objekti dedovati iz enega od prikazanih osnovnih razredov. Tabela 1 prikazuje relacijo med osnovnimi razredi in njihovim namenom.

Razred	**Namen (ko dedujemo iz razreda)**
BusinessBase<T>	Ustvarimo objekt za urejanje, kot so Uporabnik, Vloga, ...
BusinessListBase<T,C>	Ustvarimo objekt za seznam poslovnih objektov
EditableRootListBase<C>	Seznam poslovnih objektov, kjer so spremembe za vsak objekt avtomatsko shranjene, ko se uporabnik premka iz objekta na drug objekt (tabelarični prikaz in urejanje podatkov)
CommandBase	Ukaz, ki teče na aplikacijskem strežniku
ReadOnlyBase<T>	Poslovni objekt, namenjen samo branju
ReadOnlyListBase<T,C>	Seznam objektov, namenjen samo branju
NameValueListBase<K,V>	Seznam (ključ/vrednost), namenjen samo branju

Tabela 1. Osnovni razredi in njihov namen

Glede na funkcionalnost in obnašanje ločimo sledeče tipe poslovnih objektov

Stereotip	Opis	Osnovni razred
Editable root	Objekt vsebuje bralno-pisalna polja. Objekt lahko pridobimo in zapišemo direktno v podatkovno bazo.	BusinessBase<T>
Editable child	Objekt vsebuje bralno-pisalna polja. Objekt je vsebovan v drugemu objektu in ga ne moremo pridobiti/shraniti direktno v bazo. Ko se starševski objekt inicializira, ga podatkovni portal označi za »otroka«.	BusinessListBase<T,C>
Editable root list	Seznam objektov, ki vsebuje *Editable root* objekte. Seznam se lahko direktno shrani ali bere iz baze.	BusinessBase<T>
Editable child list	Seznam objektov, ki vsebuje *Editable child* objekte. Seznam je vsebovan v nekem drugem objektu.	BusinessListBase<T,C>
Dynamic root list	Seznam objektov tipa *Editable root*. Seznam se lahko direktno shrani ali bere iz baze. Uporablja se ga za sezname, prikazane v tabeli, kjer lahko podatke tudi urejamo (npr. MS SQL Server tabelarični urejevalnik podatkov).	BusinessListBase<T,C>
Command	Objekt, ki izvede ukaz na aplikacjskem strežniku in vrne rezultat.	CommandBase
Read-only root	Objekt, ki vsebuje samo bralna polja. Direktno ga beremo iz baze.	ReadOnlyBase<T>
Read-only child	Objekt, ki vsebuje samo bralna polja. Vsebovan v drugem objektu. Ne moremo ga pridobiti direktno iz baze.	ReadOnlyBase<T>
Read-only root list	Seznam objektov tipa *Read-only root*, dobimo ga direktno iz baze.	ReadOnlyListBase<T,C>
Read-only child list	Seznam objektov tipa *Read-only child*, ki je vsebovan v drugem objektu. Ne moremo ga pridobiti direktno iz baze.	ReadOnlyListBase<T>
Name/value list	Seznam objektov, ki vsebujejo ključ, vrednost. Namenjeno za prikazovanje šifrantov.	NameValueListBase<K,V>

Tabela 2. Seznam in opis CSLA stereotipov

3.3.2. Večnivojska funkcionalnost razveljavitve

Razveljavitev deluje tako, da ob vsaki spremembi podatkov v objektu vzame sliko objekta in ga serializira v podatkovni tok bitov. Za razveljavitev sprememb je potrebno iz podatkovnega toka torej kreirati objekt. Slabost te pretvorbe je v tem, da vrne novi primerek objekta in je zato potrebno vse ostale objekte, ki so imeli referenco na ta objekt, osvežiti.

Razred *BusinessBase* deduje iz *UndoableBase* in s tem pridobi zmožnost večnivojske razveljavitve. Ker pa vsi poslovni objekti dedujejo iz *BusinessBase*, imajo posledično tudi ti objekti to funkcionalnost. Za uporabo te funkcionalnosti imamo na voljo tri metode:

- *BeginEdit()* – pove objektu, da naredi sliko trenutnega stanja objekta v pripravi na urejanje. Vsakič, ko je ta metoda poklicana, se naredi nova slika. Vse te slike se hranijo v pomnilniku, tako da jih lahko obnovimo ob klicu funkcije *CancelEdit()*,
- *CancelEdit()* – pove objektu, da obnovi objekt na najnovejše stanje. To učinkovito izvede operacijo razveljavitve in prekliče spremembe do globine enega nivoja. Če je *CancelEdit()* poklican tolikokrat kot *BeginEdit()*, se objekt vrne v originalno stanje,
- *ApplyEdit()* – pove objektu, da zbriše najnovejši posnetek in pusti objekt tak, kot je. Če je *ApplyEdit()* poklican tolikokrat kot *BeginEdit()*, zbrišemo vse posnetke in tako naredimo vse spremembe dokončne.

3.3.3. Podpora povezovanju podatkov

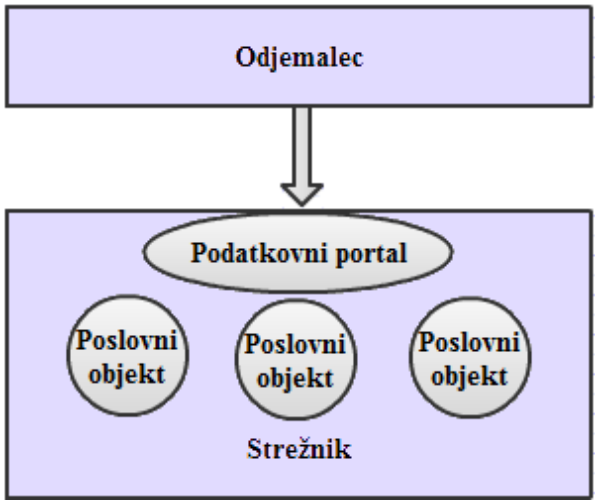
Čeprav ogrodje .NET že samo po sebi podpira to funkcionalnost, lahko sam objekt ponuja bolj popolno povezovanje podatkov z implementiranjem nekaterih *.NET Data binding* vmesnikov. Osnovni razred za poslovne objekte *BusinessBase* implementira *IEditableObject* in *IdataErrorInfo* vmesnik in tako podpira povezovanje v *WPF*, *Windows Forms* in *Web Forms* uporabniških vmesnikih.

3.3.4. Poslovna pravila in pravila preverjanja pravilnosti vnosov

Eden od ciljev ogrodja je poenostavitev in standardizacija poslovnih in preverbenih pravil kot tudi preverjanje le-teh in vodenje seznama neuspešno izvedenih pravil. Seznam slednjih je namenjen prikazu v uporabniškem vmesniku, da lahko uporabniku prikažemo, kaj točno je s podatki narobe. Da bi zagotovili to funkcionalnost vsem poslovnim objektom, je vse to implementirano v razredu *BusinessBase*.

3.3.5. Podatkovni portal

Zagotavljanje podatkovne integritete je pri shranjevanju in pridobivanju podatkov iz podatkovne baze lahko kompleksno. Ob vsem tem pa je prisotna še splošna zahteva po možnosti neodvisnosti od fizične konfiguracije. Za ta namen ponuja CSLA.NET enotno vstopno točko do aplikacijskega strežnika, na katero lahko nastavimo vsakega odjemalca. S tem je omogočen nadzor nad poslovnimi objekti za čas, ko so na strežniku in izvajajo kodo za dostop do podatkov. Osnovni koncept takega dostopa je prikazan na sliki 6.



Slika 6. Shema poenotene vstopne točke

Z uporabo podatkovnega portala pridobimo naslednje ključne prednosti:

- centralizirana varnost pri klicu funkcij na aplikacijskem strežniku,
- konsisteten mehanizem pri izvedbi CRUD (Create, Read, Update and Delete) operacij (vsi objekti na isti način),
- abstrakcija mrežnega transporta med odjemalcem in strežnikom (podpora za WCF, remoting, web services, enterprise services, ...),
- kontrola za preklop med lokalnim in oddaljenim izvajanjem kode za podatkovni dostop (ta funkcionalnost je podrobneje opisana v razdelku 3.3.6).

Podatkovni portal je razdeljen na več vsebinskih delov, in sicer:

- **Odjemalčev podatkovni portal**
  Ta del podatkovnega portala je implementiran kot statičen razred, ker lahko tako direktno dostopamo do metod, kot so: *Create(), Fetch(), Delete() in Excute()*. Ta del podatkovnega portala prebere kofiguracijsko datoteko in nato delegira strežniškemu podatkovnem portalu ali se izvaja lokalno ali na strežniku.

- **Odjemalčevi proxies**
  Ko odjemalčev podatkovni portal bere konfiguracijsko datoteko za določitev primernega protokola, pa odjemalčevi proxies poskrbijo za detajle vsake tehnologije. Dopušča tudi, da uporabimo svoje proxy razrede.

- **Sporočilni objekti**
  Kadar odjemalčev podatkovni portal pokliče strežniški podatkovni portal, se pošlje veliko informacij z odjemalca na strežnik. Poleg poslovnih objektov se prenesejo še naslednje informacije: odjemalčevi podatki (odjemalčeve lokalne nastavitve), aplikacijske kontekstne podatke (logiranje napak) in uporabnikovo indentiteto, če uporabljamo ročno avtentikacijo.

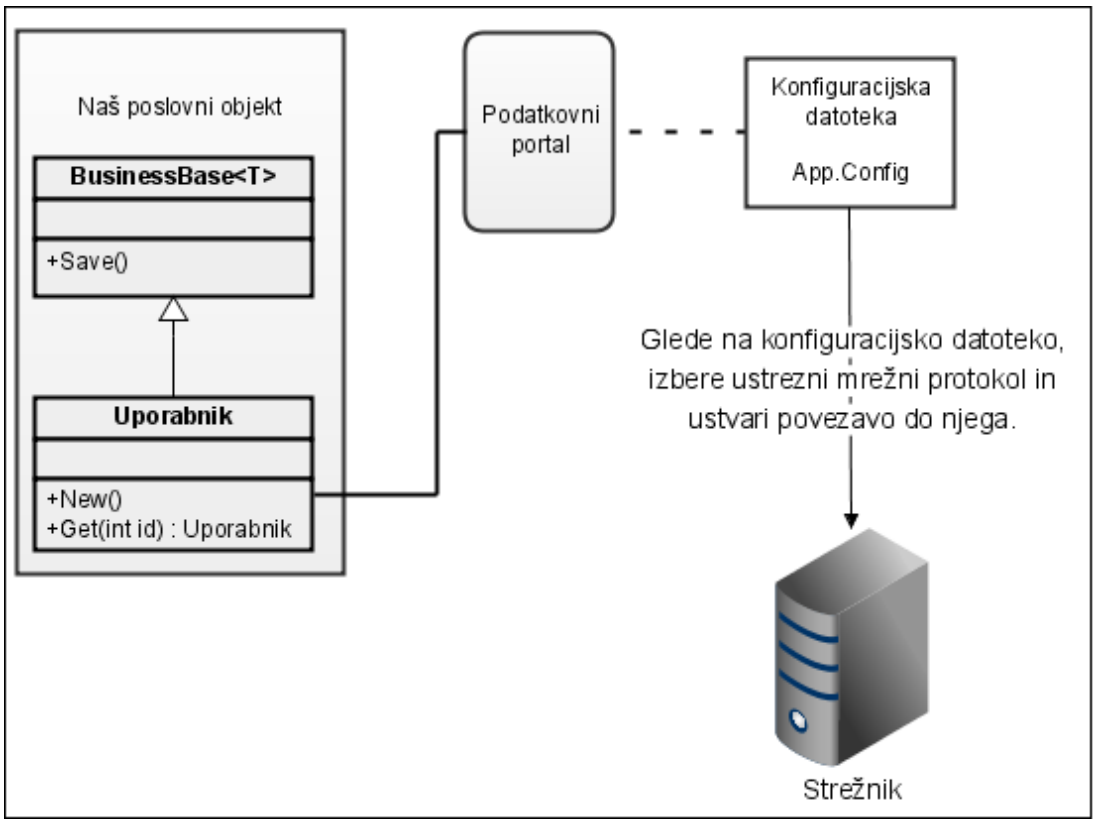
- **Strežniški gostiteljski objekti**
  Ti objekti so odgovorni za sprejemanje dohodnih zahtev preko primernih mrežnih protokolov ter posredovanje teh zahtev strežniškim komponentam. Drugo pa mora teči znotraj primerne gostiteljske tehnologije.

- **Strežniški podatkovni portal**
  V ožjem smislu strežniški podatkovni portal sprejme zahtevo od odjemalca in jo usmeri k pravilemu upravljalcu. Velja še omeniti, da lahko ta del podatkovnega portala teče tudi na odjemalčevem računalniku ali pa na oddaljenem strežniku. V obeh primerih je implementacija enaka. Deluje lahko tudi v transakcijskem načinu.

3.3.6. Preklop med različnimi fizičnimi postavitvami aplikacije

Izbira fizične konfiguracije je lahko zelo težka. Če razvijamo aplikacijo, ki jo želimo prodati večim naročnikom, se je potrebno zavedati, da fizična postavitev ne bo pri vsaki stranki enaka. Zato moramo imeti na voljo mehanizem, ki nam omogoča menjavanje fizičnih postavitev brez spremembe kode.

Na sliki 7 je prikazana shema mehanizma, ki nam omogoča tak preklop. Vsak poslovni razred, ki deduje iz osnovnih CSLA.NET razredov, dostopa preko podatkovnega portala do svojih podatkov. Podatkovni portal prebere konfiguracijsko datoteko in na podlagi vsebine ugotovi, kateri mrežni protokol mora uporabiti. Na ta način lahko dostopa do podatkovnega vira.



Slika 7. Shema mehanizma podatkovnega portala pri večnivojski aplikaciji

Razlika med takšno postavitvijo in alternativno dvonivojsko postavitvijo je ta, da namesto omrežnega protokola v konfiguracijo zapišemo povezavni niz do podatkovne baze. Tako podatkovni portal ve, da med aplikacijo in podatkovnim virom ni vmesnega nivoja in s tem ustvari povezavo do podatkovne baze sam.

3.3.7. Ročna avtentikacija

V primeru, ko nam ne zadostuje vgrajena varnostna politika, lahko naredimo svoje varnostne objekte. Dedovati morajo iz *BusinessPrincipalBase* in *Iidentity*.

4. Ovrednotenje ogrodja CSLA.NET skozi implementacijo praktične aplikacije

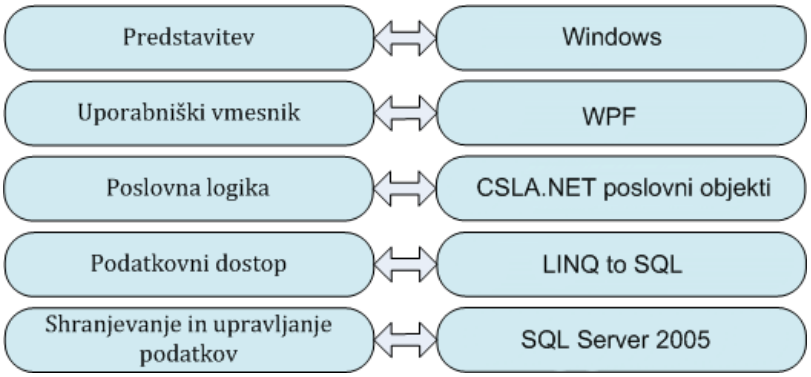
V poglavju 3 smo spoznali osnovne ideje in koncept delovanja ogrodja. V tem poglavju bomo večino teh funkcionalnosti uporabili in implementirali v programsko rešitev. V nadaljnjih razdelkih bomo predstavili tehnično zasnovo vzorčne aplikacije (razdelek 4.1), njen podatkovni nivo (razdelek 4.2) in poslovni nivo (razdelek 4.3). V slednjem si bomo pogledali dejansko implementacijo poslovnih objektov, koncept tovarniških metod, relacije med objekti, poslovna in avtorizacijska pravila ter podatkovni portal. V razdelku 4.4 sledi še pregled predstavitvenega nivoja. Želimo torej razviti vzorčno a preprosto aplikacijo z uporabo ogrodja CSLA.NET, ki bo čim bolj zajela funkcionalnosti ogrodja, omenjene v prejšnjih poglavjih.

4.1. Vsebinska in tehnična zasnova

Vsebinsko je aplikacija preprosta, zadali smo si izdelati aplikacijo, ki ima lastni nadzor uporabnikov (angl. *User management*). Aplikacija mora omogočati:

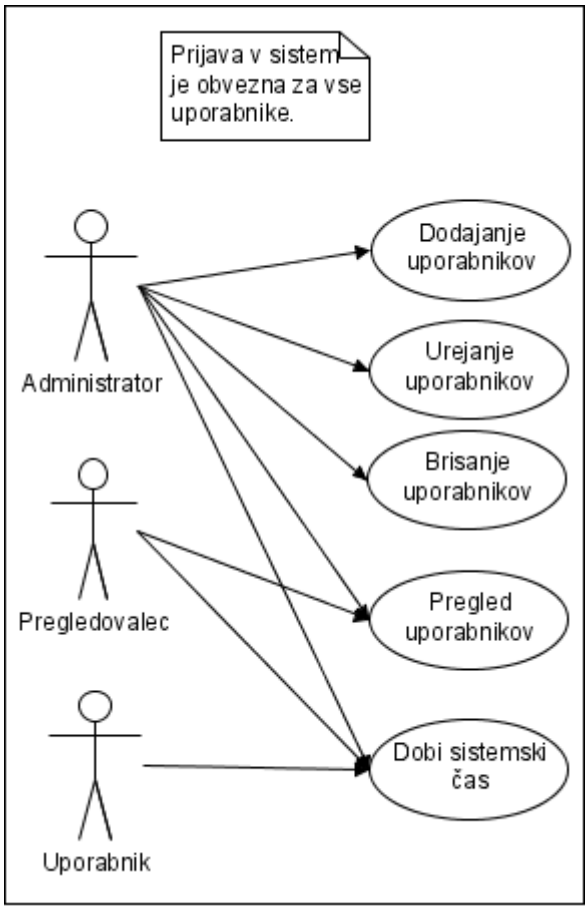
- avtentikacijo uporabnika,
- urejanje uporabnikov in njegovih vlog,
- glede na vloge je potebno kontrolirati njihove dostope v aplikaciji,
- preverbo pravilnosti vnešenih podatkov pri vnosu uporabnikov,
- možnost postavitve aplikacije na različne fizične konfiguracije.

Če tehnologije v naši aplikaciji povežemo z petplastno logično arhitekturo, ki smo jo predstavili v prejšnjih poglavjih, lahko povezave prikažemo, kot je to storjeno na sliki 8.



Slika 8. Tehnologije, uporabljene v vzorčni aplikaciji, in povezava s petplastno logično arhitekturo

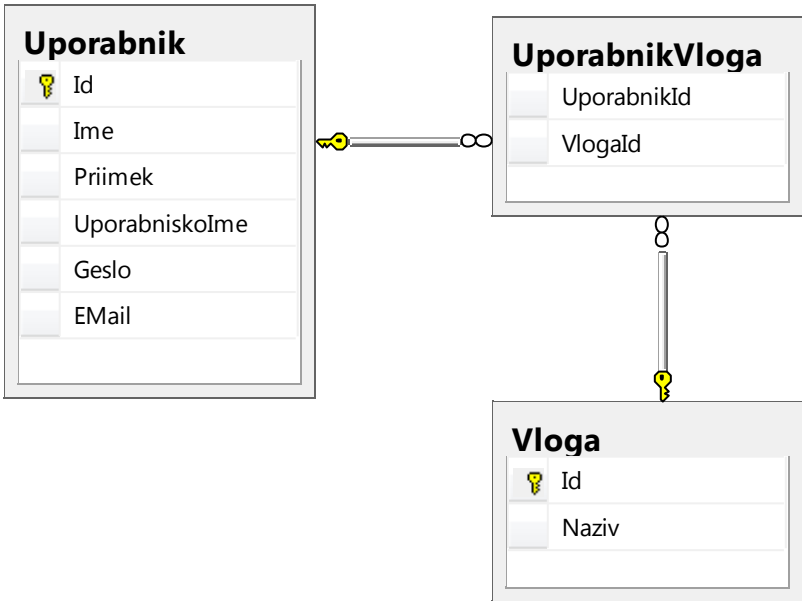
Želene funkcionalnosti aplikacije prikazujemo kar z diagrami primerov uporabe, kot je to prikazano na sliki 9.



Slika 9. Diagram primerov uporabe za primer: vloga Administrator lahko doda, ureja, briše, pregleduje uporabnike in izve sistemski čas, Pregledovalec lahko pregleduje uporabnike, vse vloge pa imajo dostop do sistemskega časa

4.2. Podatkovni nivo

Pri razvoju aplikacije smo uporabili podatkovno bazo SQL Server 2005 (logična plast shranjevanja in upravljanja s podatki). V podatkovni bazi imamo tri tabele, in sicer *Uporabnik*, *UporabnikVloga* in *Vloga*, kot prikazuje slika 10.



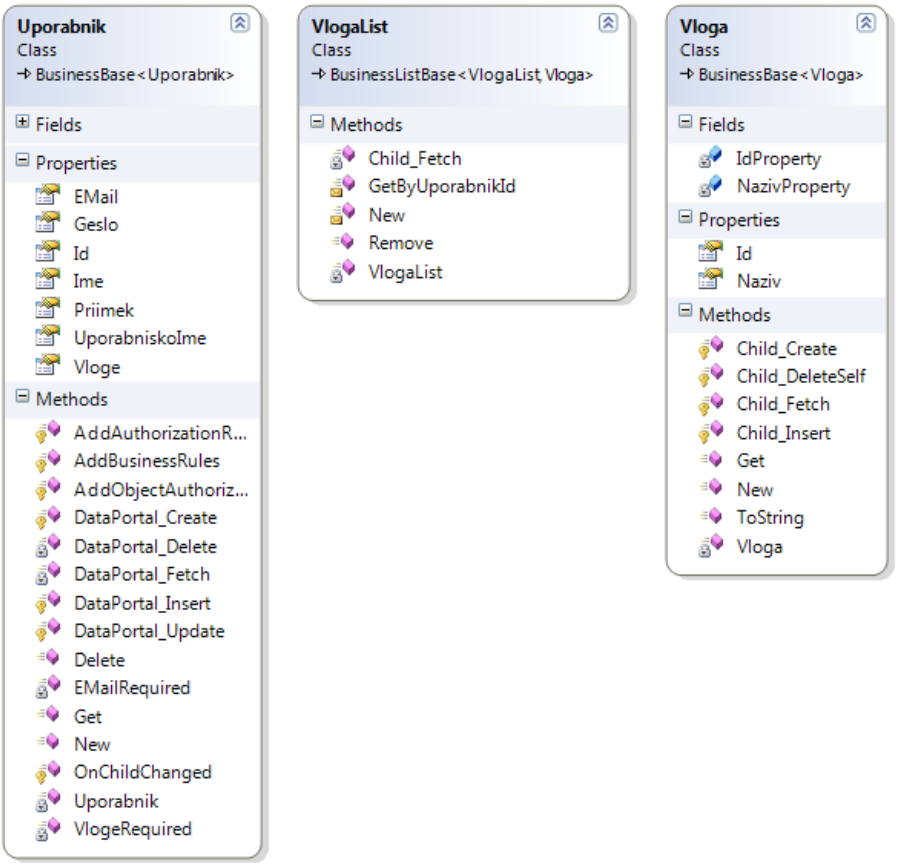
Slika 10. Podatkovni model baze

Za shranjevanje podatkov o uporabnikih uporabljamo torej tabelo Uporabnik, za shranjevanje uporabnikovih vlog pa tabelo UporabnikVloga. Vloge so shranjene v tabeli Vloga.

Za dostop do podatkovne baze smo uporabili LINQ to SQL (logična plast podatkovnega dostopa), ki je izšel skupaj z ogrodjem .NET verzije 3.5 [8] in ponuja preslikavo relacijskih baznih tabel in rezultatov baznih metod v objekte .NET.

4.3. Poslovni nivo

Po izdelavi podatkovnega modela in ureditvi dostopa do podatkovne baze, se lahko posvetimo jedru vzorčne aplikacije. Implementirali bomo poslovne objekte, potrebne za njeno delovanje. Vsi razredi, razen *VlogaSif*, dedujejo od osnovnih razredov ogrodja CSLA.NET, s čimer pridobijo osnovne funkcionalnosti, ki jih ponuja ogrodje. Slika 11 prikazuje razredni diagram teh poslovnih objektov.



Slika 11. Razredni diagram poslovnih objektov

V poslovnih razredih CSLA.NET uporabljamo regije programske kode, da lažje razdelimo funkcionalnost znotraj razreda in s tem tudi povečamo berljivost kode. Vsako izmed teh regij si bomo pogledali na primeru opisa razreda *Uporabnik*. V dodatku A pa je prikazana vsa pomembna koda iz poslovnega nivoja.

Razred uporabnik

Razred *uporabnik* je tipa *Editable root*. Deduje od osnovnega razreda *BusinessBase* ogrodja CSLA.NET. V naši aplikaciji je osnovni razred in vsebuje pet pomembnih regij programske kode:

- ***Properties*:** ta regija vsebuje zasebna (angl. *private*) polja (angl. *field*) in javne lastnosti (angl. *property*), ki jih vidimo na razrednem diagramu (slika 12). Ko želimo neko lastnost prebrati ali spremeniti, CSLA.NET najprej preveri, če ima uporabnik ustrezne pravice. Za vsako od teh lastnosti imamo v regiji *Authorization Rules* možnost nastaviti, kateri uporabniki imajo pravico do branja in kateri spreminjanja.

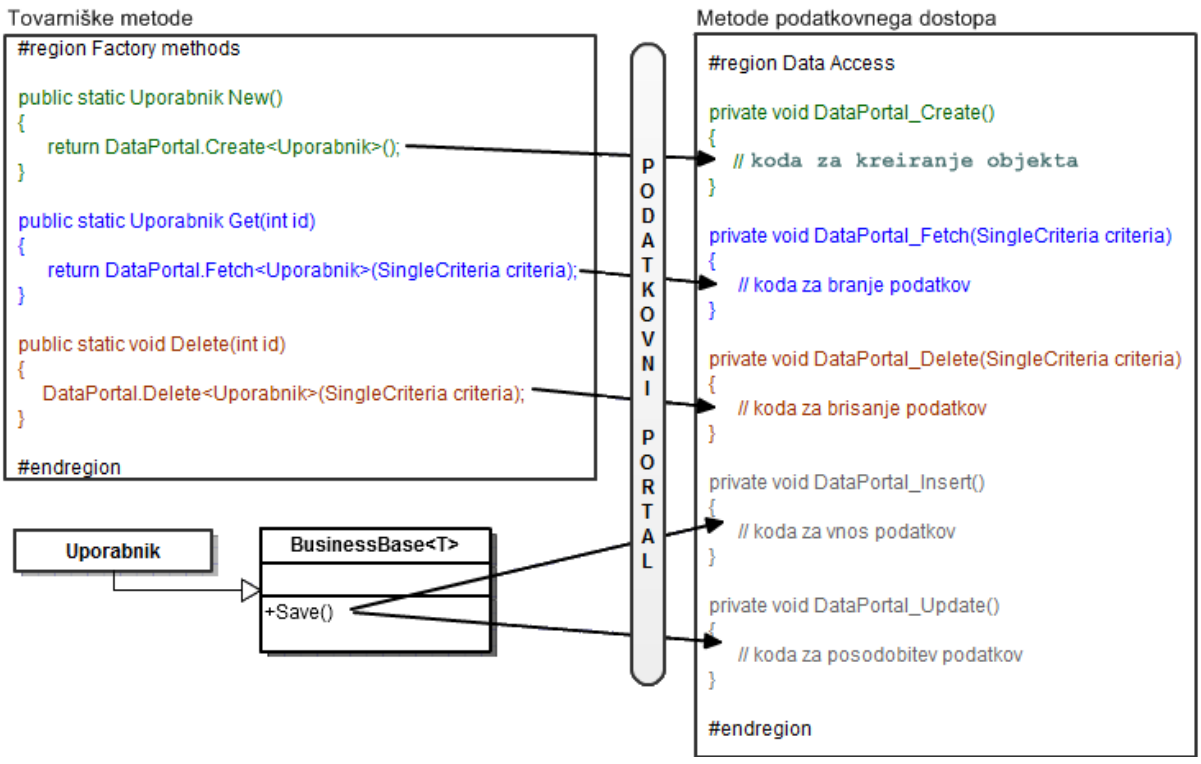
- ***Factory Methods*:** ta regija vsebuje tovarniške metode, ki so bile omenjene v razdelku 2.3.6, in zasebni konstruktor, s katerim prisilimo razvijalce, da uporabljajo tovarniške metode. V tem razredu imamo dve tovarniški metodi, in sicer *New()* in *Get(int Id)*. Ti metodi ponavadi vsebujeta klic na podatkovni portal. Ob tem klicu ogrodje

CSLA.NET preveri, če ima uporabnik pravice za branje objekta. Če uporabnik pravic nima, podatkovni portal sproži varnostno izjemo, da uporabnik nima pravic za to operacijo. Ta pravila dodamo v regiji *Authorization Rules*.

- ***Validation Rules*:** ta regija vsebuje metodo *AddBusinessRules()*, ki jo prepišemo (angl. *override*) iz osnovnega razreda ter v njej dodajamo pravila za preverbe podatkov. Pravila vežemo na posamezne lastnosti objekta.

- ***Authorization Rules*:** ta regija vsebuje dve metodi. V prvi metodi *AddAuthorizationRules()* vežemo pravila na lastnosti, v drugi *AddObjectAuthorizationRules()* pa na nivo objekta. Obstajajo tudi metode, v katerih lahko na podoben način vežemo pravila tudi samo na določen primerek (angl. *instance*) objekta.

- ***Data Access*:** ta regija vsebuje metode, ki se izvajajo na strežniškem podatkovnem portalu (na aplikacijskem strežniku), razen če označimo drugače (nad metodo napišemo atribut *[RunLocal])*. Te metode so zadolžene za komunikacijo s podatkovno bazo. V našem primeru imamo metode: *DataPortal_Fetch, DataPortal_Create, DataPortal_Insert, DataPortal_Update, DataPortal_ Delete.* Na sliki 12 vidimo povezavo med tovarniškimi metodami in metodami v tej regiji. Čeprav so napisane v istem razredu, se klic metod opravi preko podatkovnega portala. Vnos ali podsodobitev podatkov pa se izvede s klicem metode *Save()*, ki je implementirana v osnovnem razredu. Podatkovni portal je tisti, ki na podlagi stanja objekta pokliče ustrezno metodo.



Slika 12. Shema povezave med tovarniškimi metodami in metodami za podatkovni dostop

Razred vlogaList

Razred *vlogaList* je tipa *Editable child list* in vsebuje zbirko objektov tipa *Vloga*. Deduje od osnovnega razreda *BusinessListBase* ogrodja CSLA.NET. Vsebuje dve programski regiji *Factory Methods* in *DataAccess*. Ker je ta razred vsebovan v drugem razredu (v našem primeru v *Uporabniku*), moramo klic podatkovnega portala sprementiti iz *DataPortal.Fetch* v *DataPortal.FetchChild*. S tem povemo, da so ta objekt in ostali objekti v njegovi zbirki podobjekti (angl. *Child*) objekta *Uporabnik*.

Razreda vloga in vlogaSif

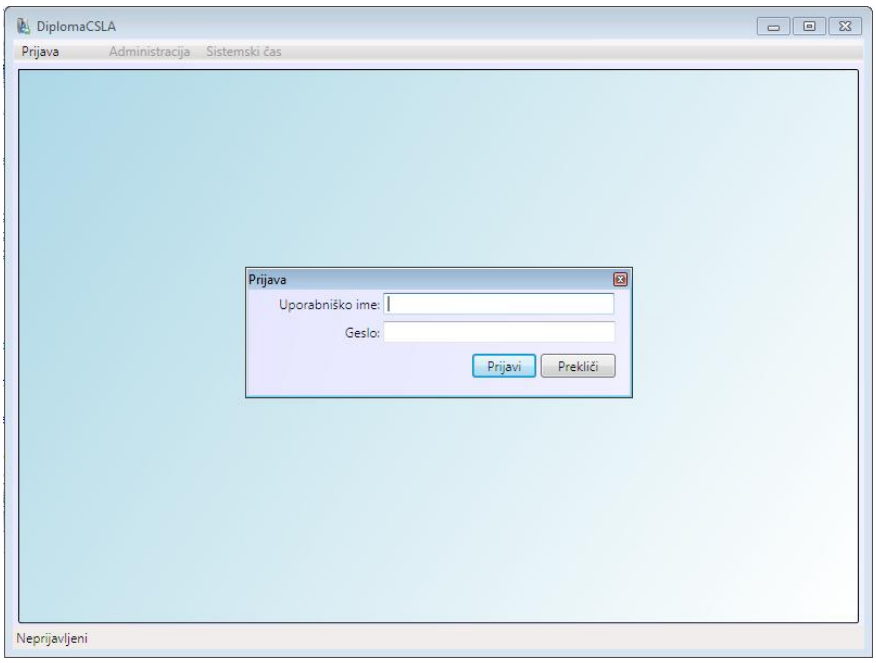
Razred *vloga* je tipa *Editable child*. Deduje od osnovnega razreda *BusinessBase* ogrodja CSLA.NET. Je samostojen objekt, vendar je tesno povezan z razredom *vlogaList*. Ta razred vsebinsko predstavlja šifrant vlog, ki ga pridobimo iz podatkovne baze. V našem primeru smo ga uporabili tudi za shranjevanje podatkov v vmesno tabelo *UporabnikVloga*. V regiji *DataAccess* tako kličemo bazne metode za branje in brisanje iz prej omenjene tabele. Obe metodi sta poklicani s strani *FieldManager*-ja, ki glede na status objekta v zbirki pokliče ustrezno metodo. Za šifrant vlog smo naredili preprost »navaden« razred *VlogaSif*, ki vsebuje statične objekte tipa *vloga*. Tega imenujemo tudi »sistemski šifrant«, ki ga ne potrebujemo pridobiti iz podatkovne baze, ampak njegove zapise ustvarimo kar v kodi. Ti zapisi morajo biti enaki, kot so v tabeli. Takšen pristop pride v poštev, če so vloge znane vnaprej.

S tem smo zaključili pregled poslovnega nivoja. Prikazane »pametne« poslovne objekte je v nadaljevanju potrebno samo še povezati z uporabniškim vmesnikom, kar je tudi tema naslednjega poglavja.

4.4. Uporabniški vmesnik

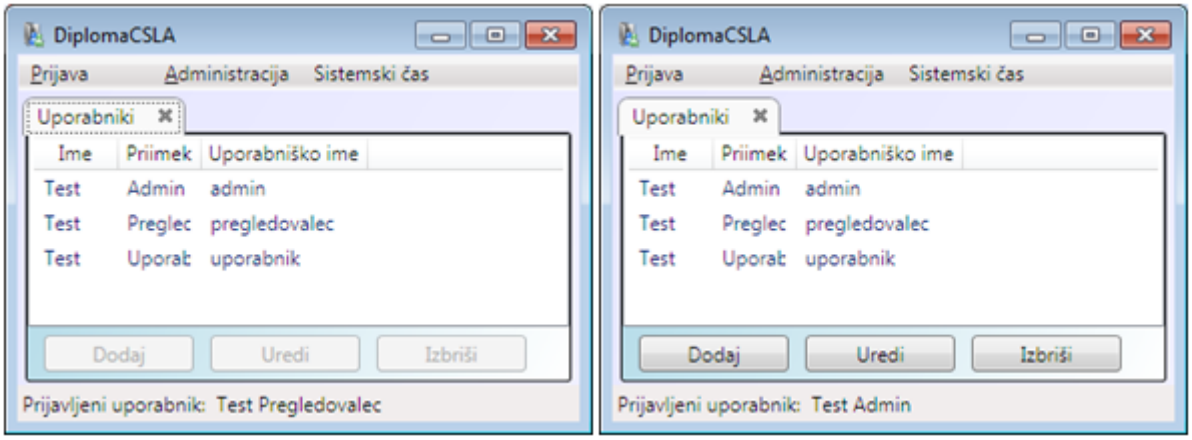
Pri razvoju uporabniškega vmesnika imamo imamo na voljo veliko tehnologij, ki nam omogočajo njegovo izdelavo, bodisi kot spletnega ali pa namiznega. Za poterebe vzorčne aplikacije smo naredili uporabniški vmesnik v tehnologiji WPF. Izgled in uporabnost izdelane vzorčne aplikacije opisujemo v nadaljevanju.

Ob zagonu aplikacije se nam pojavi glavno okno z oknom za prijavo, kot je prikazano na sliki 13. Kot se vidi na sliki, je gumb *Administracija* onemogočen, ker uporabnik še ni prijavljen, omogoči pa se šele, ko se prijavi uporabnik z vlogo *administrator* ali *pregledovalec*. Gumb *Sistemski čas* pa se omogoči, ko je uporabnik prijavljen. Vse to je zapisano v naših poslovnih razredih, tako da tukaj preverjamo samo poslovna pravila.



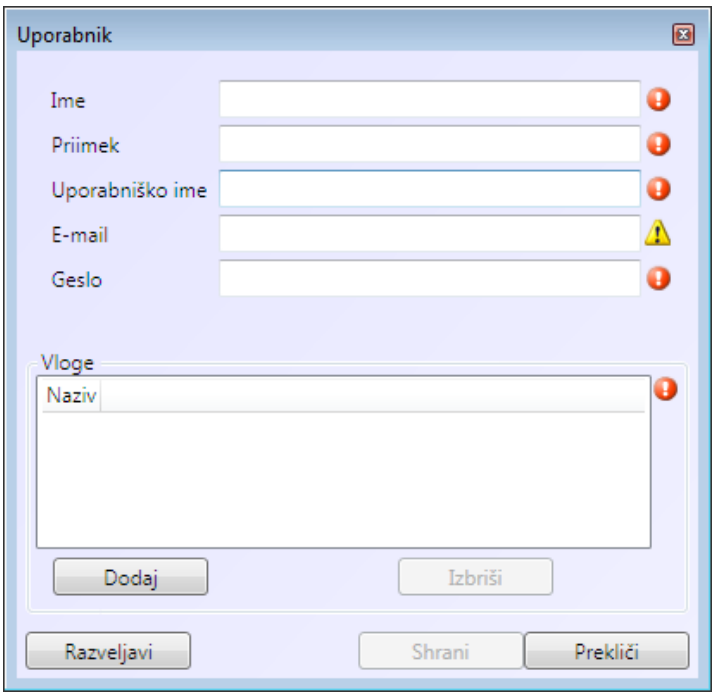
Slika 13. Glavo okno in modalno okno Prijava

Najbolj osnovna vloga uporabnika je *Uporabnik*. V njo sodijo tisti uporabniki, ki lahko opravljajo samo osnovne funkcije, v našem primeru torej vidijo sistemski čas. Medtem ko z vlogo *Pregledovalec* lahko uporabniki vidijo uporabnike sistema, ne morejo pa jih urejati, vloga *Administrator* vključuje polno funkcionalnost aplikacije. Ta razlika je vidna na sliki 14, ki prikazuje različno razpoložljivost gumbov za urejanje uporabnikov glede na vlogo prijavljenega uporabnika.



Slika 14. Uporabniški vmesnik, ko je prijavljen pregledovalec in ko je prijavljen administrator

Ko želi administrator dodati uporabnika, se prikaže vnosno okno, kot ga vidimo na sliki 15. Uporabniku vmesnik takoj prikaže, katera polja so obvezna, vmesnik pa izvaja tudi preverbo, da onemogoči vnos nekonsistentnih podatkov. Preverba podatkov vrača negativno logično vrednost, če pravilo ni izpolnjeno. Gumb *Shrani* se aktivira šele, ko ima objekt veljavno lastnost *IsSavable*.



Slika 15. Vnosno okno za podatke novega uporabnika

4.5. Primerjava vzorčne aplikacije, razvite z ogrodjem CSLA.NET, s klasičnim razvojem aplikacije

S preprostim primerom smo prikazali aplikacijo za nadzor uporabnikov, ki je uporabna tudi kot osnova za poslovne aplikacije. Ni treba, da smo razvijalci, da bi prišli do zaključka, da bi za izdelavo enako funkcionalne aplikacije brez uporabe ogrodja CSLA.NET potrebovali veliko več kode in časa. Nekaj dejstev in opražanj glede opisanega razvoja z uporabo ogrodja CSLA.NET v primerjavi z razvojem klasične aplikacije lahko strnemo in opišemo v tabeli 3 (prednosti ogrodja so prikazane z zeleno barvo, šibkosti pa z rdečo).

Kriterij	**Z uporabo CSLA.NET**	**Brez uporabe CSLA.NET**
Hitrost programiranja	CSLA.NET se lahko pohvali z visokim RAD faktorjem (angl. *Rapid Application Development*), ko se uporabe ogrodja naučimo.	Če razvijamo z lastnim ogrodjem, ki je podobno močno kot CSLA.NET, se hitrosti lahko približamo, drugače pa zelo težko (oziroma se ji ne).
Obseg kode	Obseg napisane kode je manjši, saj nam ni potrebno implementirati vseh tehničnih podrobnosti. Vse nadomentimo s klici metod, ki so v ogrodju.	Obseg kode je večji, razen če ne uporabljamo kakšno drugo ogrodje. Sicer moramo vse potrebne funkcionalnosti implementirati sami.
Berljivost kode	Koda je bolj berljiva. Vzorci programiranja so jasni. Programer se ne mora preveč oddaljiti od njih.	Če nimamo jasno zastavljenih standardov programiranja, lahko naša koda postane tako za druge razvijalce kot tudi za nas neberljiva.
Hitrost delovanja	Šibka stran ogrodja. Na veliko se uporablja zrcaljenje (angl. *Reflection*), s katerim podatkovni portal išče metode za podatkovni dostop, kar zahteva veliko časa.	Delovanje programa je lahko hitrejše kot z uporabo CSLA.NET.
Čas, potreben za učenje	Učenje ogrodja zahteva veliko časa. Poleg tega od nas zahteva, da imamo dobro predznanje o objektnem programiranju. Veliko se naučimo tudi skozi praktično uporabo.	Če znamo programirati, nam učenje novega programskega jezika ne vzame veliko časa, saj ni bistvenih konceptualnih razlik.

Tabela 3. Primerjava uporabe ogrodja z klasičnim načinom razvoja aplikacij

5. Povzetek in sklepne ugotovitve

V poglavjih 3 in 4 smo spoznali funkcionalnosti ogrodja CSLA.NET, katerega lastnosti bomo objektivno povzeli v tem poglavju. Poglavje povzema razne komentarje iz blogov razvijalcev, člankov, forumov, kakor tudi iz lastnih izkušenj ter opažanj pri uporabi ogrodja [4, 5, 7].

5.1. Prednosti uporabe ogrodja CSLA.NET

Zmanjša režijsko programiranje pri razvoju poslovnih aplikacij: Z uporabo ogrodja se znebimo izdelave programske arhitekture, kadar se lotimo razvoja poslovne aplikacije. Ni nam potrebno skrbeti, na koliko fizičnih nivojih bo aplikacija tekla, niti za implementacijo podatkovnega vezenja in za hierarhične podatke v objektih.

Poenoti vzorce programiranja: Problem pri razvoju velikih aplikacij je tudi slog programiranja. Vsak razvijalec v razvojni ekipi ima lahko drugačen slog. Zato imamo znotraj podjetja ali ekipe ponavadi prisotne tudi interne standarde programiranja. Skozi neko obdobje uporabe ogrodja CSLA.NET, kot glavnega ogrodja za razvoj aplikacij se je izkazalo, da se razvijalci ne morejo preveč odaljiti od programskih vzorcev, ki jih nudi ogrodje.

Zagotavlja prožnosti pri upravljanju z virovi podatkov in omrežnimi protokoli: Že samo ime CSLA (angl. *Component-based Scalable Logical Architecture*) omenja skalabilnost oz. nadgradljivost. Dejstvo, da sta lokacija vira podatkov in omrežni protokol deklarativni nastavitvi v konfiguracijskih datotekah *app/web.config*, ponuja veliko manevrskega prostora, kadar želimo premakniti aplikacijo z dveh nivojev na tri.

Nudi veliko razredov za pomoč: Pri razvoju aplikacij si razvijalci naredijo razrede, ki jim olajšajo delo (angl. *Helper classes*). Tudi ogrodje CSLA.NET ima na voljo nekaj razredov, ki nam olajšajo delo z ogrodjem.

Podatkovni portal: Ključ za uporabo ogrodja je razumevanje delovanja podatkovnega portala. Ker ta del ogrodja skrbi za vse dostope do podatkovnega vira, nam pri shranjevanju podatkov ni potrebno skrbeti, ali gre za nov zapis ali za posodabljanje obstoječega. Pri klicu metode *Save()* zna podatkovni portal razbrati iz objekta, ali gre za nov ali za obstoječ zapis in tako izvrši potrebno operacijo.

5.2. Slabosti uporabe CSLA.NET ogrodja

Potreben čas za učenje: Uporaba ogrodja ni intuitivna. Ima precejšno krivuljo učenja in zahteva veliko vloženega časa. Uporabnik potrebuje veliko prakse, da se ogrodja navadi, kar se še posebej pozna pri uporabi podatkovnega portala, če želimo izkoristiti njegove funkcionalnosti. Uporaba ogrodja CSLA.NET se priporoča za velike projekte, ki se izvajajo na porazdeljenih sistemih in bodo v uporabi veliko let, za majhne pa ne, ker bi učenje zahtevalo skoraj toliko časa kot sam razvoj aplikacije.

Zahteva dobra OOP znanja: Ogrodje omogoča široko uporabo vzorca tovarniških metod (angl. *factory design pattern*), zrcaljenja (angl. *reflection*), serializacije, generikov, statičnih metod in abstraktnih razredov. Vse to niso začetniške teme in če želimo uporabljati ogrodje, je dobro, da imamo vsaj enega razvijalca, ki ima s tem izkušnje in lahko piše učinkovito objektno usmerjeno kodo v okolju .NET. Sicer tvegamo, da dobimo napake pri prevajanju, ki jih je težko razumeti in rešiti.

Zahteva celovito poznavanje ogrodja: CSLA.NET je ogrodje in ne metodologija. Potrebno se ga je naučiti in ga uporabljati tako, kot je mišljeno.

Uporaba zrcaljenja namesto vmesnikov: Še ena omembe vredna lastnost ogrodja je uporaba zrcaljenja. Podatkovni portal zahteva, da imajo poslovni objekti implementirane metode za podatkovni dostop, ki so klicane med izvajanjem. Če želimo, da se metoda implementira v razredu, ponavadi to naredimo tako, da objekt deduje iz abstraktnega razreda, ki vsebuje virtualno metodo, ali pa uporabimo vmesnike. Če v tem primeru ne implementiramo zahtevane metode, dobimo napako pri prevajanju. V nasprotju s tem pristopom koda v podatkovnem portalu uporablja zrcaljenje za identifikacijo posebnih metod za shranjevanje in pridobivanje podatkov. V primeru, da metode v objektu nimamo implementirane, pa nam napako javi šele, ko podatkovni portal pokliče zahtevano metodo.

Poslovni razredi (ponavadi) vsebujejo kodo za podatkovni dostop: Tukaj gre predvsem za kodo, ki jo implementiramo za strežniški del podatkovnega portala. Ta koda vsebuje klic neke storitve ali naslavlja podatkovno bazo in je potrebno paziti, da ne napišemo kode, ki se tiče uporabniškega vmesnika.

5.3. Sklepne misli

Če si pogledamo prej naštete prednosti in slabosti, lahko vidimo, da velikih ovir pri uporabi ogrodja ni. Seveda se je potrebno vsake nove »tehnologije« naučiti, da jo znamo dobro uporabljati. V primeru ogrodja CSLA.NET imamo na voljo knjigo avtorja, ki je zelo obsežna, in ponuja dobro razlago konceptov, idej, funkcionalnosti ter opis primera programa, ki zavzema vse funkcionalnosti ogrodja. Slednje nam pride prav, ko naletimo na napako, pri kateri ne vemo, zakaj se je pojavila. Omenjena knjiga je v tem diplomskem delu služila kot vodilo za pisanje, kajti poleg elektronskih virov (forum na avtorjevi strani, video gradivo,

nekaj člankov) predstavlja skorajda edini uradni vir informacij o ogrodju CSLA.NET. Ker je to odprtokodno ogrodje, imamo na voljo tudi izvorno kodo in primer implementacije programa s tem ogrodjem v jezikih C# in VB. Da je ogrodje široko uporabno, pove že podatek, da je to ogrodje rezultat 14-letnega raziskovanja in razvoja [9]. Ogrodje CSLA.NET podpira veliko tehnologij, ki so vključene v ogrodje .NET, po novem pa tudi obstaja različica za Silverlight. Ogrodje pa se razvija naprej tudi sedaj. Pri praktičnem delu diplome smo uporabili ogrodje verzije 3.6.1, ki je izšlo februarja 2009, sedaj pa je na voljo že pred-izdaja verzije 4.0.0.

Ogrodje CSLA.NET ponuja dobro rešitev pri razvoju velikih poslovnih aplikacij, ni pa primerno za vsako aplikacijo. Dejstvo pa je, da je to dobro premišljeno in vsebinsko zelo obširno ogrodje, ki dobro služi svojemu namenu.

DODATEK A: Programska koda poslovnega nivoja vzorčne aplikacije

- **Razred *Uporabnik*, regija *Properties***

```
private static PropertyInfo<string> ImeProperty =
                        RegisterProperty(new PropertyInfo<string>("Ime"));
public string Ime
{
      get { return GetProperty(ImeProperty); }
      set { SetProperty(ImeProperty, value); }
}

// Vloge
private static PropertyInfo<VlogaList> VlogeProperty =
      RegisterProperty(new PropertyInfo<VlogaList>("Vloge"));
public VlogaList Vloge
{
    get
    {
        if (!FieldManager.FieldExists(VlogeProperty))
            if (this.IsNew)
               LoadProperty(VlogeProperty, VlogaList.New());
            else
               LoadProperty(VlogeProperty, VlogaList.GetByUporabnikId(Id));

        return GetProperty(VlogeProperty);
     }
}
```

- **Razred *Uporabnik*, regija *Validation Rules***

```
// AddAuthorizationRules
protected override void AddAuthorizationRules()
{
    // add AuthorizationRules here
    AuthorizationRules.AllowWrite(ImeProperty, "Administrator");
    AuthorizationRules.AllowWrite(PriimekProperty, "Administrator");
    AuthorizationRules.AllowWrite(UporabniskoIme, "Administrator");
    AuthorizationRules.AllowWrite(GesloProperty, "Administrator");
}
// AddObjectAuthorizationRules
protected static void AddObjectAuthorizationRules()
{
    // add object-level authorization rules here
```

```
    AuthorizationRules.AllowCreate(typeof(Uporabnik), "Administrator");
    AuthorizationRules.AllowEdit(typeof(Uporabnik), "Administrator");
    AuthorizationRules.AllowDelete(typeof(Uporabnik), "Administrator");
    AuthorizationRules.AllowGet(typeof(Uporabnik), "Administrator");
}
```

- **Razgled *Uporabnik*, regija *Factory Methods***

```
public static Uporabnik Get(int id)

{
    return DataPortal.Fetch<Uporabnik>(
    new SingleCriteria<Uporabnik, int>(id));
}
```

- **Razred *Uporabnik*, regija *DataAccess***

```
private void DataPortal_Fetch(SingleCriteria<Uporabnik, int> criteria)

{
     using (var cm = Global.ContextManager)
     {
         var item = cm.DataContext.GetUporabnikById(criteria.Value)
                      .SingleOrDefault();
         if (item == null)
             return;

           using (BypassPropertyChecks)
           {
                Id = item.Id;
                Ime = item.Ime;
                Priimek = item.Priimek;
                UporabniskoIme = item.UporabniskoIme;
                Geslo = item.Geslo;
                EMail = item.EMail;
            }
        }

      ValidationRules.CheckRules();
  }
```

- **Razred *VlogaList*, regija *Factory Methods* in *DataAccess*, kot Child objekt**

```
internal static VlogaList GetByUporabnikId(int id)
{
     return DataPortal.FetchChild<VlogaList>(id);
}

private void Child_Fetch(int id)
{
  using (var cm = Global.ContextManager)
  {
      var items = cm.DataContext.GetVlogeByUporabnikId(id);

      this.RaiseListChangedEvents = false;
      this.AddRange(items.Select(v => Vloga.Get(v.VlogaId)));
      this.RaiseListChangedEvents = true;
}}
```

- **Razred *Vloga*, regija *DataAccess***

```
protected void Child_Insert(Uporabnik parent)

{
     using (var cm = Global.ContextManager)
          cm.DataContext.UporabnikVloga_Insert(parent.Id, Id);
}

protected void Child_DeleteSelf(Uporabnik parent)
{
     using (var cm = Global.ContextManager)
         cm.DataContext.UporabnikVloga_Delete(parent.Id, Id);
}
```

- **Razred VlogaSif**

```
public class VlogaSif

{
    private static List<Vloga> _list;

    public static Vloga Admin = Vloga.New(1, "Administrator");
    public static Vloga Pregledovalec = Vloga.New(2, "Pregledovalec");
    public static Vloga Uporabnik = Vloga.New(3, "Uporabnik");
    .....
}
```

- **Razred *GetServerTimeCommand***

```
// Execute
public static DateTime Execute()
{
    GetServerTimeCommand newCmd = DataPortal.Execute(
            new GetServerTimeCommand());
    return newCmd._currentDateTime;
}

#region DataAccess
protected override void DataPortal_Execute()
{
    _currentDateTime = DateTime.Now;
}
#endregion
```

- **Razred *CustomPrincipal***

```
// Login
public static bool Login(string username, string password)
{
    return SetPrincipal(CustomIdentity.GetIdentity(username, password));
}

// SetPrincipal
private static bool SetPrincipal(CustomIdentity identity)
{
    if (identity.IsAuthenticated)
    {
        CustomPrincipal principal = new CustomPrincipal(identity);
        Csla.ApplicationContext.User = principal;
    }
    else
        Logout();

    return identity.IsAuthenticated;
}
```

Slike

Tabele

Literatura

[1] R. Lhotka, Expert C# 2008 Business Objects, Apress, 2008

[2] R. Lhotka, Expert C# 2005 Business Objects, Apress, 2005

[3] M. MacDonald, Pro WPF in C# 2008, Second Edition, Apress, 2008

[4] (2007) Programming Within the CSLA>The Pros and Cons of CSLA. Dostopno na: http://www.informit.com/articles/article.aspx?p=770361&seqNum=4

[5] (2008) Working with Rocky Lhotka's CSLA Framework. Dostopno na: http://blog.componentoriented.com/2008/07/working_with_rocky_lhotkas_csla_framework/

[6] (2008) Business object (computer science). Dostopno na: http://en.wikipedia.org/wiki/Business_object_%28computer_science%29

[7] (2006) Using CSLA as an application framework. Dostopno na: http://claytonj.wordpress.com/2006/10/17/using-csla-as-an-application-framework/

[8] (2008) LINQ to SQL. Dostopno na: http://msdn.microsoft.com/en-us/library/bb386976.aspx

[9] (2009) What is CSLA.NET?. Dostopno na: http://www.lhotka.net/cslanet/Info.aspx