UNIVERZA V LJUBLJANI
EKONOMSKA FAKULTETA

MAGISTRSKO DELO

RAZVOJ SPLETNEGA OGRODJA ZA IZVEDBO UPORABNIŠKEGA VMESNIKA DO PODATKOVNIH BAZ

Ljubljana, september 2009

MIHA NOVAK

IZJAVA

Študent Miha Novak izjavljam, da sem avtor tega magistrskega dela, ki sem ga napisal pod mentorstvom dr. Tomaža Turka in skladno s 1. odstavkom 21. člena Zakona o avtorskih in sorodnih pravicah dovolim objavo na fakultetnih spletnih straneh.

V Ljubljani, 28. september 2009

Podpis:____________________

Kazalo

1 Ogrodje „Uporabniški vmesnik do podatkovne baze“..3
1.1 Opis in značilnosti..3
1.2 Namen in uporaba...3
1.3 Podobna ogrodja in vmesniki do podatkovnih baz..4
1.3.1 Ogrodja..4
1.3.1.1 Symfony...4
1.3.1.2 Propel...5
1.3.2 Uporabniški vmesniki do podatkovnih baz...6
1.3.2.1 PhpMyAdmin..6
2 Razvoj Uporabniškega vmesnika do podatkovne baze...6
2.1 Uvod..6
2.2 Analiza uporabniških zahtev...7
2.2.1 Ugotavljanje uporabniških zahtev...8
2.2.1.1 Licitacija uporabniških zahtev..8
2.2.1.2 Usklajevanje in preverjanje uporabniških zahtev...9
2.2.1.3 Model poslovnih uporabniških zahtev...10
2.2.1.3.1 Model poslovnih primerov uporabe..10
2.2.1.3.2 Model poslovnih razredov..11
2.2.1.4 Dokument uporabniških zahtev..13
2.2.2 Specifikacija uporabniških zahtev..14
2.2.3 Razlaga primerov uporabe...15
2.3 Sistemska analiza in zasnova...15
2.3.1 Sistemska analiza...15
2.3.2 Sistemska zasnova..16
2.4 Pristop orodja in materiala v sistemski analizi in zasnovi..16
2.4.1 Uvod...16
2.4.1.1 Pogled podpore...18
2.4.2 Vodilna metafora in zasnovna metafora 18
2.4.3 Interakcija orodja in materiala...18
2.4.4 Material..22
2.4.5 Orodje..25
2.4.6 Delovno okolje...31
2.4.7 Zbiralnik..32
2.5 Testiranje...34
3 Glavni uporabljeni koncepti..35
3.1 Objektno usmerjen pristop...35
3.1.1 Razvoj objektno usmerjenega pristopa 36
3.1.2 Objektni meta model pristopa Orodje in material...37
3.1.3 Razred..38
3.1.4 Življenjski cikel objekta...41
3.1.5 Ograjevanje (ang.: encapsulation)...41

3.1.6 Tip......42
3.2 Arhitektura spletnega, objektno usmerjenega in porazdeljenega informacijskega sistema. 43
3.2.1 Porazdeljeni objekti......44
3.2.2 Oddaljeni in lokalni vmesniki......44
3.2.3 Strežniški podsistem......46
3.2.4 Odjemalčev podsistem......46
3.2.5 Teorija tridelne arhitekture......47
3.2.6 Model spletnih aplikacij......48
3.3 Ogrodje in prilagajanje spletnega, objektno usmerjenega in porazdeljenega informacijskega sistema......53
3.3.1 Ogrodje......53
3.3.2 Prilagajanje trajnega obrazca odjemalca......53
3.4 Objektno - relacijsko preslikavanje......54
3.5 Problemi trajnih podatkov v spletnem, objektno usmerjenem in porazdeljenem informacijskem sistemu......55
3.5.1 Stanje seje......55
3.6 Uporabljeni vzorci zasnove pri Uporabniškem vmesniku do podatkovnih baz......57
3.6.1 Vzorci......57
3.6.2 Vzorec Singleton......57
3.6.3 Vzorec Metoda tovarna (ang.: factory method)......58
3.6.4 Vzorec Kompozicija......59
3.6.5 Vzorec Aktivni zapis (ang.: Active Record pattern)......60
3.6.6 Vzorec Odjemalčev zbiralnik tabelnih struktur......60
3.6.7 Vzorec Odjemalčevo preverjanje podatkov......61
4 Uporaba......62
4.1 Uporaba za uporabnike......62
4.2 Uporaba za razvijalce......65
Sklep......68
Literatura in Viri......71
Priloga

Kazalo slik

Slika 1: Model poslovnih primerov uporabe UVPB......11
Slika 2: Model poslovnih razredov, 1. del......12
Slika 3: Model poslovnih razredov, 2. del......13
Slika 4: Transformacija modela domene aplikacije......17
Slika 5: Interakcija med GUV materialom, GUV kontrolnim materialom in orodjem......20

Slika 6: Interakcija med orodjem toolKeypressed_CS_MakeEntry ter GUV kontrolnim materialom trajnega obrazca makeEntry_CS[1..n]....21
Slika 7: Interakcija med orodjem ToolKeypressed_CS_MakeEntry ter GUV kontrolnim materialom MakeEntry_CS....22
Slika 8: Diagram aktivnosti nekaterih operacij na odjemalčevi strani po naložitvi spletne strani MakeEntry....25
Slika 9: Delovno okolje odjemalčevih orodij....27
Slika 10: Orodje ToolKeypressed_CS....28
Slika 11: Struktura trajnega hranjenja trajnih obrazcev kot del mehanizma trajnih obrazcev....33
Slika 12: Omejitve projekta....35
Slika 13: Transformacija domenskega modela v zasnovni model odjemalčevega podsistema....38
Slika 14: Verižni klici storitev med razredi....40
Slika 15: Model splošne arhitekture spletne aplikacije....49
Slika 16: Model arhitekture spletne aplikacije UVPB....50
Slika 17: Abstraktni strežniški razred Class_wp_SS....51
Slika 18: Abstraktni razred odjemalca Class_wp_CS....52
Slika 19: Prilagoditev trajnega obrazca odjemalca....54
Slika 20: Diagram razredov z razredom Forms_CS....59
Slika 21: Meni osnovnih operacij UBDB....62
Slika 22: Primer osnovne operacije kreiraj zapis v tabeli partner....63
Slika 23: Primer branja, dopolnjevanja ter brisanja zapisa v tabeli partner....63
Slika 24: Primer polja PartnerID kateremu se samodejno pripiše lastnost tujega ključa ter se samodejno določi povezava s tabelo partner s katero je tuji ključ povezan....64
Slika 25: Prikaz obrazca izbora (ang.: Select Record form), ki prikazuje seznam zapisov tabele partner....65
Slika 26: Primer vnosnega obrazca faktura z vstavljeno vrednostjo primarnega ključa iz obrazca izbora višjega nivoja....65

Uvod

Problematika

Informacijski sistem je sistem, v katerem se podatki oziroma informacije ustvarjajo, berejo, posodabljajo in brišejo. To so osnovne operacije vsakega informacijskega sistema. Magistrsko delo bo opisovalo razvoj ogrodja za izvedbo uporabniškega vmesnika do podatkovnih baz za izvrševanje zgoraj naštetih operacij v izbrani podatkovni bazi.

Programska rešitev se bo lahko uporabljala kot samostojna aplikacija ter kot jedro oziroma ogrodje (ang.: framework) specifičnih in bolj zapletenih informacijskih sistemov. Objektno usmerjene značilnosti ogrodja, kot je na primer dedovanje, bodo omogočale prilagoditve in tako povečevale njegovo razširljivost. Tako kot druga ogrodja bo moralo tudi to omogočati konkretne načine prilagoditev.

Avtor magistrske naloge želi, da bo razvita programska rešitev služila kot ogrodje za prilagojen informacijski sistem, potreben za podporo poslovnim procesom podjetja Klik d. o. o. Programska rešitev bo postopoma nadomestila obstoječo množico aplikacij, ki so napisane v proceduralnem jeziku in delujejo v okolju DOS. Povečana dodana vrednost bo omogočila prehod na nov poslovni model, ki naj bi izboljšal kakovost računovodskih storitev, ki jih podjetje ponuja. Funkcionalnost novega modela bo podobna staremu sistemu, ki že deluje v realnem poslovnem okolju. Dodana vrednost novega sistema bo večja dostopnost, učinkovitejše vzdrževanje, večja razširljivost in fleksibilnost. Spletno okolje sistema bo uporabniku omogočalo prosto izbiro lokacije dostopa, centralno hranjenje podatkov bo zmanjšalo stroške vzdrževanja, spletni vzorec strežnik−odjemalec (ang.: server-client) pa bo povečal fleksibilnost, saj bodo nove različice aplikacije lahko takoj prenesene in uporabljene. Večja razširljivost bo dosežena z objektno usmerjenimi značilnostmi sistema, obenem bo z mehanizmom dedovanja lahko implementirati novo funkcionalnost z nič ali minimalnimi spremembami v jedru aplikacije.

Namen in cilj magistrskega dela

Namen magistrskega dela je razviti delujočo programsko rešitev ogrodje za izvedbo uporabniškega vmesnika do podatkovnih baz in ga uporabiti kot jedro oziroma osnovo za prilagojene programske rešitve novega informacijskega sistema v podjetju Klik d. o. o.

Omeniti velja podobna ogrodja Zend, CakePHP, Symfony in Propel, ki so podrobneje opisana v poglavju 1.3.1, in uspešno služijo kot osnova za razvoj mnogih aplikacij v realnem okolju. Vsa zgoraj našteta ogrodja imajo poudarek na strežniški strani, kar lahko negativno vpliva na odzivnost aplikacij. Nobeno izmed ogrodij tudi ne zajema vseh konceptov, ki jih magistrsko delo opisuje. Propel je specifično ogrodje visoke abstrakcije, katerega uporabnost je bolj omejena kot pri ogrodju, ki ga magistrsko delo omenja.

Ogrodja Zend, CakePHP in Symfony ne omogočajo dovolj enostavnega samodejnega kreiranja HTML obrazcev, so manj optimizirana v smislu pretoka podatkov med strežnikom in odjemalcev in preveč ohlapno ločujejo podsistem strežnika in odjemalca.

Cilji magistrskega dela so:

- Implementirati določen informacijski sistem zgolj z uporabo skriptnih jezikov, ki se danes uporabljajo.
- Razvijalcem spletnih rešitev povečati možnost izbire ogrodja za hitrejši razvoj kakovostnejšega sistema.
- Končnemu uporabniku ogrodja ponuditi brezplačno orodje za izvajanje osnovnih operacij: ustvari, beri, dopolni in briši zapis.
- Deklarirati programsko rešitev kot odprto kodno ter tako prispevati k razvoju razvijalcev odprto kodnih rešitev.

Metode dela

V poglavju 1 so predstavljene glavne lastnosti in uporaba ogrodja. Znanstvena metoda za doseganje ciljev magistrskega dela je razvoj spletnega, objektno usmerjenega in porazdeljenega informacijskega sistema, ki bo generičen do te mere, da bo hkrati tudi ogrodje. Metoda razvoja ogrodja se opira na model slapa (ang.: waterfall model) z zaporednimi koraki analize uporabniških zahtev (poglavje 2.2), zasnove sistema (poglavje 2.3) in testiranje (poglavje 2.5). Poseben poudarek je na pristopu orodja in materiala (poglavje 2.4) in na opisu glavnih konceptov (poglavje 3). Uporabljena sta dva skriptna jezika: PHP na strani strežnika ter Javascript na strani odjemalca. Infrastruktura, na podlagi katere bo sistem deloval, bo Httpd kot spletni strežnik, Mysql kot podatkovni strežnik in Firefox kot delovno okolje uporabnika.

Podrobnejša analiza in razčlenitev uporabniških zahtev, ki sta predstavljeni v poglavju 2.2, bosta vodili v zasnovo ustreznih razredov, ti bodo tvorili ustrezno arhitekturo, ki bo dovolj generična, da bo z uporabo dodatnih vzorcev in implementacijo ustreznih mehanizmov tvorila ogrodje primerno za nadaljnje prilagoditve specifičnim uporabniškim zahtevam. Z uporabniškimi zahtevami je mišljena analiza potreb končnega uporabnika za dostop do podatkovnih baz in ne analiza potreb razvijalcev, ki bodo ogrodje uporabljali. Osnovna funkcionalnost ogrodja bo realizacija identificiranih ključnih primerov uporabe, ustrezna struktura arhitekture ogrodja pa bo razvijalcem omogočala realizacijo izvedenih oziroma specifičnih primerov uporabe, ki so lastni posameznim informacijskim sistemom.

Poglavitni uporabljeni koncepti pri razvoju spletnega, objektno usmerjenega in porazdeljenega informacijskega sistema - ogrodja - so opisani v poglavju 3. Sledenje objektno usmerjeni paradigmi pri razvoju ogrodja praktično nima alternative, njegove značilnosti pa so podrobneje opisane v poglavju 3.1. Je pa res, da objektno usmerjena zasnova programske rešitve ustvari neskladje do relacijsko usmerjene podatkovne baze. Ta problem rešuje koncept objektno relacijskega preslikavanja, opisan v poglavju 3.4.

Porazdeljeni sistemi, ki imajo objekte razdeljene na stran strežnika in stran odjemalca, za infrastrukturo pa uporabljajo splet, vsebujejo množico značilnosti in vzorcev, ki se razlikujejo od klasičnih neporazdeljenih sistemov, zato je podroben opis ključen in je opisan v poglavju 3.2. Pri metodi dela, to je razvoju sistema, je ugotovljeno, da rešitvi problema trajnosti in prilagajanja obrazcev za vnos podatkov, ki sta opisani v poglavju 3.3 in 3.5, predstavljata eno izmed osrednjih lastnosti ogrodja UVPB. Pomembnejši vzorci zasnove sistema, ki so ključni za implementacijo določenih rešitev, kot so vzorec Metoda tovarna, vzorec Kompozicija in drugi, so opisani v poglavju 3.6.

1 Ogrodje „Uporabniški vmesnik do podatkovne baze“

1.1 Opis in značilnosti

Da bi zagotovili dovolj visoko raven kakovosti, mora ogrodje „Uporabniški vmesnik do podatkovne baze“ (v nadaljevanju UVPB) vsebovati naslednje poglavitne značilnosti:

- Teoretično naravnane značilnosti:
  - objektna usmerjenost,
  - spletna naravnanost,
  - porazdeljena arhitektura sistema.
- Praktično naravnane značilnosti:
  - samodejno kreiranje HTML obrazcev,
  - velika odzivnost HTML obrazcev,
  - navigacija obrazcev med povezanimi tabelami,
  - izdatna uporaba tipkovnice.

Posledice zadostitve teoretičnim kriterijem objektne usmerjenosti in porazdeljene arhitekture sistema ter praktičnim kriterijem velike odzivnosti HTML obrazcev in izdatne uporabe tipkovnice so razvoj in realizacija vzorcev na odjemalčevi strani. Uporaba vzorcev (poglavje 3.6), kot so Metoda tovarna, vzorec Kompozicija in Odjemalčev zbiralnik tabelnih struktur, na primeru UVPB, so zahtevali mnogo porabljenih virov merjenih v enoti človek-ura (ang.: man-hour) v fazah analize, zasnove, implementacije in testiranja. Zaradi pomembnosti teh vzorcev bi lahko klasificirali ogrodje UVPB kot odjemalčevo ogrodje za razliko od drugih podobnih ogrodjih (poglavje 1.3.1), ki večino vzorcev implementirajo na strani strežnika.

1.2 Namen in uporaba

Ogrodje UVPB bo primarno namenjeno razvijalcem, ki želijo izdelati množico prilagojenih vnosnih obrazcev. Končni uporabnik bo lahko pričel z vnosom podatkov takoj, ko bo izdelana podatkovna baza ter vzpostavljena povezava z njo. V realnem okolju je skoraj vedno potrebno vnosne obrazce prilagoditi, na primer dodati določene kontrole podatkov, dodati omejitve vrednosti, dodati vmesne izračune itd. Veljalo bi tudi omeniti, da UVPB ne

nudi nobenega ogrodja oziroma podpore za izdelavo poročil, ki jih uporabnik poleg možnosti hitrega vnosa podatkov, v praksi vedno potrebuje.

Tabele relacijske podatkovne baze so skoraj vedno povezane med sabo s povezavami ena-proti-mnogo. UVPB naj bi to upošteval in uporabniku omogočil pridobitev vrednosti določenega polja določenega zapisa iz polja druge tabele. Še več, vmesnik naj bi uporabniku omogočil spremembo prikaza trenutnega obrazca tabele v obrazec vnosa druge tabele, kjer bi uporabnik lahko vnesel več zapisov. Zatem bi izbral določen zapis in vstavil primarni ključ tega zapisa v polje prvotnega obrazca. Opisana navigacija med obrazci bo zahtevala infrastrukturo in mehanizme trajnega hranjenja podatkov določenega obrazca.

1.3 Podobna ogrodja in vmesniki do podatkovnih baz

1.3.1 Ogrodja

Razvijalci programske opreme imajo več možnosti nakupa oziroma uporabe podobnih ogrodij. Izmed bolj popularnih so:

- Zend,
- CakePHP,
- Symfony,
- Propel.

Kot že omenjeno v poglavju Namen in cilj magistrskega dela so poglavitni vzroki za razvoj ogrodja UVPB v primerjavi uporabe že razvitih ogrodij spodaj naštete pomanjkljivosti obstoječih ogrodij:

- pomanjkanje mehanizmov in vzorcev na odjemalčevi strani,
- premajhno upoštevanje praktično naravnanih značilnosti opisanih v poglavju 1.1:
    - samodejno kreiranje HTML obrazcev,
    - velika odzivnost HTML obrazcev,
    - navigacija obrazcev med povezanimi tabelami,
    - izdatna uporaba tipkovnice;

1.3.1.1 Symfony

Ogrodje Symfony je razvito z namenom optimizacije razvoja spletnih aplikacij na podlagi nekaj ključnih lastnosti. Ogrodje uporablja model tridelne arhitekture (MVC), ki ločuje poslovna pravila aplikacije, logiko strežnika in predstavitvene poglede. Nenazadnje, ogrodje avtomatizira mnogo opravil ter tako razvijalcu dovoljuje večjo osredotočenost na poslovno logiko namesto na tehnične plati aplikacije, kot je na primer navigacija med spletnimi stranmi, ločevanje dolgih poročil na posamezne spletne strani itd. Symfony, ki je v celoti napisan v skriptnem jeziku PHP 5, je bil testiran v različnih realnih okoljih in je

kompatibilen z večino sistemov za upravljanje s podatkovnimi bazami (Potencier & Zaninotto, 2007).

Lastnosti ogrodja Symfony so naslednje: enostavna namestitev, neodvisnost od podatkovne baze, enostavna uporaba, fleksibilnost, možnost konfiguracije, stabilnost, lahko brana programska koda s phpDocumentor komentarji, vgrajen internacionalizacijski sloj tako za prevode podatkov kot za prevode vmesnika, samodejna kontrola podatkov vnosnih obrazcev, upravljanje predpomnilnika, sistem avtentikacije in avtorizacije, Ajax, vgrajeno testiranje funkcionalnosti ter testiranje enot, razhroščevalnik, open source itd. (Potencier & Zaninotto, 2007).

Uporabniška skupina Symfony je velika in omogoča potrebno podporo ter možnosti prispevanja. Ogrodje Symfony je brezplačno in zaščiteno z licenco MIT.

Symfony uporablja objektno-relacijski abstrakcijski sloj, ki ločuje SQL poizvedbe od aplikacijske logike, kar omogoča optimizacijo ter prilagoditve posameznim sistemov za hranjenje podatkov. Abstrakcijski sloj samodejno prevaja klice objektov v SQL poizvedbe, ki nato vplivajo na podatkovno bazo. Ta abstrakcijski sloj je pravzaprav samostojno ogrodje imenovano Propel, ki je lahko vključeno v ogrodje Symfony, in je podrobno opisano v poglavju 1.3.1.2.

1.3.1.2 Propel

Propel je objektno-relacijska transformacija za PHP 5 zaščiteno z licenco GNU Lesser Public General License (LGPL). Propel na osnovi modela meta podatkov podatkovne baze Propel samodejno ustvari objekte, ki ustrezajo posameznim tabelam v podatkovni bazi. Dostop do baze tako poteka le preko teh objektov. Ustvarjeni objekti so prazni podrazredi, katerih metode omogočajo dostop do podatkovne baze. Te metode je mogoče prilagoditi ali jih nadomestiti z novimi.

Propel temelji na Apache Torque, znanem ogrodju za Javo, in izvaja objektno relacijsko transformacijo. Ogrodje je razdeljeno na dva dela. Prvi del je generator vseh razredov, drugi pa je delujoč del, ki omogoča uporabo teh razredov ter njihovo interakcijo s podatkovno bazo.

Tako kot UVPB je tudi Propel vmesnik do podatkovne baze, ni pa uporabniški vmesnik. Medtem ko je UVPB vmesnik med uporabnikom in podatkovno bazo, je Propel vmesnik med razvijalcem in podatkovno bazo. Propel namreč ustvari zgolj razrede ter okolje v katerem ti razredi delujejo, ne ustvari pa dejanskih obrazcev, ki uporabniku omogočajo spreminjanje podatkov podatkovne baze.

Razvoja UVPB se je začel leta 2004, ko ogrodja, kot je na primer Propel, še ni bilo na voljo, problem pa je tudi predstavljala premalo zmogljiva strojna oprema s prepočasnimi komunikacijskimi povezavami. Majhen objekt *DbObject_SS* (poglavje 3.4) ima tudi danes nezanemarljive prednosti hitrosti in manjše porabe delovnega spomina v primerjavi z obsežnimi ogrodji kot je Propel, vendar bi bilo, zaradi zmogljivejše strojne opreme, zrelosti ogrodja in dobre podpore skupnosti razvijalcev, pri nadaljnjem razvoju UVPB smiselno podrobneje preučiti možnost vključitve ogrodja Propel v UVPB ogrodje.

1.3.2 Uporabniški vmesniki do podatkovnih baz

Eden izmed ciljev magistrskega dela je končnemu uporabniku ponuditi vmesnik do podatkovne baze, ki bo omogočal izvrševanje osnovnih operacij nad podatkovno bazo. Na voljo je mnogo vmesnikov do podatkovnih baz od katerih je PhpMyAdmin zagotovo najpopularnejši.

1.3.2.1 PhpMyAdmin

PhpMyAdmin je izredno razširjen uporabniški vmesnik do podatkovnih baz, ki poleg osnovnih UBDB (ustvari, beri, dopolni in briši zapis) operacij omogoča tudi ustvarjanje in spreminjanje struktur podatkovne baze, kot so tabele, polja, tipi tabel, tipi polj, izvrševanje SQL stavkov in drugo. Vmesnik ne vključuje navigacije med obrazci povezanih tabel, ni prilagojen delu z tipkovnico in zato ni primeren za velike količine vnosa podatkov. Vmesnik je namenjen zgolj končnemu uporabniku in se ne uvršča med ogrodja novim informacijskim sistemom.

2 Razvoj Uporabniškega vmesnika do podatkovne baze

2.1 Uvod

Razvoj informacijskega sistema je zahteven in tvegan proces. Pomemben dejavnik tveganja je sposobnost razvijalca - razvijalec mora razumeti, upravljati in nadzorovati zapletenost vseh elementov sistema ter odnose med njimi. Uspešnost procesa pa ni odvisna zgolj od usposobljenosti razvijalca oziroma ekipe razvijalcev. V primeru nezadostne podpore managementa ali v primeru pogostega spreminjanja uporabniških zahtev se znatno poveča možnost za neuspeh projekta.

Za boljši uspeh projekta je potrebno celoten proces vzeti resno, ga izvesti sistematično in ustrezno dokumentirati glede na izbrano metodo razvoja informacijskega sistema. Večina informacijsko – tehnoloških projektov danes ne dosega zastavljenih projektnih ciljev kot so: čas, strošek in kvaliteta.

Proces razvoja informacijskega sistema pričnemo z definicijo domenskega modela, ki definira koncept in interakcijo stvari (domenskih vrednosti) v domeni aplikacije. Ko presodimo, da je model domene dovolj ekspresiven, lahko proces nadaljujemo s transformacijo modela domene v model programske opreme. Medtem ko model domene uporabnik lažje razume, je model programske opreme zanj terminološko težje razumljiv, saj so uporabljeni termini bolj tehnične narave.

Napačno bi bilo sklepati, da je razvoj informacijskega sistema zgolj prepis oziroma uporaba obstoječih rešitev v model programske opreme. Razvoj programske opreme pomeni tudi dodajanje novih orodij in avtomatizacij, ki bodo skupaj z drugimi elementi integrirana v nov sistem (Zullighoven, 2005, str. 73). Proces razvoja pomeni učenje in študiranje domene sistema to je elementov sistema ter povezav med njimi. Za učinkovit prenos znanja od strokovnjakov domene sistema ter do uporabnikov starega ali podobnega sistema je ključno, da ima razvijalec ustrezne sposobnosti medosebnega komuniciranja.

2.2 Analiza uporabniških zahtev

Pri razvoju sistema moramo najprej analizirati zahteve in želje ter določiti cilje, katere mora sistem izpolniti. Zahteve moramo opredeliti tako, da jih bo razumel vsak potencialen bralec. To poglavje je razdeljeno na dva dela: 2.2.1 in 2.2.2.

Analiza uporabniških zahtev je zahtevna faza predvsem zaradi nezanesljivih virov informacij. Uporabniki pogosto ne vedo, kaj točno potrebujejo, ali pa svojih potreb ne znajo izraziti. Poleg tega so uporabniške zahteve znotraj sistema lahko med seboj nasprotujoče in izključujoče, lahko pa so tudi ekonomsko neupravičene.

Analiza zahtev je začetna točka razvoja informacijskega sistema. Analiza zahtev vključuje zbiranje informacij o sistemu v obliki opisnih stavkov, seznanjanje z njimi in njihovo preučevanje. Za obsežnejše informacijske sisteme podjetja potrebujejo več ljudi in porabijo več časa za izpeljavo tega koraka.

Rezultat analize zahtev je dokument uporabniških zahtev, ki vsebuje nekaj ključnih problemov projekta, in služi kot podlaga za nadaljnjo podrobnejše izpopolnjevanje uporabniških zahtev v dokumentu specifikacije.

Po fazi specifikacije uporabniških zahtev nastopi faza zasnove sistema, kjer se definira arhitektura sistema ter kasneje ostali podrobni elementi. Kasnejša faza implementacije pa pomeni transformacijo opisnih ter vizualnih modelov prejšnjih faz v formalno obliko s točno določeno sintakso programske kode.

2.2.1 Ugotavljanje uporabniških zahtev

V tej fazi mora sistemski analitik ali razvijalec sistema ugotoviti uporabniške zahteve ali kaj uporabnik pričakuje od končanega razvoja sistema. Zahteve se zabeležijo kot opisni stavki. Faza je najmanj tehnična, z večjim poudarkom na socialnih, komunikacijskih in voditeljskih veščinah. (Leszek, 2001, str. 80). Proces zbiranja zahtev je potrebno jemati resno, saj lahko vsak nesporazum ali nerazumevanje zahtev privede do mnogo delavnih ur razvoja funkcionalnosti, ki jo uporabnik sploh ne potrebuje.

Leszek klasificira uporabniške zahteve na naslednji način (Leszek, 2001, str. 80):

1. Trditve storitev,
   1. funkcijske zahteve,
   2. podatkovne zahteve;
2. Trditve omejitev.

2.2.1.1 Licitacija uporabniških zahtev

Ta faza temelji na posvetovanjih tako z uporabniki kot z domenskimi strokovnjaki. Komunikacija z domenskim strokovnjakom je relativno enostavno pridobivanje domenskega znanja, saj sama strokovnost narekuje visoko mero jasnih ter logičnih trditev, ki zmanjšujejo možnost nesporazuma. Komunikacija z uporabniki sistema, kjer vsak uporabnik razume in uporablja le del celotnega sistema, pa velja za zahteven in nesporazumom podvržen proces. Potrebno je dodati, da domenski strokovnjak ni vedno nujno potreben – pri enostavnih projektih zadostuje le uporabnik, še posebej, če je dobro seznanjen s sistemom in ga dobro razume.

Zbrane informacije s strani domenskih strokovnjakov imajo značilnost stalnih poslovnih pravil, ki so primerna za večino organizacij ali sistemov, kjer bodo razviti informacijski sistemi implementirani (Leszek, 2001, str. 82).

Licitacija uporabniških zahtev je izvedena s tradicionalnimi ali modernim metodami (Leszek, 2001, str. 82).

Tradicionalne metode so:

- intervju,
- vprašalnik,
- opazovanje,
- proučevanje delovnih procesov, dokumentov in drugih programskih sistemov,
- druge.

Moderne metode so:

- metoda prototipov,
- JAD,

- RAD,
- druge.

Primer UVPD

Razvoj UVPD zahteva poznavanje konceptov kot so primarni ključ, tuji ključ, povezave (ang.: association), normalizacija itd. Navedeni termini so prvotno tehnične narave, vendar so tesno povezani z resničnimi sistemi in resničnimi situacijami. Koncept podatkovne baze je izredno razširjen. Vsak informacijski sistem vsebuje množico podatkov, ki jo mora sistemski analitik pravilno strukturirati v med seboj povezane tabele. Kasneje, ko sistem že vsebuje podatke, se le-te prikliče v obliki SQL poizvedb kot na primer *SELECT * FROM tabela*.

Licitacija uporabniških potreb je v primeru razvoja UVPB izvedena v okviru tradicionalne metode proučevanja delujočega programskega sistema „Glavna knjiga“ podjetja Klik d. o. o. ter proučevanju razširjenega uporabniškega vmesnika do podatkovne baze PhpMyAdmin, opisanega v poglavju 1.3.2.1. Med fazo razvoja so se uporabniške zahteve še dodatno preverjale.

2.2.1.2 Usklajevanje in preverjanje uporabniških zahtev

Zaradi zapletenosti in nezanesljivosti informacij pridobljenih z licitacijo, ugotavljanje uporabniških zahtev ni tako enostaven in jasen „A-Z“ proces. Usklajevanje ter preverjanje zahtev se ne začne z zaključkom faze licitacije uporabniških zahtev, temveč poteka vzporedno z njo. Pridobljene informacije se mora večkrat preveriti in uskladiti med različnimi uporabniki, preden se jih lahko zbere oziroma zapiše kot zanesljiv in koherenten dokument uporabniških zahtev.

Med preverjanjem in usklajevanjem zahtev moramo določiti meje sistema, da se izognemo izstopanju iz območja (ang.: scope creep), ter določiti, katere zahteve bodo predmet implementacije in katere ne. Tukaj nam lahko zelo pomaga kontekstni diagram (ang.: context diagram), saj je zaradi visoke stopnje abstrakcije iz njega jasno razvidno, kateri elementi spadajo v sistem in kaj so zunanje entitete, ki spadajo izven območja sistema.

Zbiranje uporabniških zahtev navadno vključuje množico uporabnikov, tako da je prekrivanje in izključevanje uporabniških zahtev neizogibno. V primeru velikega števila uporabnikov v fazi določanja uporabniških zahtev, je priporočljivo kreiranje matrike odvisnosti uporabniških zahtev, kjer so zahteve podrobno in natančno oštevilčene ter strukturirane (Leszek, 2001, str. 89).

Smiselnost razvoja informacijskega sistema se ugotavlja s študijo izvedljivosti projekta. V študiji podrobno opredelimo in ocenimo faktorje tveganja, ki določajo ali nam bo sistem uspelo razviti ali ne. Faktorji tveganja so naslednji (Leszek, 2001, str. 89):

- Tehnični,
- kakovostni,
- varnostni,
- integriteta podatkovne baze,
- proces razvoja,
- politični,
- zakonski,
- nestabilnost.

Zgoraj navedeni faktorji tveganja so lahko opisno ali vrednostno ocenjeni s poljubno izbrano odločitveno metodo. Lažje ocenjevanje faktorjev nam omogoča programska oprema DEXi za podporo večparametrskemu odločanju, razvita na inštitutu Jožefa Stefana.

2.2.1.3 Model poslovnih uporabniških zahtev

V fazi ugotavljanja uporabniških zahtev zabeležimo zahteve kot opisno besedilo v obliki trditev. Poleg pisnega dela je, zaradi boljšega razumevanja proučevanega sistema, smiselno kreiranje množice slik ali diagramov, ki prav tako opisujejo dani sistem. V diagramih opredelimo meje sistema, poglavitne poslovne primere uporabe in glavne poslovne razrede. Po Leszeku se v tej fazi izdelata dva glavna diagrama: model poslovnih primerov uporabe ter model poslovnih razredov, ki bosta služila kot osnova za nadaljnjo analizo v kasnejših fazah (Leszek, 2001, str. 93).

Primer UVPB

Za lažje razumevanje osnovnih zahtev se izdela dva vizualna modela resničnega sveta, Slika 1 nam prikazuje model poslovnih primerov uporabe, Slika 2 in Slika 3 pa prikazujeta model poslovnih razredov.

2.2.1.3.1 Model poslovnih primerov uporabe

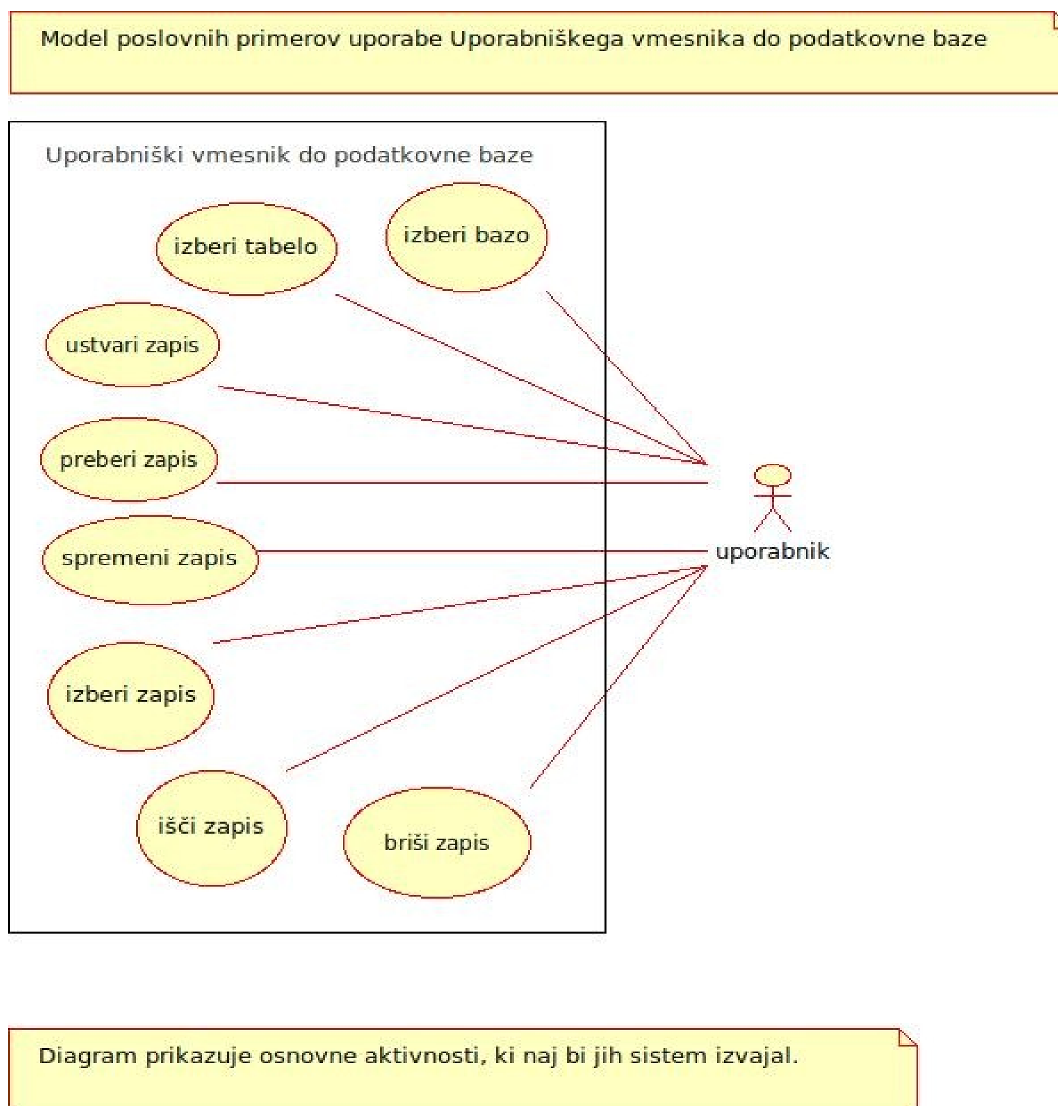
Model poslovnih primerov uporabe predstavlja visoko stopnjo abstrakcije (Kruchten, 1999). Identificirani so poglavitni poslovni procesi in na kratko opisani vsi primer uporabe. Povezave med primeri uporabe se v tem koraku še ne ugotavlja.

Primer UVPB

Spodnji diagram je model poslovnih primerov uporabe informacijskega sistema UVPB. Drugo ime za sistem bi lahko bilo tudi „UBDB brskalniški vmesnik do podatkovne baze“,

kjer kratica UBDB (ang.: CRUD) pomeni ustvari, beri, dopolni in briši zapis. To pa so tudi vse osnovne operacije, ki jih potrebujemo pri uporabi podatkovne baze.

Slika 1: Model poslovnih primerov uporabe UVPB



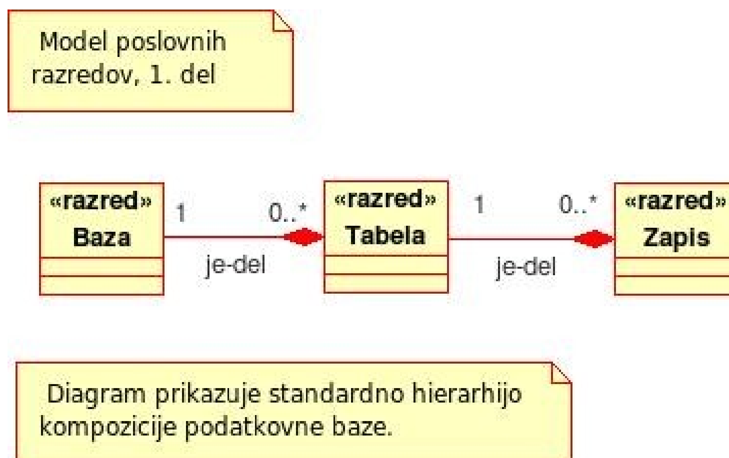
2.2.1.3.2 *Model poslovnih razredov*

Poslovni razredi so izpeljani iz poslovnih primerov uporabe. Njihovo podrobnejše proučevanje nam omogoča definiranje medsebojne povezanosti med razredi. Izdelava končnega modela razredov je zahtevna aktivnost analiziranja, v kateri je potrebno mnogo preverjanj, iteracij in popravkov. Diagram je visoka stopnja abstrakcije končnega modela razredov, tako da v tem koraku zadostuje kratek opis poglavitnih razredov prikazanih v diagramu.

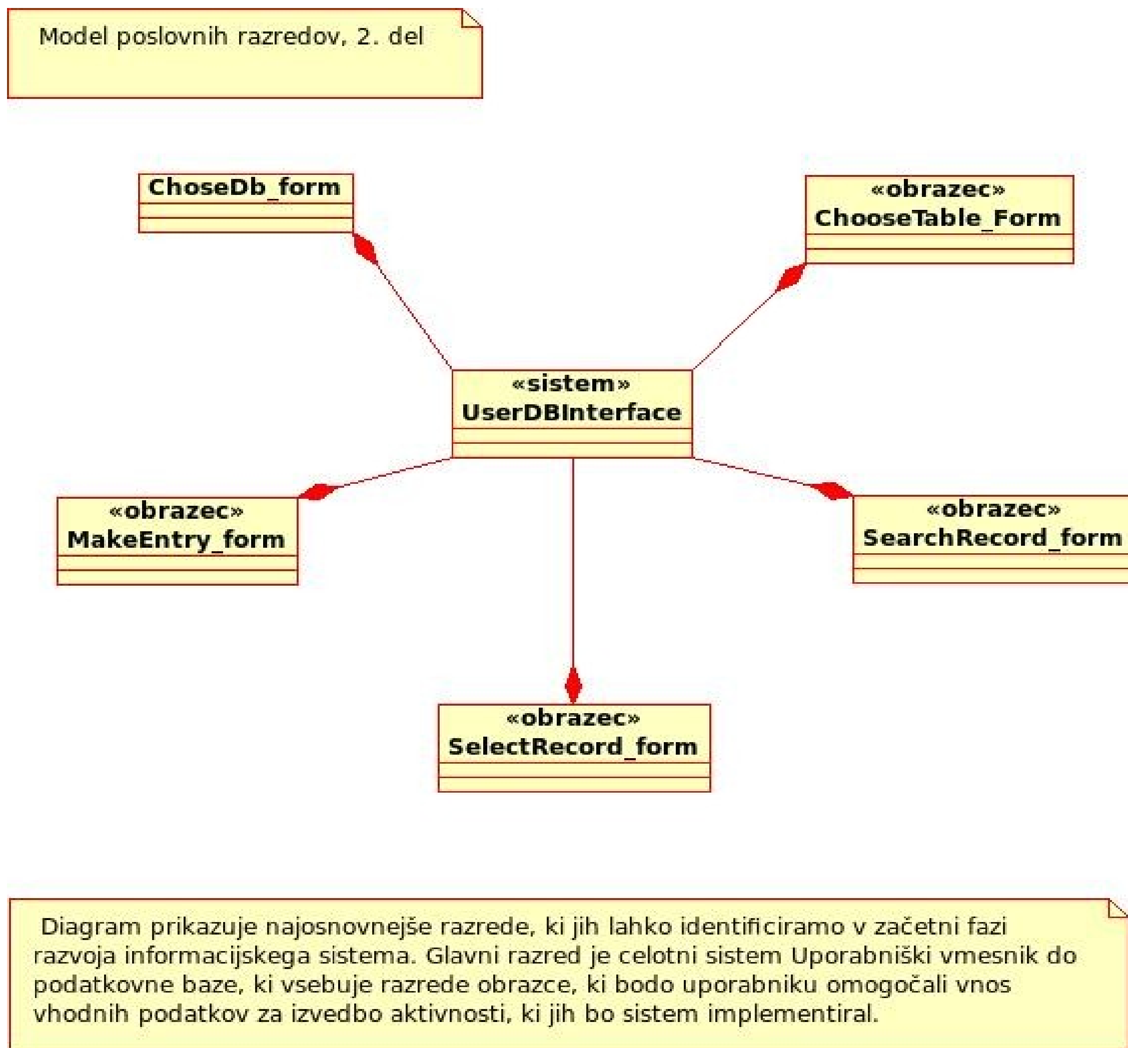
Primer UVPB

Pri proučevanju primerov uporabe ugotovimo, da obstaja več razredov, ki jih lahko razdelimo v dve skupini. Ena skupina je tesno povezana s podatkovno bazo ter preko nje s trajnostjo hranjenih podatkov. Ti razredi so prikazani na Sliki 2. Druga skupina razredov (Slika 3) je povezana z množico obrazcev in njihovimi elementi, ki bodo uporabniku omogočali vnos, izbor ter iskanje želenih podatkov.

Slika 2: Model poslovnih razredov, 1. del



Slika 3: Model poslovnih razredov, 2. del



Temeljitejša razčlenitev diagrama se izdela v fazi analize in zasnove procesa razvoja informacijskega sistema.

2.2.1.4 Dokument uporabniških zahtev

Dokument uporabniških zahtev je rezultat faze ugotavljanja uporabniških zahtev. Vsebuje: trditve zahtev, namen projekta, kontekst, urnik, proračun projekta ter obravnava druge ključne probleme projekta.

Tipični dokument uporabniških zahtev je strukturiran v naslednje sklope (Leszek, 2001, str. 99):

- **Uvod.** V prvi vrsti je namenjen managerjem in drugim odločevalcem, za katere je manj verjetno, da bodo podrobneje proučili celoten dokument.
  1. Namen in obseg projekta ali produkta.

2. Poslovni kontekst.
3. Deležniki.
4. Ideje za rešitve.
5. Povzetek dokumenta.

- **Storitve sistema.** To je glavni del dokumenta.
  1. Obseg sistema.
  2. Funkcijske zahteve.
  3. Podatkovne zahteve.
- **Omejitve sistema.** Opis, kaj omejuje sistem.
  1. Zahteve vmesnika.
  2. Zahteve učinkovitosti.
  3. Varnostne zahteve.
  4. Operativne zahteve.
  5. Politične in zakonske zahteve.
  6. Ostale omejitve.
- **Drugo.** Opis problemov, ki so pomembni za projekt in niso opisani v zgornjih poglavjih.
  1. Odprti problemi.
  2. Predhodni urnik.
  3. Predhodni proračun.

Izstopanje iz območja (ang.: scope creep). Praktično je nemogoče razviti optimalen sistem, ki bi vključeval vso želeno funkcionalnost. Res je, da je razvoj sistema pogosto iterativen proces, kjer pri vsaki iteraciji povečujemo stopnjo funkcionalnosti, vendar je potrebno definirati časovne in finančne okvirje prve ali prvih nekaj iteracij ter ugotoviti ekonomsko smiselnost projekta za dani okvir. Za dani časovni okvir je potrebno določiti primeren oziroma maksimalen obseg uporabniških zahtev zato, da se izognemo „izstopanju iz območja“. S tem zaščitimo uspeh projekta pred tveganjem, da ga ne bi dokončali v danem časovnem ali finančnem okvirju zaradi prevelike množice uporabniških zahtev (za določen odstotek le-teh se bo zagotovo izkazalo, da so nepotrebne).

2.2.2 Specifikacija uporabniških zahtev

Po tem, ko smo zbrali vse osnovne uporabniške zahteve in jih zabeležili v dokumentu uporabniških zahtev, je zbrane zahteve potrebno podrobneje specificirati s pomočjo grafičnih in drugih formalnih modelov. Ta faza pomeni razlago in temeljito analizo dokumenta uporabniških zahtev, saj predstavlja njegovo razširitev. Grafični modeli dokumenta specifikacije uporabniških zahtev so narejeni s pomočjo jezika UML, ki vsebuje znatni izbor grafičnih elementov in pravil možnih interakcij med njimi.

2.2.3 Razlaga primerov uporabe

Ko so osnovne aktivnosti sistema odkrite, sistemski analitik nadaljuje z njihovo razlago oziroma analizo. Pri tem identificira množico dodatnih razredov. Zaradi porazdeljenega sistema bodo aktivnosti potekale tako na strani strežnika kot na strani odjemalca. Vsak primer uporabe, kot je prikazan na Sliki 1, bo implementiran s pomočjo svojega razreda, ki bo povezoval dva dodatna razreda. En izmed teh dodatnih razredov bo implementiran na strani strežnika in bo izvajal aktivnosti, ki bodo potekale na strežniku, drug razred pa bo implementiran na strani odjemalca ter bo izvajal aktivnosti, ki bodo potekale na njegovi strani.

Model poslovnih razredov na Sliki 3 nam prikazuje množico osnovnih razredov, ki bodo specificirali različne tipe obrazcev. Vsak primer uporabe bo implementiran s konkretnim obrazcem, ki bo uporabniku omogočal interakcijo s sistemom. Implementacija konkretnega obrazca bo v obliki programske kode, ki je napisana v datoteki, in bo prevedena na odjemalčevi infrastrukturi kot je na primer internetni brskalnik. Primer uporabe je omejen le na odjemalčeve aktivnosti in je sestavljen iz množice manjših aktivnosti. To pomeni, da vsaki odjemalčevi zahtevi obrazca sledi prikaz le-te, inicializacija „poslušalcev dogodkov“ (ang.: event listeners), dodajanje elementov na obrazec itd. Upravljanje zahtev različnih obrazcev poteka z mehanizmom trajnostnih obrazcev.

2.3 Sistemska analiza in zasnova

2.3.1 Sistemska analiza

Analiza informacijskih sistemov pomeni formacijo razredov in konceptov interakcije, ki bodo logično odražali resnični svet. Osnova za analizo je dokument uporabniških zahtev ali pa dokument specifikacije uporabniških zahtev, ki poleg opisnega besedila vsebuje še: primere uporabe, tabele entitete, tabele aktivnosti itd. Koncepti in predmeti resničnega sveta se pogosto ne morejo direktno prenesti v objekte, elemente ter njihove povezave programskih sistemov, zato lahko za njihovo identificiranje uporabimo različne tehnike kot so CRC (ang.: Clas Responsibility Collaboration), igranje vlog (ang.: Role playing), analiza samostalnikov (ang.: Noun analysis) itd (Conallen, 1999, str. 141). Analiza zahteva razčlenitev primerov uporabe na podrobne aktivnosti interakcije med uporabniki in sistemom. Podrobne aktivnosti se lahko opiše v besedilu, lahko pa se jih predstavi kot niz UML diagramov zaporedja, UML diagramov sodelovanja ali tabele aktivnosti.

V skladu z modernimi koncepti bo informacijski sistem razvit kot objektno usmerjen sistem, medtem ko bo podatkovna baza relacijsko usmerjena. To bo povzročilo problem transformacije objektov aplikacijskega sistema v podatkovno bazo in obratno. Transformacija bi lahko potekala preko vmesnika oziroma množice specializiranih objektov, katerih naloga bi bila le pretvarjanje in prenašanje objektov med sistemoma. Primer takega vmesnika je Propel, ki je opisan v poglavju 1.3.1.2. Za potrebe

magistrskega dela se tak vmesnik ne bo podrobneje obravnaval ter ne bo implementiran. Prenos bo v razvitem sistemu implementiran s strukturiranim povpraševalnim jezikom (ang.: Structured Query Language) in aplikacijskim tipom spremenljivke, imenovanim polje (ang.: array).

2.3.2 Sistemska zasnova

Faza sistemske zasnove je proces prevajanja uporabniških zahtev v implementacijsko logiko. Preučevanje in razčlenjevanje diagrama razredov, diagrama zaporedja ter kolaboracijskega diagrama poteka toliko časa, dokler nimamo dovolj meta informacij za pričetek programiranja (Conallen, 1999, str. 147).

Objektno usmerjeni programski jeziki vsebujejo enostavne tipe spremenljivk kot so število (ang.: integer), niz (ang.: string), dvojiškost (ang.: boolean) itd. Poleg teh tipov pa je mogoče definirati tudi poljubne tipe, kar je izredno pomembna lastnost teh jezikov. S pomočjo poljubno definiranih tipov lahko določimo poljubne domenske tipe, ki ustrezajo točno določeni domeni. Domenske tipe spremenljivk implementiramo kot razrede in tako sami določimo semantiko modela zasnove ter modela implementacije. Primerki domenskih tipov so domenske vrednosti, ki se v postopku sistemske zasnove prevedejo v objekte programskega sistema.

2.4 Pristop orodja in materiala v sistemski analizi in zasnovi

2.4.1 Uvod

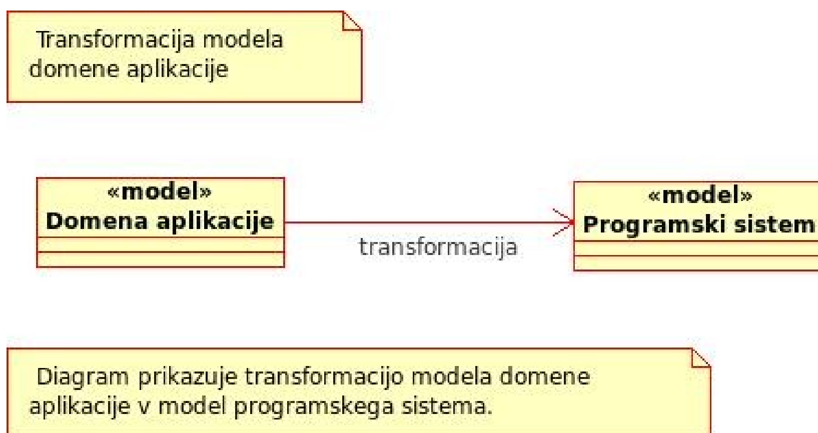
Splošna ideja o tem, kako lahko z orodji manipuliramo material, nam je vsem znana ter celo samoumevna, kljub temu pa jo lahko poimenujemo z izrazom kot je Vzorec interakcije med orodjem in materialom. Vzorec zato, ker pri razvijanju informacijskega sistema, probleme, na katere naletimo rešujemo tako, da določen problem primerjamo z že rešenimi problemi ali vzorci, nato pa poskušamo rešiti novo nastali problem na podoben način.

Razvoj informacijskega sistema s pristopom orodja in materiala se prične z identifikacijo vodilne metafore, nadaljuje pa z opisom le-te s pomočjo njene zasnove. Orodja in materiali so izredno pomembni, medtem ko so generični postopki in procesi manj (Zullighoven, 2005, str. 66). Če smatramo generične procese za avtomatizacijo, nam orodja omogočajo, da izberemo tip avtomatizacije in njihovo zaporedje. Namen pristopa orodja in materiala je, da razvijalce sili k aplikacijski usmerjenosti v procesu razvoja. Aplikacijska usmerjenost poudarja pomen nalog in procesov, ki jih bodo uporabniki sistema izvajali z razvitimi sistemi (Zullighoven, 2005, str. 4)

Eden izmed poglavitnih principov razvoja informacijskega sistema s pristopom orodja in materiala je upoštevanje dveh pogojev v procesu transformacije modela domene aplikacije v programski model (Zullighoven, 2005, str. 5):

- strukture obeh modelov morajo biti podobne,
- obstajati mora močna povezava med terminologijo modela domene aplikacije in modelom programske arhitekture.

Slika 4: Transformacija modela domene aplikacije



Napor, ki je v razvoj vložen z namenom, da bi zadostili zgoraj omenjenima pogojema, ni trivialen in določa temelj učinkovitega upravljanja kompleksnosti sistema. Razvijalec mora razumeti terminologijo obeh modelov, povezavo med resničnim svetom in programskim sistemom. Ta povezava mora biti čimbolj razumljiva.

Vodilna metafora mora biti slikovit izraz, ki predstavlja enoten pogled na celoten proces razvoja želenega sistema (Zullighoven, 2005, str. 5). Zasnovna metafora je ime objekta iz resničnega sveta, ki opisuje komponento sistema aplikacije. Najbolj pogoste zasnovne metafore so (Zullighoven, 2005, str. 6):

- material,
- orodje,
- avtomatizacija,
- delovno okolje,
- timsko delo.

Pri razvoju informacijskih sistemov uporabljamo pristope in metode. Naloga razvijalca je, da izbere primeren pristop ali primerno metodo, saj različni sistemi zahtevajo različne pristope in metode. Pristop orodja in materiala ni metoda, temveč je metodološki okvir, ki zajema naslednje elemente (Zullighoven, 2005, str. 9):

- objektno in aplikativno usmerjen pogled,
- zbirko uveljavljenih tehnik konstrukcije, analize in dokumentacije v obliki vzorcev,
- opis konceptov ter arhitekturnega modela,

- ovrednotenje množice projektnih izkušenj,
- nabor usmeritev za razvoj konkretne tehnike izgradnje.

2.4.1.1 Pogled podpore

S tehnologijo, ki se neprestano razvija, ter z vedno bolj izobraženo delovno silo se spreminja tudi narava dela. Slabo izobraženi delavci, odgovorni za rutinska opravila ali preprosto nadzorovanje in nastavljanje tehničnih naprav, se nadomeščajo z domenskimi strokovnjaki z znatnim domenskim znanjem. Potemtakem se spreminjajo tudi informacijski sistemi, katere ti delavci uporabljajo pri svojem delu - delovno-kontrolni sistemi se nadomeščajo s strokovnimi virtualnimi delovnimi mesti, s potrebnimi materiali in fleksibilnimi orodji, ki so uporabniku na razpolago. Vloga programskih sistemov začenja presegati zgolj vlogo avtomatizacije procesov ter vedno bolj prehaja v vlogo podpore visoko izobraženi delovni sili. Vloga podpore pomeni to, da naj bi programski sistemi zagotavljali nabor orodij, s katerimi bi bili uporabniki sposobni manipulacije oziroma upravljanja z domenskimi materiali.

2.4.2 Vodilna metafora in zasnovna metafora

Generično metafora pomeni besedno figuro, za katero je značilno poimenovanje določenega pojava z izrazom, ki označuje v navadni rabi kak drug podoben pojav (SSKJ, 2009). Metafora poudarja določene lastnosti ali poglede v originalnem izrazu.

Vodilna metafora je osnovni pogled, ki nam pomaga zaznati, razumeti in zasnovati del resničnosti (Zullighoven, 2005, str. 59).

Zasnovne metafore uporabljamo, da boljše izrazimo vodilno metaforo in tako povečamo izrazni učinek metafore (Zullighoven, 2005, str. 63). Zasnovne metafore nam podajo osnovne elemente naše aplikacijske domene, strukturirano izrazijo glavno idejo oziroma vodilno metaforo ter tako pojasnjujejo, kaj je potrebno modelirati, analizirati ter implementirati iz vidika domene (Zullighoven, 2005, str. 64).

2.4.3 Interakcija orodja in materiala

Povezavo med orodji in materialom imenujemo povezava uporabe, kar ustreza klasični povezavi klica, in je lahko določena s pogodbo (Zullighoven, 2005, str. 32). Ključni elementi kateregakoli informacijskega sistema so orodja in materiali, ki se nahajajo v domensko specifičnem delovnem okolju. Orodja uporabljajo materiale, materiali pa so uporabljeni, preiskani ter spremenjeni s strani orodij. Ta interakcija lahko poteka na veliko različnih načinov in kombinacij, katere je potrebno podrobneje proučiti z namenom, da bi orodja ter materiale čimbolj optimalno zasnovali.

Zasnova orodij ter materialov naj bi potekala na podlagi analize sistema, na podlagi specifikacije ter opisa primerov uporabe. Z določitvijo ter kasnejšo implementacijo orodij in materialov konstruiramo programski model, ki bo vključeval podrobnejši pogodbeni model interakcije med orodjem in materiali. Povezava uporabe je implementirana v definiciji razreda in opisuje operacije interakcije, ki so dovoljene med objekti (Zullighoven, 2005, str. 33). Dejanska povezanost med objekti je realizirana šele med delovanjem sistema po principu polimorfnosti, potem ko se kreirajo objekti (Zullighoven, 2005, str. 33).

Pogled (ang.: aspect)**.** Pogled je zasnovni vzorec, ki eksplicitno ponazarja interakcijo med orodji in materiali ter določa storitve, ki naj bi jih materiali nudili orodjem (Zullighoven, 2005).

Primer UVPB

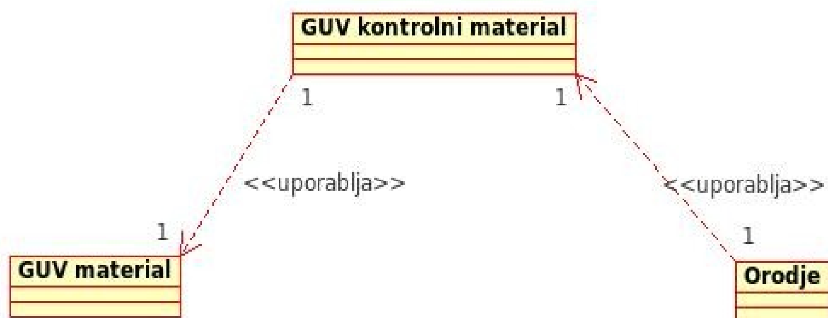
GUV (grafični uporabniški vmesnik) materiali so HTML materiali, ki sestavljajo HTML dokument in so na primer FORM, SUBMIT, TABLE itd. V našem sistemu bo GUV material manipuliran in upravljan preko trajnih obrazcev, zato bomo trajni obrazec poimenovali GUV kontrolni material. GUV kontrolni material pa bo manipuliran in upravljan z orodjem kot je prikazano na Sliki 5. Razlog za tako razdelitev in organizacijo je podrobneje opisan v poglavju 2.4.4.

Interakcijski vzorec je implementiran kot metoda *addToolKeypressed* v vsakem izmed objektov trajnega obrazca (ang.: Persistent Form) (objekt *ChooseTable_CS*, *SelectRecord_CS* itd.). V tej metodi lahko objekt povežemo z določenim orodjem (npr.: *ToolKeypressed{name of the tool}*) ter GUV materialom. Objekt orodja določenega razreda ne vsebuje nobenih atributov, za katere bi bila potrebna trajnost, zato zadostuje en primerek orodja za vsak tip trajnega obrazca. Vsak material oziroma objekt trajnega obrazca je lahko manipuliran le z enim orodjem, ki pa lahko nato delegira opravila drugim pod-orodjem.

Vsi materiali trajnih obrazcev se nahajajo v enem zbiralniku. V aplikaciji UVPB so lahko vsi objekti enega tipa uporabljeni z nič ali največ enim orodjem. To pomeni, da lahko eno orodje uporablja več objektov istega razreda. Posledično moramo zagotoviti, da bo orodje prejelo podatke ustreznega materiala trajnega obrazca (GUV kontrolni material) ter ustreznega GUV materiala.

Slika 5: Interakcija med GUV materialom, GUV kontrolnim materialom in orodjem

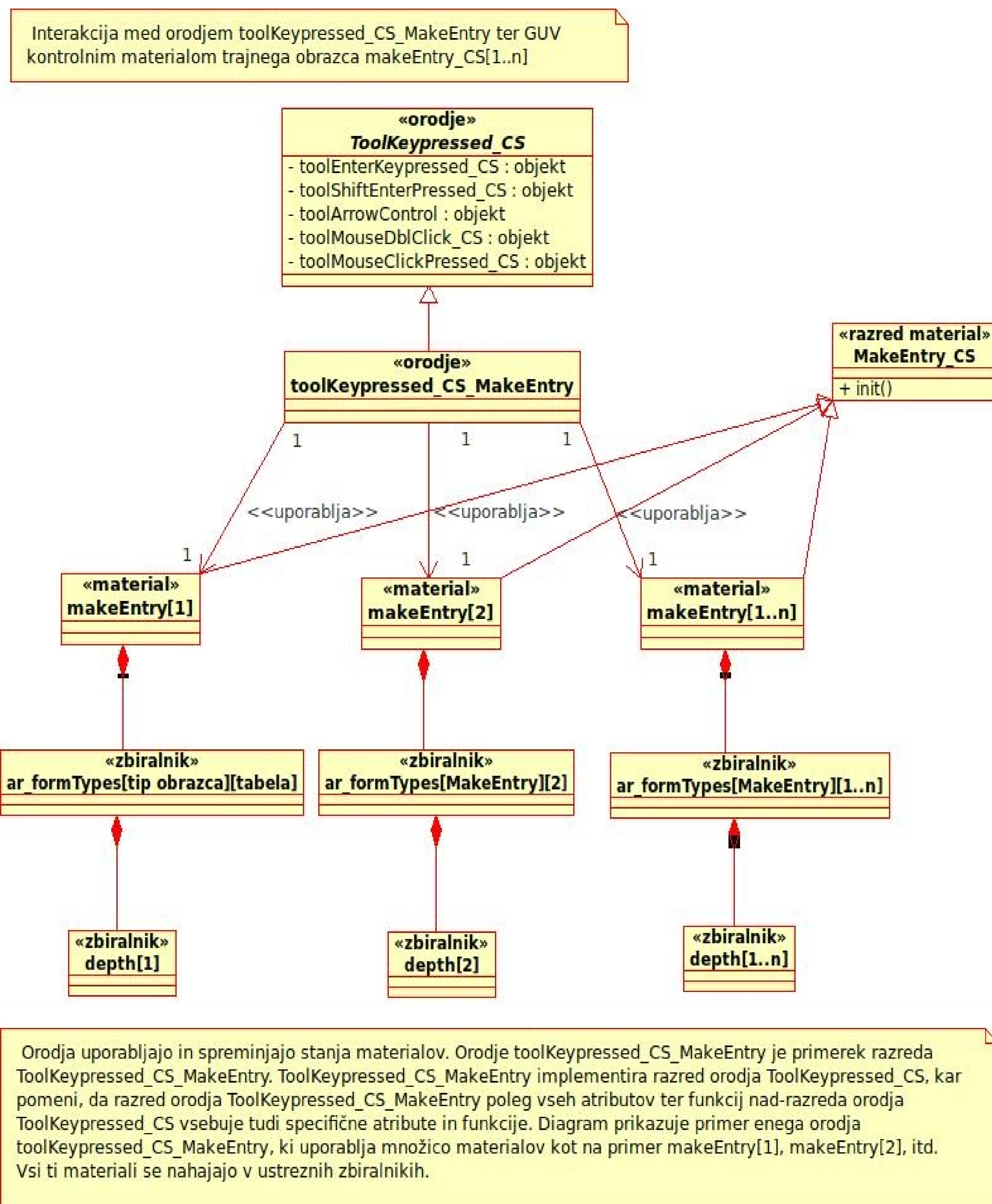
Interakcija med GUV materialom, GUV kontrolnim materialom in orodjem



Uporabnik uporablja orodja. Orodja uporabljajo GUV kontrolne materiale in preko njih GUV materiale.

Uporabnik uporablja orodja. Orodja uporabljajo GUV kontrolne materiale in preko njih GUV materiale.

Slika 6: Interakcija med orodjem toolKeypressed_CS_MakeEntry ter GUV kontrolnim materialom trajnega obrazca makeEntry_CS[1..n]

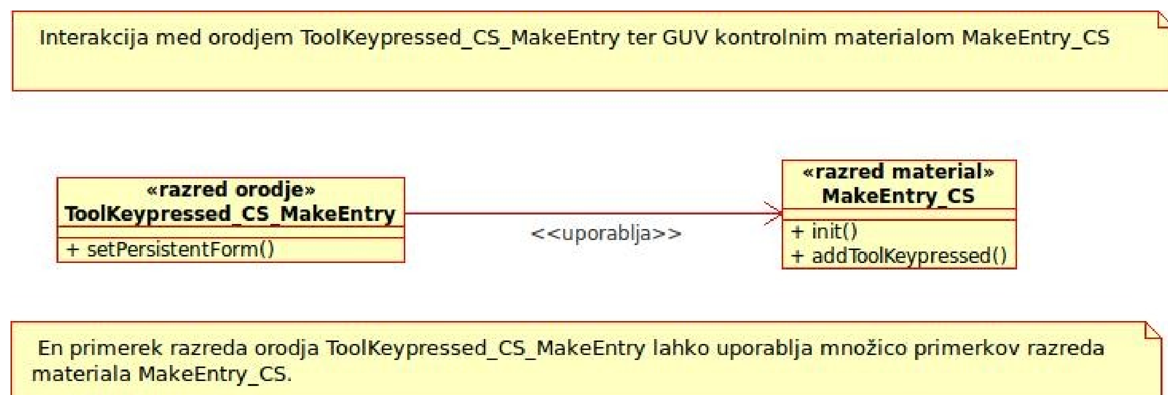


Orodja uporabljajo in spreminjajo stanja materialov. Orodje *toolKeypressed_CS_MakeEntry* je primerek razreda *ToolKeypressed_CS_MakeEntry*. *ToolKeypressed_CS_MakeEntry* implementira razred orodja *ToolKeypressed_CS*, kar pomeni, da razred orodja *ToolKeypressed_CS_MakeEntry* poleg vseh atributov ter funkcij

nad-razreda orodja *ToolKeypressed_CS* vsebuje tudi specifične atribute in funkcije. Diagram prikazuje primer enega orodja *toolKeypressed_CS_MakeEntry*, ki uporablja množico materialov kot na primer *makeEntry[1]*, *makeEntry[2]* itd. Vsi ti materiali se nahajajo v ustreznih zbiralnikih. Kratica CS v imenih razredov in objektov pomeni c*lient side* oziroma stran odjemalca.

Interakcija med orodji ter materiali kot prikazuje naslednja slika:

Slika 7: Interakcija med orodjem ToolKeypressed_CS_MakeEntry ter GUV kontrolnim materialom MakeEntry_CS



En primerek razreda orodja *ToolKeypressed_CS_MakeEntry* lahko uporablja množico primerkov razreda materiala *MakeEntry_CS*.

2.4.4 Material

Material je objekt, ki vključuje domensko specifične koncepte in bo sčasoma postal del razvitega sistema. Uporablja se ga z uporabo orodij in avtomatizacij (Zullighoven, 2005, str. 68). Razvoj materiala poteka tako, da se najprej identificira glavne materiale, ki bodo tvorili platformo, iz katere se bo naredil model programskih materialov (ki bo bodo že implementirani). V kontekstu programskega modela materiali predstavljajo objekte, ki bodo manipulirani z orodji prav tako kot v domenskem modelu (Zullighoven, 2005, str. 140).

Primer UVPB

Struktura množice materiala v porazdeljenem sistemu je naslednja :

- *Strežniška stran.*
  - *Zbiralnik seje (ang.: session container).*
    - *Obrazec (abstraktni podatkovni tip).*
      - *Start_SS form.*
        - ....
    - *Obrazec (abstraktni podatkovni tip).*
      - *ChooseDB_SS obrazec.*

            - *.......*
        - *ChooseTable_SS obrazec.*
            - *.......*
        - *MakeEntry_SS obrazec.*
            - *.......*
        - *SelectRecord_SS obrazec.*
            - *.......*
        - *SearchRecord_SS obrazec.*
- *Stran odjemalca.*
    - *Spletna stran odjemalca ali okno brskalnika.*
        - Obrazec (abstraktni podatkovni tip).
            - *Start obrazec.*
            - *ChooseDB obrazec.*
                - *.......*
            - *ChooseTable obrazec.*
                - *.......*
            - *MakeEntry obrazec.*
                - *.......*
            - *SelectRecord obrazec.*
                - *.......*
            - *SearchRecord obrazec.*
                - *.......*
        - *Trajni obrazec (abstraktni podatkovni tip).*
            - *Razred Start.*
                - *ChooseDB (zbiralnik).*
                    - *ChooseDB[1..n].*
                - *ChooseTable (zbiralnik).*
                    - *ChooseTable[1..n].*
                - *MakeEntry (zbiralnik).*
                    - *MakeEntry[1..n].*
                - *SelectRecord (zbiralnik).*
                    - *SelectRecord[1..n].*
                - *SearchRecord (zbiralnik).*
                    - *SearchRecord[1..n].*

Material spletne strani na odjemalcu

Naloga internetnega brskalnika, ki je nameščen na strani odjemalca, je kreiranje materialov spletne strani. Primerki spletnih strani so na strani odjemalcu ustvarjeni kot okna brskalnika s hierarhijo dokumentnega objektnega modela (DOM). Bolj natančno, taka hierarhija vsebuje objekt dokumenta, objekt obrazca, objekt elementa na obrazcu itd. Deklaracija spletnih strani in njihovih GUV elementov se nahaja v datotekah (html, htm, php, asp itd.), ki se nahajajo na odjemalčevem računalniku, in ki so bile poslane s strežnika.

Materiali trajnega obrazca

Spletne strani na odjemalcu niso trajnega značaja, saj je njihova vsebina trajna le toliko časa, dokler se spletna stran ne nadomesti z drugo spletno stranjo. Spletna stran je sicer tudi po nadomestitvi dostopna preko zgodovine spletnih strani, vendar brskalnikov mehanizem zgodovine spletnih strani ni namenjen in ni primeren za pogost kasnejši dostop do vrednosti posameznih elementov spletnih strani. Koncept materiala trajnega obrazca

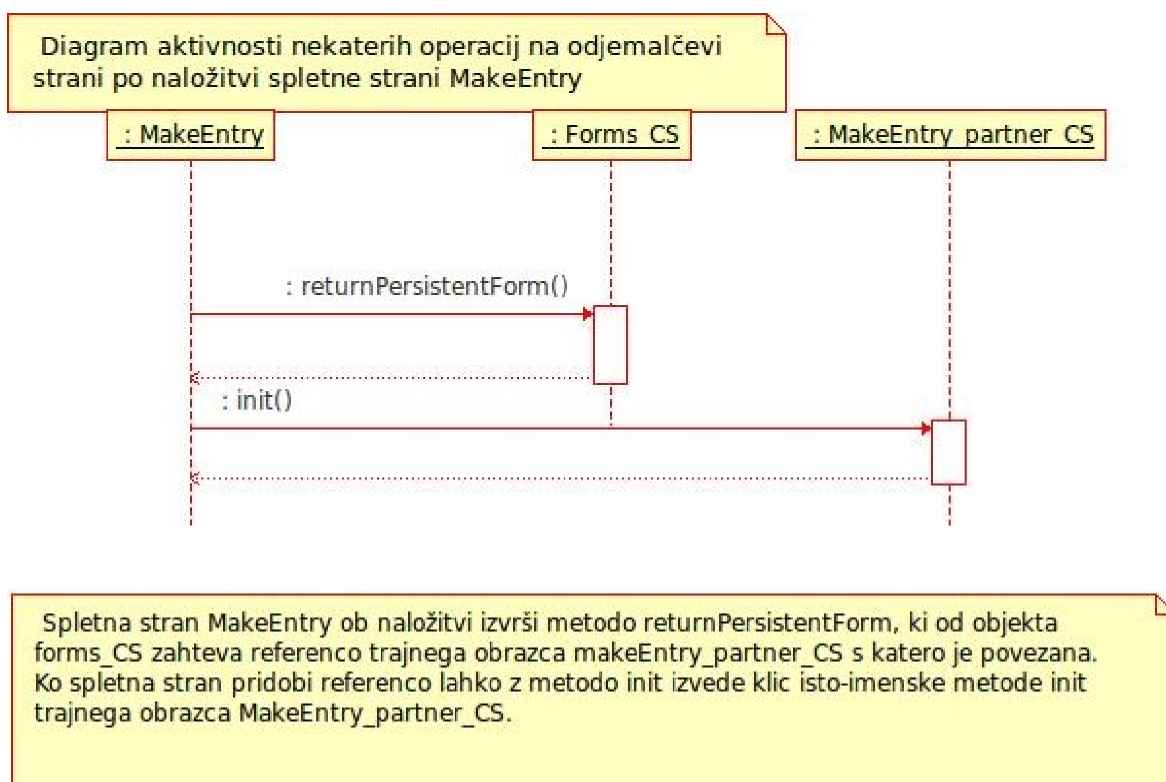
dopolnjujejo spletno stran tako, da trajno hrani vrednosti elementov spletnih strani in tudi specificira dodatne atribute ter obnašanje določenih spletnih strani na odjemalcu.

Materiali trajnih obrazcev so zelo podobni materialom GUV obrazcev, od njih pa se razlikujejo v trajnosti. Medtem ko se GUV obrazec ustvari ob vsaki naložitvi (ang.: load) spletne strani ter izbriše ob vsaki sprostitvi (ang.: unload), trajni obrazec ostane v odjemalčevem spominu med vso sejo. Ta trajnost omogoča relativno trajno hrambo podatkov ter njihovo kasnejšo obnovo. GUV obrazec je implementiran s HTML jezikom z začetno zastavico <FORM> in končno zastavico </FORM>. Upravljanje ter hranjenje materialov trajnih obrazcev je implementirano z mehanizmom trajnih obrazcev kot kompozicijski vzorec, ki je opisan v poglavju 3.6.4.

Zaradi dejstva, da se vnos v obrazec izvaja na odjemalcu, je hranjenje trajnih obrazcev tudi implementirano na strani odjemalca. Na ta način je minimalizirano število oddaljenih klicev, kar poveča hitrost izvršitve opravila. Alternativni način bi bila nadomestitev trajnega mehanizma hranjenja podatkov odjemalca s strežniškim trajnim mehanizmom, vendar bi tak pristop v veliki meri zmanjšal hitrost izvrševanja aktivnosti iz dveh razlogov. Prvi razlog je, da bi se ob vsaki zamenjavi vnosnega obrazca morala s strežnika prenesti specifikacija obrazca. Drugi razlog pa je, da bi se zahtevani obrazec moral ustvariti vsakokrat znova.

Brskalnik ob vsaki naložitvi spletne strani izvede niz splošnih aktivnosti, ki so implementirane v jeziku JavaScript. Ena izmed splošnih aktivnosti je klic objekta *forms_CS*, ki kreira oziroma poveže ustrezen trajni obrazec z določeno spletno stranjo. Kot je opisano v poglavju 3.6.4, objekt *forms_CS* vsebuje vse trajne obrazce, hierarhično strukturirane v ustreznih zbiralnikih. Druga pomembna splošna aktivnost je klic metode *init* v pripadajočih trajnih obrazcih. Zgoraj naštete splošne aktivnosti se izvršijo ob vsaki naložitvi spletnih strani, ki realizirajo ključne primere uporabe: izberi bazo, izberi tabelo, išči zapis, izberi zapis in dopolni zapis, kot je prikazano na Sliki 1. Poleg teh dveh metod pa metoda init vsebuje še dve splošni aktivnosti, in sicer *getRequiredInput* in *addToolKeypressed*. Metoda *getRequiredInput* vsebuje pravila pridobitve vhodnih podatkov, metoda *addToolKeypressed* pa kreira oziroma poveže trajni obrazec z ustreznim orodjem.

Slika 8: Diagram aktivnosti nekaterih operacij na odjemalčevi strani po naložitvi spletne strani MakeEntry



Spletna stran *MakeEntry* ob naložitvi izvrši metodo *returnPersistentForm*, ki od objekta *forms_CS* zahteva referenco trajnega obrazca *makeEntry_partner_CS*, s katero je povezana. Ko spletna stran pridobi referenco, lahko z metodo *init* izvede klic istoimenske metode *init* trajnega obrazca *MakeEntry_partner_CS*.

2.4.5 Orodje

Definicija. V kontekstu pristopa orodja in materiala je orodje objekt, ki ga ljudje lahko uporabljajo za spreminjanje in pregledovanje materiala (Zullighoven, 2005, str. 67).

Orodje predstavlja ponavljajočo se manipulacijo materiala (Zullighoven, 2005, str. 141). Poimenujemo ga s samostalnikom (npr.: tipkovnica), s katerim izpolnimo želena opravila ali množico opravil. Orodje naj bi bilo uporabnika prijazno in naj bi imelo več opcij. Orodja so centralni vmesnik med uporabnikom in sistemom.

Oblika orodja. Okno brskalnika je objekt - zbiralnik, ki vsebuje druge objekte. Kako lahko uporabnik ve, kateri objekt trenutno orodje tipkovnica ali miška manipulira? Vsako orodje naj bi imelo vizualno obliko na računalniškem zaslonu.

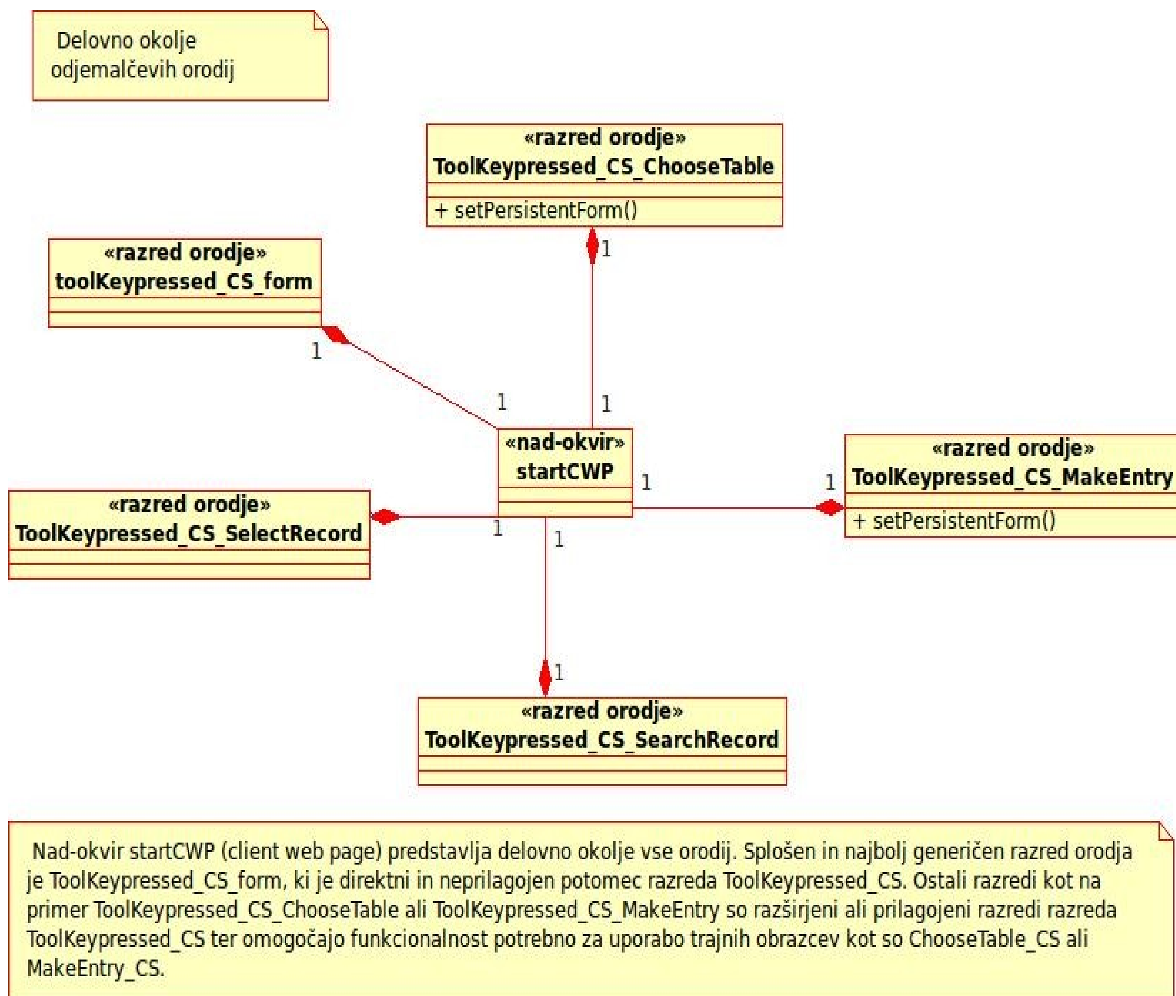
Primer UVPB

Upravljanje z materiali obrazca se izvaja z domenskim orodjem računalniška tipkovnica. Tipkovnica zagotavlja vmesnik za dosego različnih opravil, kot so na primer: izberi tabelo, dodaj zapis itd. Domensko orodje fizična tipkovnica je transformirano v objekt, ki ga v programskem sistemu imenujemo *toolKeypressed_CS*.

Kot je navedeno zgoraj, morajo orodja imeti vizualno obliko na zaslonu. Tako ima orodje tipkovnica obliko utripalke, ki z utripanjem opozarja, kateri objekt (vnosno polje) bo orodje trenutno spreminjalo (vpis vrednosti). Orodje miška je na zaslonu predstavljeno v obliki puščice, katere konica kaže na objekt, ki se bo ob uporabi orodja (pritisk leve tipke na miški) spremenil oziroma odreagiral na določen način.

Delovno okolje je prostor, kjer se delo izvršuje ter zaključi, in kjer so na voljo orodja ter materiali (Zullighoven, 2005, str. 6). UVPB je zasnovan kot tri nivojski sistem, kar napeljuje na dejstvo, da so orodja in materiali raztreseni med temi tremi podsistemi. Na Sliki 9 je prikazano delovno okolje odjemalca poimenovano *startCWP* (CWP - client work place).

Slika 9: Delovno okolje odjemalčevih orodij



Nad-okvir (ang.: frameset) *startCWP* (client web page) predstavlja delovno okolje vseh orodij. Splošen in najbolj generičen razred orodja je *ToolKeypressed_CS_form*, ki je direkten in neprilagojen potomec razreda *ToolKeypressed_CS*. Ostali razredi, kot na primer *ToolKeypressed_CS_ChooseTable* ali *ToolKeypressed_CS_MakeEntry*, so razširjeni ali prilagojeni razredi razreda ToolKeypressed_CS ter omogočajo funkcionalnost, ki je potrebna za uporabo trajnih obrazcev kot so *ChooseTable_CS* ali *MakeEntry_CS*.

Trajnost orodij. Orodja, potrebna za izvršitev glavnih primerov uporabe (izberi tabelo, ustvari zapis, išči zapis, izberi zapis), se ustvarijo na osnovi očetovskega razreda orodja *ToolKeypressed_CS* ter z razširitvenim in specifičnim razredom orodja, ki se nahaja v ločeni datoteki. Za primer uporabe "izberi tabelo" se tako ob naložitvi nad-okvirja *startCWP* kreira objekt orodja *toolKeypressed_CS_ChooseTable*, ki najprej od očetovskega razreda prevzame privzeto obnašanje ter privzete atribute. Takoj za določitvijo podedovanih lastnosti pa z nadomestitvijo privzetega orodje prevzame še specifično obnašanje in specifične atribute od specifičnega razreda orodja *ToolKeypressed_CS_ChooseTable*. Objekt odgovoren za ustvarjanje objektov orodij je

application_CS, ki je del nad-okvirja *startCWP*. Ker je nad-okvir *startCWP* trajen, so tudi vsi objekti, ki so del nad-okvirja, trajni, tako, da orodja, ko so enkrat ustvarjena, ostanejo na razpolago v ozadju med celotno sejo. Orodja se aktivirajo na zahtevo uporabnika preko brskalnikovega mehanizma dogodkov (ang.: events), kot je na primer pritisk tipke (ang.: keypress).

Razred orodja *ToolKeypressed*. Povezan je z razredom materiala *Form*. Povezava je implementirana z metodo *addToolKeypressed* obrazca *Form*, ki doda poslušalec dogodkov (ang.: event listener) GUV HTML obrazcu. Poslušalec dogodkov je vgrajena (ang.: built-in) funkcija programskega jezika JavaScript, ki se odziva na določene dogodke (npr: pritisk tipke) in nato kliče ustrezno metodo (npr: *keypressed*) ustreznega orodja (*ToolKeypressed_CS*).

Abstraktno orodje *ToolKeypressed_CS* je orodje, ki vključuje niz pod-orodij. Naloga orodja *toolKeypressed_CS* je pridobitev podatkov o dogodku, ki ga je sprožil uporabnik, aktivirati ustrezno pod-orodje in mu delegirati del opravil.

ToolKeypressed je sestavljen iz naslednjih pod-orodij:

- *ToolEnterPressed_CS*,
- *ToolShiftEnterPressed_CS*,
- *ToolMouseDblClick_CS*,
- *ToolArrowControl*.

Razred orodje *toolKeypressed_CS* se uporablja kot zasnovno ogrodje za druga specializirana orodja, kot na primer *toolKeypressed_ChooseTable*. Na Sliki 10 je vidna delna specifikacija razreda orodja. V primeru, da specializirano orodje uporablja druga pod-orodja, se jih v specializiranem orodju definira ter tako prepiše privzeta orodja, kot definirana v orodju *toolKeypressed_CS* ali pa doda nova.

Slika 10: Orodje ToolKeypressed_CS

«orodje» **ToolKeypressed_CS**
- toolEnterKeypressed_CS : objekt
- toolShiftEnterPressed_CS : objekt
- toolArrowControl : objekt
- toolMouseDblClick_CS : objekt
- toolMouseClickPressed_CS : objekt

V razredu *ToolKeypressed_CS* atribut *toolEnterKeypressed* predstavlja primerek orodja *ToolEnterKeypressed*. Implementacija v jeziku JavaScript je naslednja:
this.toolEnterPressed = new *ToolEnterPressed()*;

Dostop do povezanih tabel z orodjem *ShiftToolEnterKeypressed*

Orodje *ShiftToolEnterKeypressed* uporabniku omogoča izvrševanje UBDB (ang.: CRUD) operacij na več tabelah preko enega obrazca s predpostavko, da so tabele med sabo povezane. Vsa polja prvotne tabele, ki se končajo s črkama ID, predstavljajo polje povezave (tuji ključ) z drugo tabelo z istim imenom kot je polje povezave, vendar brez črk ID. Take povezave so posledica normalizacije podatkovne baze in med drugim optimizirajo vnose podatkov, saj v primeru vnašanja množice računov v vnosni obrazec Vnos računa v polje *PartnerID*, vnesemo le šifro zahtevanega partnerja in ne vseh podatkov partnerja. Tabela oziroma obrazec računa namreč preko polja *Šifra partnerja* iz tabele *partner* pridobi vse podatke partnerja.

Orodje *ShiftToolEnterKeypressed* se na dogodek hkratnega pritiska tipke *shift* in *enter* v določenem polju odzove tako, da se prikaže obrazec Izberi zapis, ki prikaže seznam zapisov v tabeli s poljem, na katerem je bil izvršen dogodek. Torej, hkratni pritisk tipke *shift* in *enter* v polju *PartnerID* v vnosnem obrazcu Vnos računa bo prikazal seznam zapisov vseh partnerjev (šifrant partnerjev). Ob izboru določenega partnerja se bo šifra le-tega vpisala v polje *PartnerID* v prejšnjem obrazcu Vnos računa. V tem primeru smo uporabili obrazce na dveh nivojih: vnosni obrazec Vnos računa je prvi nivo, medtem ko je obrazec izbora Izberi partnerja drugi nivo. Število nivojev je teoretično neomejeno, saj lahko na primer na obrazcu Vnos partnerja preko polja *CountryID* preidemo na obrazec Izberi državo, ki bi pomenil tretji nivo.

Zaradi ne–trajne narave spletnih strani oziroma GUV elementov ter njihovih vrednosti se pri vračanju na nižje nivoje, kot na primer pri izboru partnerja na obrazcu Izbor partnerja ter vnovičnem prikazu obrazca Vnos računa, vrednosti GUV elementov obrazca Vnos računa, ne bi ohranile. Da bi ohranili vrednosti elementov obrazcev na nižjih nivojih, je potrebno implementirati mehanizem trajnega hranjenja podatkov oziroma obrazcev, kot je predstavljen na Sliki 11.

Orodje *ToolKeypressed_CS_MakeEntry* in *ToolEnterKeypressed_CS_MakeEntry*

Specializirano orodje *ToolKeypressed_CS_MakeEntry* je pod-orodje orodja *ToolKeypressed_CS*, saj od njega podeduje vse atribute ter funkcionalnost. Ker pa mora biti obnašanje orodja *ToolKeypressed_CS_MakeEntry* malce drugačno kot orodja *ToolKeypressed_CS*, specificiramo to specifično oziroma dodatno obnašanje v razredu orodja *ToolKeypressed_CS_MakeEntry*.

Razred orodje *ToolEnterKeypressed_CS_MakeEntry* specificira obnašanje oziroma odzivanje na dogodke pritiskov tipke *enter* in je del orodja *ToolKeypressed_CS_MakeEntry*. Povezava orodja *ToolKeypressed_CS_MakeEntry* in orodja *ToolEnterKeypressed_CS_MakeEntry* je torej je-del (ang.: is-part-of) in je implementirana kot vzorec kompozicije. V razredu orodja *ToolKeypressed_CS_MakeEntry*

atributu *toolEnterPressed* določimo referenco do primerka (ang.: instance) razreda orodja *ToolEnterKeypressed_CS_MakeEntry* ter tako prepišemo podedovano oziroma privzeto vrednost atributa, ki se nanaša na privzet razred orodja *ToolEnterPressed_CS*. Implementacija orodja in materiala, ki je povezan z določenim primerom uporabe, se nahaja v datoteki *{ime primera uporabe}_CS.js*.

JavaScript implementacija

Kot je opisano zgoraj, je povezava je-del implementirana kot atribut z vrednostjo, ki kaže na določen primerek razreda. Primerek razreda se ustvari na podlagi specifikacije razreda. JavaScript dovoljuje specifikacijo razreda znotraj specifikacije očetovskega razreda, kar pa znatno zmanjša stopnjo ponovne uporabne razredov (ang.: re-usability), zato se možnost gnezdenja (ang.: nesting) specifikacij ne bo uporabljala.

Obnašanje orodja proti obnašanju materiala obrazca

Orodja so še posebej uporabna za ponavljajoča se opravila, ki jih tudi delno ali v celoti implementirajo (Zullighoven, 2005, str. 67).

Tip materiala trajnega obrazca (npr.: *MakeEntry_CS*) ima lahko mnogo pod-razredov, katerih objekti bodo vsi implementirali enake odzive na določene aktivnosti. Primer ponavljajočih se aktivnosti je fokusiranje (ang.: focus) in od-fokusiranje (ang.: blur) GUV HTML vnosnih elementov ali preklapljanje med različnimi padajočimi seznami (ang.: combo box) GUV obrazca. Aktivnosti so skupne vsem obrazcem sistema in naj bi se izvedle kot odziv na pritiske smernikov tipka gor in tipka dol. Drug primer niza aktivnosti, ki pa so implementirane v metodi *addRecord* razreda *MakeEntry_CS*, so aktivnosti povezane z dostopom in manipulacijo GUV HTML elementov ter uporabljalo množico atributov trajnih obrazcev. Ali naj zgoraj opisane aktivnosti pripišemo orodjem ali materialu?

Odvisno. Enostavno obnašanje, ki ne zahteva velike količine trajnih podatkov primerka trajnega obrazca, naj bo implementirano v orodju. V nasprotju s tem, pa naj bo obnašanje, ki zahteva velike količine trajnih podatkov določenega trajnega obrazca, implementirano v trajnem obrazcu, orodja pa naj služijo le za sprožitev (ang.: trigger) tega obnašanja. To povzroči zmanjšanje sklopa (ang.: coupling) med orodjem in trajnim obrazcem. Orodje namreč pokliče metodo trajnega obrazca, ta metoda izvrši niz aktivnosti, ki vključujejo množico podatkov, ki so del trajnega obrazca, zatem metoda vrne rezultat orodju. V primeru, da bi bila manipulacija GUV HTML obrazca implementirana direktno v orodju, bi orodje moralo pridobiti vso množico podatkov trajnih obrazcev, kar bi povečalo sklop. Na ta način bi se kohezija oziroma povezanost spremenljivk in obnašanja znotraj trajnega obrazca tudi zmanjšala, saj bi bil del obnašanja trajnega obrazca implementiran v orodju namesto v trajnem obrazcu.

2.4.6 Delovno okolje

Delovno okolje je lokacija, kjer so orodja, materiali ter ostali objekti, ki se nanašajo na določeno opravilo, dostopni in urejeni na način, kot ga določa specifična domena. Dejansko delo oziroma izvršitev aktivnosti je izvršena na delovnem prostoru (ang.: workplace), medtem ko je delovno okolje širši pojem in vključuje dodatne lokacije, ki so dostopne znotraj delovnega prostora (Zullighoven, 2005, str. 96).

Primer UVPB

V primeru UVPB je delovno okolje kot širši pojem realizirano v obliki porazdeljenega informacijskega sistema. Sistem je porazdeljen med strežniški delovni prostor in prostor odjemalca. Delovni prostor strežnika je nadalje razdeljen na storitve strežnika podatkovne baze in storitve spletnega strežnika.

Delovni prostor odjemalca, kot dejanski hranitelj odjemalčevih orodij in materiala, je trenutno odprto okno brskalnika oziroma bolj specifično nad-okvir (ang.: *frameset*) v oknu.

Strežniški delovni prostor storitev spletnega strežnika, poimenovan httpd storitev, omogoča zaščiten primerek zbiralnika podatkov oziroma objektov, poimenovanega *Session*. Zbiralnik *Session* je na strežniškem sistemu implementiran kot tip podatkovnega polja (ang.: array data type) in je povezan le z enim sistemom odjemalca, s čimer je zagotovljena zasebnost podatkov.

Delovni prostor strežnika podatkovne baze je trajna storitev (ang.: persistent service) razpoložljivih zbiralnikov deljenih podatkov (ang.: shared data) oziroma podatkovne baze, tabele in polj. Zasebnost tega delovnega prostora se zagotavlja z naprednim sistemom avtorizacije uporabnikov.

Vnašanje, izbiranje, pregledovanje zapisov tabele se izvaja v nad-okvirju, poimenovanim startCWP, znotraj spletne strani odjemalca, kar pomeni, da nad-okvir predstavlja delovni prostor. Tako delovni prostor odjemalca kot objekti odjemalca imajo posreden dostop do strežniških objektov ter relacijske podatkovne baze, kar razširja delovni prostor odjemalca na celotno delovno okolje, ki vključuje objekte tako na strani odjemalca kot na strani strežnika. Nad-okvir na strani odjemalca ter objekti na strani strežnika so zaščiteni z geslom, ki je lastno določenemu uporabniku.

Informacijski sistem temelji na spletu, kar pomeni, da podpira hkratno delo več uporabnikov, vendar to ne pomeni, da so vsi ustvarjeni objekti deljeni (ang.: shared). Za vsakega uporabnika se na odjemalcu ustvari njemu lasten nad-okvir. Razredi, na podlagi katerih se ustvarijo objekti odjemalca ter strežnika, so deljeni in skupni vsem ustvarjenim

objektom, kar pomeni, da vsi uporabniki uporabljajo isti informacijski sistem, vsak uporabnik z njemu lastnimi domenskimi vrednostmi. Delovno mesto podatkovne baze kot jedro deljenih objektov skupaj s svojim varnostnim sistemom omogoča deljene podatkovne strukture s poudarkom na deljenih podatkih.

Tip delovnih mest. Tip delovnega mesta je abstrakcija delovnega mesta v resničnem svetu. Določajo ga naslednji dejavniki:

- Število fleksibilnih ter ponavljajočih se aktivnosti, ki so povezane z delovnim mestom.
- Tip in količina opreme, ki je uporabljena na delovnem mestu.
- Obseg domenskega znanja in sposobnosti delavcev na delovnem mestu.
- Obseg znanja informacijske tehnologije delavcev na delovnem mestu.

Primer UVPB

Tip delovnega mesta UVPB je funkcionalno delovno mesto. Prepoznamo ga lahko po naslednjih značilnosti (Zullighoven, 2005, str. 75):

- Obseg situacijske fleksibilnosti (ang.: situational flexibility) in aktivnosti, ki se ponavljajo: nizka fleksibilnost; majhno število ponavljajočih se opravil.
- Tip in količina opreme, ki je uporabljena na delovnem mestu: majhno število materialov; majhno število optimiziranih in specializiranih orodij.
- Obseg zahtevanih domenskih veščin: srednje do mnogo; ni potrebno, da sistem podpira domensko usmerjanje uporabnika.
- Obseg veščin informacijske tehnologije: malo do srednje; značilnosti ter funkcionalnost sistema naj bi bila intuitivna.

2.4.7 Zbiralnik

Zbiralnik

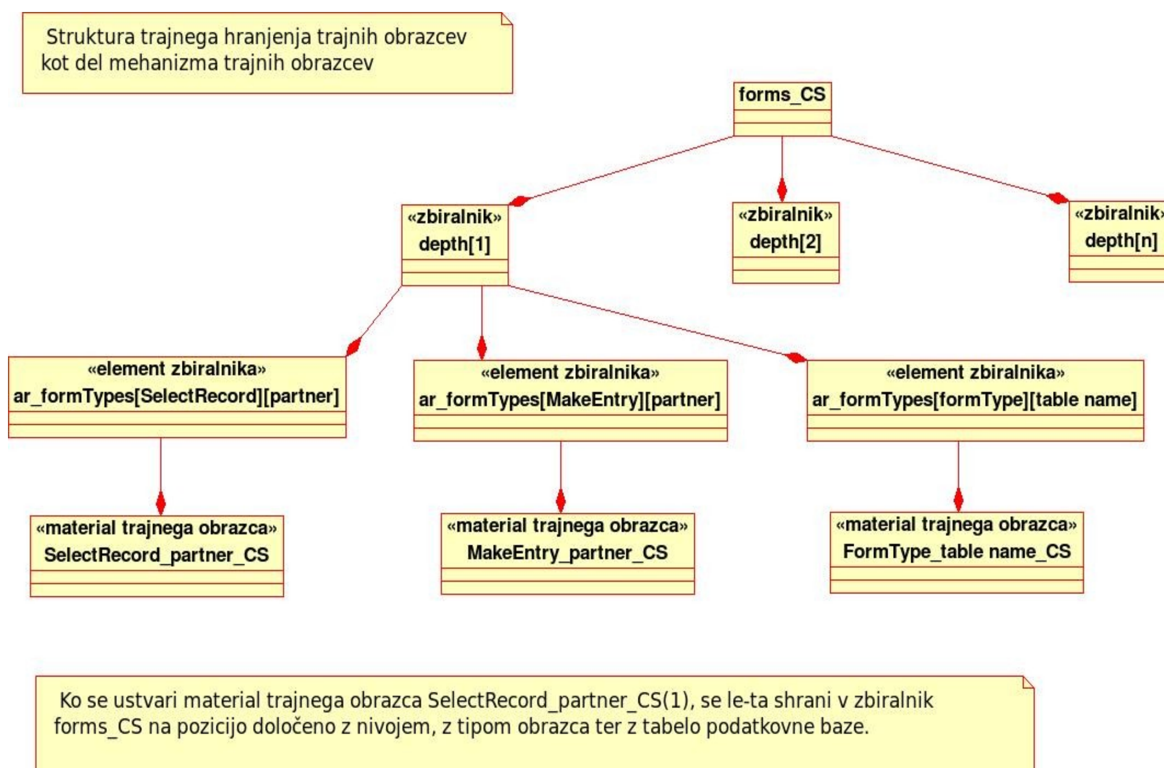
Nekateri razredi trajnih obrazcev kot je na primer obrazec *MakeEntry_CS*, imajo množico primerkov, in sicer za vsak nivo vnašanja po enega in znotraj vsakega nivoja en primerek za vsako tabelo, v katero vnašamo podatke. Koncept nivojev je opisan v poglavju 3.6.4. Medtem ko uporabnik vnaša podatke v trajne obrazce različnih nivojev, naj bi sistem ohranil domenske vrednosti oziroma vnesene vrednosti polj vsakega obrazca, zato se morajo obrazci shraniti v zbiralnike, ki so organizirani v ustrezno strukturo, prikazano na Sliki 11.

Zbiralnik lahko vsebuje, upravlja in ponuja dostop do materialov. Je zelo primeren za shranjevanje stvari oziroma objektov, ki jih trenutno ne uporabljamo, vendar vemo, da jih bomo potrebovali kasneje (Zullighoven, 2005, str. 72).

Primer UVPB

Zbiralnike, implementirane kot polja (ang.: array), bomo uporabili za hranjenje primerkov trajnih obrazcev. Zbiralniki bodo skupaj z metodami dostopa do njihovih vsebin tvorili mehanizem trajnih obrazcev. Vsak trajni obrazec se shrani v množico zbiralnikov, ki so organizirani v drevesno strukturo in realizirani z vzorcem kompozicije, ki je opisan v poglavju 3.6.4.

Slika 11: Struktura trajnega hranjenja trajnih obrazcev kot del mehanizma trajnih obrazcev



Primerki trajnih obrazcev *ChooseDB_CS*, *ChooseTable_CS*, *MakeEntry_CS*, *SelectRecord_CS* in *SearchRecord_CS* imajo trajno življenjsko dobo in se nahajajo v naslednjih zbiralnikih:

- primerki *ChooseDB_CS* -> zbiralnik *ChooseDB*,
- primerki *ChooseTable_CS* -> zbiralnik *ChooseTable*,
- primerki *MakeEntry_CS* -> zbiralnik *MakeEntry*,
- primerki *SelectRecord_CS* -> zbiralnik *SelectRecord*,
- primerki *SearchRecord_CS* -> zbiralnik *SearchRecord*.

Primerki zgoraj naštetih razredov so implementirani kot elementi podatkovnega tipa polja (ang.: array), kjer vsak element predstavlja primerek razreda določenega tipa obrazca ter določenega nivoja.

2.5 Testiranje

Testiranje programske opreme oziroma že implementiranega informacijskega sistema lahko glede vrste testov in njihovih kriterijev razdelimo na dva sklopa:

- notranja učinkovitost (ang.: internal efficiency),
- zunanja uspešnost (ang.: external effectiveness).

Notranja učinkovitost obsega optimizirano, standardizirano in dobro dokumentirano programsko kodo, medtem ko zunanja uspešnost podaja in ocenjuje kriterije o stopnji usklajenosti programske rešitve z uporabniškimi zahtevami. Kot omenjeno v uvodu poglavja 2.4, je pristop Orodja in materiala aplikacijsko usmerjen, kar pomeni, da je za tak pristop bolj pomembna zunanja uspešnost. Pri tolikšnem številu neuspešnih projektov razvoja informacijskih sistemov, katerih vzrok neuspeha je nezadostna koristnost oziroma nezadostna zadovoljitev uporabniških zahtev, velja nameniti dodatno pozornost doseganju kriterijev zunanje uspešnosti. V primeru neuporabnosti razvite programske rešitve lahko tako rešitev imenujemo polična programska oprema (ang.: shelfware), saj lahko medije (zgoščenke, usb ključe itd), ki vsebujejo tako rešitev, po zaključku razvoja postavimo kar na polico (Chapman, 2006, str. 277).

Testiranje razvite programske rešitve zahteva velik delež sredstev celotnega proračuna projekta ter ponavljajoče aktivnosti testiranja ob vsaki novi različici oziroma iteraciji. Ena izmed možnosti zmanjšanja stroškov testiranja in hkratnega povečanja zanesljivosti sistema je avtomatizacija testov. Avtomatizacija testov pomeni, da množico testov implementiramo kot programsko kodo.

Avtomatizacija testov lahko znatno poveča zanesljivost sistema, kar je lahko zelo koristno pri pogostih preurejanjih (ang.: re-factoring) programske kode in kratkih iteracijskih ciklih (ang.: iteration cycle). Problem avtomatizacije testov je povečanje števila vrstic programske kode, več porabljenega časa za njihovo pisanje ter več časa oziroma stroškov za njihovo vzdrževanje. Vendar so pri sistemih, kjer je zanesljivost delovanja ena izmed pomembnejših zahtev, ti dodatni stroški vzdrževanja praviloma bistveno manjši od stroškov, ki nastanejo zaradi neodkritih napak pri spreminjanja programske kode. Vsakokrat, ko se spremeni funkcionalnost sistema, se mora spremeniti tudi programska koda testov. Spremenjena programska koda za izboljšanje notranje učinkovitosti ne potrebuje spreminjanja programske kode testov, saj izhodni podatki testov na podlagi določenih vhodnih podatkih ostanejo nespremenjeni. Programska koda testov pogosto zahteva celo več vrstic programske kode kot koda, ki jo testiramo, saj mora programer napisati teste za mnogo različnih kombinacij vhodnih podatkov in pri vsaki različni možnosti vhodnega podatka določiti pravilen izhodni podatek. V primeru neustrezno zbranih uporabniških zahtev se pri avtomatizaciji testov strošek te napake poveča, saj so bila, poleg sredstev za pisanje programske kode, vsa sredstva porabljena za avtomatizacijo testov zaman. Orodja za samodejno generacijo programske kode za testiranje je mogoče kupiti, vendar so praviloma draga.

3 Glavni uporabljeni koncepti

3.1 Objektno usmerjen pristop

Objektno usmerjen pristop je največji dosežek razvoja programskih sistemov v obdobju devetdesetih. Pristop je spremenil način snovanja sistemov ter način njihove komunikacije. Poleg tega je spremenil tudi način snovanja, upravljanja in prenove poslovnih procesov (Lee & Tepfenhart, 1997, str. IV).

Uspešen razvoj ali prenova poljubnega sistema zahteva izgradnjo modela, ki bo zajemal vse potrebne aktivnosti, podatke ter poslovna pravila informacijskega sistema. Omejitve vsakega projekta razvoja ali prenove lahko razdelimo na tri dele: proračun, čas in kvaliteta.

Slika 12: Omejitve projekta

Strošek vzdrževanja in strošek prihodnjih izboljšav informacijskega sistema je določen s kriteriji kakovosti, ki naj bi se upoštevali pri razvoju sistema. Kriteriji kakovosti so: zanesljivost, fleksibilnost in enostavno vzdrževanje. Za doseganje ustrezne višine kriterijev so potrebne veščine in tehnike, ki se nanašajo na obvladovanje kompleksnosti sistemov. Objektno usmerjen pristop je odličen način obvladovanja kompleksnosti. Prednosti objektno usmerjenega pristopa v primerjavi z ne-objektnim so (Lea & Tepfenhart, 1997, str. 4):

- Hitro razumljiv model razredov in objektov pomeni hitrejši in cenejši razvoj, prenovo in vzdrževanje informacijskih sistemov.
- Možnost ponovne uporabe konceptualnih, zasnovnih ter programskih vzorcev povečuje vrednost razvitega sistema.
- Lokalizacija potencialnih sprememb sistema povečuje zanesljivost ter robustnost sistema.
- Pristop zagotavlja načine obvladovanja kompleksnosti, ki jih potrebuje projektni management razvoja sistemov.

Kot je omenjeno zgoraj, je ena izmed prednosti objektno usmerjenega pristopa ponovna uporaba vzorcev, vendar je tukaj potrebno opozoriti na omejitve ponovne uporabe. Pri določenem obsegu že razvitih vzorcev se ekipa razvijalcev sooči z obvladovanjem te množice vzorcev (v obliki razredov, objektov, programske kode). Porabljen čas iskanja že

razvitih vzorcev je lahko v primeru neobvladovanja množice vzorcev daljši od ponovnega razvoja vzorca.

Pri razvoju sistema sta poleg že omenjenih zelo pomembna kriterija kakovosti: kohezija in sklop.

Kohezija je notranja povezanost med atributi oziroma lastnostmi zasnovnih ali konstrukcijskih enot. Princip maksimalne kohezije zahteva močno povezavo med zasnovnimi ali konstrukcijskimi enotami (Zullighoven, 2005, str. 37). Ko razvijalec identificira določen atribut, mora pravilno ugotoviti, kateri enoti le-ta najbolj pripada. V primeru, da razvijalec kreira modele, kjer atributi niso ustrezno porazdeljeni po enotah, se kohezija objekta in s tem tudi sistema zmanjša.

Sklop je povezava ali odvisnost med različnimi zasnovnimi ali konstrukcijskimi enotami. Minimalen sklop enot pomeni čim manjšo povezanost enot (Zullghoven, 2005, str. 37). Stopnja sklopa je še posebno pomembna pri porazdeljenih informacijskih sistemih, kjer je hitrost povezave med strežnikom in odjemalcem znatno manjša kot pri neporazdeljenih sistemih. Posledično je potrebno sklop strežnika in odjemalca zmanjšati na minimum, kar z drugimi besedami pomeni, da je potrebno omejiti število klicev med njima na najmanjšo možno količino.

3.1.1 Razvoj objektno usmerjenega pristopa

Če upoštevamo današnje standarde, so bile aplikacije razvite v prejšnjih desetletjih majhne in enostavne. V 70-ih 20.stoletja sta Al Constantine in Ed Yourdon začela uporabljati funkcije kot osnovno konstrukcijsko enoto aplikacije. To je ena izmed poglavitnih značilnosti strukturne analize sistema. Nadzor dela oziroma aktivnosti aplikacije je postal enostavnejši, kar je bilo nujno potrebno zaradi povečanega števila podatkov ter spremenljivk, kar je posledično povečevalo stroške vzdrževanja ter zmanjševalo razumljivost programske kode (Lee & Tepfen Hart, 1997, str. 4).

Obseg ter kompleksnost programske kode večine aplikacij sta naraščala, s tem pa je naraščala tudi potreba po večji strukturiranosti kode. Individualni programerji so se začeli združevati v ekipe, strukturna analiza pa je služila kot metoda za lažjo koordinacijo, nadzor in analizo procesa razvoja sistema. Razlog združevanj in uporabe strukturne analize ni bila večja sinergija, temveč zgolj odgovor na povečano kompleksnost sistemov ter s tem večjo potrebo po obvladovanju kompleksnosti.

Znanstvene aplikacije uporabljajo relativno statične in nespremenljive funkcije, zato so zelo primerne za strukturno analizo in lahko znatno zmanjšajo stroške vzdrževanja. Poslovne organizacije pa se soočajo z bolj dinamičnim okoljem in morajo pogosteje prilagajati svoje funkcije. Po drugi strani so podatki poslovnih organizacij bolj stabilni, kar je povzročilo razvoj entitetno relacijskega modela (ang.: entity-relation model). Razvil ga

je Peter Chan v 80-ih prejšnjega stoletja. Entitetno relacijski model grafično prikazuje podatkovno strukturo sistema, vendar ne prikazuje množice poslovnih pravil implementiranih v funkcijah. Pri Chan-nu je osnovna konstrukcijska enota postala entiteta, ki je določena z unikatnim identifikacijskim ključem in njegovimi odvisnimi atributi.

Trenutna paradigma izgradnje programskih sistemov je objektno usmerjen pristop, ki je uporabljen tudi v tem magistrskem delu. Pristop združuje tako funkcijski pogled kot podatkovni pogled informacijskega sistema. Osnovna konstrukcijska enota je postala razred, ki vsebuje podatke ter deklaracije funkcij, objekt pa je postal primerek konkretne realizacije enote razreda.

3.1.2 Objektni meta model pristopa Orodje in material

Najpomembnejši elementi sistema so razredi in objekti povezani s povezavo uporablja (ang.: use relationship) in dedovanjem (Zullighoven, 2005, str. 18). Pri modeliranju resničnega sveta lahko uporabimo pojme, ki so prikazani v spodnji tabeli:

Povezave resničnega sveta z objektnim meta modelom	
Domenski model	Tehnični model
Stvar	Objekt
Interakcija	Operacija
Koncept	Razred
Generalizacija, specializacija	Dedovanje
Kompozicija	Agregacija, povezava (ang.: Aggregation, Association)
Pod in nad koncept (ang.: sub and super concept)	Hierarhija razredov

Pomembno pravilo domenskega modeliranja je identificiranje domensko specifičnih generičnih izrazov, njihovo podrobnejše specificiranje in njihovo nadaljnje strukturiranje z uporabo dedovanja ali povezave uporablja v modele z nižjo stopnjo abstrakcije (Zullighoven, 2005, str. 18).

Primer UVPB. V domeni aplikacije lahko identificiramo naslednje generične izraze:

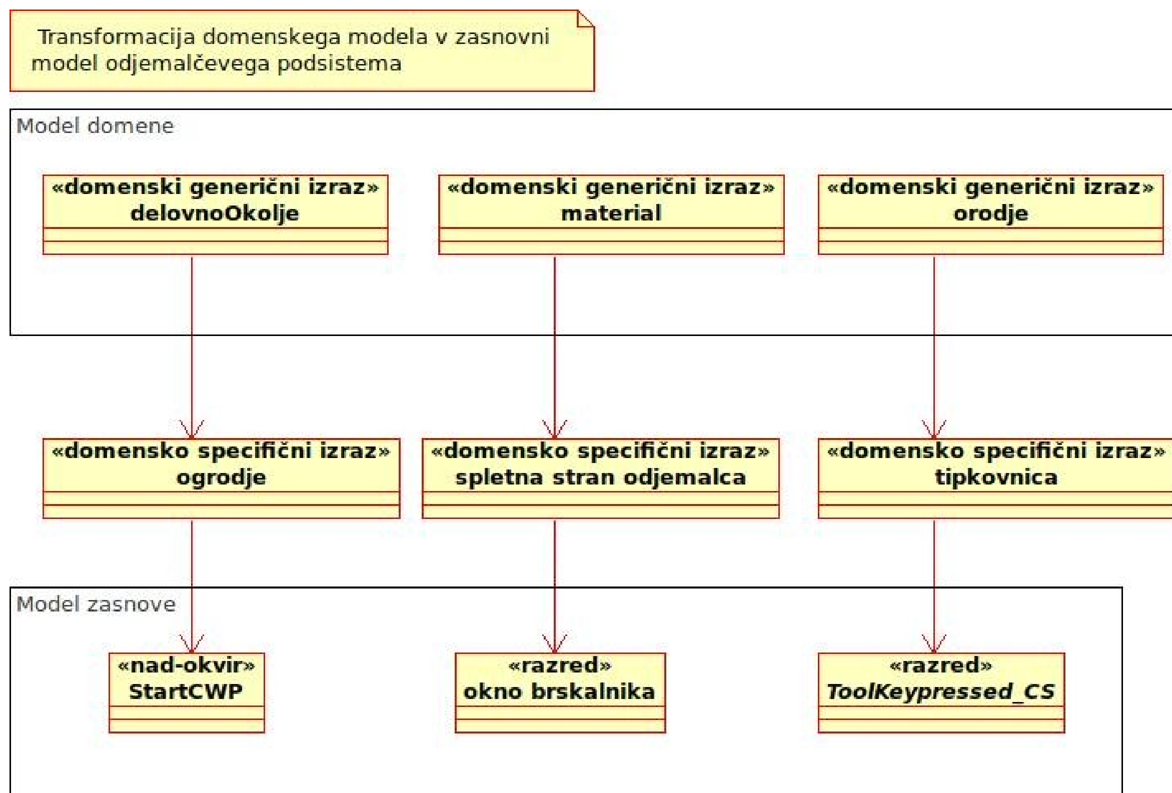
- delovno okolje,
- orodje,
- material.

Nato poskušamo ustvariti povezavo med temi generičnimi izrazi in domensko specifičnimi. Zaradi narave porazdeljenih sistemov ter relativno velike neodvisnosti med njihovimi

podsistemi, bo analiza omejena na zgolj en podsistem. Domensko specifični izrazi v podsistemu odjemalca so naslednji:

- nad-okvir,
- spletna stran,
- tipkovnica.

Slika 13: Transformacija domenskega modela v zasnovni model odjemalčevega podsistema



Slika 13 prikazuje transformacijo oziroma prehod izrazov iz domenskega modela, ki je najbližji resničnemu svetu, do modela zasnove, katerega izrazi odslikavajo specifične razrede sistema.

3.1.3 Razred

Razredi modelirajo koncepte, objekte ter njihove elemente. So abstrakcija objektov s podobnim obnašanjem. V programskem modelu je razred del programskega besedila, ki opisuje poglavitne atribute ter vmesnike (ang.: interface) objektov, ki jih lahko kreira. Razred je definiran s svojim imenom, z nadrazredi (ang.: superclass), z atributi in vmesniki (Zullighoven, 2005, str. 27). Za razliko od tipa (ang.: type) je razred tudi implementacija objekta ali vmesnika in vsebuje izvršljivo programsko kodo (ang.: executable code).

Prevajalnik (ang.: compiler) in izvršilni sistem (ang.: runtime system) zagotavljata pravilnost vseh kreiranih primerkov razredov, ki temeljijo na popolnem opisu razredov.

Vmesnike razredov klasificiramo po sledečih kategorijah (Zullighoven, 2005, str. 26):

- javni vmesniki (ang.: public interfaces),
- notranji vmesniki (ang.: internal interfaces).
  - Podedovani vmesniki (ang.: inheritance interfaces),
  - privatni vmesniki (ang.: private interfaces).

Drugi način klasifikacije vmesnikov razredov je (Zullighoven, 2005, str. 25):

Klasifikacija vmesnikov razredov	
Domensko specifični pogled	Programsko specifični pogled
Navodilo (ang.: instruction)	Procedura
Zahteva (ang.: request)	Funkcija
Test	Predikat (ang.: predicate)

JavaScript primer

Javni vmesniki so operacije, dostopne izven območja objekta. S programskim jezikom JavaScript to dostopnost dosežemo z naslednjim načinom:

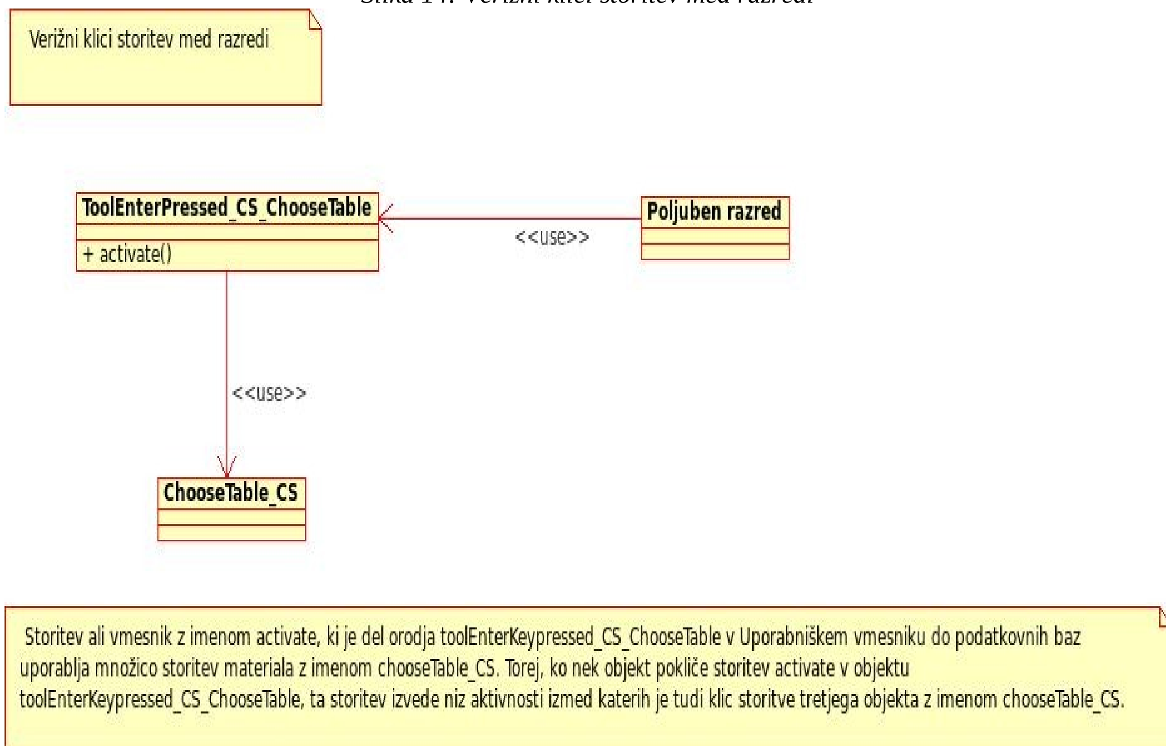
- vmesnik gnezdimo znotraj razreda, ki se definira s konstrukcijsko funkcijo,
- vmesnik definiramo z uporabo rezervirane besede „this“.

Notranji vmesnik se v JavaScriptu kreira enako kot zgoraj opisani javni vmesnik vendar brez uporabe rezervirane besede „*this*“.

Storitve so vse javne operacije določenega objekta. Z drugimi besedami, vmesniki objekta definirajo njegove storitve. Storitve objekta so pogosto realizirane na način, da storitve določenega objekta uporabljajo tudi storitve drugih objektov (Zullighoven, 2005, str. 24).

Primer UVPB

Slika 14: Verižni klici storitev med razredi



Storitev ali vmesnik z imenom *activate*, ki je del orodja *toolEnterKeypressed_CS_ChooseTable* v UVPB, uporablja množico storitev materiala z imenom *chooseTable_CS*. Torej, ko nek objekt pokliče storitev *activate* v objektu *toolEnterKeypressed_CS_ChooseTable*, ta storitev izvede niz aktivnosti, izmed katerih je tudi klic storitve tretjega objekta z imenom *chooseTable_CS*.

Podoben primer je, ko klic storitve z imenom *addRecord* v orodju *chooseTable_CS* izvede tudi aktivnost, ko klicana storitev zahteva novo storitev objekta GUV materiala z imenom *submit*.

Generične operacije. Par operacij beri in piši, ki se nanašata na atribute, imenujemo generične operacije. Implementacija teh operacij kot javnih vmesnikov delno krši objektno usmerjen princip skrivanja informacij (ang.: information hidding), zato se naj razvijalec izogiba takšni implementaciji (Zullighoven, 2005, str. 26).

3.1.4 Življenjski cikel objekta

Operacije, ki določajo in omejujejo življenjski cikel objekta, se smatrajo za kritične pri modeliranju tehničnega oziroma programskega modela. Te operacije ne morejo biti del objekta ampak so del njegovega okolja. Te operacije so:

- kreiranje,
- brisanje,
- transformacija v druge objekte.

Primer UVPB

Omenili in analizirali bomo kritične situacije naslednjih različnih skupin objektov:

- **Objekti HTML obrazcev (implementirani s HTML zastavico FORM)**. Kreirajo se ob vsaki naložitvi spletne strani, izbrišejo oziroma uničijo pa se ob vsaki sprostitvi (ang.: unloading) spletne strani. Sprostitev spletne strani se zgodi ob zaprtju okna ali tik pred naložitvijo nove spletne strani.
- **JavaScript materiali trajnih obrazcev**. Vsak objekt HTML obrazca je povezan in nadzorovan z enim materialom trajnega obrazca. To pomeni, da trajni obrazec ne sme imeti enake življenjske dobe kot objekt HTML obrazca in se ne sme izbrisati ob izbrisu HTML obrazca. Trajnost obrazca v okolju brskalnika odjemalca je omogočena z vzorcem nad-okvirja (ang.: frameset), implementiranega s HTML zastavico FRAMESET. Z naložitvami in sprostitvami spletnih strani ali obrazcev znotraj dela okvirja (ang.: frame) podatki, ki so prisotni v nad-okvirju (ang.: frameset), ostanejo trajni.

3.1.5 Ograjevanje (ang.: encapsulation)

Ograjevanje lahko prevedemo v princip skrivanja informacij, kar pomeni, da struktura objekta ni dostopna odjemalcem objekta. Odločitve glede zasnove znotraj objekta so tako (ang.: design decisions) lokalizirane, zatorej sme razvijalec poljubno spreminjati implementacijo objekta, ne da bi s tem izgubil konsistentnost z odjemalčeve perspektive, dokler ne spreminja imena oziroma podpisa operacij ali vmesnikov (Zullighoven, 2005, str. 24).

Zaradi pogosto velike medsebojne odvisnosti atributov objekta, bi bilo neprimerno neposredno spreminjati atribute ali podatke le-tega. Pametneje je spreminjanje atributov omejiti na določene točke oziroma na le tiste operacije, ki so javne. Tako se da doseči lažjo kontrolo nad objekti ter lažje vzdrževanje našega sistema.

3.1.6 Tip

Tip predpisuje niz vrednosti ter množico dovoljenih operacij nad vrednostmi. Specificira obnašanje objektov kot sintakso njihovih vmesnikov, kar pomeni, da poimenuje njihove operacije, ki se jih lahko kliče. Tip je definiran kot abstraktni podatkovni tip. Tip ne more imeti primerkov in ne vsebuje podatka, kakšno naj bo stanje operacije ter kako naj bo operacija implementirana (Zullighoven, 2005, str. 41).

Namen uvedbe tipov kot del objektno usmerjenega pristopa je povečati razumljivost programskih sistemov (ang.: software system). Tipi predstavljajo abstrakcijo razredov. Brez možnosti dedovanja je lahko objekt le enega tipa, z dedovanjem pa lahko objekt pripada množici tipov, saj podeduje vmesnike vseh določenih tipov. Tip lahko vključuje mehanizme za preverjanje objektov, ki so z njim povezani. V tipu je lahko implementiran pogodbeni model (ang.: contract model) z mehanizmom lovljenja napak ter množico trditev, katere se nato preverja ali so pravilne ali ne. Omeniti velja, da so jeziki, ki omogočajo prenos aksiomatske semantike abstraktnih podatkovnih tipov h kreiranim objektom, redki.

Primarno je tip koncept specifikacije, ki določa eksterni pogled deklariranih identifikatorjev (ang.: identifiers) in objektov. Tipi določajo in zagotavljajo pravilno strukturo aplikacije. Razlike med tipom in razredov so (Zullighoven, 2005, str. 42):

- Medtem ko tip definira le vmesnike, razred definira tudi implementirano interno podatkovno strukturo in implementirane operacije.
- Tip objekta se prvotno navezuje le na njegove vmesnike.
- Primerki tipov so razredi, ki implementirajo vmesnike in potencialno določajo dodatno obnašanje oziroma razširijo vmesnik.
- En objekt ima lahko mnogo tipov, objekti različnih razredov so tudi lahko istega tipa.
- Del vmesnika objekta je lahko določen z enim tipom, drugi deli vmesnika pa z drugimi tipi.
- Pogoj za pripadnost objektov istemu tipu je nezadosten, če imata dva različna objekta enak določen del vmesnika.

Primer UVPB

V uporabniškem vmesniku identificiramo naslednje dva tipa, ki tvorita najvišji abstrakcijski nivo strani sistema odjemalca:

- Form (slo: obrazec),
- toolKeypressed (slo: orodje pritiska tipke).

Oba tipa sta abstrakcijska razreda in nimata primerkov. Tip Form je osnova za množico razredov kot so ChooseDB_CS, ChooseTable_CS itd. Ti razredi podedujejo splošni

vmesnik, deklariran kot tip, ga občasno redefinirajo (ang.: redefine) ali dodajo določenemu obrazcu specifične operacije in atribute.

3.2 Arhitektura spletnega, objektno usmerjenega in porazdeljenega informacijskega sistema

Definicij arhitektur je mnogo, vendar se po Fowlerju lahko razdelijo v dve glavni trditvi. Prva trditev je, da je arhitektura razdelitev sistema na več delov na najvišjem abstrakcijskem nivoju, druga pa, da je arhitektura niz sprejetih odločitev oziroma pravil, ki jih je kasneje težko spreminjati (Fowler, 2003). Arhitektura je tudi skupno razumevanje razvijalcev, kakšna je zasnova sistema, kar pomeni, da je arhitektura določenega sistema subjektivne narave in je lahko drugačna v primeru, ko jo določa druga skupina razvijalcev. Conallen definira arhitekturo aplikacije kot najvišji abstrakcijski pogled najbolj pomembnih komponent sistema, ki jih ni več mogoče grupirati (Conallen, 1999, str. 97).

Kvaliteta programske opreme je določena in merjena z množico parametrov in karakteristikami. Porazdeljeni sistemi imajo lahko v primerjavi z neporazdeljenimi pozitivne ali negativne vplive na različne parametre. Spodaj so razloženi nekateri vplivi na posamezne parametre.

Latenca (ang.: latency). Je minimalen čas, ki preteče od uporabniške zahteve do kakršnegakoli odziva sistema, od katerega odziv pričakujemo (Fowler, 2003). Pri tem ni pomembno ali sistem razume uporabniško zahtevo ali ne. V primeru, da je uporabniška zahteva nesmiselna, mora sistem o tem obvestiti uporabnika. Eden izmed primerov je, ko uporabnik preko spletnega brskalnika zahteva določeno spletno stran, vendar preteče na primer 10 sekund, preden dobi od strežnika odziv, da stran ni dosegljiva. V tem primeru je latentnost sistema 10 sekund.

Nadgradljivost (ang.: scalability). Je mera za vpliv dodajanja informacijskih sredstev (ang.: resources) k sistemu na zmogljivost sistema (Fowler, 2003). Višja stopnja nadgradljivosti pomeni, da sistem lažje izvede dodaten obseg dela, pri čemer ni pomembno ali je potrebno dodajati dodatna informacijska sredstva ali ne. V primeru dodajanja informacijskih sredstev je dodajanje modernejše in učinkovitejše strojne opreme običajno stroškovno bolj učinkovito kot dodajanje nove programske opreme. Razlog za to je hitro nižanje relativne cene strojne opreme proti ceni človek-ura pisanja programske opreme.

Razširljivost (ang.: extensibility). Je mera težavnosti ali napora, ki ga je potrebno vložiti v implementacijo razširitve sistema. V primeru, da se razširitev nanaša na porazdeljene sisteme, je običajno potrebna implementacija dodatnih vzorcev za doseganje kriterijev zmogljivosti.

3.2.1 Porazdeljeni objekti

Arhitektura porazdeljenega sistema ima v najvišjem abstrakcijskem nivoju dva dela, prvi del je strežniški podsistem, drugi del je podsistem odjemalca. Naloga ter izziv razvijalca je v tem, da ustrezno in čimbolj optimalno razdeli podatke ter obnašanje sistema med omenjena podsistema. Taka razdelitev poveča zapletenost porazdeljenega sistema v primerjavi z neporazdeljenimi sistemi. Deljeni objekti, kot so podatkovna baza, objekti zadolženi za dostop do nje ter objekti zadolženi za zahtevnejše kalkulacije, se nahajajo na strežniškem podsistemu, medtem ko se objekti zadolženi za takojšnje kontrole podatkov in za grafične predstavitve nahajajo na odjemalčevem podsistemu.

V spletnih aplikacijah je komunikacija med podsistemoma implementirana s HTTP protokolom, ki poleg množice pravil, ki definirajo protokol, vsebuje tudi dve operaciji GET in POST. Ti dve operaciji strežniku omogočata pridobitev podatkov, vnesenih s strani uporabnika, na odjemalčevem spletnem obrazcu na podsistemu odjemalca. Spletna povezava med podsistemoma ni stalna, odzivnost pa za približno 10000krat manjša kot pri povezavah med lokalnimi objekti oziroma lokalnimi sistemi. Povezava se realizira tako, da odjemalec pošlje zahtevo strežniku, ki mora takšno zahtevo pričakovati oziroma čakati.

Znane arhitekture porazdeljenih sistemov so (Conallen, 1999, str. 46) :

- RMI - Remote Method Invocation, Java.
- CORBA - Common Object Request Broker Architecture, OMG.
- DCOM - Distributed Component Object Model, Microsoft.

Ena izmed pomembnih lastnosti zgoraj naštetih arhitektur je infrastruktura, imenovana vmesna oprema (ang.: middleware), ki implementira in poenostavlja klice med lokalnimi in oddaljenimi objekti, kar lahko znatno zmanjša zahtevnost dela razvijalca. Transparentnost, ki jo zagotavlja vmesna oprema, omogoča klice objektov sistema ne glede na to ali se sodelujoči objekti nahajajo v lokalnem ali oddaljenem sistemu (Fowler, 2003, str. 89). Kljub relativno enostavni implementaciji porazdeljenih objektov in njihovih klicev, mora razvijalec dobro ločiti med lokalnimi klici in oddaljenimi klici. V nasprotnem primeru se lahko odzivnost sistema občutno zmanjša, saj so klici oddaljenih objektov časovno mnogo bolj potratni kot klici med lokalnimi objekti.

Zaradi standardnega in razširjenega brskalnika, ki podpira HTTP protokol in niz standardnih skriptnih jezikov kot so HTML, XML, CSS, DOM, JavaScript in drugi, lahko zdaj realiziramo porazdeljene aplikacije s centralno in deljeno bazo podatkov, z nizom deljenih objektov na strani strežniška in odjemalca.

3.2.2 Oddaljeni in lokalni vmesniki

Porazdeljenost objektov vsekakor ne sme biti poljubna, ampak mora biti dobro premišljena in logična. Glavni razlog je zmogljivost oziroma hitrost izvrševanja aktivnosti določenega

sistema. Klici procedur znotraj enega procesa so zelo hitri, medtem ko so klici med različnimi procesi bistveno počasnejši. Način porazdelitve objektov vpliva na tip lokalnih in oddaljenih vmesnikov in je lahko fin (ang.: fine-grained interface) ali grob (ang.: coarse-grained interface) (Fowler, 2003, str. 89).

Fin vmesnik (ang.: fine-grained interface). Fini vmesniki se skladajo s principi objektno usmerjenega pristopa, ki narekujejo zelo podrobno razdelitev funkcionalnosti ter podatkov med objekti. Rezultat take razdelitve je množica metod z omejeno funkcionalnostjo ter mnogo kratkih klicev med objekti.

Grob vmesnik. (ang.: coarse-grained interface).

Kot omenjeno zgoraj, je močna povezanost oziroma visoka stopnja sklopa med objekti na različnih vozliščih povzroči veliko obremenitev sistemskih zmogljivosti, zato so prilagoditve v obliki novih vzorcev zasnove nujno potrebne za njihovo obvladovanje. Prilagoditve zasnove kot na primer vzorec Oddaljena fasada (ang.: Remote facade) ali vzorec Objekt podatkovnega prenosa (ang.: Data transfer object) v takih primerih pogosto pomenijo manjši odmik od strogih principov objektno usmerjenega pristopa. Da bi dosegli minimalno število klicev med objekti na različnih vozliščih, je včasih potrebno zasnovati grobe vmesnike oziroma metode, ki bodo z enim klicem pridobile več podatkov ali izvedle več opravil kot pri strogem upoštevanju objektne usmerjenosti. Tu bi na primer opravila bila bolj logično in podrobno porazdeljena med objekti, s čimer bi se posledično povečalo število povezav med njimi. Slabost zasnove grobih vmesnikov je manjša fleksibilnost in razširljivost ter hkrati povečana stopnja kompleksnosti.

Oddaljeni vmesniki so bili običajno implementirani kot oddaljeni klici globalnim metodam ali kot direktni klici metodam objektov. V zadnjih letih se veča število HTTP komunikacije, ki temeljni na jeziku XML. Objekt, ki se nahaja na določenem sistemu, pokliče metodo objekta na drugem sistemu tako, da prvi objekt pokliče določeno spletno stran oziroma vstopna točko definirano z URL naslovom, ki nato pokliče drugi objekt. Strežnik sprejme parametre klica na tri načine: napisane v besedilu URL-ja, določene znotraj polja (ang.: array) POST ali določene znotraj polja GET. Rezultat klica je v obliki besedila, ki je oblikovan v XML jeziku. Prednost XML oblike besedila je, da poleg podatkov v obliki vrednosti določa tudi pomen teh podatkov, kar odjemalcu ali prvemu objektu v našem primeru omogoča interpretacijo le-teh. Slabost HTTP komunikacije, ki temelji na XML-ju, je njena počasnost.

Če povzamemo: razvijalec naj bi zasnoval sistem na način, ki je skladen z objektno usmeritvijo in uporablja fine vmesnike, kadar se vmesniki uporabljajo znotraj objektov enega sistema. Kadar pa bodo vmesnike uporabljali objekti iz oddaljenih sistemov, naj se zasnuje grobe vmesnike. Vzorec Oddaljena fasada predvideva obstoj objekta, čigar edini namen je transformacija finih vmesnikov v grobe vmesnike, medtem ko vzorec Objekt

podatkovnega prenosa predvideva objekt, ki služi kot paket za množico različnih podatkov, ki jih je potrebno prenesti (Fowler, 2004, str. 92).

3.2.3 Strežniški podsistem

Obnašanje ali funkcionalnost strežniškega podsistema je določena z množico strežniških razredov in množico njihovih medsebojnih povezav. Metode za kreiranje objektov se pokličejo s spletnih strani, ki se naložijo na zahteve uporabnikov. Ustvarjeni objekti so lahko trajni ali ne-trajni. V primeru, da so ne-trajni, se objekti ob koncu naložitve spletne strani uničijo. V primeru, da so objekti trajni, strežnik ustvari sejo (ang.: session) za vsakega uporabnika posebej, kamor se shranjujejo objekti in kjer so na voljo za morebitne kasnejše dostope.

Primer UVPB

Modul php, ki je del strežnika Httpd, ima na voljo niz nastavitev v zvezi s strežniškimi sejami, ki omogočajo registracijo poljubnega števila spremenljivk, katerih vrednosti se ohranijo ob večkratnih zahtevah spletnih strani (PHP manual). Vsaka skripta, ki vsebuje reference do objektov, ki se nahajajo v seji, mora imeti pred deklaracijo seje („*session start*“), definirane reference datotek, ki vsebujejo specifikacije razredov od teh objektov.

3.2.4 Odjemalčev podsistem

Ko podsistem odjemalca od strežnika zahteva objekt v obliki spletne strani, strežnik najprej izvede vse metode strežniških objektov kot so določene na spletni strani. Kot že omenjeno, se objekti, ki so lastniki teh metod, lahko kreirajo ob vsaki zahtevani spletni strani znova ali pa so trajni znotraj določene seje. Po izvršitvi vseh aktivnosti, ki so določene s programsko kodo, strežnik odstrani vso programski kodo, namenjeno strežniku s spletne strani ter vrne spletno stran odjemalcu. Mogoče je tudi, da strežnik tik pred poslanim odgovorom na zahtevo odjemalca dinamično spremeni specifikacije razredov odjemalca in tako vpliva na kasnejše kreiranje objektov odjemalca.

Po prejemu spletne strani odjemalčevo delovno okolje Firefox izvrši vse metode, ki so določene na spletni strani. Te metode so definirane v specifikacijah razredov, ki so lahko fiksni ter se ob prenosu shranijo na odjemalcu ali pa se lahko prenašajo s strežnika ob vsakem prejemu spletne strani. Z namenom minimalizacije klicev oziroma količine prenesenih podatkov med podsistemom strežnika in odjemalca, lahko uporabimo dva vzorca za ohranitev specifikacije razredov na odjemalcu tudi po naložitvi naslednje spletne strani, in sicer vzorec Nad-okvira (ang.: Frameset) ter vzorec Predpomnilnik (ang.: cache).

Vzorec Nad-okvira predvideva obstoj HTML elementa FRAMESET, ki lahko vsebuje množico oken, ki lahko vsebujejo različne spletne strani. Poleg oken vzorec Nad-okvir vsebuje tudi del, kjer se lahko nahaja odjemalčeva skripta. Ker so razredi odjemalca deklarirani s pomočjo njegove skripte in smiselno razdeljeni tako, da se deklaracija vsakega razreda nahaja v ločeni datoteki, razvijalec definira reference teh datotek znotraj nad-okvira. Ob nalaganju spletnih strani nad-okvir ostaja isti, tako da tudi specifikacija razredov ostaja fiksna.

Vzorec Predpomnilnik je implementiran v samem brskalniku Firefox. V primeru, da je možnost predpomnilnika vključena, brskalnik shranjuje prejete spletne strani v dinamični spomin odjemalčevega sistema. To pomeni, da se v primeru zaprtja brskalnika, vsi objekti shranjeni v predpomnilniku, izbrišejo. Dokler pa je brskalnik odprt, je mogoč dostop do vseh specifikacij razredov, ki so se shranili v predpomnilnik ob prejetju.

3.2.5 Teorija tridelne arhitekture

Tridelna arhitektura je najbolj tipičen primer večslojne arhitekture in je največkrat uporabljena pri odjemalec-strežnik aplikacijah. Na vrhu arhitekture, najbližje uporabniku, je predstavitveni sloj, ki je odgovoren za grafični videz aplikacije, na sredini je funkcijski sloj, ki implementira poslovno logiko in pravila, na dnu pa je podatkovni sloj, ki vsebuje podatke in nudi vmesnik do podatkovne baze (Zullinghoven, 2005, str. 297).

Tridelna arhitektura je skladna s temeljnimi principi slojne arhitekture le v primeru, da lahko en sloj dostopa do elementov tega sloja in do elementov spodnjega sloja. Torej, srednji funkcijskih sloj lahko dostopa le do elementov lastnega sloja in do elementov podatkovnega sloja, ki je nivo nižje, medtem ko do elementov predstavitvenega sloja ne more dostopati. Vsak sloj tako nudi vmesnik višjemu sloju, kjer vmesnik predstavlja abstrakcijo karakteristik sloja v obliki storitvenega vmesnika (Zullighoven, 2005, str. 298).

Poleg slojne arhitekture, ki logično razdeljuje dani informacijski sistem, ne smemo zanemariti odvisnosti med sloji in elementi. Domenski in podatkovni vir nikakor ne smeta biti odvisna od predstavitvenega dela sistema. To pomeni, da objekti funkcijskega in podatkovnega sloja ne smejo klicati objektov predstavitvenega sloja (Fowler, 2003). Razlog je očiten, saj več klicev oziroma povezav pomeni močnejši sklop slojev sistema, kar povečuje verjetnost, da bo sprememba predstavitvenega sloja povzročila spremembo v podatkovnem sloju. Stabilnosti slojev ali, z drugimi besedami, pogostost sprememb zasnove in implementacije slojev se precej razlikuje, pri čemer je podatkovni sloj podvržen najmanj spremembam, predstavitveni pa je najbolj nestabilen s pogostimi spremembami. Te razlike so razlog za omejitve klicev, v zgoraj omenjeni smeri, ali objektov nižjih, bolj stabilnih slojev, objektom višjih, manj stabilnih slojev.

Primer UVPB

1. Predstavitveni sloj vsebuje vse GUV HTML elemente kot na primer obrazce, vhodna okenca (ang.: input boxes), gumbe itd. Infrastruktura brskalnika zagotavlja vmesnik (ang.: interface), imenovan DOM (ang.: document object model), za dostop in manipulacijo teh GUV elementov.
2. Funkcijski sloj na odjemalcu implementira poslovno logiko, ki se izvršuje na odjemalcu, kot je na primer kontrola podatkov (ang.: data validation), nadzor in upravljanje obrazcev itd. Z uporabo DOM vmesnika lahko ta sloj manipulira GUV elemente, ki so prisotni v predstavitvenem sloju, medtem ko lahko z uporabo HTTP protokola pridobiva ter pošilja podatke s strežnika oziroma na strežnik.
3. Funkcijski sloj na strežniku implementira poslovno logiko, ki se izvršuje na strežniku in je lahko tudi kontrola podatkov, sistemske omejitve (ang.: system constraints), kalkulacije, varnostno preverjanje (ang.: security validation), manipulacija podatkovne baze itd. Sloj zagotavlja HTTP protokol za pridobivanje ter pošiljanje podatkov od odjemalca in k odjemalcu.
4. Podatkovni sloj predstavlja dejanske podatke in nudi vmesnik za njihovo manipulacijo.

3.2.6 Model spletnih aplikacij

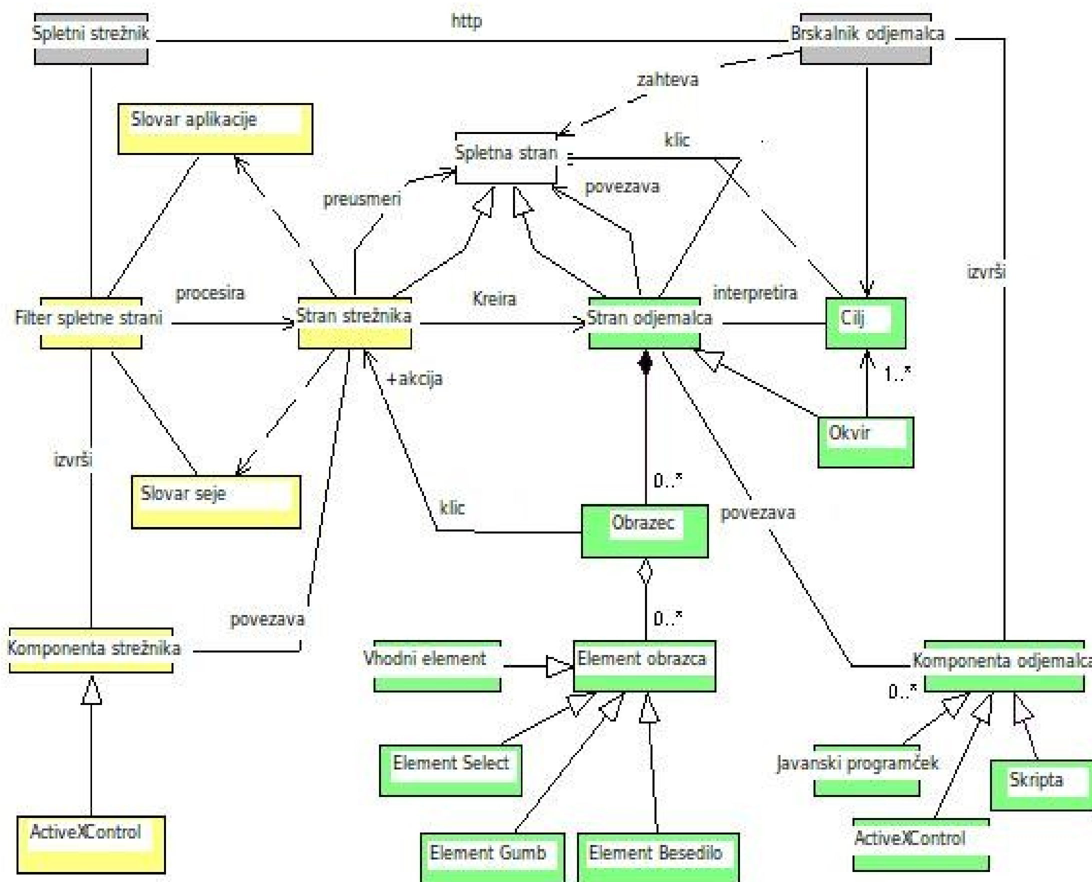
Po Conallen-u je spletna aplikacija spletna stran oziroma so spletne strani, ki imajo vpliv na poslovanje, ki ga podpirajo. Zato naj bi vsebovale poslovno logiko in ne zgolj predstavitvene vidike z navigacijo strani. Navadno te spletne strani vsebujejo navodila za izvrševanje aktivnosti na strežniku in na odjemalcu (Conallen, 1999b).

Uporabnik je v interakciji z brskalnikom, ki je aplikacija, in deluje na računalniku odjemalca. Brskalnik je odgovoren za manipulacijo različnih grafičnih elementov, najpogosteje HTML elementov. Poleg tega, brskalnik komunicira s strežnikom tako, da od njega zahteva spletne strani, jih interpretira (ang.: render) ter prikaže uporabniku. Komunikacija je diskretne narave, saj strežnik in odjemalec nista stalno povezana, ampak se povežeta ob določenih dogodkih, na primer ob zahtevi odjemalca, da bi pridobil določene spletne strani od strežnika oziroma konkretno ob vpisu naslova spletne strani v naslovno vrstico brskalnika ter pritiskom tipke *enter*. Drug, zelo uporaben način povezave, je s klicem metode *load* objekta XMLHttpRequest (AJAX), ki zahteva spletno stran brez ponovne naložitve spletne strani.

Spletni strežnik ponuja spletne strani tako, da stalno posluša oziroma čaka na zahteve poslane na strežnikov spletni naslov. Vsak spletni naslov ustreza določeni spletni strani, ki se nahaja na strežniškem datotečnem sistemu. Strežnik naloži datoteko in izvrši vsa programska navodila, ki so napisana v segmentu, katerega začetek in konec določa ustrezna ključna beseda. Bolj točno, programska navodila izvrši modul, ki je del spletnega strežnika. Medtem ko aktivnosti odjemalca ter programska navodila temeljijo bolj na

dogodkih (ang.: event-driven), so strežniška navodila v prvi vrsti proceduralne narave (Conallen, 1999b).

Slika 15: Model splošne arhitekture spletne aplikacije

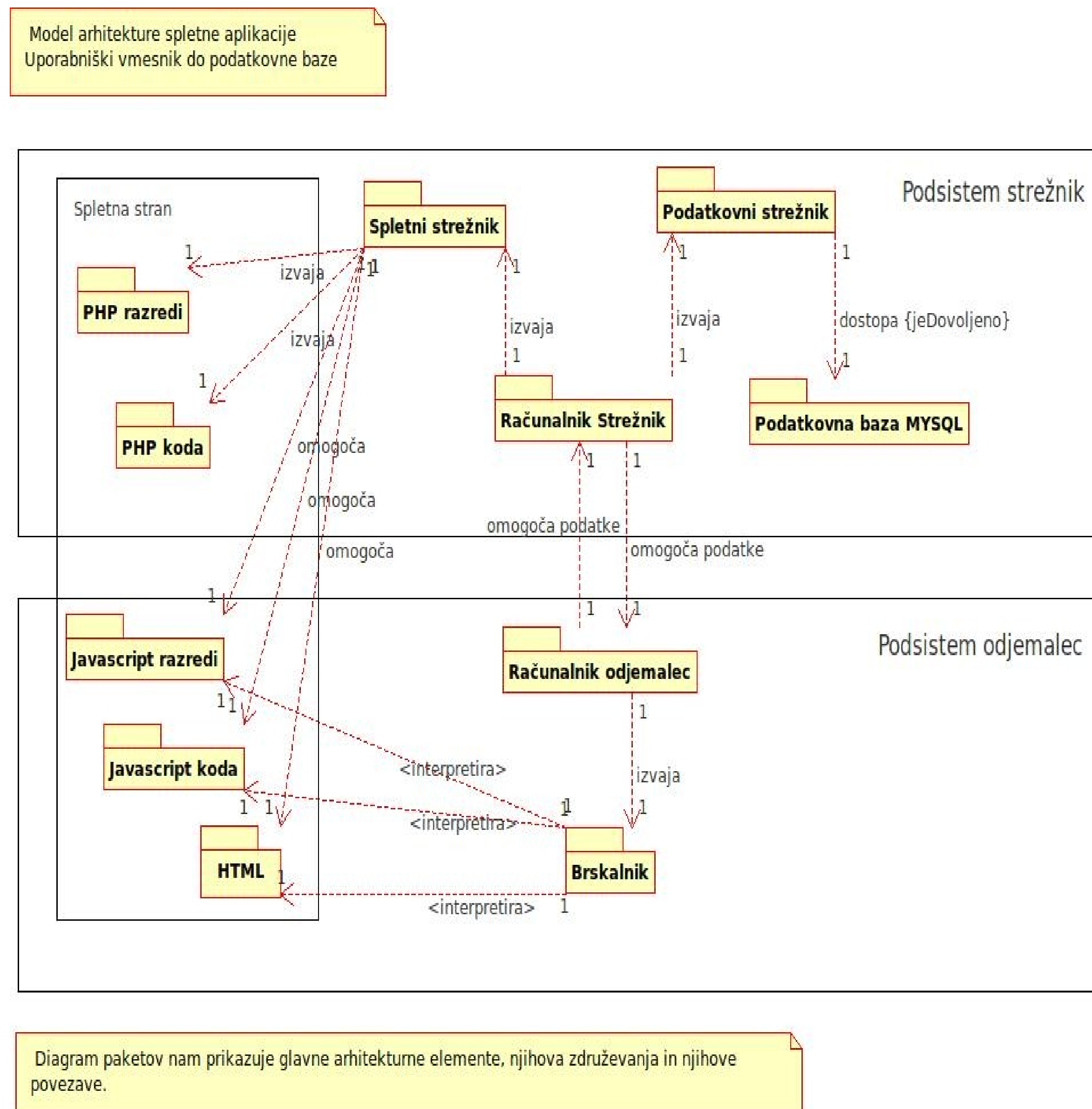


Vir: Conallen, Model of a generalized web application architecture, 1999b.

Na Sliki 15 Conallen grafično upodobi model splošne arhitekture spletne aplikacije. Glavna os arhitekture sta dva podsistema: strežnik ter odjemalec, ki je na sliki prikazan kot brskalnik odjemalca.

Primer UVPB

Slika 16: Model arhitekture spletne aplikacije UVPB



Na Sliki 16 je prikazan model arhitekture spletne aplikacije UVPB. Enako kot pri Conallenu iz Slike 15 je glavna os arhitekture relacija računalnik strežnik in računalnik odjemalec, ki si izmenjujeta podatke na podlagi HTTP protokola. Na sliki je jasno viden element spletna stran, ki vsebuje elemente oziroma pakete elementov obeh sistemov. V osnovi se spletna stran kot besedilo programskih navodil fizično v celoti nahaja na strežniškem podsistemu, ob zahtevi odjemalca pa se na nanj prenese le del spletne strani, ki je na sliki prikazan kot podsistem odjemalca. Izvajanje programskih navodil paketov, ki so

del spletne strani, poteka delno na podsistemu strežnik, delno na podsistemu odjemalec, kot je prikazano na sliki.

Programska navodila, ki se nahajajo na spletnih straneh, v primeru UVPD izvršuje skriptni izvrševalec (ang.: scripting engine) v obliki php modula, ki je del spletnega strežnika. Ključna beseda, ki določa začetek segmenta programskih navodil je <?php, končna pa ?>.

Identifikacija spletnih strani in njihovih funkcionalnosti je osnovana na podlagi primerov uporabe. Primer uporabe "ustvari zapis", na primer, določa spletno stran *MakeEntry*, ki svojo funkcionalnost implementira v strežniškem razredu *MakeEntry_SS (kratica SS - server side)* in razredu odjemalca *MakeEntry_CS*.

Slika 17: Abstraktni strežniški razred Class_wp_SS

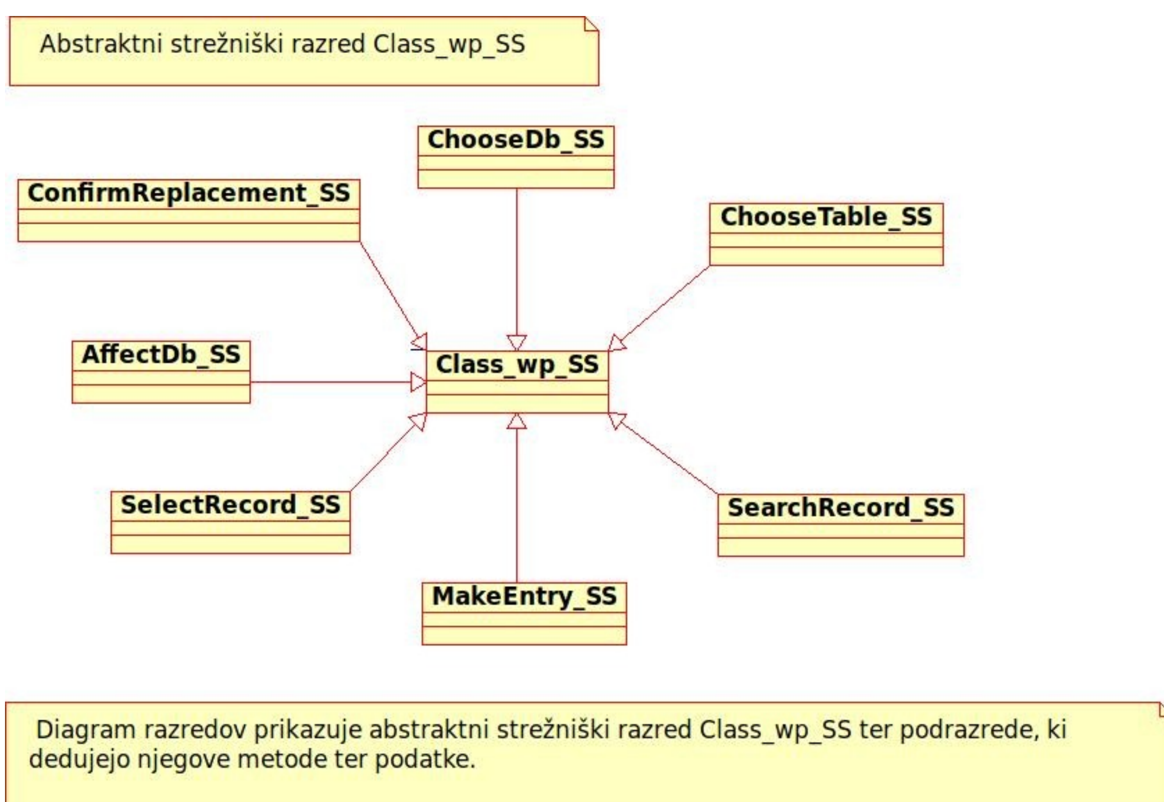
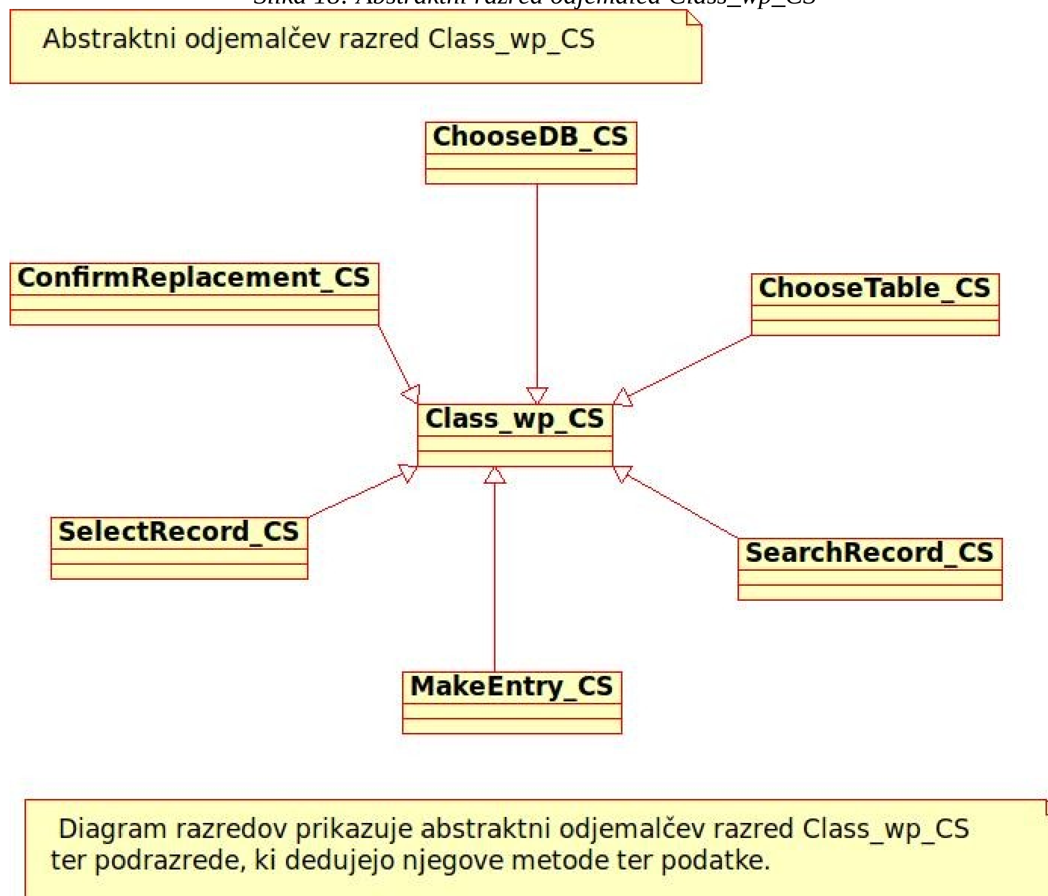


Diagram razredov na Sliki 17 prikazuje abstraktni strežniški razred *Class_wp_SS (kratica wp - web page)* ter podrazrede, ki dedujejo njegove metode ter podatke. Abstraktni razred nima svoje implementacije, temveč lahko služi le kot osnova za dedovanje drugim razredom, ki imajo svoje konkretne implementacije. Specifikacije metod so v abstraktnem razredu sicer lahko podane, vendar mora v razredu obstajati vsaj ena abstraktna metoda oziroma metoda brez specifikacije, ki bi lahko bila osnova za implementacijo. Razred

Class_wp_SS torej služi le kot predstavitev koncepta, ki je v našem primeru nek splošni razred na strežnikovi strani, ki ima to lastnost, da v sebi združuje vso specifikacijo funkcionalnosti strežniškega dela določene spletne strani. Konkretni razredi, ki vsebujejo specifikacijo določene spletne strani, so MakeEntry_SS, SelectRecord_SS in ostali kot je razvidno iz diagrama.

Koncept oziroma splošni abstraktni razred, ki določa funkcionalnost spletne strani na strani odjemalca je zasnovan enako kot pri strežniškem razredu in je prikazan na naslednji sliki:

Slika 18: Abstraktni razred odjemalca Class_wp_CS



Omeniti velja, da razred, ki implementira primer uporabe "dopolni zapis" na strani odjemalca, ne obstaja, medtem ko je na strežniku prisoten. Primer uporabe "dopolni zapis" je lahko klican od mnogih primerov uporabe kot na primer "ustvari zapis" ali "izberi zapis", zato mora biti definiran kot samostojna entiteta. Ko se pokliče primer uporabe "dopolni zapis", se celotna funkcionalnost primera uporabe izvrši na strežniku.

3.3 Ogrodje in prilagajanje spletnega, objektno usmerjenega in porazdeljenega informacijskega sistema

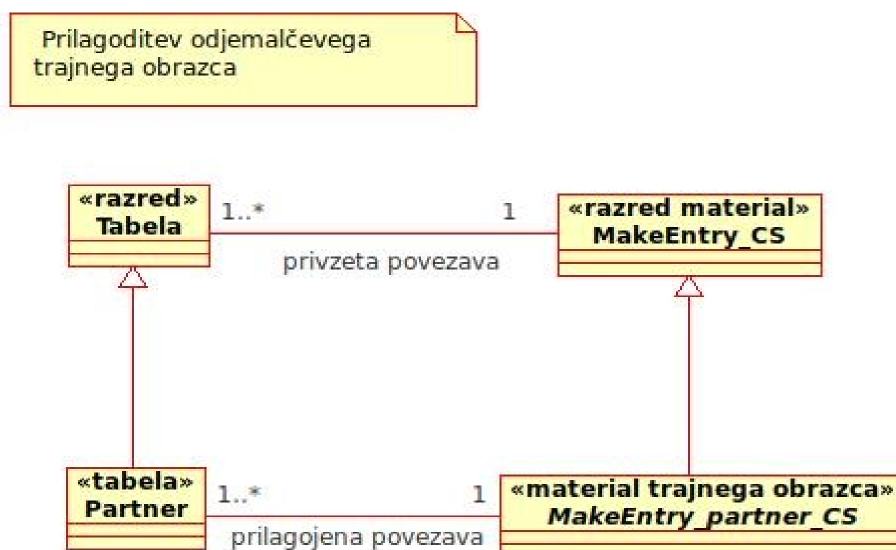
3.3.1 Ogrodje

Uporaba ogrodij pri razvoju informacijskega sistema ima velik vpliv na hitrost razvoja ter na razširljivost sistema, saj zagotavlja preverjeno osnovno arhitekturno strukturo ali ogrodje z vključeno množico mehanizmov in „kljuk“ za razširitve ter spremembe sistema brez spreminjanja osnovne infrastrukture sistema.

3.3.2 Prilagajanje trajnega obrazca odjemalca

Za vnose zapisov v tabele ogrodje uporablja enoten, privzet trajni obrazec odjemalca *MakeEntry_CS*, čeprav ima v praksi vsak trajni obrazec odjemalca, ki povzroči vnos zapisa v določeno tabelo specifično obnašanje. Najbolj tipična prilagojena funkcionalnost takega obrazca je preverjanje vnesenih podatkov in jo je potrebno implementirati praktično na vsakem obrazcu posebej. Hitrost razvoja novih obrazcev se bo povečala, če bo ogrodje sistema vključevalo mehanizem, ki bo predvideval in poenotil implementacijo zgoraj omenjenih prilagoditev obrazcev. Tak mehanizem ogrodja UVPB je implementiran v razredu *Forms_CS* z uporabo vzorca Metode tovarna in je podrobneje opisan v poglavju 3.6.3. Mehanizem predvideva specifikacijo funkcionalnosti ali, v našem primeru, specifikacijo prilagoditve določenega obrazca v razredu z določenim imenom, nato pa definiranje povezave med obrazcem vnosa zapisa v določeno tabelo ter razredom v razredu *Forms_CS*.

Slika 19: Prilagoditev trajnega obrazca odjemalca



3.4 Objektno - relacijsko preslikavanje

Kot je omenjeno v poglavju 2.3, morajo biti objektno usmerjeni aplikacijski podatki preslikani med ustrezne elemente podatkovno-relacijskega modela. Prevajanje objektne strukture podatkov v relacijsko, se v UVPB izvršuje v objektu, imenovanem *DbObject_SS*. Metoda objekta, imenovana *evaluateAndWriteToDB*, poleg drugih parametrov zahteva podatke v objektni strukturi, na podlagi teh podatkov kreira SQL stavek in ga pošlje relacijski podatkovni bazi, ki na podlagi tega stavka vstavi, prebere, dopolni ali izbriše zapis. Vsi identificirani objekti na strežniški strani, ki zahtevajo trajnost podatkov, kot je značilna za podatkovne baze, so v razvitem sistemu potomci objekta *DbObject_SS*. To pomeni, da podedujejo vse metode očetovskega objekta, tudi metodo *evaluateAndWriteToDB*.

Transformacije podatkov so, med zgoraj omenjenimi različnimi načini organiziranosti podatkov, zapletena ter pogosta opravila, zato je še kako smiselno uporabiti ustrezno ogrodje, ki vsebuje že implementirane vzorce ter mehanizme za reševanje in obvladovanje povečane stopnje kompleksnosti sistema.

3.5 Problemi trajnih podatkov v spletnem, objektno usmerjenem in porazdeljenem informacijskem sistemu

3.5.1 Stanje seje

Stanje seje je množica vseh podatkov, ki se nahajajo v določeni seji. Ena izmed lastnosti seje je relativno velika trajnost podatkov, ki se nahajajo v njej, vendar je manjša kot trajnost podatkov, shranjenih v podatkovni bazi. Začetek seje je določen ob prijavi uporabnika v sistem, zaključek seje pa je čas po preteku maksimalnega časa neaktivnosti uporabnika. Teoretično se podatki seje sicer lahko shranjujejo na trdi disk, pri čemer bi se trajnost podatkov seje in trajnost podatkov podatkovne baze izenačila, vendar trajnost podatkov ni primaren namen obstoja seje. Primerek seje naj bi se kreiral za vsakega uporabnika posebej. Čas obstoja določene seje naj bi bil omejen, določal naj bi ga maksimalen čas neaktivnosti uporabnika, preden se določena seja uniči. Na primer, če uporabnik 1 uro ne uporablja sistema se seja uniči in z njo tudi vsi podatki, ki se nahajajo v njej.

Stanje seje je lahko tudi nepravilno. V primeru, da uporabnik v določen obrazec vnese napačno vrednost, jo odjemalec pošlje strežniku, podatki pa postanejo del seje na strežniški strani. Strežnik izvede preverjanje podatkov seje, ugotovi napačno stanje seje ter vrne napako. Uporabnik sicer dobi obvestilo o napačnem stanju seje, stanje seje pa je še vedno v napačnem stanju.

Seja s sejnim stanjem. Seja s sejnim stanjem je zmožna ohranjati določeno množico podatkov med več dostopi ali zahtevami, vendar je število teh dostopov praviloma omejeno in vsekakor ne dosega nivoja trajnosti podatkov v podatkovni bazi. Strežnik s sejnim stanjem je strežnik, ki omogoča dostop do množice podatkov, ki so vezani na določenega uporabnika preko več zaporednih zahtev. Tak strežnik zahteva precejšnje pomnilniške kapacitete, ki bodo zadoščale hranjenju podatkov seje vsem uporabnikov strežniških storitev.

Seja brez sejnega stanja. Taka seja ne vsebuje nobenih uporabniških podatkov, kar pomeni, da mora strežnik brez sejnega stanja ob vsaki zahtevi odjemalca vsakokrat znova pridobiti vse podatke, ki so potrebni za izvedbo zahtevane storitve.

Primer UVPB

V primeru UVPB je na tako na strani strežniška kot na strani odjemalca uporabljena seja s sejnim stanjem.

Stran strežnika

Od veliko objektov strani strežnika se pričakuje oziroma zahteva, da ohranijo določeno trajnost svojega stanja ter s tem dostopnost podatkov objekta v določenem časovnem obdobju. Ko na primer ob prijavi uporabnik vpiše svoje uporabniško ime ter geslo, mora objekt hraniti te podatke določen čas, saj bi v nasprotnem primeru uporabnik moral ime in geslo vpisovati ob vsaki poslani zahtevi. Trajnost podatkov omogoča programska oprema oziroma infrastruktura na strežniku s pomočjo PHP polja (ang.: array), ki se imenuje SESSION. Vsi podatki shranjeni v tem zbiralniku bodo dostopni tudi ob vseh naslednjih poslanih zahtevah pod pogojem, da se ob vsaki zahtevi pošlje tudi identifikacijska številka seje (ang.: session id). Za določanje, šifriranje ter pošiljanje identifikacijske številke seje poskrbi infrastruktura odjemalca oziroma samo okolje brskalnika Firefox.

Stran odjemalca

Ko uporabnik izbere podatkovno bazo, ki jo želi uporabiti se ime podatkovne baze shrani v spremenljivko *currentDB* v trajnem odjemalčevem objektu *application_CS*. Ob dejstvu, da je trajnost ter dostopnost tega podatka na strežniški strani že omogočena, se lahko vprašamo, zakaj bi potrebovali trajnost podatka tudi na strani odjemalca. Po poslani zahtevi od odjemalca na strežnik, se podatki na odjemalcu izgubijo, medtem ko mora aplikacija ves čas prikazovati ime trenutno izbrane podatkovne baze na zaslonu ekrana. Trajnost na strani odjemalca je zagotovljena s pomočjo nad-okvira (ang.: frameset) ter zgornjega okvira (ang.: frame), ki sta oba neodvisna od procesa potrjevanja (ang.: submiting) ter ponovnega nalaganja (ang.: reloading) okvira, ki prikazuje obrazce operacij UBDB. Besedilo imena trenutne podatkovne baze prikazano v zgornjem okvirju, ki poleg imena trenutne podatkovne baze, prikazuje tudi glavni meni aplikacije, je statično, saj se tega okvirja nikoli ne potrjuje (ang.: submit) ali ponovno nalaga. Poleg statičnega besedila v zgornjem okvirju trajnost podatkov omogoča nad-okvir, v katerem se nahajajo vsi trajni objekti in trajne spremenljivke kot na primer objekt *application_CS*, ki poleg drugih spremenljivk hrani tudi spremenljivko *currentDB.*

Hranjenje trajnih podatkov obrazcev je implementirano na strani odjemalca, čeprav bi bil mehanizem lahko implementiran tudi na strani strežnika. Velika slabost takšnega načina bi bil velik padec zmogljivosti sistema, saj bi se trajni obrazci morali prenašati od strežnika k odjemalcu vsakokrat, ko bi uporabnik spremenil vnosni obrazec. Kot prenos obrazcev pa ni mišljen le prenos podatkov dobljenih iz podatkovne baze skupaj z nekaj parametri, vendar tudi prenos kode oziroma besedila za kreiranje vseh grafičnih elementov, kot so na primer HTML elementi. Poleg padca oziroma odzivnosti sistema bi se zelo povečala obremenitev komunikacijskih poti med strežnikom in odjemalcem.

Če povzamemo, trajnost podatkov na strani odjemalca je dosežena z uporabo nad-okvira, ki je del okolja brskalnika. Nad-okvir vsebuje enega ali več okvirjev (ang.: frame). Vsak okvir predstavlja določeno spletno stran, katere podatki se uničijo ob ponovnem nalaganju,

medtem ko podatki shranjeni v nad-okviru ohranijo svoje vrednosti. Iz tega sledi, da je večina objektov, ki jih aplikacija uporablja, shranjenih v nad-okviru. Z namenom nadaljnjega zmanjševanja obsega prenosa podatkov, se v nad-okvir brskalnika ob prvem prenosu meta podatkov določene tabele le-ti shranijo v nad-okvir. Drugi načini doseganja trajnosti podatkov na odjemalčevi strani so piškotki (ang.: cookies) ali vključitev podatkov v internetni naslov (ang.: URL).

3.6 Uporabljeni vzorci zasnove pri Uporabniškem vmesniku do podatkovnih baz

3.6.1 Vzorci

Vsak vzorec opisuje problem iz našega okolja, ki se ponavlja, in rešitev tega problema tako, da se vsakokrat lahko uporabi isto rešitev problema na drugačen način (Gamma et al., str. 2).

Vzorci objektivizirajo oziroma posplošijo izkušnjo programskih inženirjev na strukturiran in lahko dostopen način. So abstrakcija programske kode in ponujajo glavno idejo dobre rešitve konstrukcijskim problemom, ki se ponavljajo. Klasificiramo jih lahko v 3 kategorije (Zullighoven, 2005, str. 79):

- **Konceptualni.** Temeljijo na domenskem pogledu in so torej vidni tudi uporabniku sistema.
- **Zasnovni.** Opisujejo osnovne elemente programske zasnove.
- **Programski.** Kreirajo in implementirajo osnovne elemente zasnove.

3.6.2 Vzorec Singleton

Vzorec opisuje razred, ki ima le en primerek in nudi globalen dostop do podatkov, ki jih vsebuje (Gamma, 1999, str. 127). Prednost vzorca je med drugim zmanjšan obseg imenskega prostora aplikacije, ker se množica spremenljivk z njihovimi vrednostmi omeji znotraj primerka razreda in tako ne „onesnažuje“ imenskega prostora. Druga prednost je kontroliran in jasen dostop do enega samcatega primerka. Zaradi globalne dostopnosti je primerek primeren za hranjenje konfiguracijskih spremenljivk.

Primer UVPB

Singleton vzorec je mogoče zaslediti v naslednjih razredih:

- Podsistem odjemalca
  - *Application_CS*. Ograjuje oziroma vsebuje materiale, orodja in ostale singleton primerke kot so *Forms_CS* in *CheckInput_CS*.

    - *PotOfTables_CS*. Razred vsebuje meta podatke od tabel ter metode za dostop do njih.
    - *Forms_CS*. Razred vsebuje metode za nadzor nivojev, ki je opisan v poglavju 3.6.3 ter hrani specifične spremenljivke za vsak nivo posebej.
    - *Utility_CS*. Razred vsebuje globalne metode in spremenljivke.
    - *CheckInput_CS*. Vsebuje splošne metode za kontrolo datuma, števil itd.
- Podsistem strežnika
    - *Application_SS*. Ograjuje oziroma vsebuje materiale in ostale singleton razrede kot sta na primer *PotOfTables_SS*, *Utility_SS*, ki se nanašata na stran strežnika.
    - *Utility_SS*. Razred vsebuje globalne metode in spremenljivke strani strežnika.

3.6.3 Vzorec Metoda tovarna (ang.: factory method)

Definira vmesnik za kreiranje objektov, vendar se dejansko kreiranje objekta ter odločitev o vrsti objekta izvrši v podrazredu. Metoda torej dovoljuje kreiranje objektov v podrazredih (Gamma, 1995, str.107).

Primer UVPB

V primeru UVPB je mnogo razredov, ki se nanašajo na isti primer uporabe (ang.: use case). Na primer primer uporaba Ustvari zapis predvideva vnos podatkov v poljubno tabelo. Ker pa različne tabele zahtevajo različno obnašanje vnosnih obrazcev, potrebujemo mnogo razredov, kjer vsak razred implementira specifično obnašanje primera uporabe vnosa podatkov določene tabele. Primera konkretne implementacije specifičnega obnašanja sta razreda *MakeEntry_invoice_CS* in razred *MakeEntry_partner_CS*, ki sta oba potomca generalnega razreda *MakeEntry_CS*.

Implementacija mehanizma kreiranja ustreznih podrazredov razreda *MakeEntry_CS* je narejena v kompozitnem razredu *Forms_CS*, ki je strukturiran tako, da poenostavlja kreiranje novih objektov ter določanja povezave med tabelami in razredi, ki implementirajo specifične vnosne obrazce. Definicija Metode tovarna predvideva kreiranje objektov v podrazredih, vendar se v našem primeru objekti kreirajo znotraj okvira, ki ni podrazred razreda *Forms_CS*. Kreirajo se tako, da se metodi poda parametre o tipu razreda, ki naj se kreira. Ime vmesnika v vzorcu je v našem primeru *createPersistentObject* in pripada razredu *Forms_CS*. V metodi *setTableSpecificClasses* razreda *Forms_CS* razvijalec definira povezavo med tabelo in razredom za vsak primer uporabe povezan z operacijami UBDB (ang.: CRUD) posebej.

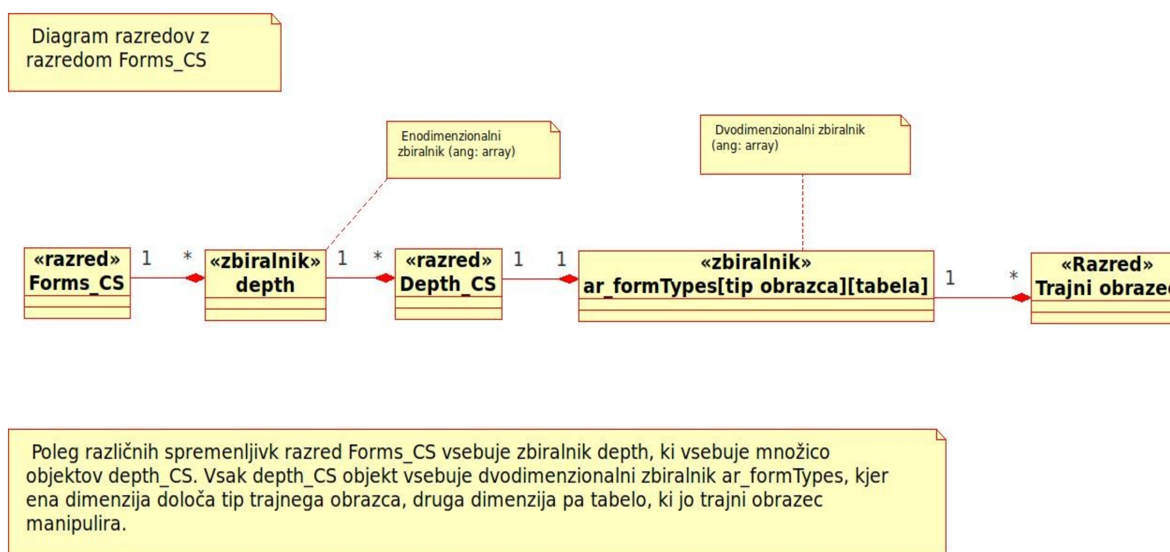
3.6.4 Vzorec Kompozicija

Vzorec Kompozicije omogoča enak način ravnanja s posameznimi objekti kot s skupinami objektov (Gamma, 1995, str. 163).

Primer UVPB

Vsak objekt trajnega obrazca, ki omogoča manipulacijo določene tabele, je določenega tipa. Glede na tip obrazca se v razredu *Forms_CS* sestavi hierarhija objektov.

Slika 20: Diagram razredov z razredom Forms_CS



Poleg različnih spremenljivk razred *Forms_CS* vsebuje zbiralnik *depth*, ki vsebuje množico objektov *depth_CS*. Vsak *depth_CS* objekt vsebuje dvodimenzionalni zbiralnik *ar_formTypes*, kjer ena dimenzija določa tip trajnega obrazca, druga dimenzija pa tabelo, ki jo trajni obrazec manipulira. Za primer vzemimo nastanek trajnega obrazca *MakeEntry_partner_CS*, ki se kreira ob uporabniškem izboru opcije Vnos v tabelo *partner*. Po nastanku se le-ta shrani v objektu *forms_CS*, ki se zaradi trajnosti na strani odjemalca nahaja v nad-okviru *start_CWP*. Tip materiala trajnega obrazca *MakeEntry_partner_CS* je *MakeEntry_CS*, saj gre za vnos zapisa in predstavlja prilagoditev tipičnega trajnega obrazca MakeEntry_CS, kot je opisano v poglavju 3.3.2. Dvodimenzionalni zbiralnik pa poleg informacije o tipu vsebuje tudi ime tabele v katero se podatki na obrazcu vnesejo, ki pa se v primeru obrazca *MakeEntry_partner_CS* imenuje *partner*. Namen zbiralnika *depth* je posledica večnivojskih trajnih obrazcev, ki so podrobneje opisani v podpoglavju Dostop do povezanih tabel z orodjem *ShiftToolEnterKeypressed* poglavja 2.4.5.

Realizacija vzorca kompozicije je dosežena z uporabo zbiralnika *forms_CS*, ki poleg hranjenja vseh materialov trajnih obrazcev, vsebuje tudi generične operacije za določanje ter pridobivanje vrednosti spremenljivk, ki jih uporabljajo vsi materiali oziroma objekti v kompoziciji. Objekti, kot sta *SelectRecord_partner_CS* ali *MakeEntry_partner_CS*, so shranjeni v kompoziciji *forms_CS* in dostopni na isti način preko vmesnika *createPersistentObject* in *returnPersistentForm*.

3.6.5 Vzorec Aktivni zapis (ang.: Active Record pattern)

Vzorec Aktivnega zapisa je pristop dostopa do podatkovne baze. Predvideva obstoj objekta, ki predstavlja določen zapis tabele podatkovne baze ali določen pogled podatkovne baze. Objekt domenskega modela, ki je implementiran na podlagi opisanega vzorca, vsebuje poslovno logiko in logiko dostopa do podatkov. Prav zaradi dejstva, da objekt vsebuje logiko dostopa do podatkov, ima objekt visoko stopnjo sklapljanja (ang.: coupling) s podatkovno bazo. Vzorec Aktivnega zapisa ima naslednje tipične značilnosti (Fowler, 2003):

- Kreiranje primerka Aktivnega zapisa na podlagi zapisa v rezultatu (ang.: resultset), ki ga vrne izvršen SQL stavek.
- Kreiranje novega primerka objekta z vrednostmi, ki so podlaga za vnos v podatkovno bazo.
- Statične metode, ki zajemajo pogosto uporabljene SQL stavke ter vrnejo objekte Aktivnega zapisa.
- Možnost spreminjanja podatkovne baze na podlagi vrednosti podatkov objektov Aktivnega zapisa.
- Posodabljanje ter branje polj tabel podatkovne baze.
- Implementacija poslovne logike.

Primer UVPB

Objekt, ki realizira vzorec Aktivnega zapisa, je *DbObject_SS* in vsebuje metode kot so *connectToDB, setID, returnID, isRecordAlreadyInDb, insertObjectOnFirstEmptyID, evaluateAndWriteToDB, getLastIdOfField*, *returnObjectFields* in druge. Metodi *setID* ter *returnID* sta primera posodobitve določenega polja tabele podatkovne baze. Iskanje zapisa v podatkovni bazi poteka s pomočjo metode *isRecordAlreadyInDb*.

3.6.6 Vzorec Odjemalčev zbiralnik tabelnih struktur

1. Problem

Zaradi omejenosti strojnih in komunikacijskih zmogljivosti je potrebno upoštevati pogostost posodobitve meta podatkov tabel na odjemalcu s strani strežnika.

2. Kontekst

Razvoj spletnih aplikacij strežnik-odjemalec, ki temeljijo na protokolu HTTP.

3. Sile (ang.: forces)

Komunikacija med strežnikom in odjemalcem je odvisna od sprememb, povzročenih na določeni relaciji, in zahtevano ažurnostjo vsebin. Razvijalec naj bi optimiziral potek komunikacije na način, ki bo minimaliziral njeno redudantnost, kar pomeni, da naj se pri zaporednih prenosih ne bi prenašali venomer isti podatki, temveč le spremembe podatkov.

4. Rešitev

Struktura tabel v podatkovnih bazah je izredno stabilen element v programski opremi. To pomeni, da lahko zlahka omejimo pogostost posodobitve meta podatkov tabel na odjemalcu s strani strežnika na le en prenos na sejo. Podatki o strukturi tabele se tako prenesejo s strani strežnika k odjemalcu le ob prvi zahtevi po začetku seje in se shranijo v pomnilnik odjemalca.

5. Primer UVPB

Razred *PotOfTables_CS* na strani odjemalca kreira objekt zbiralnika imenovanega *potOfTables_CS*. Zbiralnik je medpomnilnik, v katerega se ob prvi zahtevi odjemalca prenesejo meta podatki od tabel podatkovne baze. Medpomnilnik omogoča bistveno hitrejše kasnejše poizvedbe drugih odjemalčevih objektov kot pa, da bi odjemalčevi objekti vsakokrat zahtevali podatke od strežnika.

3.6.7 Vzorec Odjemalčevo preverjanje podatkov

1. Problem

Omejene zmogljivosti strojne opreme ter komunikacijskih poti narekujejo minimalizacijo prometa med različnimi arhitekturnimi komponentami. Zaradi dejstva, da je določen odstotek uporabniških vnosov napačen, je nujna sestavina vsakega informacijskega sistema preverjanje kreiranih oziroma vnesenih podatkov. Ugotoviti je potrebno, kje je bolj smiselno izvajati postopek preverjanja podatkov, na strežniku ali na odjemalcu.

2. Kontekst

Razvoj spletnih aplikacij strežnik-odjemalec, ki temeljijo na protokolu HTTP.

3. Sile

Ob bolj pogostem pojavljanju napak bo preverjanje na strežniku zelo povečalo promet komunikacijskih poti, saj se bo poleg poslanega obsega pravilnih podatkov prenesel tudi ves niz napačnih podatkov.

4. Rešitev
Postopek preverjanja podatkov, ki ne zahteva podatkov tabel podatkovne baze, naj bo implementiran na strani odjemalca. Obvestila o napakah se tako takoj prikažejo na zaslonu, kar poveča odzivnost sistema ter izboljša uporabniško izkušnjo. To seveda ne pomeni, da se podatki na strežniški strani ne preverjajo, ampak naj se preverjajo na obeh podsistemih.

4 Uporaba

Ogrodje UVPB ima 2 načina uporabe. Lahko ga uporablja končni uporabnik za manipulacijo podatkov podatkovnih baz, lahko pa ga uporablja razvijalec kot ogrodje za razvoj specifičnih vnosnih obrazcev.

4.1 Uporaba za uporabnike

S pomočjo UVPB lahko uporabnik, ki razume osnovno strukturo podatkovne baze, do katere dostopa, na njej izvaja osnovne operacije ustvarjanja, branja, dopolnjevanja ter brisanja zapisov, shranjenih v tabelah. Poleg tega lahko uporabnik to naredi v znanem delovnem okolju internetnega iskalnika.

Na začetni spletni strani UVPB se nahaja meni z opcijami osnovnih operacij UBDB, ki jih spletna aplikacija nudi: Izberi podatkovno bazo, Izberi tabelo, Išči zapis in Dodaj zapis.

Slika 21: Meni osnovnih operacij UBDB

Operacija kreiranja novega zapisa je izvedena z izborom opcije „Dodaj zapis“, ki povzroči prikaz vnosnega obrazca tabele *partner*. Slika 22 prikazuje primer izbora tabele *partner* neke podatkovne baze.

Slika 22: Primer osnovne operacije kreiraj zapis v tabeli partner

Branje zapisa poteka preko izbora opcije „išči zapis“ v meniju, čemur sledi vpis pogojev za prikaz zapisov. Pritisk tipke *enter* na zapisu prikaže vnosni obrazec "dopolni zapis" z vrednostmi polj, ki določajo iskani zapis, kot je prikazano na Sliki 23. Opcije na koncu polj omogočajo dopolnitev ali bris zapisa.

Slika 23: Primer branja, dopolnjevanja ter brisanja zapisa v tabeli partner

Pomembna lastnost ogrodja UVPB je podpora navigaciji med povezanimi tabelami. Povezavo tabel ni potrebno določiti eksplicitno. Privzeta logika ogrodja pripisuje lastnost tujega ključa vsakemu polju, ki ni primarni ključ, in ki se končuje s črkama ID. V primeru vnosnega obrazca tabele *faktura*, ki ga prikazuje Slika 24, so tuji ključi polja *NarocilnicaID*, *DobavnicaID*, *PredracuID*, *RacunZaPredplaciloID*, *PartnerID*, *KontoID*, *StroskovnoMestoID*, *NacinPlacilaID* ter polje *NogaRacuna2ID*.

Slika 24: Primer polja PartnerID kateremu se samodejno pripiše lastnost tujega ključa ter se samodejno določi povezava s tabelo partner s katero je tuji ključ povezan.

Na podlagi imena polja se določi tudi ime primarnega polja ter tabela, ki je povezana s tujim ključem. Za ime primarnega polja, ki je povezano s tujim ključem, se predpostavlja, da je enako imenu tujega ključa. Ime tabel pa se sestavi tako, da se odstrani zadnji 2 črki ID ter vse črke spremeni v male tiskane, torej je povezana tabela v primeru polja *PartnerID* tabela *partner*. Ogrodje implementira enotno funkcionalnost polj, ki predstavljajo tuje ključe tako, da hkraten pritisk tipke *shift* ter tipke *enter* povzroči prehod na obrazec naslednjega nivoja (Slika 25), ki prikazuje seznam zapisov v tabeli povezani s poljem tujega ključa, na katerem smo izvedli pritisk tipk. Izbor zahtevanega zapisa na seznamu zopet povzroči prehod na prejšnji obrazec, prikazan na Sliki 26, in vstavi vrednost primarnega ključa izbranega zapisa v polje tujega ključa začetnega obrazca.

Slika 25: Prikaz obrazca izbora (ang.: Select Record form), ki prikazuje seznam zapisov tabele partner

http://localhost - Uporabniški vmesnik do baze podatkov - Bon Echo

KLIK Operacije Razno

IZBERI ZAPIS V TABELI v tabeli partner:
1/1

PartnerID	Davcna	Naziv	Naslov	Posta	Kraj	DrzavaID	TRR	Tele
1	SI83423565	Testni Partner	Jelovškova 5	1000	Ljubljana	SI		

Slika 26: Primer vnosnega obrazca faktura z vstavljeno vrednostjo primarnega ključa iz obrazca izbora višjega nivoja

DatumDobavez		
RokPlacila		
PartnerID	1	Testni Partner
KontoID		
StroskovnoMestoID		
NacinPlacilaID		

4.2 Uporaba za razvijalce

Ogrodje UVPB vsebuje niz objektov pripravljenih za prilagajanje. Spodaj je naveden skrajšan postopek za prilagoditev vnosnega obrazca tabele *partner*.

Kreira se podatkovna baza *Def1*, *gkstranke* in tabele

Podatkovna baza *Def1* je privzeta baza, kjer razvijalec ustvari tabele. Ko obstaja več naročnikov projekta, je običajno, da se za vsakega naročnika kreira podatkovna baza. Globalni parametri, ki so dostopni kateremukoli objektu aplikacije ter lastni vsaki bazi posebej, se nahajajo v bazi *gkstranke* v tabeli stranke. V primeru projekta, ki vsebuje več modulov, se kreira nova tabela *modulizastranko*, kjer se vpiše vse module, ki jih stranka potrebuje. Za potrebe ponazoritve kreiramo tabelo *partner* v bazi *Def1*.

Kreirajo se mape ter datoteke

Ustvari se nova mapa *CustomizedModule*, ki mora vsebovati mape z imeni *Class_PHP*, *Class_JS* in *Class_JS_TableSpecific*.

Mapa *Class_PHP* bo vsebovala datoteke kot sta *Application_customizedModule_SS* in *Start_customizedModule_SS*. Razred strežniške strani *Application_customizedModule_SS*, ki se nahaja v istoimenski datoteki, je podrazred razredu *Application_SS, medtem, ko je* razred *Start_customizedModule_SS* podrazred razredu *Start_SS. Oba nadrazreda sta* specificirana v mapi *UserToDBInterface*. Oba novo ustvarjena podrazreda sta specifična za modul *CustomizedModule*, kar pomeni, da vsebujeta obnašanje ter podatke, ki so specifični za ta modul na splošno. Podrobne značilnosti obnašanja posameznih tabel se specificirajo v drugih razredih.

Mapa *Class_JS* naj vsebuje datoteke z razredi odjemalčeve strani kot sta *Application_customized_CS* in *Forms_customizedModule_CS*. Podobno kot zgoraj omenjeni razred strežniške strani Application_Customized_CS, opisuje podatke in funkcionalnost specifično za posamezen modul. *Forms_customizedModule_CS* vsebuje „kljukice“ (ang.: hooks), kamor se priključijo prilagojeni objekti. Te „kljukice“ so implementirane kot zbiralniki, kjer mora razvijalec določiti povezave med tabelami in razredi, shranjenimi v mapi *Class_JS_TableSpecific*.

Podatki ter funkcionalnost, ki je specifična za tabelo *partner*, se nahajajo v datoteki *MakeEntry_partner_CS.js* mape *Class_JS_TableSpecific*. Ta razred se uporablja le pri vnosu v tabelo *partner* in le pri tipu vnosnega obrazca *MakeEntry_CS*. Ostali, prilagojeni obrazci, kot sta obrazec izbora (*SelectEntry_CS*) in obrazec iskanja (*SearchRecord_CS*), se naj nahajajo v datoteki *SelectEntry_partner_CS* oziroma v datoteki *SearchRecord_partner_CS*.

Kopiranje datotek iz privzetega modula k prilagojenemu

Datoteke se kopira iz mape *UserToDBInterface* v novo ustvarjeno mapo *CustomizedFolder*.

Sprememba programske kode v *.php* datotekah

Ime seje (ang.: session name) naj se spremeni v vseh datoteka s končnico *.php* iz *UserDBInterface* v *CustomizedModule*. Ime seje določa imenski prostor, kjer se nahajajo objekti, ki se nanašajo na določen modul. Vsak modul ima na primer objekt, ki se imenuje *Application_SS*, vendar se vsak od njih nahaja v svojem imenskem prostoru.

Doda in spremeni naj se programska koda v datoteki *menu.php*

V datoteki *menu.php* naj se doda ali spremenijo poti datotek, ki specificirajo funkcionalnost posameznih opcij menija.

Spremeni naj se programska koda v datoteki *menu.php in MenuFunctions.js*

Spremeni naj se programska koda, ki bo določala obliko in funkcionalnost menija ter opcij.

Ustvari naj se nova datoteka, ki bo vsebovala prilagojen obrazec

Ustvari naj se nova datoteka *MakeEntry_partner_CS*, ki bo vsebovala razred s podatki ter specifično funkcionalnost vnosnega obrazca v tabelo partner. Metoda *addToolKeypressed* mora biti ponovno definirana ter prepisana z namenom, da poveže trenutni obrazec, viden na zaslonu z ustreznim trajnim orodjem.

Doda naj se programska koda v datoteko Forms_customizedModule_CS.js

Dodana programska koda bo določala povezavo novo ustvarjenih razredov in tabel.

Dodaj in spremeni programsko kodo v datoteki start.php

Potrebno je dodati sklice do vseh na novo ustvarjenih datotek kot na primer *Application_customizedModule_SS*, *Start_customizedModule_SS*, *Application_customizedModule_CS* in *Forms_customizedModule_CS*. Spletna stran *start.php* je odgovorna za kreiranje večine objektov aplikacije, zato mora vsebovati informacijo oziroma pot do datotek, kjer se nahajajo specifikacije razredov.

Kreira naj se objekt *$_SESSION['application_SS']* na podlagi razreda *Application_customizedModule_SS.*

SKLEP

Ugotovljeno je, da je namen, vključno z cilji, magistrskega dela izpolnjen. Delo podrobno razloži teoretične koncepte, ki so uporabljeni v ogrodju, in ponuja delujočo programsko rešitev, ki, v času oddaje magistrskega dela, že predstavlja jedro nekaterim razvitim prilagojenim programskim rešitvam v podjetju Klik d. o. o. Delujoča rešitev je razvita zgolj z uporabo skriptnih jezikov in je deklarirana kot odprto kodna. V poglavju 4 sta opisana konkretna primera uporabe tako za razvijalce kot za končne uporabnike. Kot odprto kodna rešitev je ogrodje prosto za uporabo vsej svetovni skupnosti razvijalcev. Samodejno kreiranje HTML obrazcev na podlagi meta podatkov podatkovne baze dovoljuje končnemu uporabniku uporabo programske rešitve kot orodja za izvrševanje osnovnih operacij ustvarjanja, branja, dopolnjevanja in brisanja podatkov.

Doseženi cilji magistrskega dela so podjetju Klik d. o. o. omogočili dosego poslovnih ciljev povečanja dohodka, rasti števila zaposlenih, posodobitev obstoječih programskih rešitev ter možnost realizacije novega poslovnega modela z novimi oblikami dodane

vrednosti, kot so cenejše vzdrževanje, večja fleksibilnost, neodvisnost lokacije dostopa do aplikacije in druge.

Praktični rezultati so dosegljivi na:

- Predstavitvena različica je dostopna preko naslova: http://www.klik.si/spletnestoritve/Stranke/Mag1/index.html.
- Datoteke ogrodja se lahko prenese preko naslova: http://www.stamihan.frin.us/spletnestoritve/Stranke/Mag1/Download/index.html.
- Osnutek dokumenta magistrskega dela v angleškem jeziku se lahko prenese preko naslova: http://www.stamihan.frin.us/spletnestoritve/Stranke/Mag1/Doc/ang/MasterDegreethesis.pdf.
- Dokument magistrskega dela v slovenskem jeziku, potrjen s strani Ekonomske fakultete, se lahko prenese preko naslova: http://www.stamihan.frin.us/spletnestoritve/Stranke/Mag1/Doc/slo/MasterDegreethesis.pdf.

Velikost projekta praktičnega dela magistrskega dela je podan z naslednjimi kriteriji:

- število programskih vrstic: cca. 10000,
- število razredov: cca. 40,
- neposredni oziroma končni uporabniki: 0,
- posredni uporabniki: 14 podjetij oziroma cca. 20 uporabnikov.

Spletna aplikacija, ki deluje v okolju razširjenega internetnega brskalnika s tridelno, porazdeljeno arhitekturo, vsekakor poveča dodano vrednost v primerjavi s starimi informacijskimi sistemi. Standardni protokol kot je internet HTTP in standardni internetni brskalnik sta ustvarila potrebo po razvoju informacijskih sistemov, ki so prilagojeni in optimizirani tem standardom. UVPB je odgovor na to potrebo.

Podatke in funkcionalnost aplikacije zagotavlja niz objektov, razdeljenih na stran odjemalca ter stran strežnika, medtem ko je trajnost podatkov implementirana v relacijski podatkovni bazi. Funkcionalnost kot je preverjanje podatkov, navigacija med obrazci, direktni odzivi na uporabniška dejanja so naloge strani odjemalca. Naloge strani strežnika pa so interakcija s podatkovno bazo, del poslovne logike, zahtevnejši izračuni itd.

Ena izmed slabosti spletnega, porazdeljenega sistema so vsekakor višji stroški povezani z varnostjo in zasebnostjo podatkov ter tudi večja zapletenost porazdeljenega sistema. Varnostni problemi so večinoma rešeni s prijavnimi obrazci, varnostnim sistemom podatkovnih baz in strežniško metodo šifriranja mrežnih komunikacij. Kompleksnost se obvladuje z objektnim pristopom in tridelno arhitekturo ločevanja podatkov, kode oziroma funkcionalnosti in oblike podatkov.

Glavnina danes delujočih programskih rešitev spletnih aplikacij se fizično nahaja na strežniških podsistemih. Prednost trenutne arhitekture je, da uporabnik dobi najnovejšo različico vsakokrat, ko odpre začetno spletno stran aplikacije, saj se tudi koda, ki deluje na strani odjemalca prenese vsakokrat znova. Prednost ni zanemarljiva v okoljih, kjer ima razvoj aplikacije mnogo iteracij, in kjer je uporabnik resnično nevešč uporabe računalniške tehnologije.

Slabost spletnega objektno usmerjenega in porazdeljenega informacijskega sistema je manjša varnost podatkov zaradi zelo povečane dostopnosti sistema, ki deluje v spletu. Upoštevati moramo tudi bistveno manjše hitrosti spletnih in porazdeljenih sistemov od tistih, ki se v celoti nahajajo zgolj na enem računalniku ter za komunikacijo ne uporabljajo HTTP protokola. Še več, izvajanje navodil oziroma ukazov skriptih jezikov je počasnejše od izvajanja binarne kode, saj mora sistem vsak ukaz sproti prevesti in ga nato izvesti, medtem, ko lahko binarno kodo izvaja brez prevajanja. To še dodatno zmanjša hitrost sistema in komaj zadosti potrebam hitrega vnosa ter izpisov podatkov. Brez tehnološkega napredka zadnjih let kot so hitrejše internetne povezave, hitrejši procesorji ter hitrejši RAM realizacija poslovnega modela oziroma realizacija sistema ne bi bila mogoča oziroma smiselna, saj bi sistem deloval prepočasi.

Razvito ogrodje ni dovolj fleksibilno, da bi bilo kompatibilno z drugimi sistemi za upravljanje podatkovne baze (SUBP) niti ne deluje v drugih okoljih kot je Firefox 2.0 ali Firefox 3.0. Pomanjkljivost implementacije bi lahko bila tudi uporaba skriptnega jezika JavaScript, saj je ekspresivnost tega jezika omejena.

Faza razvoja informacijskega sistema zbiranje uporabniških zahtev ni potekala popolnoma v skladu z navodili, ki so opisanimi v poglavju 2.2. Razvoj ogrodja je majhen projekt, za katerega ni smiselno pisanje obsežne in podrobne dokumentacije, saj bi to po nepotrebnem zavrlo razvoj sistema.

Možnost nadaljnjega razvoja ogrodja UVPB bi bila v samodejnem kreiranjem obrazcev, ki bi omogočali hkraten vnos v dve ali več tabeli, v eno glavno in eno pod-tabelo. Primer takega obrazca je obrazec vnosa računa, ki v zgornjem delu vsebuje podatke kupca, datum, veze z drugimi dokumenti in druge podatke, v spodnjem delu pa vsebuje množico vrstic, kjer vsaka vrstica vsebuje podatke določene vrste artikla. Zgornji del predstavlja eno tabelo, na primer poimenovano „račun“, spodnji del pa predstavlja drugo tabelo, na primer poimenovano „artikli na računu“.

V prihodnosti bi bilo UVPB smiselno uvrstiti v zbirko odprto kodnih rešitev na spletni strani *http://sourceforge.net/* in k razvoju poskušati pritegniti druge razvijalce programskih rešitev. Izboljšati bi bilo mogoče grafično podobo UVPB, preurediti programsko kodo menijev in drugo.

Praktični del magistrskega dela je rezultat mnogo iteracij v časovnem obdobju nekaj let in je rezultat stalnega preurejanja kode (ang.: refactoring). Avtor dela si želi, da bi tako teoretični del kot tudi praktični del, poleg podjetju Klik d. o. o., koristila tudi drugim razvijalcem informacijskih sistemov.

Literatura in viri

1. *About JavaScript.* (2009) *[Mozilla developer center]*. Najdeno 17.9.2009 na spletnem naslovu https://developer.mozilla.org/en/About_JavaScript.
2. Achour M., Betz F. & Dovgal A. (2009). *PHP Manual*. The PHP Documentation Group. Najdeno 17.9.2009 na spletnem naslovu http://www.php.net/manual/en/index.php.
3. Boss B., Celik T. & Hickson I. *(2007). Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification [W3C].* Najdeno 17.9.2009 na spletnem naslovu http://www.w3.org/TR/CSS2/.
4. Brus M. (2001). *Orodja za uporabo metode 'poenoten jezik modeliranja' (UML) pri izgradnji informacijskih sistemov.* Ljubljana: Ekonomska fakulteta.
5. Chapman M. (2003). *In Search of Stupidity: Over 20 Years of High-Tech Marketing Disasters.* Apress.
6. Conallen J. (1999). *Building Web Applications with UML.* Addison Wesley Longman.
7. Conallen J. (2006). *Modeling Web Application Design with UML*. Najdeno 27. septembra 2009 na spletnem naslovu http://www.itmweb.com/essay546.htm
8. Damij T. (2001). *Tabular application development for information systems: an object-oriented methology*. New York: Springer, Cop.
9. Damij K. (2001). *Razvoj informacijskega sistema bolnice z metodologijo TAD*. Ljubljana: Ekonomska fakulteta.
10. Duffy T. (2001). *Encapsulate Your JavaScript: Keep Private Methods Private.* Najdeno 17.9.2009 na spletnem naslovu http://www.devx.com/getHelpOn/10MinutesSolution/16467/17663/page/1.
11. Edwards J. & Adams C. (2006). *The JavaScript Anthology: 101 Essential Tips, Tricks & Hacks.* SitePoint Pty. Ltd.,
12. Fowler M. (2003). *UML Distilled A Brief Guide to the Standard Object Modeling Language.* (3rd ed.) Addison-Wesley Professional.
13. Fowler M. (2005). *Patterns of Enterprise Application Architecture.* Addison-Wesley Professional.
14. Gamma E., Helm R. & Johnson R. (1995). *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional.
15. Groff R. & Weinberg P. (2002). *SQL: The Complete Reference.* (2nd ed.) The McGraw - Hill Companies.
16. Heilmann C. (2006). Beginning JavaScript with DOM Scripting and Ajax: From Novice to Professional. Apress.
17. Hornby A. (2007). *Oxford Advanced Learner's Dictionar*y. (7th ed.) Oxford University Press.
18. Kruchten P. (2000). *The Rational Unified Process*. (2nd ed.) Addison - Wesley Professional.
19. Musciano C. & Kennedy B. (2000). *HTML & XHTML. The Definite Guide.* O'Reilly & Associates, Inc.

20. Harold E. & Means S. (2001). *XML in a Nutshell. A Desktop Quick Reference.* O' Reilly & Associates, Inc.
21. *PhpPatterns*. Najdeno 27. septembra 2009 na spletnem naslovu http://www.phppatterns.com/docs/design/php_and_uml_class_diagrams
22. Jacobson I. & Booch G. & Rumbaugh J. (1999). *The Unified Software Development Process.* Addison Wesley Longman inc.
23. Leszek A. (2001). *Requirements Analysis and System Design. Developing Information Systems with UML*. England: Pearson Education.
24. *Ltfe ITKT slovar* [LTFE]. (2008). Najdeno 17.9.2009 na spletnem naslovu http://slovar.ltfe.org.
25. Lindsey K. (2000-2003). *JavaScript. Inheritance in JavaScript.* Najdeno 27. septembra 2009 na spletnem naslovu http://www.kevlindev.com/tutorials/javascript/inheritance/inheritance10.htm.
26. Potencier F. & Zaninotto F.. (2007). *The Definitive Guide to symfony (Definitive Guide)*.
27. Richard C. & William M. (1997). *UML and C++. A practical guide to object-oriented development.* New Yersey: Prentice-Hall, Inc.
28. Shelly G., Cashman T. & Rosenblatt H. (2003). *System Analysis and Design course Technology.* (5th ed.) International Thomson Publishing
29. Siau, K. & Cao, Q. (2001). Unified Modeling Language (UML) – a complexity analysis. *Journal of Database Management*, 12 (1).
30. Slovar slovenskega knjižnega jezika. (2008). Ljubljana: DZS.
31. Slovensko društvo Informatika. (2009). *Islovar*. (2nd ed.) Najdeno 17.9.2009 na spletnem naslovu http://www.islovar.org/
32. *Systems Analysis and Design. Data Flow Diagrams. (1998).* Najdeno 17.9.2009 na spletnem naslovu http://www.cs.mdx.ac.uk/staffpages/geetha/bis2030/DFD.html.
33. Ullman C. & Dykes L. (2007). *Beginning Ajax.* Wiley Publishing, Inc.
34. Zullighoven H. (2005). *Object-Oriented Construction Handbook. Developing application-oriented software with the tools and materials approach.* ZDA: Elsevier Inc.