UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Siniša Ribić

Aplikacija za likvidacijo faktur

DIPLOMSKO DELO

NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: doc. dr Rok Rupnik

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

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

Št. naloge: 00003/2010
Datum: 01.10.2010

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

Kandidat: **SINIŠA RIBIĆ**

Naslov: **APLIKACIJA ZA LIKVIDACIJO FAKTUR**
APPLICATION SYSTEM FOR INVOICE CLOSURE

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Razvijte aplikacijo za masovno likvidacijo faktur. Pri tem uporabite .NET tehnologijo in MS SQL podatkovno bazo.

Mentor:

doc. dr. Rok Rupnik

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a SINIŠA PIBIĆ,

z vpisno številko 63050098,

sem avtor/-ica diplomskega dela z naslovom:

APLIKACIJA ZA LIKVIDACIJO FAKTUR

S svojim podpisom zagotavljam, da:

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

  doc. dr. Rok Rupnik

  in somentorstvom (naziv, ime in priimek)

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 11. 3. 2011 Podpis avtorja/-ice: Siniša Pibić

Zahvala

Zahvaljujem se doc. dr. Roku Rupniku za pomoč in nasvete v procesu izdelave diplomske naloge.

Posebno zahvalo bi rad namenil svoji družini in punci, za pomoč in spodbudo ne samo pri izdelavi diplomske naloge ampak v času celotnega študija.

KAZALO VSEBINE

Seznam uporabljenih kratic in pojmov

BPM	upravljanje poslovnih procesov
CSS	predloge ki določajo videz spletih strani
HTML	označevalni jezik za izdelavo spletnih strani
SP	funkcije, izdelane z uporabo SQL jezika, ki delujejo nad podatkovno bazo
SQL	strukturirani povpraševalni jezik za delo s podatkovnimi bazami
TIFF	format za shranjevanje slik
XML	razširljiv označevalni jezik
WS	proces operacijskega sistema Windows
.NET	ogrodje za razvoj aplikacij

1. Povzetek

Podjetje CREA d.o.o., kjer sem opravljal svoje praktično izobraževanje, je bilo zadolženo za izdelavo celotne informacijske rešitve za enega večjih upraviteljev nepremičnin na področju Ljubljane. V okviru diplomske naloge sem sodeloval pri razvoju omenjene informacijske rešitve, ki je predstavljala prehod na elektronsko poslovanje.

Aplikacija je bila razvita na podlagi .NET platforme v programskem jeziku C#. Pomemben del aplikacije je predstavljala tudi podatkovna baza Microsoft SQL, kjer so bili shranjeni vsi podatki. Upravljanje poslovnih procesov je bilo izvedeno s pomočjo orodja Ultimus Business Process Management Studio. Izdelani iskalnik je omogočal hitro in natančno iskanje s pomočjo dodatnih pogojev. Rezultate iskanja so predstavljali elektronsko obdelani prispeli dokumenti. Poleg iskalnika je bil v sklopu celotne informacijske rešitve izdelan tudi poslovni proces, ki je skrbel za likvidacijo prispelih masovnih faktur na podlagi določenih poslovnih pravil.

Opisani moduli so bili izdelani predvsem zaradi hitrejše obdelave in lažjega iskanja po arhivu prispelih dokumentov. Razviti moduli so odpravili težavo zakasnitve, ki je nastajala pri klasični likvidaciji in obenem poskrbeli za avtomatsko elektronsko arhiviranje prispelih dokumentov. Izdelava iskalnika s pogoji je odpravila nepotrebno in zamudno prikazovanje vseh rezultatov ob vsakem iskanju.

Ključne besede

Celotna informacijska rešitev, poslovni proces, likvidacija masovnih faktur, iskalnik, Ultimus BPM Studio

2. Abstract

Company CREA d.o.o. where I did my practical training was in charge of making a complete information solution for one of the biggest real-estate agencies in Ljubljana. Within the thesis I was involved in development of complete information solution whose main part was eBusiness.

The application was developed on .NET platform in programming language C#. An important part of the application was its database – Microsoft SQL, where all data was stored. Business process management was performed by Ultimus Business Process Management Studio. Search engine was designed with search filter, which allowed us to search quickly and precisely. Results of search query were represented as electronically process invoice documents. In addition to search engine the business process was developed as well. New business process was in charge of liquidation of bulk invoices based on certain business rules.

The described modules were designed for improving the process of invoices and to enable faster search through archive of income invoices. They also eliminated the problems of delay caused by classical liquidation and at the same time, their duty was to automatically create archives from income documents. Search engine contained built in search filter which improved search speed with selection of results based on filter.

Key words

Complete information solution, business process, liquidation of bulk invoices, search engine, Ultimus Business Process Management Studio

3. Uvod

Današnji način poslovanja je podoben verigi, kjer je pomembno, da vsak člen verige opravi svojo nalogo, da bi na koncu lahko prišli do željenih rezultatov. V kolikor eden od členov odpove in ne opravi zadane naloge, kot rezultat dobimo negativno zaključen cikel. Potrebno je torej poskrbeti za nemoteno delovanje vseh členov, ne glede na čas in lokacijo. Za primer vzemimo uslužbenca v pisarni, ki ima na svoji delovni mizi kup papirja, ki ga je potrebno temeljito pregledati in na podlagi rezultatov posredovati v nadaljnje obdelave. Vsakomur od nas je tako delo odvečno in vsi se trudimo, da ga čim bolj minimiziramo. Uslužbenec v pisarni je le eden od mnogih členov verige poslovanja. Predstavljajmo si, da ta isti uslužbenec sodeluje na več poslovnih področjih hkrati vsak dan. Seveda pa od uslužbenca pričakujemo maksimalno učinkovitost, ne glede na količino njegovega dela.

Pri vsakodnevnih ponavljajočih opravilih pride do pravega izraza sistem avtomatizacije. Določene člene oz. korake poslovne verige lahko avtomatiziramo s čimer dosežemo visoko kakovost in zanesljivost. V določenih primerih bi bilo mogoče avtomatizirati celoten poslovni proces, vendar to v vseh primerih ni zaželeno. Kljub vsej tehnologiji, ki nam je ponujena, je v določenih primerih človeški faktor ključnega pomena za uspešen končni rezultat.

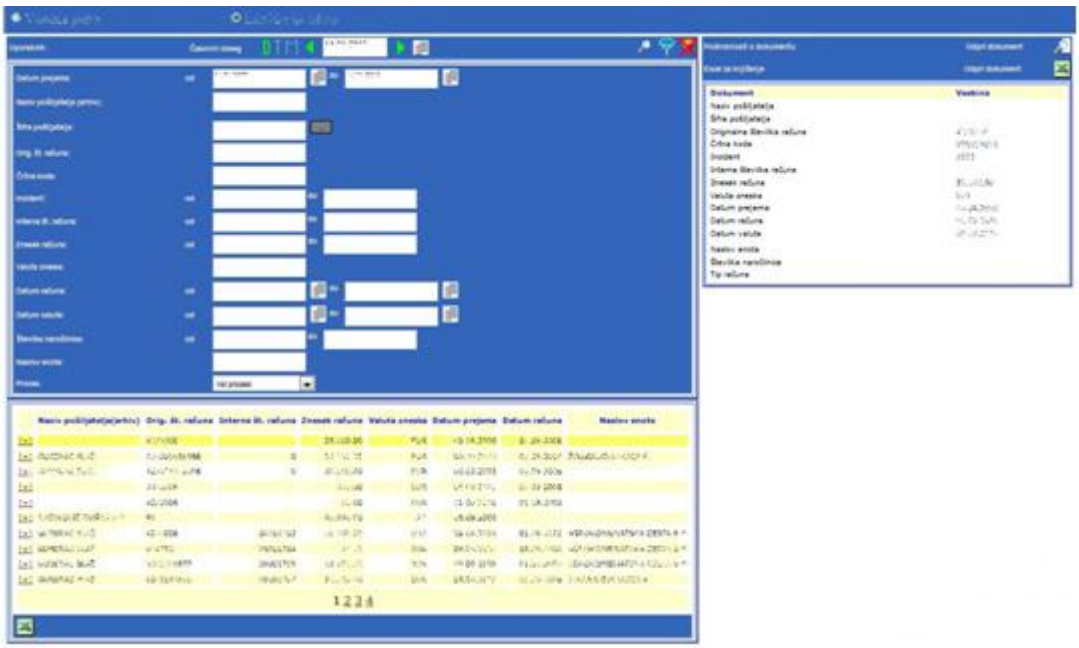
Na podoben primer sem naletel tudi sam, pri avtomatizaciji določenega člena v verigi poslovanja. Zaradi velike količine prejetih dokumentov, je bilo potrebno avtomatizirati proces prejemanja računov s strani podjetij in obenem ugotoviti ali je prejet račun že poravnan. Vse skupaj je bilo potrebno med seboj povezati in urediti, končne rezultate pa vpisati v arhiv, ki je dostopen vsem zaposlenim znotraj podjetja. Ključnega pomena je bila tudi konsistenca podatkov, za katero je moral skrbeti sistem sam. Kljub avtomatiziranem delovanju procesa je nad vsem tem bil potreben nadzor s strani zaposlenega, ki je v primeru težav imel ključno vlogo. Obdelava vseh dokumentov je potekala avtomatsko. Navadno poslovanje preko papirja se je v tem koraku spremenilo v elektronsko poslovanje kajti vsi dokumenti so se pretvorili v elektronsko obliko. Način poslovanja se je v tem koraku spremenil, kar je pomenilo olajšanje. Obenem se je pospešil tudi sam potek, vendar kljub vsem prednostim je problem količine dokumentov ostal nerešljiv.

Predstavljajte si, da je neko podjetje zahtevalo od našega uslužbenca vpogled v prejete dokumente za nekaj let nazaj. Za uslužbenca to pomeni prebijanje skozi goro dokumentov, rezultati pa bi bili dostopni z večjo zakasnitvijo. V primeru elektronskega poslovanja bi bili vsi dokumenti arhivirani elektronsko, naš uslužbenec bi potreboval le dober iskalnik po arhivu in dokumenti bi bili na voljo v nekaj trenutkih. Torej sam proces avtomatizacije ni osredotočen le na eno samo področje, kajti na tak način ne pridobimo veliko. Potrebno je poskrbeti za avtomatizacijo celotne hierarhije določenega člena v verigi poslovanja. Moja naloga je torej bila poskrbeti za avtomatizacijo poslovnega procesa likvidacije masovnih faktur za enega večjih upraviteljev nepremičnin na področju Ljubljane. Na podlagi te naloge sem poskrbel za avtomatizacijo ne samo na področju likvidacije masovnih faktur ampak tudi na področju iskanja in pregledovanja že obdelanih faktur. Zato je moj projekt bil razdeljen na

dva dela, kjer je prvi del predstavljal iskalnik, drugi del pa avtomatizacijo poslovnega procesa likvidacije masovnih faktur.

4. Iskalnik prejetih dokumentov

Začetni korak pri izdelavi prenovljenega iskalnika je bil pregled in preučevanje obstoječe aplikacije – »starega« iskalnika. Predhodna aplikacija je bila zasnovana kot spletna aplikacija, sestavljena hpa je bila iz dveh strani. Vsaka od strani je iskala po določenih dokumentih glede na njihov izvor. Začetna stran je vsebovala iskalnik, ki je omogočal iskanje po dokumentih iz vhodne pošte, iskalnik na drugi strani pa je omogočal iskanje po dokumentih likvidacije faktur. Oba iskalnika sta vsebovala tudi iskalni filter, ki je pripomogel k naprednem iskanju. Prikaz rezultatov je bil ločen na dva dela. Primer iskalnika prikazuje slika 1.

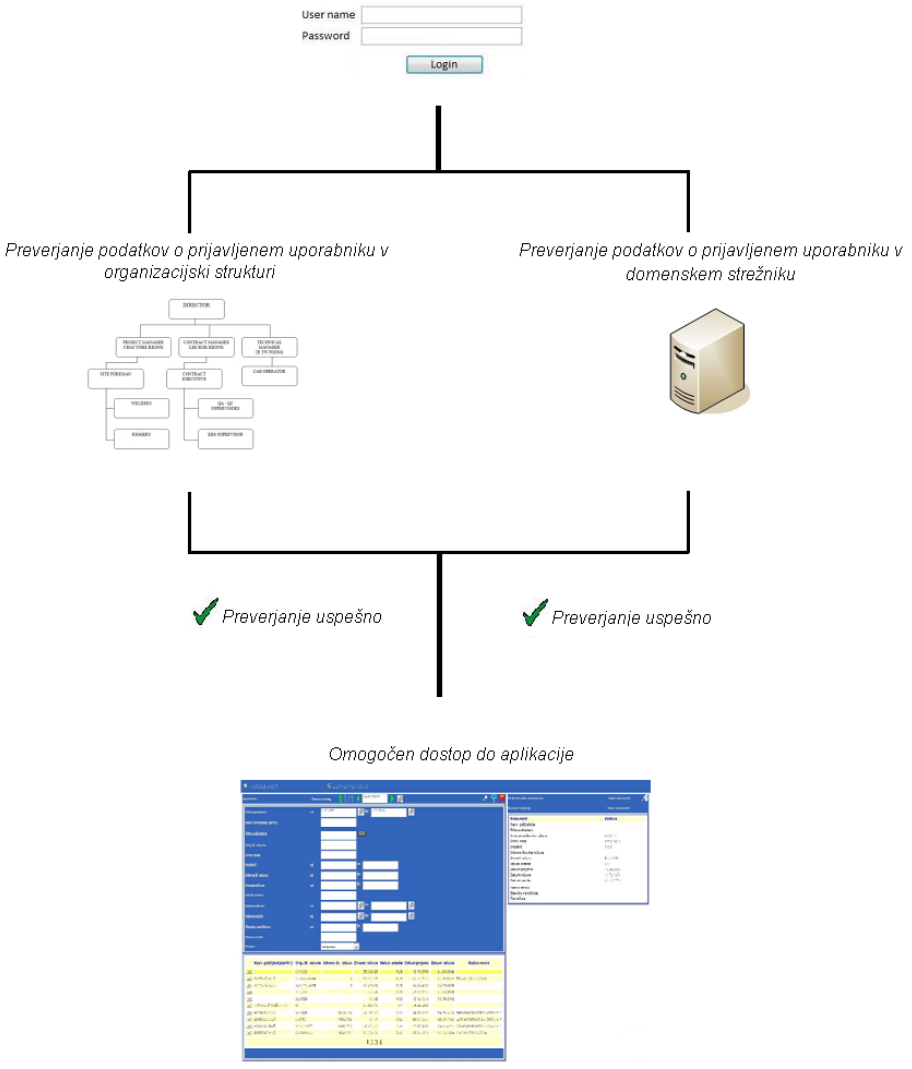


Slika 1: Primer starega iskalnika.

Iskalnik dokumentov vhodne pošte je poleg klasičnega iskalnika imel obenem implementiran tudi modul za dodeljevanje pravic pregledovanja najdenih dokumentov. Vsak uslužbenec tako ni imel vpogleda nad vsemi rezultati iskanja. Oba iskalnika sta omogočala tudi dodatno funkcionalnost objave najdenih dokumentov na portalu, ki je bil izdelan v sklopu celotne informacijske rešitve. Moja naloga je bila preučiti delovanje obeh iskalnikov, najti boljše in optimalnejše rešitve za določene funkcionalnosti, predvsem pa pohitriti iskanje po bazi prejetih dokumentov.

4.1. Opis obstoječe aplikacije - iskalnika, na podlagi katerega sem naredil prenovo.

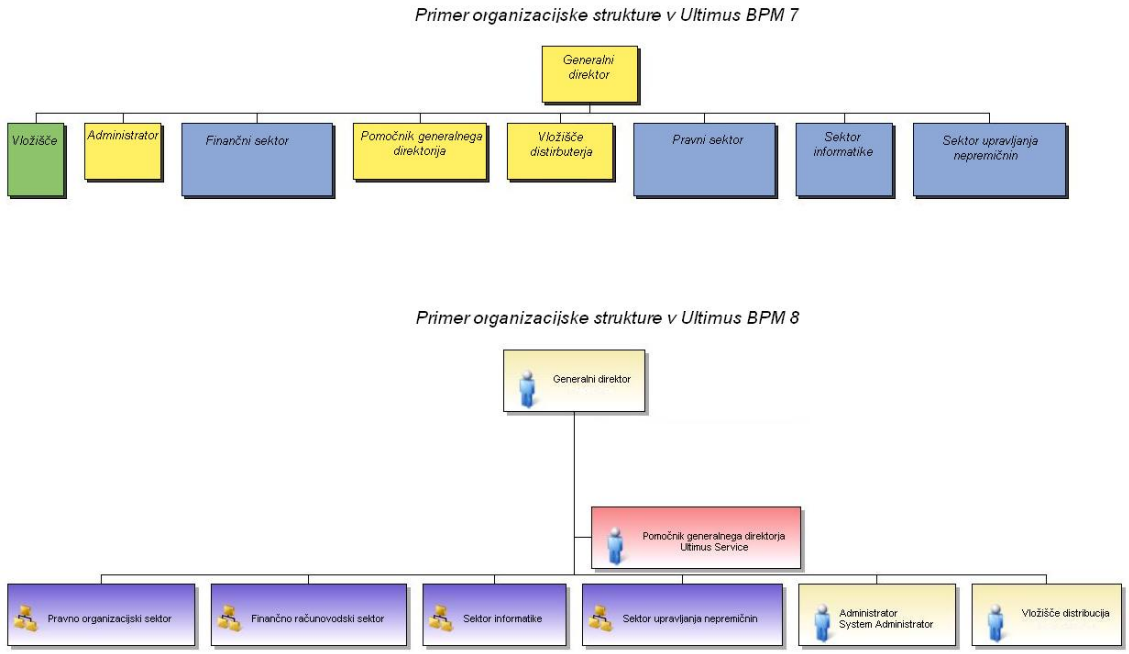
Obstoječa aplikacija je bila izdelana kot spletna aplikacija, ki je vsebovala dva različna iskalnika. Aplikacija je bila dostopna le zaposlenim, s kombinacijo uporabniškega imena in gesla. Vsi zaposleni so bili udeleženci določenega poslovnega procesa v podjetju, torej so bili člani organizacijske strukture podjetja. Organizacijska struktura podjetja je bila vezana na poslovni proces, kjer so vsi zaposleni – udeleženci poslovnega procesa pripadniki organizacijske strukture. Dostop do iskalnika je bil mogoč le v primeru, da je uporabnik, ki je želel dostopati do aplikacije, vključen v organizacijsko strukturo podjetja in ima pravico dostopa. To pomeni, da je uporabnik pripadnik določene skupine v organizacijski strukturi in ima omogočen dostop do iskalnika. Spodnja slika prikazuje način preverjanja dostopa do aplikacije – slika 2.



Slika 2: Preverjanje dostopnih pravic ob prijavi v sistem.

Obstoječ iskalnik je v ozadju deloval na podatkovni bazi Oracle, njegova glavna slabost pa je bilo njegovo počasno delovanje. Počasnost je prišla do izraza zlasti pri veliki količini zadetkov iskanja, saj so se vsi rezultati zapisali v uporabnikovo sejo, kar je slabo vplivalo na odzivnost pri pregledovanju (več strani) rezultatov. V principu je to pomenilo, da se je iskanje opravilo samo enkrat (rezultati so se zapisali v uporabnikovo sejo). Ker pa je bilo rezultatov preveč za prikaz na eni strani, se je uporabniku prikazala opcija prebiranja rezultatov po straneh. Ob kliku na naslednjo stran (rezultatov iskanja) se ni opravilo novo iskanje, ampak so se iz uporabnikove seje prebrali obstoječi rezultati. Slaba odzivnost pa ni predstavljala edine težave, večji problem je v tem primeru predstavljala pravilnost rezultatov. Tak način iskanja nam ne zagotavlja vedno pravilnih – svežih podatkov. Med brskanjem po straneh rezultatov shranjenih v uporabnikovi seji se lahko zgodi, da se določen zapis v podatkovni bazi doda ali posodobi. Razlike nam niso vidne, kajti ne opravimo novega poizvedovanja – iskanja, ampak brskamo po "zastarelih" rezultatih zapisanih v uporabnikovi seji. Torej prva izboljšava pri postopku prenove iskalnika je bila popravljanje načina iskanja. Ob vsaki spremembi strani se je opravilo novo iskanje v podatkovni bazi. Na tak način lahko zagotovimo ažurnost prikazanih rezultatov iskanja ob prehodu med stranmi.

Prehod na novo aplikacijo je obenem predstavljal tudi prehod na novo verzijo orodja za upravljanje s poslovnimi procesi. V mojem primeru je to pomenilo prehod iz verzije 7 na verzijo 8 (torej prehod iz Ultimus BPM Studia 7 na Ultimus BPM Studio 8), kar je za seboj prineslo kar nekaj sprememb. Potrebna je bila implementacija novih vmesnikov za pravilno delovanje komponent nove različice Ultimus BPM orodja z iskalnikom. Največjo spremembo je predstavljala organizacijska struktura BPM orodja (Organisation Chart), ki se je močno razlikovala od predhodne, zato je bilo v novi različici iskalnika potrebno poskrbeti za izgradnjo nove organizacijske strukture. Primer razlike med strukturama je prikazana na sliki 3.



Slika 3: Primer razlike med staro in novo organizacijsko strukturo

Obstoječa aplikacija je poleg iskanja vsebovala tudi modul dodeljevanja pravic pregleda rezultatov iskanja. S prehodom na novo verzijo orodja za upravljanje s poslovnimi procesi je obstoječ modul dodeljevanja pravic, zaradi prenovljene organizacijske strukture, postal neuporaben. Obstoječa funkcionalnost je bila za novo organizacijsko strukturo neprimerna, zato je bila potrebna implementacija nove logike preverjanja pripadnosti določenim skupinam in organizacijskim oddelkom. Na podlagi preverjanja so se uporabniku prikazali rezultati iskanja.

Zahteva naročnika je bila, da bi celotna informacijska rešitev, katere del je tudi iskalnik, zaradi kompatibilnosti z ostalimi moduli delovala na podatkovni bazi Microsoft SQL Server. Obstoječi iskalnik je deloval na podatkovni bazi Oracle 9i, zato je ob prehodu na novo različico iskalnika bilo potrebno poskrbeti za migracijo obstoječih podatkov. Prepisa podatkovne baze ni bilo mogoče izvesti sistematsko, kajti podatkovne baze niso povsem kompatibilne med seboj. Zataknilo se je pri določenih atributih, ki jih nova ali stara podatkovna baza nista podpirali. Za prepis je bilo potrebno izdelati ločeno konzolno aplikacijo, ki je poskrbela za konverzijo podatkov med obema podatkovnima bazama. Posebno pozornost pa je bilo potrebno nameniti pravilnem prepisu podatkov zaradi možnosti popačenosti le teh pri pretvorbi. Za novo različico iskalnika je bilo potrebno kreirati nov podatkovni model, ki bi ustrezal spremenjeni arhitekturi novega iskalnika hkrati pa bi bil primeren za nadaljnji razvoj ostalih modulov.

Prenova iskalnika se je razdelila na dva manjša dela – na prepis podatkovne baze s kreiranjem novega podatkovnega modela in na novo implementacijo iskanja in dodeljevanja pravic.

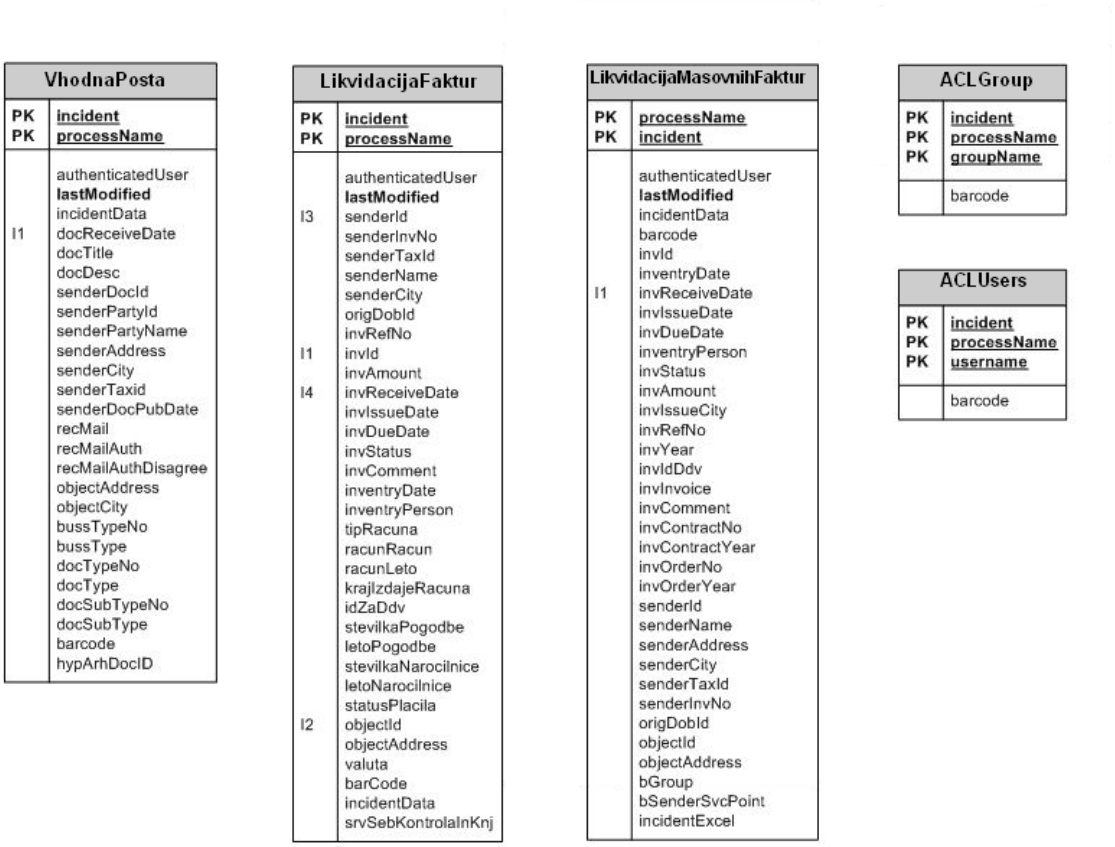
4.2. Kreiranje podatkovnega modela za prenovljen iskalnik in poslovni proces likvidacije masovnih faktur

Celotna obstoječa informacijska rešitev je v ozadju delovala na podatkovni bazi Oracle 9i, zato je ob nastanku nove različice bil potreben prehod na novo podatkovno bazo Microsoft SQL Server. Obstoječi podatkovni model novi aplikaciji ni ustrezal zaradi nekompatibilnosti obstoječih ali novih modulov, obenem pa je prišlo do velikega števila sprememb, kar je posledično privedlo do izdelave novega podatkovnega modela. Kreiranje novega modela je bilo potrebno zastaviti bolj na široko, načrtovati je bilo potrebno tudi nadgradnjo v smislu dodajanja novih modulov in posledično širjenja podatkovne baze. Obstoječi podatki, ki jih je za svoje delovanje uporabljal iskalnik, morajo biti prav tako dostopni ostalim aplikacijam, predvsem poslovnemu procesu likvidacije masovnih faktur v sklopu celotne informacijske rešitve. Pri kreiranju samega modela sem si lahko pomagal z obstoječim modelom iz podatkovne baze Oracle. Pozoren sem moral biti na število in tip atributov, kajti določeni tipi, ki so podprti pri podatkovni bazi Oracle niso podprti pri podatkovni bazi Microsoft SQL in obratno. Določene atribute je bilo potrebno nadomestiti z novimi, primernimi novi različici programa. Posebno pozornost sem moral nameniti pravilni preslikavi starih zapisov v novo podatkovno bazo.

V podatkovni bazi Oracle so bile kreirane tri glavne tabele, ki so hranile podatke glede na njihov izvor ali tip: Vhodna pošta, Likvidacija faktur, Likvidacija masovnih faktur. Poleg teh treh tabel je obstoječ model vseboval tudi tabele, ki so hranile podatke o pravicah nad pregledom rezultatov iskanja. Implementacijo dodeljevanja pravic je v novi različici iskalnika bilo potrebno izdelati na novo, obenem pa ohraniti obstoječe zapise pravic. Za pravilno delovanje je bilo potrebno kreirati strukturo, ki bo primerna za shranjevanje tako novih kot tudi starih podatkov.

Podatkovni model sem zaradi kompatibilnosti izdelal z orodjem Microsoft Visio 2000 [10], kjer sem kot tip podatkovne baze izbral Microsoft SQL – slika 4. Sledilo je kreiranje tabel, z vsemi zahtevanimi atributi in njihovimi tipi. Tu je prišlo do omenjenih težav s kompatibilnostjo med obema bazama. V podatkovni bazi Oracle obstaja podatkovni tip imenovan »CLOB« [7], katerega vsebina je zapisana v binarni obliki. Podatkovna baza Microsoft SQL takega tipa ne pozna, zato je bilo potrebno zapisovati binarne podatke na drugačen način. Podobne težave z nekompatibilnostjo med podatkovnima bazama so se pojavile tudi pri pravilni obliki zapisa datuma, pri čemer se obe podatkovni bazi razlikujeta. Razlika pri datumu pa ni bila le pri obliki zapisa, ampak tudi pri razponu datuma. Podatkovna baza Oracle 9i je v tem primeru bolj široko zastavljena in dopušča vnos zapisov od 1.1.4712 pred našim štetjem do 31.12.9999 [7], v nasprotju z podatkovno bazo Microsoft SQL, ki dopušča vnos zapisov datuma od 1.1.1753 do 31.12.9999 [2]. Obstoječi starejši nepravilni zapisi so v tem primeru predstavljali problematične podatke, na katere sem moral biti pozoren, zlasti pri prepisu podatkovne baze. V času razvoja celotnega projekta je podatkovni model bil nekajkrat posodobljen, zaradi novih zahtev s strani naročnika ali kompatibilnosti z ostalimi moduli. Arhitektura podatkovnega modela je ostala nespremenjena, do sprememb je prišlo le pri imenih določenih obstoječih

atributov ali pri dodajanju novih. Zaradi razširitve funkcionalnosti in nadgrajevanja modulov, je bila v obstoječ podatkovni model dodana tudi kakšna nova tabela.



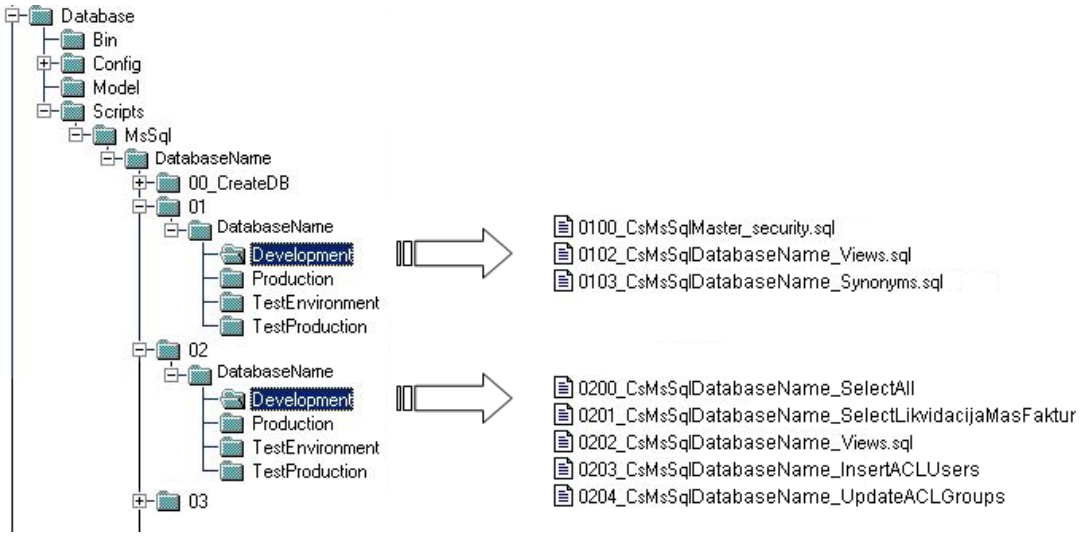
Slika 4: Podatkovni model za podatkovno bazo Microsoft SQL

4.3. Prepis obstoječe podatkovne baze iz sistema Oracle na sistem MS SQL – (skrb za konsistenco podatkov)

Po uspešnem kreiranju novega podatkovnega modela, je sledilo kreiranje podatkovne baze in prepisovanje obstoječih podatkov. Stara podatkovna baza je vsebovala tri glavne tabele, ki so vsebovale podatke glede na njihov izvor ali tip. Poleg glavnih tabel je stara podatkovna baza vsebovala tudi dve pomožni tabeli v katerih so bili shranjeni podatki o pravicah. Glavne tabele so bile precej obsežne, zato sem se odločil za postopno prepisovanje le teh. Za začetek sem iz treh glavnih tabel izbral vzorec velikosti od 50 do 100 zapisov, kar je predstavljalo moje testne podatke. Načrt je bil, da prepisovanje opravim implicitno torej, da povežem novo in staro podatkovno bazo med seboj in opravim direkten prepis podatkov, vendar se je tu zalomilo. Naletel sem na težavo pri nekompatibilnosti med podatkovnimi tipi pri Oracle in Microsoft SQL. Šlo je za zgoraj opisan problem, in sicer podatkovni tip, ki obstaja v okolju Oracle – CLOB se ne preslika implicitno v noben podatkovni tip na podatkovni bazi Microsoft SQL. Podatkovni tip

CLOB vsebuje binarni zapis podatkov, kar se je v mojem primeru uporabljalo za shranjevanje preslikanega prispelega dokumenta. Vsak zapis v podatkovni bazi je vseboval vse podatke, ki so lahko bili razbrani iz preslikanega prispelega dokumenta. Zaradi pomembnosti preslikanega dokumenta je bil zapis slike v podatkovni bazi obvezen. Za uspešen prepis je bilo potrebno najti alternativno rešitev – programsko prepisovanje podatkov.

Postopek prepisa sem opravil eksplicitno, torej preko programa, ki je opravljal pretvarjanje podatkov za vsak zapis posebej. Eksplicitno prepisovanje je dobilo nove razsežnosti, zato sem ga v skladu s politiko podjetja izdelal v obliki ločene konzolne aplikacije. Politika v podjetju je točno določala strukturo takega tipa projekta. Projekt se je v principu ločil na dva dela, od katerega je en del vseboval programsko kodo, drug del pa poizvedbe SQL. Poleg programske kode in poizvedb SQL sem moral zagotoviti tudi ustrezno strukturo map in nastavitvenih datotek. Pozoren sem moral biti tudi na ime projekta in njegovo dolžino (vse v skladu s politiko podjetja). Struktura map je bila zgrajena iz glavne mape, čigar ime je bilo enako imenu projekta. Glavna mapa je nato vseboval podmape, ki so v imenu vsebovali zaporedne številke izvajanj poizvedb SQL, torej 001, 002 … Vsaka podmapa je vseboval datoteke z poizvedbami SQL, katerih prva tri znaka v imenu so morala vsebovati ime mape, sledili so podčrtaji in zaporedne številke datotek, kot prikazuje slika 5. Po pravilnem oštevilčevanju je sledilo še ime datoteke, ki je čim bolj nazorno povedalo, kaj določena datoteka SQL naredi. Tak način imenovanja in oštevilčevanja je bil pomemben zaradi vrstnega reda izvajanja poizvedb SQL. Celotna struktura poizvedb SQL se je izvedla avtomatično s pomočjo programa, ki je na podlagi nastavitvene datoteke izvedel poizvedbe SQL po vrstnem redu glede na strukturo map.



Slika 5: Primer strukture map in oštevilčenih poizvedb SQL.

Za izdelavo programa za eksplicitno prepisovanje sem uporabil .NET 3.5 platformo, aplikacijo pa sem izdelal v okolju Visual Studio 2008. Politika podjetja teži k enotnem razvoju, zato sem prepis podatkov opravil po uveljavljenem postopku. Za komunikacijo s podatkovno bazo sem uporabil določene funkcionalnosti knjižnice Crea.Common, ki so poskrbele za branje in zapisovanje podatkov. Knjižnica Crea.Common združuje velik nabor funkcij, ki se uporabljajo v večini projektov izdelanih znotraj podjetja. Z uporabo omenjene knjižnice se zagotovi vedno enak način delovanja določene funkcionalnosti aplikacije.

Ločiti sem moral implicitni in eksplicitni prepis, zato sem vse poizvedbe SQL, ki so se lahko opravile implicitno shranil v določeno podatkovno strukturo. Vse eksplicitne poizvedbe so poleg klasičnih poizvedb SQL vsebovale tudi datoteko, ki je poskrbela za zagon programa, ki je opravil prepis implicitno. Ob izdelavi aplikacije mi ni bilo potrebno skrbeti za izdelavo grafičnega vmesnika, kajti aplikacija je bila pognana kot modul pri celotnem postopku prepisa. Postopek prepisa je potekal na tak način, da sem vsak zapis iz stare podatkovne baze programsko s pomočjo knjižnice Crea.Common prebral in iz njega izluščil podatke, ki so vsebovali vsebino slike preslikanega dokumenta. Sledila je transformacija prebranih podatkov v tabelo nizov (tabelo podatkovnega tipa bajt). Rezultat prepisa je bila tabela – polje, sestavljena iz niza bajtov. Sledilo je sestavljanje razbitega zapisa nazaj v celoto, le da je sedaj binarni zapis slike bil zamenjan z nizom bajtov, ki je ob pravi transformaciji (v obratni smeri) predstavljal preslikan dokument. Spremenjen zapis je bilo potrebno s pomočjo knjižnice Crea.Common zapisati v novo podatkovno bazo. Postopek branja, transformacije in vpisovanja predstavlja slika 6.

Slika 6: Postopek eksplicitnega prepisovanja podatkov iz podatkovne baze Oracle 9i v podatkovno bazo Microsoft SQL.

Naslednja napaka, na katero sem naletel pri prepisovanju podatkov je bil format datuma. Podatkovna baza Oracle vsebuje drugačen format datuma kot Microsoft SQL. Obenem se med dvema bazama razlikuje tudi razpon datumov, kot je opisano v prejšnjem razdelku. Zaradi takšnih napak, sem se odločil za izdelavo univerzalne funkcije, ki bi služila preverjanju pravilnosti zapisa datuma in njegovega razpona. Funkcija je bila na začetku namenjena zgolj prepisu podatkov. Kasneje sem njeno funkcionalnost razširil in jo

uporabil tudi pri kasnejših transformacijah in preverjanju pravilnosti zapisa datuma pri iskalniku.

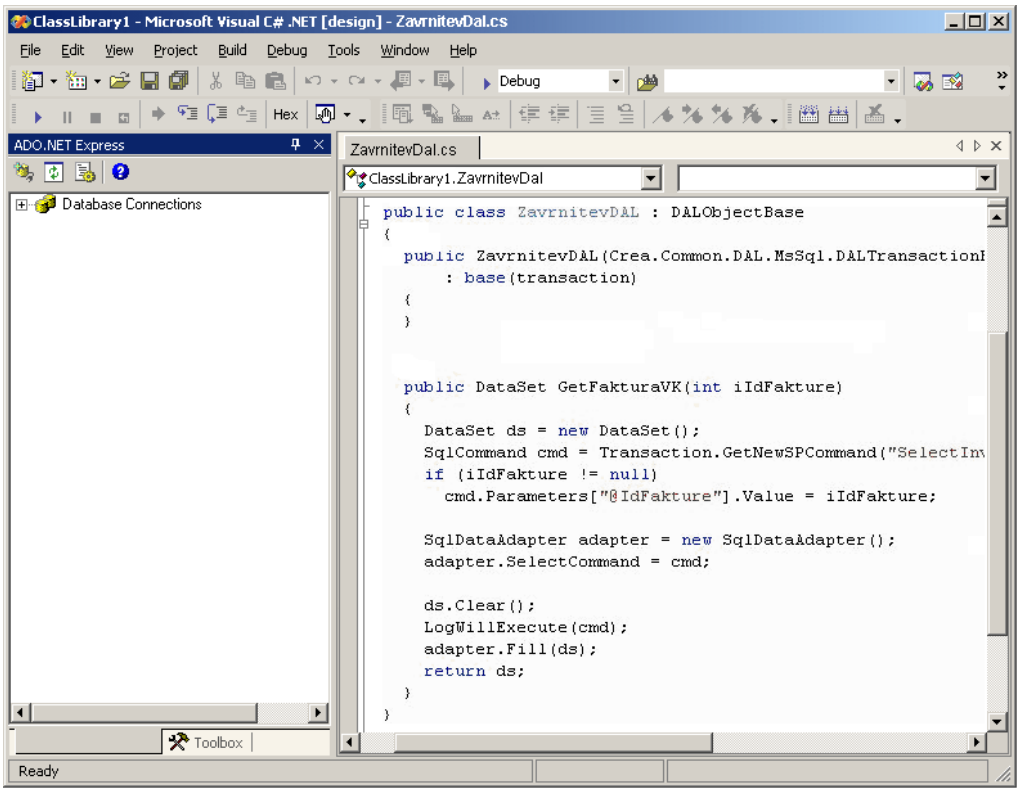
Težave z datumi in nekompatibilnimi podatkovnimi tipi, pa niso bile edine. Poskrbeti je bilo potrebno tudi za vse »klasične« možne napake. Zagotoviti je bilo potrebno, da ja vsebina podatkov enaka pred in po transformaciji, da ni prišlo do vpisovanja praznih polj podatkov v novo podatkovno bazo, da v primeru številk z decimalnimi vrednostmi ni prišlo do težav z postavitvijo pozicije decimalnega mesta itd.

Kot zadnje dejanje v postopku prepisovanja podatkov, sem opravil prepisovanje dveh tabel, ki so bile namenjene dodeljevanju pravic. Star iskalnik je za delovanje uporabljal starejšo različico orodja za upravljanje z poslovnimi procesi, kar je vplivalo na pravice uporabnikov. Dve tabeli, ki sta bili namenjeni dodeljevanju pravic, sta vsebovali podatke glede na organizacijsko strukturo in njihove člane. Problematični so bili obstoječi zapisi, kajti le ti ne bi bili pravilni v novi različici spletnega iskalnika kar bi pomenilo, da obstoječi uporabniki ne bi imeli enakih pravic. Zato sem ob samem prepisu moral preveriti vsebino in poskrbeti za pravilno dodeljevanje (preslikavo) pravic glede na novo organizacijsko strukturo, kajti le na tak način sem lahko zagotovil pravilnost prepisanih podatkov na novem sistemu.

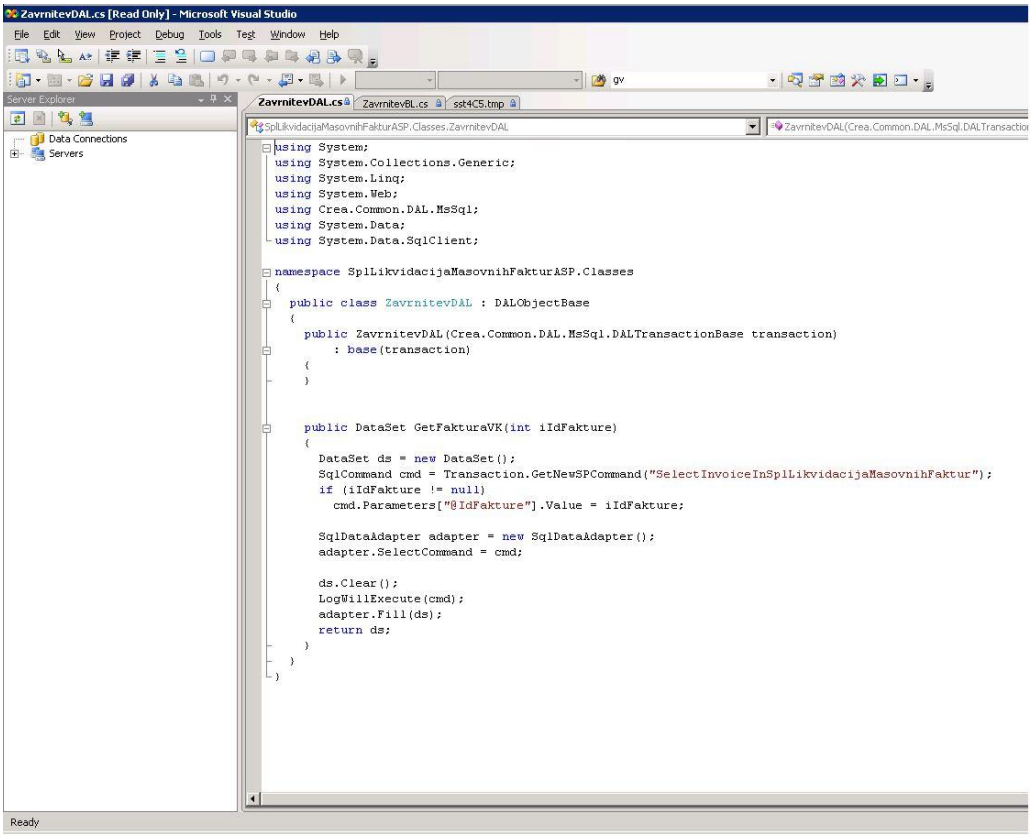
Pri samem postopku prepisa sem se veliko naučil, predvsem pa sem dobil veliko izkušenj o varnosti in pravilnosti pri procesu migracije podatkov.

4.4. Prenova spletne aplikacije iskalnika – opis novega principa iskanja in delovanja

Pogoj pri prehodu na novo verzijo iskalnika je bil, da videz starega iskalnika ostane nespremenjen, obenem pa je velik poudarek bil na hitrosti iskanja. Stara aplikacija je bila napisana v starem okolju Visual Studio (2003) [5], kar je pomenilo da so bile uporabljene komponente, ki so v današnjem času nadomeščene z novimi – bolj uporabnimi. Glede na to, da je moja primarna naloga bila pohitritev delovanja celotne aplikacije je za uspešno izvedbo bila potrebno uporaba novega okolja, nove arhitekture in novejših pristopov gradnje aplikacije. Za novo različico iskalnika sem uporabil .NET 3.5 platformo, aplikacijo sem izdeloval v okolju Visual Studio 2008 [6]. Sliki 7 in 8 prikazujeta primera starega in novega razvojnega okolja – Visual Studio. Prenovo iskalnika sem si razdelil na tri glavna področja: prepis podatkov, iskanje in prikaz rezultatov, dodeljevanje pravic pregleda rezultatov.



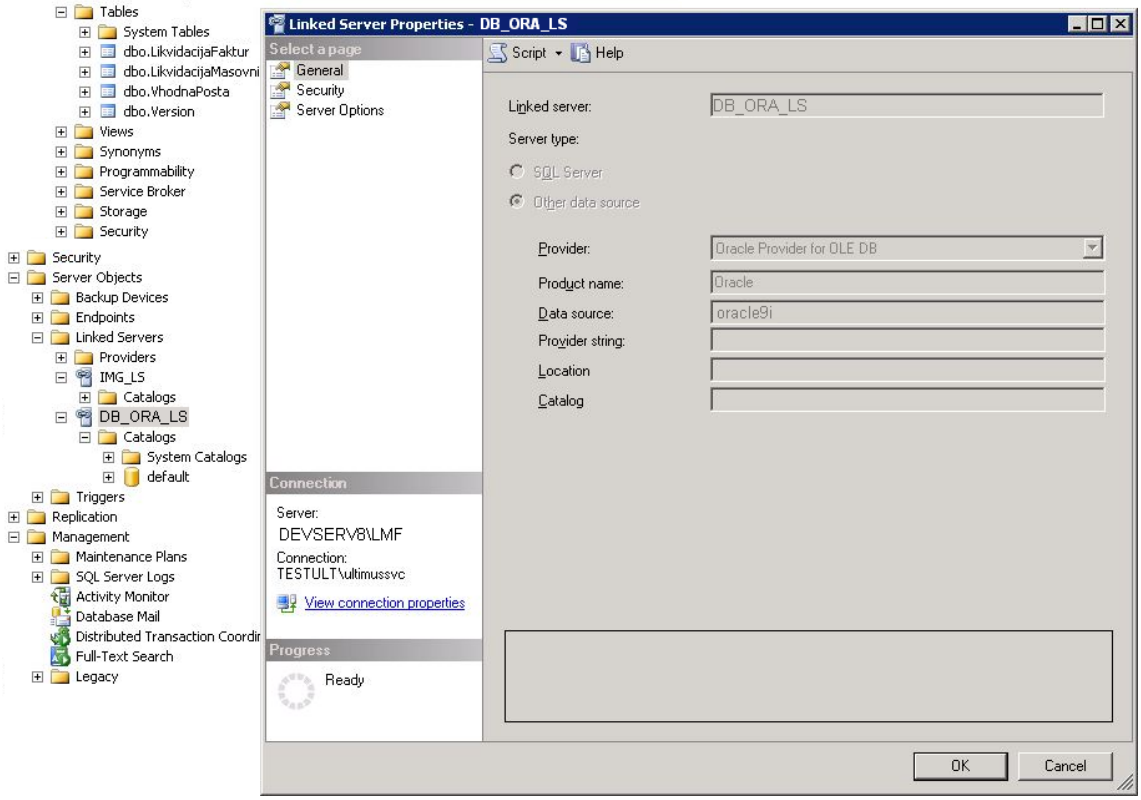
Slika 7: Primer razvojnega okolja Visual Studio 2003



Slika 8: Primer razvojnega okolja Visual Studio 2008

4.4.1. Prepis podatkov (postopek prepisa)

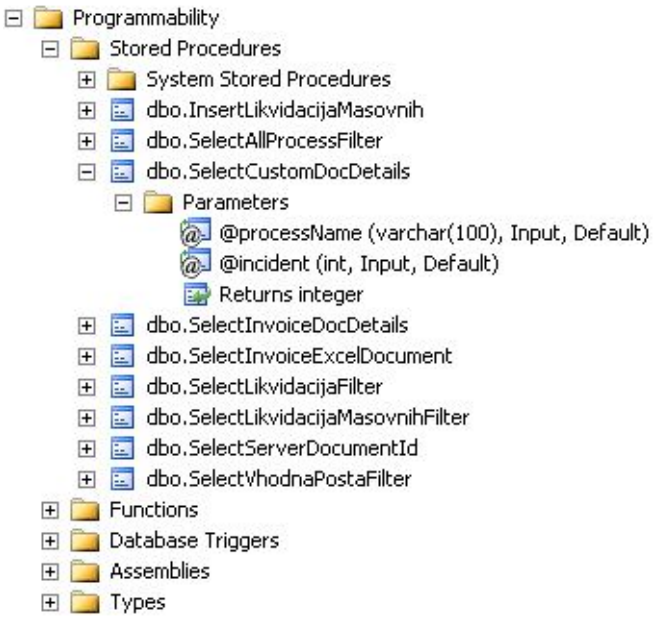
Dela sem se lotil na začetku, torej pri prepisu oz. prenosu obstoječih podatkov iz stare v novo podatkovno bazo, saj je nova različica iskalnika bila zastavljena za delo z podatkovno bazo Microsoft SQL. Pri postopku prepisa podatkov sem dobil dostop do testnih podatkov podatkovne baze Oracle [8]. Povezovanje dveh podatkovnih baz na različnih platformah je predstavljalo novo oviro. Poskrbeti sem moral za dodelitev ustreznih pravic na obeh podatkovnih bazah, da bi lahko dostopal do podatkov in nad njimi izvajal poizvedbe SQL. Zataknilo se je že pri implicitnem poizkusu dostopa iz Microsoft SQL podatkovne baze do prebiranja podatkov na Oracle 9i podatkovni bazi. Za pravilno komunikacijo je bilo potrebno vzpostaviti povezavo med obema strežnikoma, tako imenovani »Linked server« [4]. S pomočjo orodja za upravljanje podatkovne baze – Microsoft SQL Server Management Studio Express [3] sem kreiral »Linked server«, ki poskrbi za vzpostavitev povezave med obema strežnikoma, po kateri se lahko pretakajo podatki iz enega strežnika na drug. Za uspešno vzpostavitev povezave med dvema strežnikoma je potreben določen gonilnik, ki je sposoben na nižjem nivoju vzpostaviti povezavo in zagotoviti nemoten prenos podatkov. V mojem primeru je bil to »OLE DB provider for Oracle 9i«, ki sem ga moral posebej namestiti na strežnik, kjer je bil nameščen Microsoft SQL Server – Slika 9.



Slika 9: Primer povezave »Linked server«, z uporabo omenjenega gonilnika, med podatkovno bazo Oracle 9i in Microsot SQL.

V testne namene sem »Linked server« kreiral ročno in poskušal iz Microsoft SQL podatkovne baze dostopati do podatkovne baze Oracle. Po uspešno prestanem testu, je bilo potrebno na podlagi politike podjetja avtomatizirati postopek vzpostavitve povezave med dvema strežnikoma. Torej, napisati sem moral poizvedbo SQL, ki bo poskrbela za kreiranje povezave »Linked server«. Na podlagi postopka o določeni strukturi map, opisanega v drugem odstavku razdelka 1.3, je bilo potrebno SQL poizvedbo o kreiranju povezave med sistemom Oracle in Microsoft SQL Server umestiti na prvo mesto, takoj po kreiranju nove podatkovne baze. Glede na to, da je postopek prepisa definiran na tak način, da izvede poizvedbe SQL po določenih mapah, je pomembno kdaj kreiramo povezavo med obema strežnikoma. Samo na tak način sem lahko zagotovil, da je povezava (med obema strežnikoma) bila vzpostavljena pred začetkom prepisovanja podatkov. Za začetek prepisovanja je bilo potrebno napisati poizvedbe SQL, ki izvedejo branje iz Oracle podatkovne baze. Prebrane podatke lahko direktno zapišejo v Microsoft SQL podatkovno bazo ali s pomočjo programa podatke najprej preoblikujejo in nato zapišejo.

Vse poizvedbe SQL so bile kreirane v obliki »Stored procedure« – SP. »Stored procedure« je neke vrste funkcija v jeziku SQL, katera izvede poizvedbo SQL. Ob kreiranju SP se le te shranijo v obliki funkcije podatkovne baze nad katero so bile kreirane, kot je razvidno na sliki 10. Dobra prednost takih poizvedb je njena dostopnost (s pomočjo knjižnic) direktno iz programske kode, saj so SP ločene poizvedbe SQL od programske kode. Ločevanje poizvedb SQL od programske kode pripomore k hitrejšemu izvajanju, obenem pa tudi boljši strukturiranosti in lažji berljivosti programske kode. Na podlagi napisanih SP je sledilo prepisovanje podatkov – implicitno, če je le to bilo mogoče, in za tiste bolj kompleksne podatke eksplicitno.



Slika 10: Primer »Stored procedure«, ki sprejme dva parametra in izvede SQL poizvedbo nad podatkovno bazo, kjer je dodana.

4.4.2. Iskanje in prikaz rezultatov

Po uspešnem prepisu podatkov je bilo okolje za začetek prenove iskalnika pripravljeno. Videz novega iskalnika je moral ostati nespremenjen, zato sem se osredotočil na uporabo kar najbolj podobnih gradnikov kot pri predhodni različici iskalnika. Na tak način sem zagotovil nespremenjen videz, a sem moral biti pozoren na prilagoditev določenih nastavitev, da bodo ustrezale novim komponentam. Po končani izdelavi grafičnega vmesnika sem se odločil za postopno izgradnjo spletne strani. Komponente (iz predhodne različice), ki so kompatibilne sem postopoma prenašal na novo različico aplikacije. Tako sem kaj kmalu ugotovil, da bo zaradi nekompatibilnosti starejših komponent z novim okoljem potrebno celotno aplikacijo predelati. Glede na to, da je star iskalnik bil razdeljen na dve ločeni strani, so se določene funkcionalnosti ponavljale na obeh straneh. Ideja za nov iskalnik je bila ta, da bi oba iskalnika združil v enega – eno spletno stran, eno aplikacijo. Problema sem se lotil tako, da sem izdelal glavno stran – »Master page«, ki je delovala kot predloga za obe strani. Predloga je vsebovala vse komponente, ki so enake obema stranema in zagotovila enak videz le teh. Nadaljnji razvoj sem ločil na dve podstrani (glede na izvor prejetih dokumentov) – »Vhodna pošta« in »Likvidacija faktur«. Vsako od strani sem razdelil na manjše dele, glede na funkcionalnost, jih predelal in na koncu sestavil vse skupaj v končno aplikacijo. Najprej sem se lotil skupnih komponent kot so: prijava, napredni iskalnik, prikaz rezultatov in podrobnejši prikaz rezultatov.

Prijava v aplikacijo je delovala na podlagi domenskega uporabniškega imena. Vsak zaposleni je imel svoje uporabniško ime in geslo, katero mu je bilo dodeljeno in katerega je uporabljal v podjetju. Znotraj podjetja je torej obstajala neka domena npr. »podjetje.local« in nek »Active Directory«, kjer so bili shranjeni vsi člani domene, torej vsi zaposleni v tem podjetju. Vsak uporabnik se je s svojim domenskim uporabniškim imenom in geslom lahko prijavil na računalnik, ki je bil dodan v domeno. Na podlagi prijave so bile uporabniku dodeljene določene pravice. Obenem je vsak uporabnik – zaposlen v podjetju, bil član organizacijske strukture podjetja. Pravice uporabe določenih aplikacij so bile omejene glede na položaj in delovno mesto v organizacijski strukturi podjetja. Podjetje je bilo razporejeno na več oddelkov in pododdelkov, kar je pomenilo, da je vsak zaposleni imel določene pravice. Za uporabo aplikacije iskanja je bilo potrebno preverjanje pristnosti uporabnika na podlagi domenskega uporabniškega imena. Poleg preverjanja domenskega uporabnika, je bilo potrebno preverjanje v organizacijski strukturi podjetja. Postopek preverjanja je prikazan na sliki 2. V kolikor je zaposleni, ki je hotel uporabljati iskalnik, pripadal določenemu oddelku ali pododdelku v organizacijski strukturi in v kolikor je določen oddelek ali pododdelek imel pravice do uporabe aplikacije iskalnika, je bilo preverjanje pristnosti zaposlenega uspešno in bila mu je omogočena uporaba iskalnika. Programersko logiko preverjanja pristnosti uporabnika sem implementiral s pomočjo knjižnice »Crea.Common«, kjer sem s pomočjo klicev določenih (znotraj podjetja že uveljavljenih) funkcij dostopal do podatkov organizacijske strukture. V primeru neuspešne prijave, je bil uporabnik o tem ustrezno obveščen in uporaba

aplikacije mu ni bila odobrena. Po uspešni prijavi je sledil glavni del iskalnika – napredno iskanje.

Napredno iskanje je bilo namenjeno preglednejšemu izpisu podatkov, torej samo tistih ki jih želimo videti. Obe podstrani sta vsebovale možnost naprednega iskanja, a se je le to razlikovalo glede na kriterije iskanja, vendar so določene komponente bile enake. Najprej sem se odločil izdelati komponente, ki so enake pri obeh načinih iskanja. Ena izmed takih komponent, ki je bila del funkcionalnosti naprednega iskanja je tudi pregledovanje po datumih, katerega videz sem prevzel od predhodne različice iskalnika. Pregledovanje po datumih prispele pošte, je omogočalo pregledovanje po določenih datumskih intervalih, torej od določenega do določenega datuma, ali pregledovanje po dnevih, mesecih, letih. V vseh primerih je bilo potrebno poskrbeti za pravilen zapis oblike datuma v iskalna polja – potrjevanje iskalnih polj, kajti v nasprotnem primeru bi se iskanje opravljalo po napačnih datumih. Zaradi omejitve velikosti prikazanih podatkov je bil datum iskanja pomembna omejitev, saj sem na tak način zagotovil zgornjo in spodnjo mejo iskanja. Z datumom sem na nek način uporabnika omejil in se s tem zavaroval, da ne bi prišlo do prevelike količine najdenih rezultatov. To bi povzročilo preveliko obremenitev za podatkovno bazo in aplikacijo samo, rezultat tega bi bila upočasnitev delovanja celotne aplikacije.

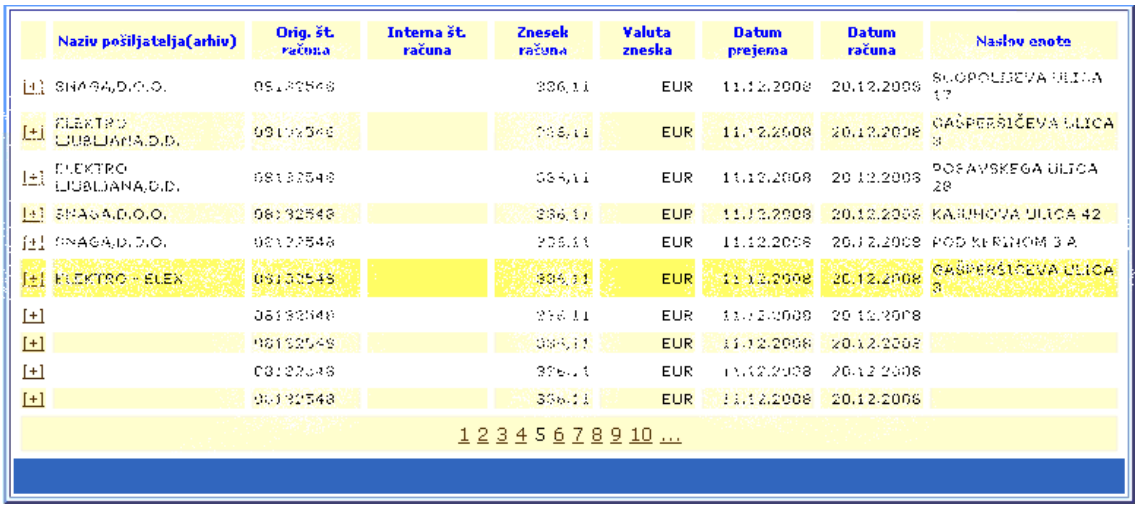
Kriteriji iskanja so bili na obeh podstraneh različni, zato je bilo potrebno narediti dva ločena kriterija. Kriterij sem naredil kot sekcijo, ki jo je bilo možno skriti ali prikazati, glede na željo uporabnika. Polja, ki jih je kriterij moral vsebovati sem pridobil iz predhodne različice iskalnika. Zaradi zagotavljanja hitrejšega iskanja sem spremembe pri iskanju s kriterijem implementiral na tak način, da so direktno vplivale na vsebino poizvedbe SQL. To je pomenilo, da v primeru iskanja brez kriterija so bile vrednosti spremenljivk prazne, kar ni vplivalo na rezultat poizvedbe SQL. V kolikor je uporabnik iskal z določenim kriterijem, so se le te vrednosti prenesle v pogoj poizvedbe SQL in kriteriju primeren je bil tudi rezultat poizvedbe. Tak način iskanja je pripomogel pri hitrosti, kajti vsi pogoji, ki so omejevali iskanje, so se izvedli v poizvedbi SQL na nivoju podatkovne baze in ne v sami aplikaciji. Torej aplikacija ni bila odvisna od kompleksnosti pogojev, njena edina naloga je bila podatke v kriteriju pravilno formirati in posredovati v SP. Rezultat poizvedbe SQL je bil že »prečiščen« rezultat (glede na kriterije), ki ga je aplikacija pravilno prikazala. V tem primeru sem izkoristil možnost porazdeljenega izvajanja operacij, kajti poizvedbe SQL so se izvajale na ločenem strežniku z podatkovno bazo, kar je pomenilo, da strežnik, kjer je bila nameščena aplikacija ni bil dodatno obremenjen. Izvajanje poizvedb na ločenem strežniku od aplikacije same je dobilo poseben pomen v primeru podstrani »Likvidacija faktur«. Kriterij na tej podstrani je vseboval možnost iskanja po zapisih iz dveh tabel – »LikvidacijeFaktur« in »LikvidacijMasovnihFaktur«, kar je pomenilo še večjo zakasnitev pri prikazu rezultata zaradi kompleksnosti same poizvedbe. V tem primeru je poizvedba izvedla unijo (slika 11) nad rezultati iz tabele »LikvidacijaFaktur« in »LikvidacijaMasovnihFaktur« glede na kriterije, ki jih je bilo potrebno upoštevati.

```
WHERE ((lr.processName='LikvidacijaFaktur') OR @processName IS NULL)
  AND ((lr.senderName LIKE '%' + @senderName + '%' )OR @senderName IS NULL)
  AND ((lr.senderId = @senderId )OR @senderId IS NULL)
  AND ((lr.senderInvNo = @senderInvNo )OR @senderInvNo IS NULL)
  AND ((lr.barcode LIKE '%' + @barcode + '%') OR @barcode IS NULL)
  AND ((lr.incident >= @incidentFrom) OR @incidentFrom IS NULL)
  AND ((lr.incident <= @incidentTo)  OR @incidentTo IS NULL)
  AND ((lr.invId >= @invIdFrom) OR @invIdFrom IS NULL)
  AND ((lr.invID <= @invIdTo) OR @invIdTo IS NULL)
  AND ((lr.invAmount >= @invAmountFrom) OR @invAmountFrom IS NULL)
  AND ((lr.invAmount <= @invAmountTo) OR @invAmountTo IS NULL)
  AND ((lr.valuta LIKE '%' + @invValuta +'%')OR @invValuta IS NULL)
  AND ((lr.invReceiveDate BETWEEN @invReceiveDateFrom AND @invReceiveDateTo))
UNION ALL

SELECT
  lm.[processName],
  lm.[incident],
  lm.[barcode],
  lm.[senderName],
  lm.[senderId],
  lm.[origDobId],
  lm.[senderInvNo],
  lm.[invId],
  lm.[invAmount],
  'EUR' AS valuta,
  lm.[invReceiveDate]
FROM [database].[dbo].[LikvidacijaMasovnihFaktur] AS lm
WHERE ((lm.processName='LikvidacijaMasovnihFaktur') OR @processName IS NULL)
  AND ((lm.senderName LIKE '%' + @senderName + '%' )OR @senderName IS NULL)
  AND ((lm.senderId = @senderId )OR @senderId IS NULL)
  AND ((lm.senderInvNo = @senderInvNo )OR @senderInvNo IS NULL)
  AND ((lm.barcode LIKE '%' + @barcode + '%') OR @barcode IS NULL)
  AND ((lm.incident >= @incidentFrom) OR @incidentFrom IS NULL)
  AND ((lm.incident <= @incidentTo)  OR @incidentTo IS NULL)
  AND ((lm.invId >= @invIdFrom) OR @invIdFrom IS NULL)
  AND ((lm.invID <= @invIdTo) OR @invIdTo IS NULL)
  AND ((lm.invAmount >= @invAmountFrom) OR @invAmountFrom IS NULL)
  AND ((lm.invAmount <= @invAmountTo) OR @invAmountTo IS NULL)
  AND ((lm.valuta LIKE '%' + @invValuta +'%')OR @invValuta IS NULL)
  AND ((lm.invReceiveDate BETWEEN @invReceiveDateFrom AND @invReceiveDateTo))
```

Slika 11: Primer »Stored procedure«, kjer se izvede unija nad rezultati dveh poizvedb nad ločenima tabelama z upoštevanjem pogojev.

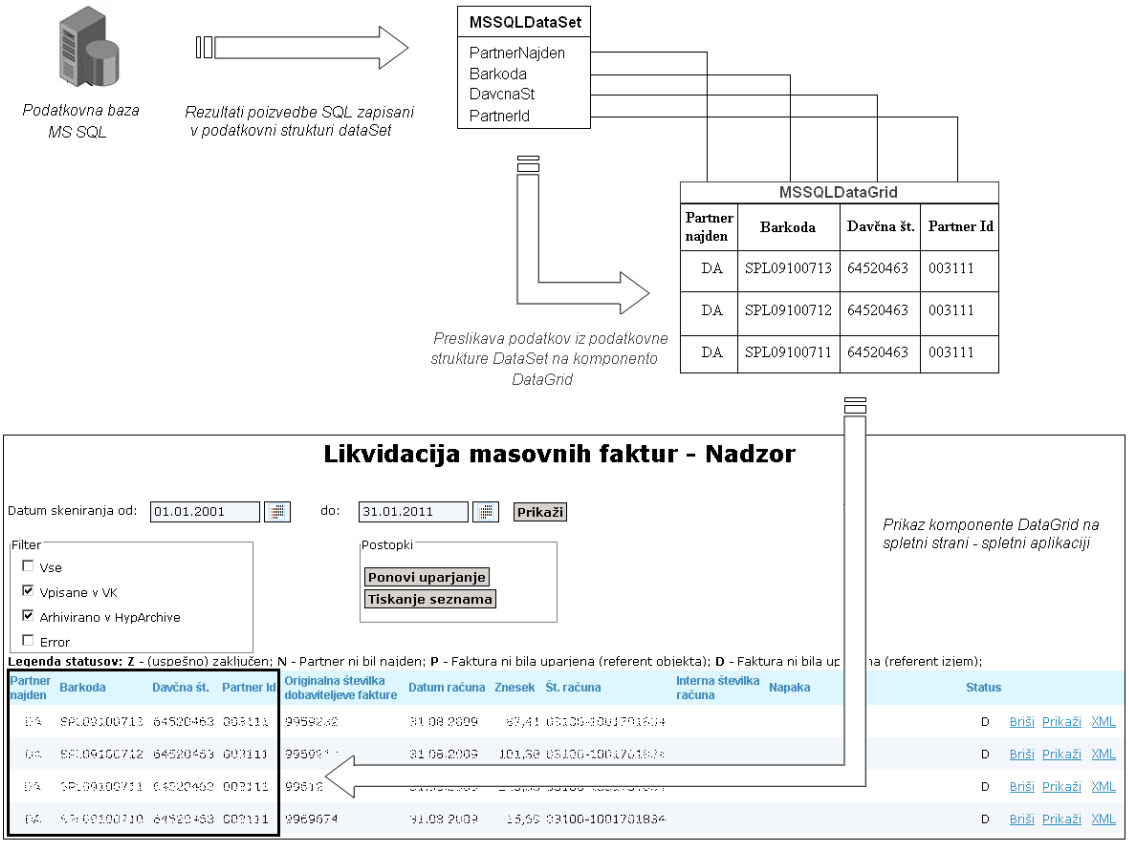
Sam postopek iskanja se je izvajal s pomočjo knjižnice »Crea.Common«, ki je vsebovala funkcionalnosti za komunikacijo s podatkovno bazo. S klicem željene funkcije iz aplikacije se je izvršil klic določene SP (na nivoju podatkovne baze), kot rezultat klica je funkcija vrnila podatkovno strukturo tipa »DataSet«, ki je vseboval vse željene podatke napisane v poizvedbi določene SP. Vsi rezultati iz podatkovne strukture »DataSet« so bili nato prikazani v spodnjem delu aplikacije. V kolikor je bilo rezultatov preveč za prikaz na eni strani, so se na prvi strani prikazali začetni rezultati, vsi ostali pa so bili shranjeni v strukturi »DataSet«, ki se je ob prehodu na novo stran rezultatov na novo napolnil – opravilo se je novo iskanje. Na tak način sem zagotovil prikaz vedno svežih rezultatov – konsistentnost podatkov (slika 12).



Slika 12: Prikaz najdenih zadetkov glede na kriterij iskanja.

Preslikavo podatkovne strukture »DataSet« na spletno stran sem izvedel s pomočjo komponente »DataGrid« [1], ki je na prvi pogled podobna tabeli, z naslovi in polji za prikaz podatkov. Grafična komponenta »DataGird« za prikaz potrebuje nek vir podatkov, ki je v mojem primeru bil podatkovna struktura »DataSet«, z vsebino rezultatov iskanja. Za lažjo povezljivost med komponento »DataGrid« in podatkovno strukturo »DataSet« sem moral biti pozoren pri rezultatu poizvedb. Imena stolpcev pri komponenti »DataGrid« so se morala ujemati z imeni stolpcev pri rezultatu poizvedb. Rezultat poizvedbe je bil shranjen v podatkovno strukturo »DataSet« kar je pomenilo, da so se stolpci pri obeh komponentah, »DataGrid« in »DataSet«, med seboj ujemali. Slika 13 prikazuje primer povezovanja dveh omenjenih komponent. Z ujemanjem stolpcev v obeh komponentah so se podatki med pravilno preslikali. V primeru večje količine prikazanih podatkov je berljivost le teh zelo pomembna, zato sem večjo berljivost podatkov pri izpisu zagotovil s pravilnim formatiranjem izpisa. To je veljalo predvsem za prikaz datumov in decimalnih števil. Formatiranje izpisa je ena od lastnosti komponente »DataGrid«, prav tako kot sortiranje podatkov po določenih stolpcih in pregledovanje strani rezultatov. Poleg prikaza osnovnih podatkov, sem kot dodatno funkcionalnost implementiral tudi prikaz vseh lastnosti najdenih rezultatov. V praksi to pomeni, da je uporabnik opravil iskanje in za rezultat dobil veliko število zadetkov. Najdeni zadetki so vsebovali samo najpomembnejše podatke, kajti v nasprotnem primeru so rezultati neberljivi. To je uporabniku omogočalo hiter pregled najpomembnejših lastnosti najdenih rezultatov. V kolikor bi želeli videti vse podrobnosti bi morali izbrati – klikniti na določen zadetek. Prikazovanje podrobnosti rezultatov iskanja sem implementiral tako, da se je ob kliku na najdeni rezultat uporabniku na strani prikazal dodatni modul z podatki. Slika 14 prikazuje primer podrobnosti izbranega rezultata. Šlo je za novo komponento tipa »DataGrid«, ki se je prikazala na desni strani iskalnika, kjer so se izpisale vse podrobnosti izbranega zadetka. Funkcionalnost prikazovanja sem naredil s pomočjo »JavaScript« funkcije, ki je za vsak zapis v rezultatu iskanja dinamično generirala novo povezavo. Vsak zapis je vseboval nek enoličen ključ – id, ki je bil vsebovan v povezavi. Na tak način sem ob vsakem izboru

vedel kateri rezultat je bil izbran. Ob kliku na to povezavo – rezultat iskanja, se je izvedla določena poizvedba SQL v obliki SP na nivoju podatkovne baze. Rezultat te poizvedbe je bila tabela podrobnosti, ki je napolnila vsebino zgoraj omenjene komponente »DataGrid«. Podatkovni vir tabele podrobnosti je bila ponovno struktura »DataSet«, čigar podatki so bili rezultat izvedene poizvedbe.



Slika 13: Primer preslikave podatkov »DataSet« in »DataGrid« in prikaz komponente »DataGrid« na spletni strani.

Slika 14: Prikaz podrobnosti o najdenem zapisu – dokumentu.

4.4.3. Dodeljevanje pravic pregleda rezultatov

Kot zadnji del pri prenovi iskalnika sem se lotil pravic pregleda rezultatov iskanja. Prikaz rezultatov iskanja je bil omejen glede na pripadnost zaposlenega v organizacijski strukturi podjetja. Starejša različica iskalnika je za dodeljevanje uporabljala določene komponente, ki so bile z novo aplikacijo nekompatibilne, zato sem se odločil za novo izdelavo modula za dodeljevanje pravic.

Pravice so bile razdeljene glede na štiri glavne zahteve naročnika in sicer:

- glede na skupino, katere član je uporabnik,
- glede na eksplicitne pravice določenega uporabnika,
- glede na prejemnika (če je prijavljen uporabnik tudi prejemnik),
- glede na pravice administratorja

Vsak uporabnik je kot rezultat iskanja dobil vse zadetke v primeru da je izpolnjeval enega od zgoraj naštetih pogojev. Za pravilno prikazovanje rezultatov, je bila potrebna implementacija modula, ki bo preverjal ali je za prijavljenega uporabnika eden od štirih zgoraj naštetih pogojev izpolnjen. Glavni del modula je bila funkcionalnost preverjanja pripadnosti uporabnika v določeni skupini. Za pravilno preverjanje članstva sem potreboval podatke iz organizacijske strukture podjetja. S pomočjo knjižnice »Crea.Common« sem dostopal do podatkov organizacijske strukture podjetja in na ta način preveril pripadnost uporabnika v določeni skupini, obenem pa dobil tudi njegove skupine (skupine katerih član je prijavljeni uporabnik). Sledilo je preverjanje vseh skupin, ki imajo pravico pregledovanja rezultatov. To preverjanje se je opravilo s pomočjo poizvedb nad vsebino tabele »ACLGroups«, ki je hranila zapise o pravicah določene skupine. V kolikor se je prijavljeni zaposleni znašel v skupini, ki je imela pravice, mu je

bilo omogočeno prikazovanje vseh rezultatov. V nasprotnem primeru bi imel vpogled do tistih rezultatov, nad katerimi je imel pravice. Najmočnejša od vseh pravic je bila seveda ta, da je prijavljeni zaposleni bil član skupine upravljalcev – administratorjev. V tem primeru dodatno preverjanje ni bilo potrebno. Za vse ostale tri pogoje to pravilo ni veljalo, zato je bilo potrebno preverjati vse ostale možnosti.

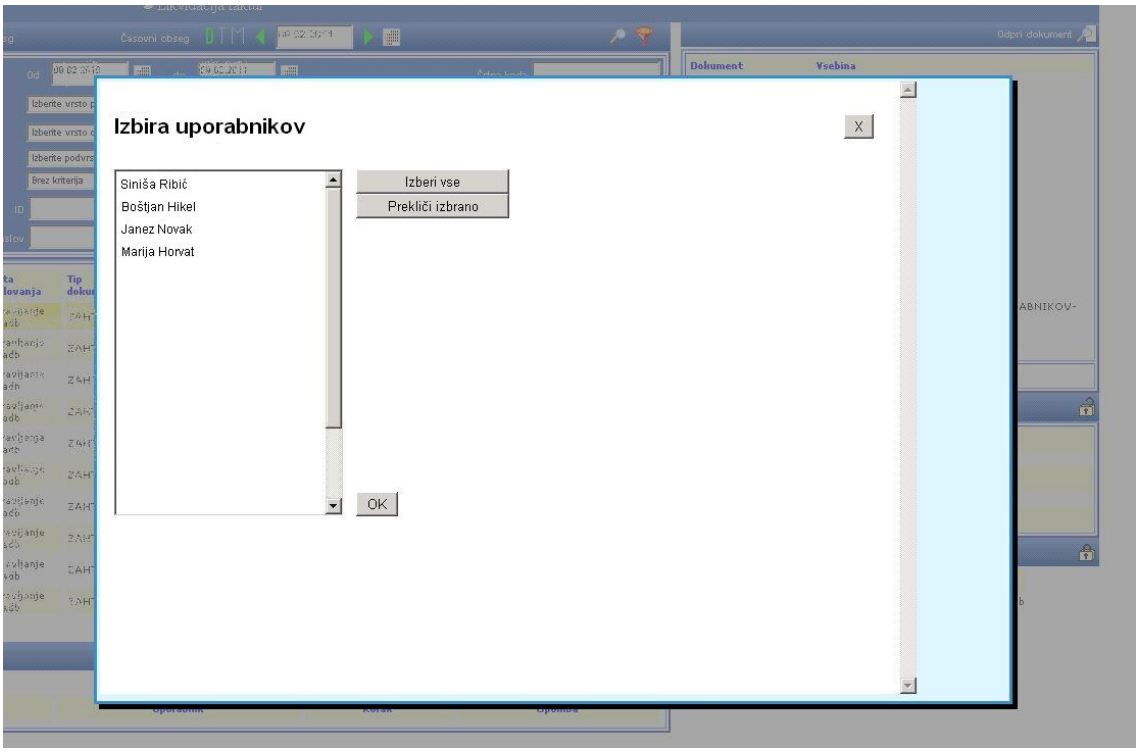
Naslednji korak pri pravilnem prikazu rezultatov iskanja je bilo preverjanje eksplicitne pravice uporabnika. To je pomenilo, da je prijavljeni zaposleni bil član skupine, ki ni imela pravice nad pregledom rezultatov iskanja, istočasno pa mu je bila dodeljena pravica (posameznemu uporabniku), s strani upravljalca - administratorja, do vpogleda nad določenimi rezultati. Torej tak uporabnik je imel eksplicitno dodeljeno pravico s strani upravljalca in članstvo v skupinah v tem primeru ni imelo pomena. Podatki o eksplicitnih pravicah nad določenimi rezultati so bili shranjeni v podatkovni bazi, v tabeli uporabnikov – »ACLUsers«.

Zadnja pravica pregleda rezultata iskanja je bila, da je bil določen dokument dodeljen določenemu zaposlenemu v postopku elektronske preslikave dokumenta in vpisa v podatkovno bazo. V tem primeru je zaposlenemu avtomatsko bila dodeljena eksplicitna pravica nad pregledom določenega dokumenta (kot rezultat iskanja), kajti prijavljeni zaposleni je bil označen kot prejemnik prispelega dokumenta.

Prioriteta pogojev si je sledila na tak način, da je najmočnejši pogoj bil članstvo uporabnika v skupini upravljalcev. Sledilo je preverjanje eksplicitne pravice zaposlenega, nato preverjanje, če je prijavljeni zaposleni tudi prejemnik in nazadnje preverjanje članstva v določenih skupinah (najnižja prioriteta). V kolikor je eden izmed pogojev bil izpolnjen, je uporabnik videl rezultate iskanja. Pogoji so vplivali na vse rezultate iskanja (v primeru članstva v določenih skupinah ali pravic upravljalca), ali pa na določene zapise (v primeru eksplicitnih pravic in v primeru, da je bil prijavljen zaposleni tudi prejemnik).

Implementacija preverjanja pravic pa je bila le del celotne funkcionalnosti modula dodeljevanja pravic. Moja naloga je obenem bila izdelati modul, s katerim bo možno tudi dodeljevati pravice pregledovanja nad določenimi rezultati iskanja. Modul dodeljevanja pravic je bil namenjen upravljavcem – administratorjem, zato sem v testne namene svojega testnega uporabnika v okviru organizacijske strukture dodal v skupino upravljavcev. Na tak način sem dobil privilegij upravljavca in pravico do prikazovanja modula dodeljevanja pravic, ki je za vsak prikazan rezultat prikazal tudi pravice pregleda. Sledila je implementacija dodajanja pravic, katero sem realiziral na dva načina, in sicer: dodajanje uporabnika ali dodajanje skupine. Pri implementaciji dodeljevanja pravic sem si pomagal z modulom izdelanim znotraj podjetja, ki je z dostopom do organizacijske strukture prikazal okno (s pomočjo »JavaScript«), kjer so bile razvrščene vse skupine in vsi uporabniki. Na podlagi izbranega načina dodajanja (uporabnik ali skupina) je uporabnik lahko dodeljeval pravice. Po končani izbiri dodajanja pravic nad pregledom rezultatov iskanja določenemu članu ali skupini, so se spremembe zapisale v podatkovno bazo in prikazale na strani. Sama funkcionalnost dodajanja pavic je bila povezana z zapisi v podatkovno bazo. Šlo je za zapisovanje v dve tabeli, ki sta bili ob prehodu na novo

podatkovno bazo prepisani. Obstoječi zapisi do bili pravilno preslikani, kar je pomenilo, da so se stare pravice obdržale. Ob dodajanju novih pravic (slika 15) se je izvedla določena poizvedba SQL, ki je poskrbela za nov zapis v tabelo, glede na to ali smo dodali pravico uporabniku ali skupini.



Slika 15: Primer dodeljevanja pravic določenim uporabnikom.

Na enak način je bila implementirana funkcionalnost brisanja določenih uporabnikov ali skupin. V primeru izbire brisanja se je pojavilo še obvestilo, kjer je uporabnik potrdil ali prekinil brisanje.

Vsi podatki o pravicah so bili prikazani v obliki tabele, kjer je za vsak zapis (pravico določenega uporabnika ali skupine) obstajala tudi možnost brisanja le te. Brisanje je bilo implementirano s klikom na gumb, ki je vseboval dinamično generirano povezavo. Povezava se je generirala dinamično glede na določen rezultat iskanja. Ob kliku na povezavo brisanja se je izvedel klic funkcije, ki je posledično izvedel poizvedbo SQL, ta pa je poskrbela za brisanje določenega zapisa v podatkovni bazi. Glede na to, da se je funkcionalnost dodajanja in brisanja pravic izvajala neposredno na podatkovni bazi, je to pomenilo hitro odzivnost za uporabnika. Takoj ko so uporabniku bile dodeljene pravice je bila sprememba vidna pri pregledovanju rezultatov, obenem pa so nove pravice stopile v veljavnost nemudoma. Enako je veljalo v primeru brisanja pravic.

5. Likvidacija masovnih faktur

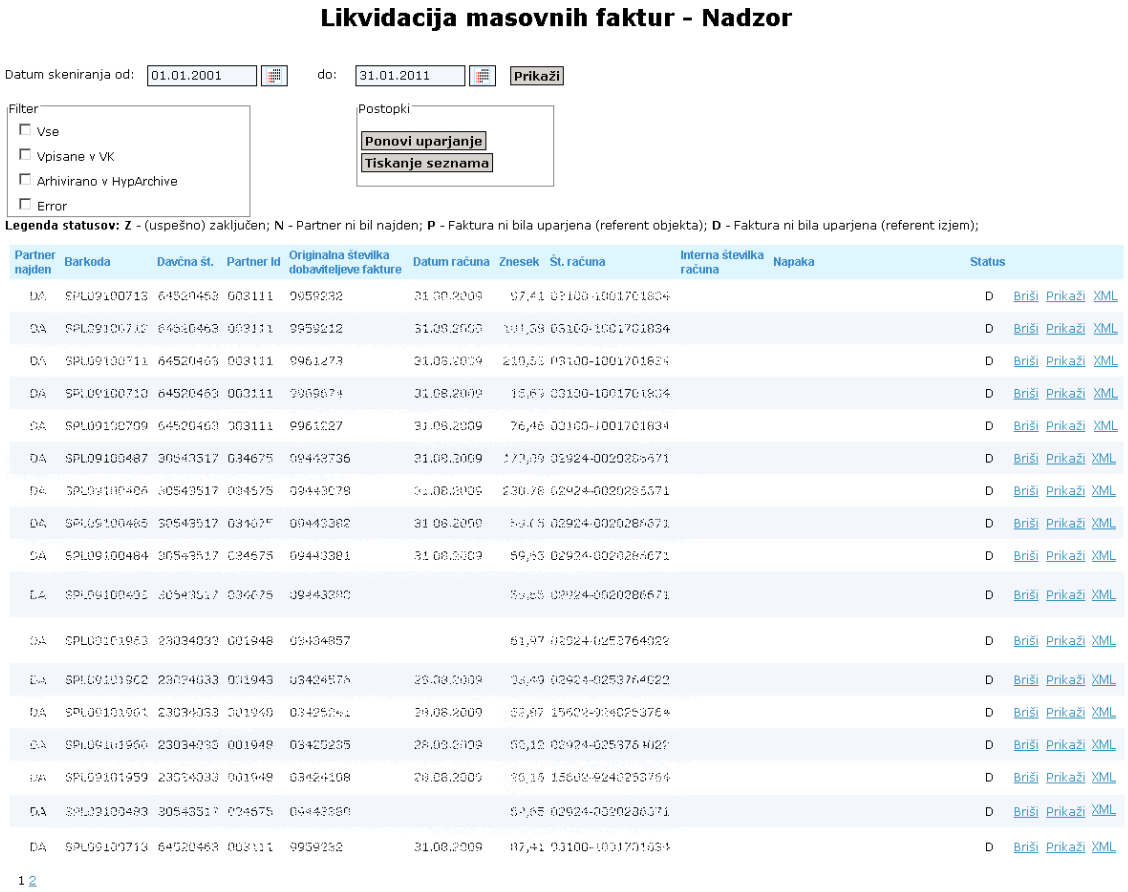
Informacijska rešitev, v sklopu katere sem izvedel prenovo iskalnika, je bila precej obsežna in sestavljena iz več delov. Eden takih je bil tudi izdelava poslovnega procesa likvidacije masovnih faktur. Celoten poslovni proces je bil zastavljen precej na široko, moja naloga pa je bila le del tega. Šlo je za postopek likvidacije prejetih faktur za obračunavanje tekočih mesečnih stroškov stanovanjskih enot. Vzdrževalci nepremičnin običajno skrbijo za več stanovanjskih naselij, vsako naselje pa vsebuje veliko število stanovanj. Proces preverjanja plačila je zaradi števila prejetih dokumentov toliko bolj kompleksen. Poslovni proces je bil zaradi lažje predstavitve razdeljen na dva dela, pri čemer je prvi del predstavljal spletno aplikacijo – nadzorni modul, ki je omogočal uvoz in iskanje (uvoženih) faktur. Druga polovica poslovnega procesa pa je predstavljala njegovo funkcionalnost – avtomatizacijo zavrnitve masovnih faktur, ki je bila sestavljena iz več korakov. Koraki poslovnega procesa so bili v mojem primeru predstavljeni kot uporabniški vmesniki, implementirani kot spletne strani.

Vsi poslovni procesi so bili modelirani s pomočjo BPM (Business Process Management) orodja. V mojem primeru je bilo to orodje Ultimus BPM Suite 8.2, ki je bilo v podjetju že uveljavljeno. Za lažjo predstavitev poteka poslovnega procesa sem s strani naročnika programske opreme dobil točna navodila – funkcionalno specifikacijo, ki je poleg podrobnega opisa zajemala tudi primere uporabniških vmesnikov.

5.1. Nadzorni modul likvidacije masovnih faktur

Naslednji izziv v sklopu celotne informacijske rešitve je bila izdelava tako imenovanega nadzornega modula. Nadzorni modul je bil izdelan kot spletna aplikacija, njegova naloga je bila omogočiti nadzor in avtomatizacijo pri procesu dodajanja – uvažanja in elektronske obdelave novega svežnja prejetih masovnih faktur. Vsak prejet dokument je elektronsko obdelan – preslikan in v obliki datoteke tipa XML shranjen na vnaprej določeno lokacijo na strežniku. Vsak dokument je nato s pomočjo nadzornega modula uvožen v sistem. Funkcionalnost uvoza poskrbi za preverjanje preslikanega dokumenta XML in s pomočjo uparjanja vsebino dokumenta vpiše v določeno tabelo v določeni podatkovni bazi. Uparjanje dokumenta se izvaja na podlagi vnaprej določenih kriterijev, ki so določeni s strani naročnika. V kolikor je bilo uparjanje uspešno se je izvedel vpis elektronske oblike prejetega dokumenta. Za uspešen postopek uvoza je poleg uparjanja in vpisovanja v tabele v podatkovni bazi bilo potrebno poskrbeti tudi za arhiviranje elektronskega dokumenta. V sklopu izdelave celotne informacijske rešitve je bil izdelan tudi arhiv dokumentov, ki je hranil vse prejete dokumente preslikane v elektronski obliki. Naloga nadzornega modula je poleg uvoza bila tudi nadzor nad uvoženimi fakturami. Torej ali je bila faktura pravilno preslikana, ali je bila pravilno arhivirana itd. Funkcionalnost nadzornega modula je omogočala pregled slike in dokumenta v preslikani obliki XML, ter v primeru napak

ponovni uvoz določenega prispelega dokumenta. V sklopu nadzornega modula pa je bil implementiran tudi iskalnik s kriterijem, ki je omogočal iskanje po vseh uvoženih dokumentih. Primer nadzornega modula je prikazan na sliki 16. Nadzorni modul je torej omogočal popolni nadzor nad postopkom uvoza prejetih masovnih faktur.

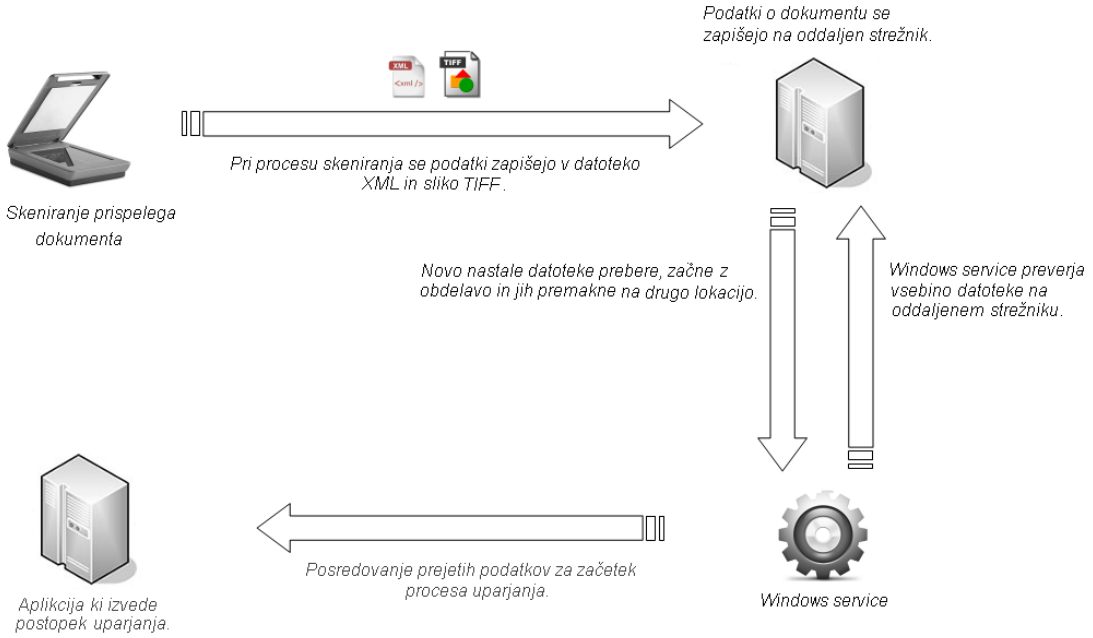


Slika 16: Nadzorni modul s prikazom obdelanih masovnih faktur v določenem časovnem obdobju.

5.1.1. Implementacija uvoza in uparjanja masovnih faktur

Postopek uvoza masovnih faktur je pomenil začetek prebiranja mape, na oddaljenem strežniku, ki je vseboval elektronsko preslikan prejeti dokument. V sklopu elektronske preslikave dokumenta se je kreirala datoteka tipa XML. Preslikana datoteka je vsebovala vse preslikane podatke in binarni zapis slike elektronsko obdelanega dokumenta. Zaradi dinamične implementacije je bil postopek uvoza izdelan kot proces operacijskega sistema Windows – »Windows service«. Implementacija postopka uvoza datotek je pomenila avtomatizacijo procesa, kajti proces WS je v ozadju tekel neprestano. Torej izdelal sem proces, ki je v ozadju neprestano preverjal določeno mapo, kjer so se po uspešni

elektronski preslikavi shranjevale datoteke XML preslikanih dokumentov prispelih masovnih faktur. V kolikor se je v mapi pojavila nova datoteka XML je proces WS začel z postopkom prebiranja. To je pomenilo, da je aplikacija prebrala vse podatke, ki so pomembni za postopek uparjanja, ki je sledil. Poleg podatkov je datoteka XML vsebovala tudi binarni zapis, ki je ob pravilnem dekodiranju prikazal sliko preslikanega dokumenta v formatu »TIFF«. Pri prebiranju datotek tipa XML sem moral posebno pozornost nameniti klasifikaciji, ki je prišla v poštev pri nadaljnjem procesu uparjanja. Poleg tega sem moral skrbeti tudi za pravilno prebiranje vseh zapisov, kajti pri postopku elektronske preslikave lahko pride do napak, ki lahko povzročijo popačenost podatkov in lahko slabo vplivajo na nadaljnje postopke. Ko sem imel vse podatke iz datoteke XML prebrane v pravilnem formatu, je sledilo uparjanje (slika 17).



Slika 17: Primer postopka uparjanja prejetega svežnja masovnih faktur.

Uparjanje je bil postopek povezovanja novih dokumentov z zapisi v obstoječih podatkovnih bazah, na podlagi česar bi bil postopek uvoza označen kot uspešno zaključen. Ker je bil postopek uparjanja precej zapleten sem za lažje razumevanje dobil s strani naročnika diagram poteka (slika 18), ki mi je bil v pomoč za boljšo predstavo. Sam postopek uparjanja je bil povezan tudi z zalednim sistemom, ki je bil lociran pri naročniku. Za uspešno preverjanje sem torej potreboval podatke iz zalednega sistema, ki sem jih dobil na podlagi vnaprej izdelanih SP. Dostop do le teh sem si zagotovil s kreiranjem poizvedbe »SQL View«, ki mi je omogočal prilagojen pogled nad podatki, na

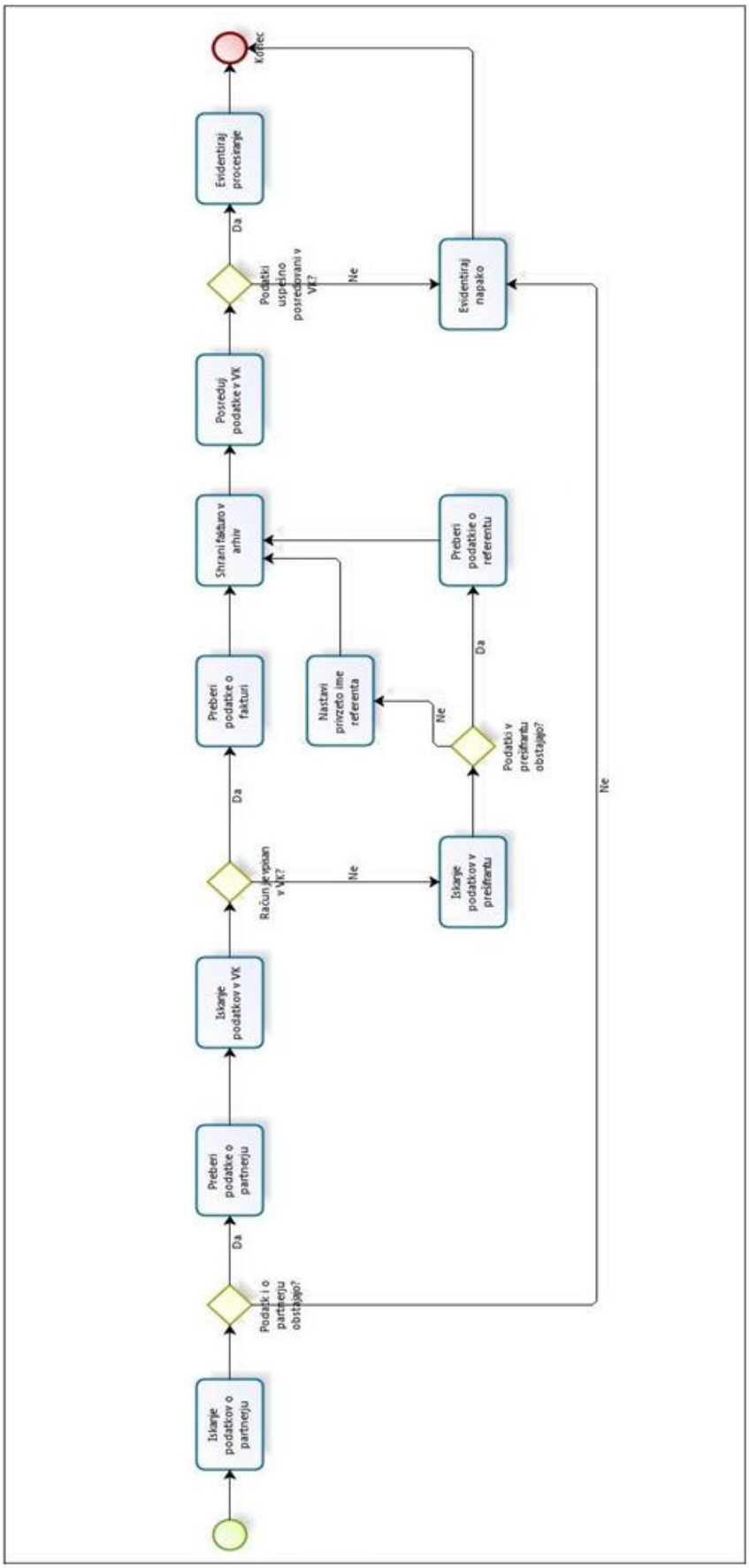
podlagi česar sem opravil preverjanje v zalednem sistemu naročnika. Postopek uparjanja sem na podlagi zahtev in diagrama poteka razdelil na štiri glavne dele:

- preverjanje v katero kategorijo partnerjev spada prispeli dokument,
- iskanje podatkov o prispelem dokumentu v tabeli »VhodnaKnjiga« v zalednem sistemu,
- iskanje podatkov v tabeli »presifrant« zalednega sistema
- iskanje podatkov o referentu

Prvi korak pri postopku uparjanja je bila identifikacija, za kakšno vrsto plačila storitev gre. Obstajale so štiri glavne skupine partnerjev glede na storitev: elektrika, vodovod, plin, ogrevanje. Na podlagi preverjanja partnerjev v tabeli šifrantov, je bilo mogoče dobiti določene podatke o partnerju (npr: njegovo ID številko in podobno). Pridobljene podatke sem potreboval za nadaljnji uspešen postopek uparjanja. Naslednji korak v procesu uparjanja je bil povezovanje z zalednim sistemom. Podobno kot pri procesu prenove iskalnika sem tudi tu moral narediti določene SP preko katerih sem lahko dostopal do podatkov v zaledni bazi. Te SP so se razlikovale od ostalih, kajti njihova naloga je bila izdelati pogled – »SQL View« nad tabelo v zaledni bazi. S pomočjo poizvedbe tipa »SQL View« sem si zagotovil prikaz samo tistih podatkov, ki jih za potrebe uparjanja potrebujem. V testne namene sem dobil kopijo podatkovne baze zalednega sistema, ki pa se ni izkazala za najboljšo, kajti kasneje je prišlo do velikih sprememb, kar je posledično slabo vplivalo tudi na mojo aplikacijo. V podatkovni bazi zalednega sistema je obstajala tabela z imenom »VhodnaKnjiga«. V kolikor je tabela »VhodnaKnjiga« vsebovala zapis o dokumentu, ki sem ga trenutno obdeloval, je to pomenilo, da je prispela faktura že plačana. V tem primeru je bil postopek uparjanja za trenutni dokument uspešno zaključen. Za uspešen zaključek obdelave dokumenta je bilo potrebno dodati sliko, pridobljeno iz elektronsko obdelane fakture, v arhiv, hkrati pa v tabeli »VhodnaKnjiga« posodobiti – vpisati podatke o črtni kodi prejete fakture skupaj s povezavo do slike v arhivu. S tem vpisom je bil postopek uparjanja elektronsko preslikanega dokumenta končan. Vendar pa to je bil najbolj optimalen scenarij, kjer je šlo vse po načrtu. Težava se je pojavila, v kolikor v zalednem sistemu v tabeli prispelih dokumentov – »VhodnaKnjiga« ni bilo najdenega zapisa o dokumentu, ki sem ga trenutno obdeloval. V tem primeru se je opravilo dodatno iskanje podatkov v tabeli »prešifrant«. Ključ za iskanje je bila številka odjemnega mesta, ki se je nahajala v datoteki XML. V kolikor je podatek o številki odjemnega mesta, ki je bil ključ za iskanje podatkov o referentu, najden v tabeli »prešifrant« je to pomenilo, da je uparjanje uspelo. Na podlagi uspešnega uparjanja so se v tabelo »VhodnaKnjiga« vpisali podatki o referentu in črtni kodi, s tem pa tudi slika fakture v arhiv. V nasprotnem primeru, ko podatki o referentu niso najdeni, so se v tabelo »VhodnaKnjiga« vpisali podatki o privzetem referentu in povezava do preslikanega dokumenta v arhivu.

Vsi elektronsko preslikani dokumenti se poleg vpisovanja v tabelo »VhodnoKnjiga« vpisujejo tudi v mojo podatkovno bazo, natančneje v tabelo imenovano »LikvidacijaMasovnihFaktur«. Težava je bila v tem, da je bila tabela kreirana že na

samem začetku, v postopku prenosa podatkov iz stare podatkovne baze v novo. Zaradi kompatibilnosti s poslovnim procesom sem moral obstoječo tabelo imenovano »LikvidacijaMasovnihFaktur« preimenovati v tabelo »LikvidacijaMasovnihFakturArhiv«. Na tak način sem zagotovil kompatibilnost pri kasnejši implementaciji poslovnega procesa likvidacije masovnih faktur, obenem pa je prenovljen iskalnik deloval nemoteno z razliko v tem, da je sedaj iskal po tabeli z drugačnim imenom – »LikvidacijaMasovnihFakturArhiv«.



Slika 18: Predstavlja primer diagrama poteka za postopek uparjanja.

5.1.2. Implementacija iskanja uvoženih masovnih faktur

Nadzorni modul omogoča tudi funkcijo iskanja po elektronsko uvoženih preslikanih dokumentih v podatkovni bazi. Namen iskalnika je nadzor nad uvoženimi dokumenti in možnost preverjanja napak nastalih pri procesu uvoza. Iskalnik torej na podlagi poizvedb SP opravi iskanje v tabeli z imenom »LikvidacijaMasovnihFaktur«, kjer se ob uvozu zapisujejo vsi podatki trenutno obdelovanega dokumenta. Število zadetkov pri iskanju je omejeno glede na datum uvoza prejetih faktur. Za lažje in bolj pregledno iskanje je iskalnik vseboval tudi kriterij iskanja, ki je zajemal štiri možnosti iskanja:

- prikaz vseh,
- prikaz neuspelega uparjanja s tabelo »VhodnaKnjiga«,
- prikaz neuspelega vpisovanja v arhiv,
- prikaz neuspešnih uvozov

Prikaz zadetkov v primeru prikaza vseh je najbolj enostaven, kajti ni potrebno dodatno preverjanje, kar pomeni, da se preko klica poizvedbe SP izpišejo vsi zadetki najdeni v podatkovni bazi, ki ustrezajo podanemu časovnemu pogoju (datumu uvoza). V primeru izpisa zadetkov z neuspelim uparjanjem v tabeli »VhodnKnjiga« je stvar malo bolj kompleksna. Tu je potrebno s pomočjo, v prejšnjem poglavju omenjenih povezav na zaledni sistem, preveriti uspešnost uparjanja. Uspešnost je v našem primeru prepoznana po atributu »črtna koda«, ki v tabeli »VhodnKnjiga« vsebuje zapis v kolikor je uparjanje uspelo in ne vsebuje zapisa – je prazno polje v kolikor uparjanje ni bilo uspešno. Podobna zahtevnost poizvedbe je v primeru preverjanja neuspelega vpisa v arhiv. Arhivski sistem teče na zalednem strežniku in ob vsakem uvozu prejete fakture se le ta doda v arhiv. V podatkovni tabeli »VhodnKnjiga« se kot rezultat uspešnega vpisa v arhiv vpiše povezava do dokumenta v arhivu. V kolikor torej v podatkovni tabeli »Vhodna knjiga« za določeno uvoženo fakturo obstaja zapis o povezavi na dokument v arhivu vemo, da je vpisovanje v arhiv bilo uspešno (slika 19). Kriterij za prikaz vseh neuspešnih uvozov torej s pomočjo poizvedbe SP preveri atributa za arhiv in črtno kodo v podatkovni tabeli »Vhodna knjiga«. V kolikor zgoraj omenjena atributa ne vsebujeta zapisov to pomeni, da uvoz ni uspel in na podlagi tega prikažemo rezultate, ki ustrezajo temu pogoju.

Slika 19: Primer uporabe filtra za prikazovanje določenih uvoženih masovnih faktur nadzornega modula.

Način iskanja v sklopu izdelave nadzornega modula je bil implementiran tako, da so se rezultati iskanja prikazovali na komponento »DataGrid«. Glede na to, da je iskalnik vseboval tudi kriterij, ki je v določenih primerih potreboval več časa za prikaz rezultatov, sem se odločil za drugačen pristop. V podatkovni bazi sem vedno opravil iskanje z omejitvami datuma, (glede na uporabnikove zahteve) kriterija pa v tem primeru nisem upošteval. Na tak način sem vedno dobil izpis vseh uvoženih prejetih faktur v obsegu datuma, ki ga je uporabnik določil ne glede na napake – kriterij. Nato sem uporabil komponento »DataView«, s katero sem naredil selekcijo pri prikazu podatkov na komponenti »DataGrid«. Nad komponento »DataView« sem torej upošteval uporabnikov kriterij in na tak način prikazal samo določene rezultate iskanja, ki so ustrezali izbranemu kriteriju. V primeru prikazov večjega števila zadetkov sem zaradi boljše preglednosti implementiral tudi prikazovanje rezultatov po straneh.

Poleg izpisa podrobnosti določenega zapisa sem za vsak zapis implementiral tudi povezavo na datoteko XML, ki je bila kreirana ob elektronski preslikavi dokumenta, sliko dokumenta, ter brisanje določenega zapisa. Pri implementaciji prikaza slike in datoteke XML določenega dokumenta, sem naletel na težave. Pri prikazovanju vsebine datoteke XML sem moral poskrbeti za pravilno prebiranje same strukture, kajti datoteka XML je zgrajena kot neke vrste drevo, ki vsebuje manjša poddrevesa. Težave sem imel zaradi same strukture datoteke, ki je zgrajena ob elektronski preslikavi. Podobna naloga me je čakala pri prikazu slike, kjer sem moral v testne namene implementirati svojo različico arhiva.

Naloga arhiva je, da hrani podatke o vsakem prispelem dokumentu v elektronski obliki (preslikan dokument). V testne namene sem izdelal implementacijo arhiva, ki je bil zmožen prikazati sliko pridobljeno iz datoteke XML. Za pravilno simulacijo arhiva je bilo potrebno pravilno pretvoriti zapis, ki se je nahajal v vsakem XML dokumentu in je

predstavljal sliko preslikanega dokumenta v sliko tipa TIFF. Pretvorba je pomenila dekodiranje slike v pravilen format, ki ga je brskalnik znal prikazati.

Povezava za brisanje najdenega zapisa je bila namenjena predvsem tistim zadetkom, ki so vsebovali napake. V primeru brisanja le teh je to pomenilo, da se postopek elektronske preslikave ponovi. Nadzorni modul je poleg vseh funkcij omogočal tudi možnost tiskanja najdenih zadetkov. V tem primeru je šlo za prikaz trenutnega seznama rezultatov, v novem oknu. S tehnične strani je to pomenilo, da sem moral na novo kreirati določene komponente – »DataGrid« in »DataView«, obenem pa sem moral rezultate prikazati z določenimi omejitvami širine in višine, glede na format primeren za tiskanje (A4). Glede na način postavitve spletnih strani je to pomenilo, da sem moral imeti dve strani tipa »Master page«, kjer je ena predstavljala nadzorni modul, druga pa prilagojeno kopijo namenjeno strani za tiskanje najdenih rezultatov. To je pomenilo, da je stran, ki je bila namenjena za prikaz rezultatov za tiskanje bila grafično okrnjena – prikazana brez uporabe »CSS« videza.

5.2 Poslovni proces likvidacije masovnih faktur

Izdelava novega poslovnega procesa je predstavljala zadnji del v sklopu celotne informacijske rešitve. Šlo je za poslovni proces likvidacije masovnih faktur, ki je omogočal likvidacijo prejetih masovnih faktur za obračun tekočih mesečnih stroškov stanovanjskih enot. V tem primeru ni šlo za nadgradnjo obstoječega poslovnega procesa ampak za izdelavo novega poslovnega procesa, ki bi olajšal in pohitril sam postopek likvidacije. V sklopu izdelave novega poslovnega procesa sem se srečal z novimi orodji in novimi pristopi. Ker je šlo za nov poslovni proces sem dobil točna navodila – funkcionalno specifikacijo, kjer so bili opisani vsi koraki procesa in izdelane maske uporabniških vmesnikov. Maske so predstavljale videz posameznih korakov.

Poslovni proces je sestavljen iz korakov, ki tvorijo celoto. To pomeni, da je nek poslovni proces zaključen šele takrat, ko se je postopek odvil od prvega do zadnjega koraka. V mojem primeru so bili koraki predstavljeni kot uporabniški vmesniki – spletne strani, kjer je uporabnik na vsaki strani – v vsakem koraku opravil določeno nalogo v sklopu dokončanja poslovnega procesa. Moj poslovni proces je vseboval štiri uporabniške korake in en avtomatski korak. Začetni korak je bil uvoz masovnih faktur z uporabo nadzornega modula, ki je sprožil kreiranje incidentov poslovnega procesa. Kreiran incident je pomenil začetek poslovnega procesa in je deloval kot instanca le tega. Vsakič, ko se je uspešno uvozil nov dokument se je avtomatsko kreiral nov incident. S pomočjo orodja »Ultimus BPM Studio« so se prožili novi incidenti in zaključevali stari. Vsi koraki in vsa funkcionalnost poslovnega procesa je potekala preko orodja »Ultimus BPM Studio«. Seveda je to pomenil nov zalogaj zame, kajti nikoli prej se nisem srečal z upravljanjem poslovnih procesov (ang. *BPM, Business Process Management*). Da je celotna stvar lahko stekla nemoteno je bilo potrebno poskrbeti za marsikaj. Poleg samega poslovnega procesa

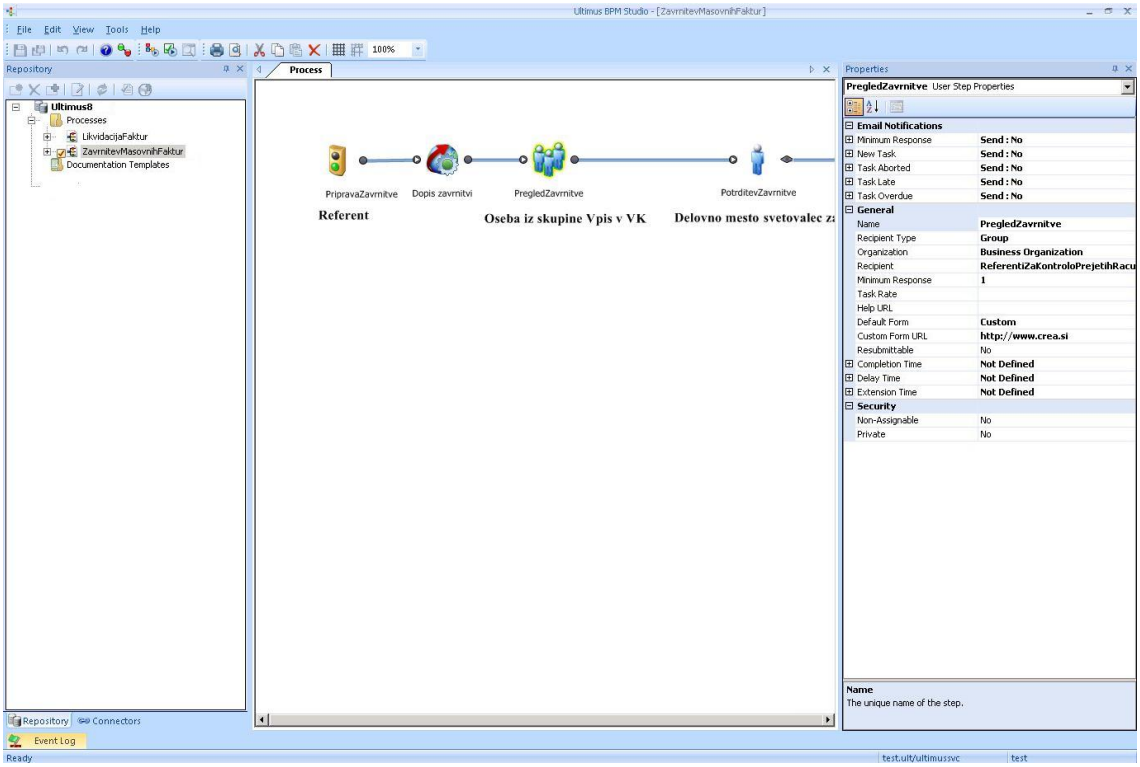
je bila tu še komunikacija med aplikacijo in orodjem »BPM«, implementacija in povezava »BPM« z podatkovno bazo, implementacija korakov – spletnih vmesnikov. Glede na to, da se podjetje, kjer sem opravljal praktično izobraževanje s tem ukvarja že vrsto let so mi njihove izkušnje in znanje prišle prav pri reševanju mojih »začetniških« problemov.

5.2.1. Spoznavanje orodja »Ultimus BPM Studio 8«

Velik del moje naloge je pri tem delu opravljalo orodje »Ultimus BPM Studio« [9], zato sem se moral najprej spoznati z njim. Za začetek sem dobil dostop do spletnega portala kjer se je nahajala vsa dokumentacija zgoraj omenjenega orodja [11]. Orodje »Ultimus« je zelo obsežno in sestavljeno iz več različnih delov tako, da pokriva več področij – od kreiranja in načrtovanja poslovnega procesa do izvajanja le tega. Da bi vse skupaj lažje razumel sem na podlagi znanja, ki sem ga dobil na spletnem portalu izdelal testni poslovni proces. Šlo je za enostaven primer z enim samim korakom, ki je kot rezultat izpisal vneseno vrednost določenega atributa. Vse skupaj je bilo videti precej enostavno, a temu ni tako. Ker je orodje »Ultimus« precej obsežno sem se naloge lotil po kosih glede na njegovo funkcionalnost. Najprej sem se osredotočil na kreiranje poslovnega procesa s pomočjo orodja iz paketa »Ultimus«.

Orodje za načrtovanje in izdelavo poslovnih procesov (angl. *Ultimus Process Designer)* vsebuje grafični vmesnik, ki je na prvi pogled močno podoben orodju »Visual Studio« (slika 20). Pri uporabi orodja za načrtovanje poslovnih procesov ne gre za programiranje ampak je to orodje namenjeno grafičnemu načrtovanju. Orodje vsebuje gradnike poslovnega procesa, različne tipe korakov – avtomatske in pogoje ter povezave med njimi. Vsak proces vsebuje najmanj tri korake, pri čemer je prvi vedno začetni, ki je lahko sprožen avtomatsko – s pomočjo procesa operacijskega sistema Windows – »Windows service« ali s pomočjo spletnega procesa – »Web service«. Drugi korak je običajno človeški, kar je predstavljeno z masko – spletna stran tipa »HTML«. Drugi korak je lahko tudi avtomatski, v tem primeru gre ponovno lahko za proces operacijskega sistema Windows ali za spletni proces. Zadnji korak je vedno enak in predstavlja konec instance poslovnega procesa. Vsi vmesni koraki lahko vsebujejo avtomatske korake imenovane »FlowBots«, ki so namenjeni avtomatiziranim procesom kot so pošiljanje pošte, vpisovanje podatkov v podatkovno bazo, generiranje raznih poročil, dokumentov (Microsoft Word, Excel) itd. Poleg avtomatskih korakov lahko poslovni proces vsebuje tudi ocenjevalni korak (angl. *Evaluation Step*), ki se na podlagi vnaprej določenega poslovnega pravila zna odločiti kateri bo naslednji korak, ki se bo izvedel v sklopu poslovnega procesa. Koraki so med seboj povezani na tak način, da ima vsak korak določenega svojega naslednika – ima določen korak, ki se bo izvedel za njim. V primeru ocenjevalnega koraka, kjer je naslednji korak znan na podlagi poslovnega pravila, so koraki med seboj povezani z navideznimi povezavami. Eden izmed bolj kompleksnih korakov poslovnega procesa je korak »*Junction«*. Korak »*Junction«* ima več funkcionalnosti in se uporablja v primerih, ko želimo združiti več vzporednih procesnih

korakov v enega samega ali obratno, ko želimo procesni korak razdružiti. Korak »*Junction«* se prav tako lahko uporablja za izvajanje akcij (sproži začetek določenega koraka) ali za odločanje – dodeljeni so mu robni pogoji, na podlagi katerih se zna odločiti za naslednji korak.



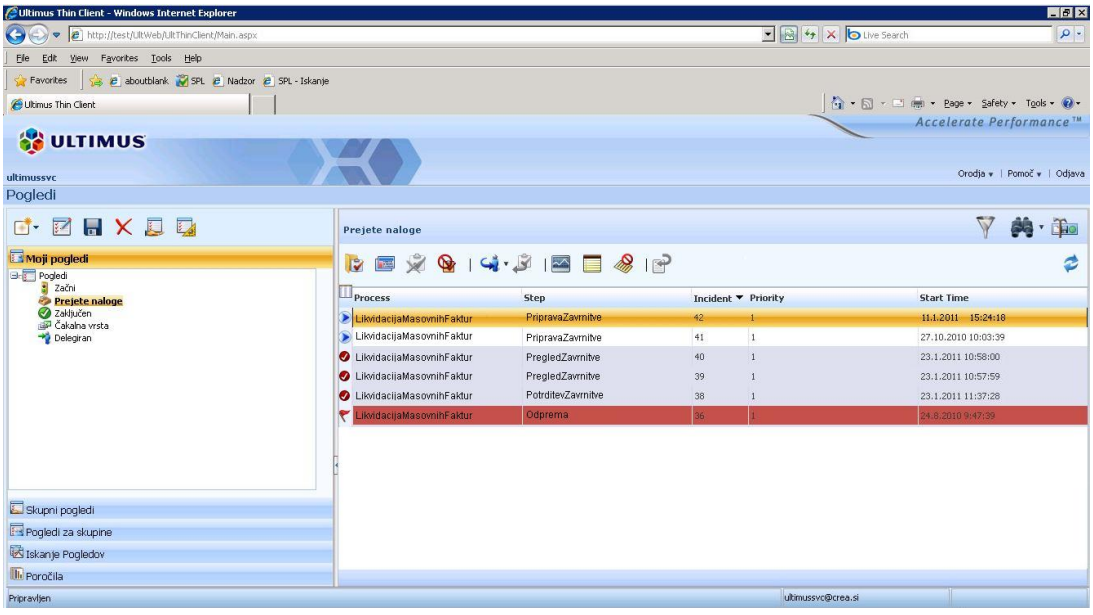
Slika 20: Primer načrtovanja poslovnega procesa »LikvidacijaMasovnihFaktur« z uporabo orodja Ultimus Process Designer.

Poleg načrtovanja in uporabe korakov nam orodje za načrtovanje omogoča tudi cel kup drugih funkcionalnosti, ki so vezane na korake poslovnega procesa. Ena takih lastnosti koraka je tudi pravica do začetka. Določen korak lahko začne samo določena uporabniška skupina, v sklopu organizacijske strukture podjetja. V primeru avtomatskih korakov je potrebno določiti tudi dejanja, ki se avtomatsko izvajajo, torej če gre za zagon nekega WS, je potrebno določiti, kdo ima pravice do zagona le tega in to lastnost pripisati avtomatskem koraku. V primeru ocenjevalnih korakov je poslovno pravilo obvezen pogoj, kajti na podlagi tega se opravi pravilno odločanje za naslednji korak. Orodje »Process Designer« nam poleg vsega naštetega omogoča tudi mnogo dodatnih možnosti, ki pa jih jaz za potrebe svojega poslovnega procesa nisem potreboval.

Ko imamo poslovni proces kreiran nam orodje »Process Designer« omogoča tudi testiranje le tega. Na tak način se lahko izognemo napakam pri nadaljnjem postopku, saj poslovni proces testiramo že v fazi načrtovanja. Torej ko je naš poslovni proces kreiran in ko so testi uspešno prestani je le ta pripravljen za delo. Kot zaključek pri kreiranju

poslovnega procesa sledi izvoz na »repozitorij« orodja »BPM«. S tem dejanjem je naš poslovni proces pripravljen za uporabo z orodjem »Ultimus Thin Client«, ki nam omogoča nadzor nad izvajanjem poslovnega procesa preko uporabniškega vmesnika.

»Ultimus Thin Client« je spletna aplikacija, ki v ozadju deluje na podlagi poslovnega procesa, ki se izvaja s pomočjo poslovne logike. Namenjena je uporabnikom – zaposlenim, ki za opravljanje svojega dela uporabljajo »BPM«. V mojem primeru je šlo za skupino uporabnikov, ki so imeli pravice do upravljanja likvidacije masovnih faktur. Uporabniški vmesnik aplikacije »Ultimus Thin Client« je zelo enostaven, uporabnikom pa omogoča sledenje – spremljanje dogajanja določenega poslovnega procesa (slika 21). S pomočjo »Ultimus Thin Client« lahko spremljamo napake pri poslovnem procesu, preverjamo v katerem koraku je določen poslovni proces, komu je bil dodeljen itd. Orodje »Ultimus Thin Client« nam poleg spremljanja več korakov različnih poslovnih procesov hkrati omogoča tudi prilagoditev prikaza podatkov na način, ki nam najbolj ustreza. Korak poslovnega procesa je predstavljen kot ločena spletna stran, ki jo odpremo preko aplikacije »Ultimus Thin Client«. Na odprti spletni strani se običajno nahajajo določena polja (v obliki spletnega obrazca), ki so pomembna za izvajanje poslovnega procesa. Ko izpolnimo spletni obrazec in s pomočjo gumba končamo trenutni korak poslovnega procesa, avtomatsko zaženemo naslednji korak. V ozadju naše aplikacije se vsi podatki obdelajo in na podlagi načrta poslovnega procesa dodelijo določenemu uporabniku, kar pomeni zagon novega koraka poslovnega procesa. Proženje novega koraka je zabeleženo v aplikaciji »Ultimus Thin Client«, kjer se s ponovitvijo zgoraj opisanega postopka odpiranja koraka poslovni proces izvede do konca.



Slika 21: Primer orodja »Ultimus Thin Client«, s katerim lahko sledimo več korakom poslovnega procesa.

Orodje »Ultimus BPM Studio« ponuja poleg zgoraj opisanih komponent tudi razne komponente, ki so uporabne predvsem v času razvoja poslovnega procesa. Gre za komponente, ki so zmožne slediti toku podatkov, ki se prenašajo med koraki poslovnega procesa in pripomorejo k hitrejšemu odpravljanju napak. Tudi »Ultimus Thin Client« je zelo uporabna aplikacija v stanju razvoja, predvsem v primeru proženja avtomatskih korakov, kajti omogoča interaktivni pregled nad končanjem enega in začetkom drugega koraka. V sklopu »Ultimus Thin Client« obstaja tudi grafični prikaz poteka poslovnega procesa, ki je zmožen zabeležiti napako na določenem koraku, kar je za razvijalca dobrodošlo.

Za pravilno prenašanje podatkov med dvema korakoma orodje »Ultimus BPM« uporablja podatkovno bazo, kjer hrani podatke med vmesnimi koraki. Ob namestitvi orodja na strežnik se namesti tudi privzeta podatkovna baza, kjer so zabeležene vse nastavitve*. Poleg podatkovne baze, ki je kreirana ob namestitvi orodja »Ultimus BPM« na strežnik je za normalno delovanje potrebno vpisovanje v uporabniško podatkovno bazo. Podatkovna baza mora biti zasnovana tako, da ima vsak poslovni proces svojo ločeno podatkovno tabelo, katere ime mora biti identično imenu poslovnega procesa. Zaradi takih zahtev sem v primeru prenove spletnega iskalnika moral preimenovati tabelo »LikvidacijaMasovnihFaktur« v tabelo »LikvidacijaMasovnihFakturArhiv«. V nasprotnem primeru bi prišlo do napak pri vpisovanju podatkov poslovnega procesa v tabelo, ki je bila kreirana zgolj za potrebe iskalnika. Poleg podatkov prenesenih iz spletnih obrazcev so se v podatkovno bazo zapisovali tudi podatki, ki so vezani za določen poslovni proces npr: št. instance koraka poslovnega procesa, čas začetka koraka itd.

Kljub vsem sposobnostim orodja »Ultimus BPM« brez aplikacije, ki je tekla v ozadju ni šlo. Šlo je za nadzorni modul, ki je s pomočjo WS avtomatsko prožila incidente poslovnega procesa. Poleg samega proženja incidentov je bilo na obrazcu določenega koraka potrebno prikazati tudi samo določene podatke, ki so vezani na ta korak. Prenos podatkov med aplikacijo in spletno aplikacijo »Ultimus Thin Client« je bil omogočen preko izmenjave datotek XML. Povezava med orodjem »BPM« in aplikacijo je bila izdelana že na začetku, v fazi načrtovanja poslovnega procesa. Pri načrtovanju poslovnega procesa je bilo potrebno razmišljati tudi o tem kateri podatki so za določen korak pomembni in kateri ne. Na podlagi teh podatkov se je zgradila struktura datotek XML, preko katere so se izmenjevali podatki. S tehnične strani gledano je to pomenilo, da je bilo potrebno izdelati razred, ki bi kot atribute vseboval vse podatke potrebne za komunikacijo s poslovnim procesom. Ob avtomatskem proženju incidentov poslovnega procesa, je tako bilo potrebno kreirati instanco ustvarjenega razreda, napolniti to instanco razreda s potrebnimi podatki in na podlagi teh avtomatsko generirati dokument XML, ki bo ustrezal dogovorjeni strukturi. Ker je sama struktura dokumenta bila vnaprej znana, je orodje »BPM« na drugi strani lahko sprejelo le točno določeno obliko dokumenta XML. Da bi prenos podatkov potekal pravilno (da bi se vsak atribut XML dokumenta preslikal v točno določen atribut v orodju BPM) je bilo potrebno poskrbeti za pravilno konfiguracijo v orodju BPM. Nastavitve preslikave so se urejale v sklopu načrtovanja poslovnega procesa. Kljub temu je obstajala možnost dodajanja novih atributov v strukturo XML dokumenta,

vendar je v tem primeru bilo potrebno poskrbeti za pravilno preslikavo. V testne namene je bil kreiran dokument XML, ki je deloval kot predloga za avtomatsko generirane dokumente v sklopu aplikacije. Na podlagi izdelanega XML se je v orodju BPM ročno določila preslikava podatkov iz XML datoteke v »spremenljivke«, ki jih BPM Studio pozna. To je omogočilo, da so podatki iz aplikacije (preko XML datoteke) bili v enaki obliki in formatu dosegljivi tudi BPM Studiu. Pomembnost pravilnega formata podatkov je ključnega pomena, predvsem v primeru uporabe t.i. »ocenjevalnega koraka«.

Ocenjevalni korak deluje kot pogoj IF in v primeru, da je pogoj izpolnjen izvede scenarij »A«, v nasprotnem primeru scenarij »B«. Pogoj preko katerega se odloča je vnaprej določen, npr: če je vrednost neke spremenljivke večja od vrednosti »x«. Vrednost spremenljivke je v vsakem incidentu drugačna, ker je odvisna od aplikacije, torej se le ta prenese preko XML datoteke. V tem primeru je nadaljnji potek poslovnega procesa odvisen od spremenljivke, ki ima različno vrednost za vsak incident kar pomeni, da je pravilen format spremenljivk XML datoteke ključnega pomena. Podatki, ki se iz aplikacije prenesejo v BPM Studio se uporabijo za prikazovanje na določenih korakih – odvisno od koraka. Tudi v tem primeru je ključnega pomena pravilni format podatkov, kajti od njih je odvisna nadaljnja usoda določenega dokumenta – določenega naročnika.

V celoti je orodje Ultimus BPM zelo obsežno in zahtevno, obenem pa nudi veliko podpore kar nam močno olajša delo. Poleg samega izvajanja poslovnih procesov ponuja veliko orodij za načrtovanje poslovnih procesov, sledenje izvajanja, upravljanje poslovnih procesov itd. Z uporabo orodij Ultimus BPM sem pridobil veliko znanja predvsem na področju načrtovanja, povezovanja in komunikacije drugih orodij z aplikacijo, ki sem jo razvijal.

5.2.2. Kreiranje poslovnega procesa

V sklopu funkcionalne specifikacije aplikacije, ki sem jo moral izdelati, sem dobil tudi načrt poslovnega procesa in njegovo funkcionalnost. Poslovni proces je bil sestavljen iz sedmih korakov – začetnega koraka, štirih uporabniških korakov, enega avtomatskega in končnega koraka. Opisoval je postopek likvidacije masovnih faktur skozi proces pregleda. Poslovni proces, ki je prikazan na sliki 22, je bil zgrajen s pomočjo orodja »Ultimus Process Designer«. Načrtovanja poslovnega procesa sem se lotil na začetnem koraku. Najprej sem določil začetni in končni korak in jima določil način in pravice zagona. Začetni korak je bil avtomatski, sprožil ga je nadzorni modul ob uvozu novega svežnja masovnih faktur. Pri tem se je kreirala nova instanca – incident poslovnega procesa, ki je bil viden v »Ultimus Thin Client«-u. Začetni korak je torej bil avtomatski in pravice za zagon poslovnega procesa je imela posebna skupina uporabnikov – »LikvidacijaMasovnihFakturWinService«. S tehnične strani je to pomenilo, da je proces operacijskega sistema Windows – WS, ki je skrbel za uvoz masovnih faktur bil zagnan pod uporabnikom, ki je bil član zgoraj omenjene skupine. Pravica zagona določenega

koraka je bila pomembna lastnost posameznega koraka, kajti na podlagi teh pravic je bilo potrebno poskrbeti za pravilno zaporedje korakov. Vsak korak je imel svoje pravice zagona, kar je pomenilo, da je ob zaključku enega koraka bilo potrebno poskrbeti za pravilno dodeljevanje pravic naslednjega koraka v sklopu povezave med dvema korakoma. Z drugimi besedami, zaporedje korakov je bilo odvisno od dodeljevanja korakov in pravic zagona določenega koraka.

Ob končanem začetnem koraku je sledil nov korak, ki je bil namenjen pripravi zavrnitve fakture. Za proženje novega koraka imenovanega »Priprava zavrnitve« je bila zadolžena določena skupina uporabnikov. Ta skupina je imela nalogo pregleda določene fakture in vpisovanje opombe zavrnitve fakture. Po dokončanju koraka »Priprava zavrnitve« je sledil nov korak, ki se je sprožil avtomatsko. Proženje avtomatskega koraka je pomenilo zagon novega »servisa«, v mojem primeru »Web service«, ki je na podlagi vsebine koraka »Priprava zavrnitve« poskrbel za avtomatsko generiranje dokumenta »Word«. Glede na to, da je bil ta korak avtomatski in ga uporabniki niso bili zmožni sami zagnati je pravica zagona bila dodeljena skupini »LikvidacijaMasovnihFakturWebService«. To je pomenilo, da je moral biti »Web service«, ki je skrbel za generiranje Word dokumenta predčasno zagnan s strani uporabnika, ki je bil član zgoraj omenjene skupine. V nasprotnem primeru avtomatizacija zaradi dostopnih pravic ne bi bila uspešna. Po uspešno generiranem dokumentu tipa »Word«, se je sprožil nov korak – »Pregled zavrnitve«. V tem koraku se opravi pregled generiranega dokumenta, ki je priložen na maski koraka – HTML spletna stran. Obenem se v sklopu koraka opravi tudi potrditev zavrnitve. Korak »Pregled zavrnitve« je namenjen uporabnikom skupine »Vpis v vhodno knjigo« v sklopu organizacijske strukture podjetja. Sledi zagon koraka »Potrditev zavrnitve«, kjer zaposleni potrdi ali zavrne prejeto fakturo. Pravice do potrditve ima zaposleni, čigar delovno mesto pripada skupini »Svetovalec za davke« v sklopu organizacijske strukture podjetja. Naloga zaposlenega je pregled prejetega dokumenta in na podlagi le tega odločanje o potrditvi ali zavrnitvi. V kolikor je generiran dokument primeren in izpolnjuje vse potrebne zahteve, lahko zaposleni potrdi zavrnitev in s tem sproži nov korak. V nasprotnem primeru, v kolikor opazi da dokument ni popoln, lahko sproži zavrnitev dokumenta. To pomeni, da se namesto zagona naslednjega koraka poslovnega procesa ponovno zažene predhodni korak – »Pregled zavrnitve« – kar pomeni vračanje v predhodni korak. Obenem mora v okno opombe opisati razlog zavrnitve, kajti na tak način sporoči sodelavcu, ki pripada skupini »Vpis v vhodno knjigo« razlog zakaj dokument ni bil posredovan v nadaljnjo obravnavo. Vračanje korak nazaj, poleg obvestila med sodelavci, za seboj potegne marsikaj. Korak, v katerega se je mogoče vrniti, je bilo potrebno posebej prilagoditi. Podatki, ki so prikazani kot vsebina koraka, so morali biti pravilni ne glede na to ali bo izvedeno vračanje korak nazaj ali ne. V tem primeru pride do izraza tudi pravica do zagona določenega koraka, kajti v našem primeru gre za dve različni skupini – odvisno od poteka poslovnega procesa. V praksi to pomeni, da je potrebno izdelati novo skupino, ki bi vsebovala vse člane obeh zgoraj omenjenih skupin – »Vpis v vhodno knjigo« in »Svetovalec za davke«. Le na tak način lahko zagotovimo pravilno delovanje v primeru dveh različnih scenarijev. Sam postopek zavrnitve dokumenta – vračanje korak nazaj, se lahko ponovi večkrat in vsakič z drugačnimi podatki.

Po uspešno končanem postopku dopolnjevanja podatkov v koraku »Pregled zavrnitve« in uspešno zaključenem koraku »Potrditev zavrnitve«, je sledil zagon koraka »Odprema«. V koraku »Odprema« se je opravil še končni pregled dokumenta in zaključek poslovnega procesa. Korak »Odprema« se po funkcionalnosti ni veliko razlikoval od ostalih. Njegova glavna razlika je bila pravica do zagona. Odpremo določenega dokumenta je lahko sprožil isti zaposleni, ki je opravljal pregled dokumenta. Korak »Odprema« je bil dodeljen vsakič drugemu uporabniku dinamično glede na korak »Pregled dokumenta«. Po uspešnem zaključku koraka »Odprema« je bil poslovni proces končan. Sledil je zagon koraka »End«, ki je poskrbel za končanje določene instance in s tem zaključek poslovnega procesa.

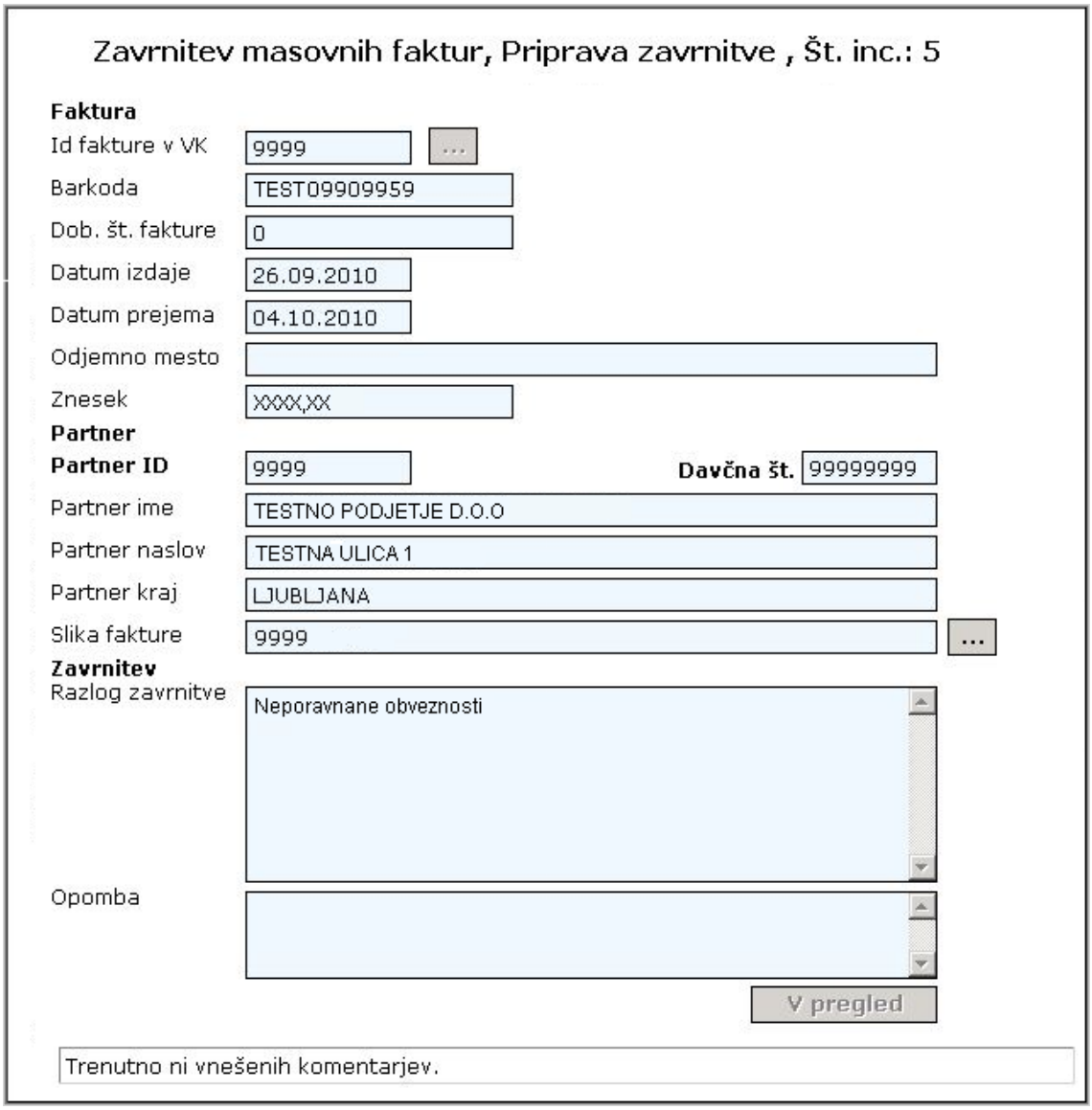
Slika 22: Primer poslovnega procesa kreiranega s pomočjo orodja »Ultimus Process Designer«.

5.2.3 Implementacija spletnih vmesnikov, namenjenim korakom poslovnih procesov

Vsak poslovni proces je sestavljen iz več korakov, ki se izvajajo zaporedno in predstavljajo posamezne dele funkcionalnosti poslovnega procesa. Koraki so med seboj povezani, kajti na tak način je določeno njihovo zaporedje. Korakov poslovnih procesov se ne da preskakovati, zato se vedno izvajajo zaporedno. V mojem primeru so koraki poslovnih procesov bili predstavljeni v obliki spletnih obrazcev. Vsak korak je imel svojo funkcionalnost, zato so se obrazci med seboj razlikovali tako po obliki, kot tudi po kompleksnosti. Spletne obrazce sem izdelal z orodjem Microsoft Visual Studio 2008, vsak izmed njih je bil sestavljen iz dveh delov. Prvi del je predstavljal sam videz obrazca, ta del sem izdelal s pomočjo HTML in ga oblikoval z uporabo CSS. Drugi del je predstavljal logiko samega obrazca torej, kaj se zgodi ob kliku na določen gumb. Ta del je (kot večino projekta) bil izdelan v programskem jeziku »C#«. Videz spletnih obrazcev je bil do neke mere vnaprej določen, kajti v sklopu funkcionalne specifikacije sem dobil tudi primer videza vsakega spletnega obrazca posebej. Kljub navodilom je bilo potrebno obrazce pravilno oblikovati, v sklopu enotnega videza celotne informacijske rešitve. Pri tem sem si pri določenih komponentah lahko pomagal z že izdelanimi videzi, a kljub temu sem moral dosti postoriti tudi sam.

Poslovni proces je obsegal pet korakov, od katerih je bil en korak avtomatski, zato sem izdelal štiri spletne obrazce. Avtomatski korak ni obsegal spletnega obrazca, kajti njegova naloga je bila generiranje dokumenta tipa »Microsoft Word« na podlagi vsebine pridobljene iz prvega koraka. Glede na to, da je bil enotni videz obrazcev pomemben, sem

se dela lotil na podoben način kot pri prenovi spletnega iskalnika. Najprej sem si izdelal spletno stran, ki sem jo uporabljal kot predlogo – »Master page«. Ta spletna stran je vsebovala vse komponente in vsa polja, ki so bila enaka vsem obrazcem. Vsak naslednji spletni obrazec je nastal na podlagi predloge, kar je pomenilo da so si vsi obrazci med seboj podobni in zgrajeni na enak način. Po končani predlogi sem se lotil izdelave prvega in hkrati najobsežnejšega koraka imenovanega »Priprava zavrnitve« (slika 23). Prvi korak je vseboval veliko pomembnih polj podatkov, ki pa so bili ob zagonu koraka prazni. Glavno polje na prvem obrazcu je bilo imenovano »ID Fakture v Vhodni knjigi«. To polje je predstavljalo vnosno polje za iskanje določenega naročnika po njegovi identifikacijski številki. Ob vnosu identifikacijske številke naročnika in kliku na gumb »Iskanje« se je v ozadju opravilo iskanje vseh podatkov o naročniku z vneseno »ID« številko. S tehnične strani gledano je to pomenilo, da se je ob kliku na gumb s pomočjo SP opravilo iskanje v tabeli »VhodnKnjiga«. Rezultat iskanja, v primeru najdenega zadetka, je pomenil pretvorbo in formatiranje najdenih podatkov v obliko primerno za prikaz na spletnem obrazcu. Torej gledano s strani uporabnika – ob uspešnem iskanju so se vsa polja (ki jih je obrazec vseboval) avtomatsko napolnila z določeno vsebino. Na tak način smo v enem koraku dobili vse potrebne podatke o naročniku. Vsa polja, ki so se napolnila z vsebino ob najdenem rezultatu so bila onemogočena za popravljanje, s čimer je bilo poskrbljeno za konsistentnost podatkov o naročniku. Obrazec na koraku »Priprava zavrnitve« je poleg vseh onemogočenih polj vseboval tudi polje, imenovano »Razlog zavrnitve«. V to polje je referent, ki je zagnal korak moral vpisati razlog zavrnitve določene fakture naročnika. Poleg vseh omenjenih vnosnih polj, je obrazec na prvem koraku vseboval tudi možnost pregleda slike prispele fakture. Šlo je za prikaz elektronsko preslikanega dokumenta, ki je bil shranjen v arhivu. Ob dokončanem pregledu in vnosu razloga zavrnitve je bil s klikom na gumb »V pregled« prvi korak zaključen.



Slika 23: Primer koraka »Priprava zavrnitve« poslovnega procesa »Likvidacija masovnih faktur«.

Zaključek prvega koraka je pomenil zapiranje spletnega obrazca in shranjevanje sprememb v podatkovno bazo, kajti na tak način so bile spremembe vidne ob zagonu novega koraka. Zaključek vsakega koraka pomeni zagon naslednjega koraka, kar je v našem primeru pomenilo zagon avtomatskega koraka. Avtomatski korak je poskrbel za generiranje dokumenta izdelanega na podlagi vsebine koraka. Avtomatsko generiran dokument, je deloval na podlagi »Web servica«, ki je vsebino prvega koraka v pravilni obliki in formatu zapisal v dokument. Dokument je bil generiran na podlagi predloge, s čimer sem zagotovil vedno enak videz ne glede na vsebino in naročnika. Po uspešnem generiranju je bil dokument shranjen na določeno lokacijo na strežniku, kar je bilo pomembno predvsem za vsebino nadaljnjih obrazcev – korakov.

Po uspešno generiranem dokumentu je bilo vse pripravljeno za zagon novega koraka imenovanega »Pregled zavrnitve«. V tem primeru je šlo predvsem za pregled nastalega dokumenta in dopolnjevanje razloga zavrnitve. Spletni obrazec je vseboval veliko vnosnih polj, katerih vsebina je bila napolnjena že ob odprtju spletne strani. Šlo je za podatke, iz predhodnega koraka, ki so se ob zagonu koraka prebrali iz podatkovne baze in se prepisali na spletni obrazec. Poleg podatkov je spletni obrazec vseboval tudi polje, kjer je bila shranjena pot do generiranega dokumenta. S klikom na gumb »Odpri«, se je odprl Microsoft Word dokument, v katerem se je nahajala generirana vsebina. Po končanem pregledu dokumenta in razloga zavrnitve je uslužbenec s klikom na gumb »Potrditev zavrnitve« zaključil trenutni korak, shranil spremembe in sprožil zagon novega.

Šlo je za korak »Potrditev zavrnitve«, čigar vsebina spletnega obrazca je bila zelo podobna vsebini predhodnega koraka. Nov spletni obrazec je ohranil funkcionalnost predhodnega, kar pomeni, da so ob zagonu koraka bila vsa vnosna polja zapolnjena z vsebino. Poleg možnosti pregleda slike elektronsko preslikanega dokumenta in generiranega dokumenta je spletni obrazec pri tem koraku vseboval tudi možnost dopolnitve zavrnjene fakture. To je pomenilo, da se je vsebina obrazca shranila v podatkovno bazo, obenem pa se je poslovni proces vrnil korak nazaj in sicer v »Pregled zavrnitve«. Implementacija vračanja je bila izdelana na podlagi spremenljivke, čigar vrednost je pri prehodu med koraki določala ali gre za vračanje v predhodni korak, ali za napredovanje v naslednji. Vrednost spremenljivke je bila zapisana tudi v datoteki XML, ki je služila za prenos podatkov med aplikacijo in »Ultimus BPM Studio«. V kolikor je bil sprožen postopek vračanja v predhodni korak, je to pomenilo ponovitev koraka »Pregled zavrnitve«. V nasprotnem primeru je klik na gumb »Odprema zavrnitve« sprožil zaključek trenutnega koraka, kar je pomenilo shranjevanje podatkov in zagon novega – zadnjega koraka.

V zadnjem koraku imenovanem »Odprema zavrnitve« je šlo za odpiranje enakega obrazca kot pri predhodnem koraku, edina razlika je bila v tem, da vračanje v predhodni korak tu ni bilo možno. Ta korak je bil dodeljen istemu referentu, ki je opravljal tudi pregled. Njegova naloga je bila opraviti zadnji pregled pripravljenega dokumenta, ga natisniti in ga po pošti poslati naročniku. S klikom na gumb »Pošta poslana« se je zaključil trenutni korak in zagnal nov korak – »End«, ki je pomenil zaključek poslovnega procesa. Z zaključkom zadnjega koraka je bil končan poslovni proces za določenega naročnika. V sklopu izvajanja korakov je bil izveden celoten proces zavrnitve fakture od začetka do konca. Po končani instanci poslovnega procesa, so se lahko prožile nove instance za istega ali drugega naročnika, odvisno od prispelih dokumentov s strani nadzornega centra.

6. Zaključek

Živimo v času, kjer je čas izvajanja določenega procesa bistvenega pomena. To se najbolj odraža na poslovnem področju. Moja diplomska naloga zato predstavlja postopek avtomatizacije določenega poslovnega procesa. Za postopek avtomatizacije se odločamo zaradi skrajšanja časa obdelave in povečanja natančnosti, kar pripomore k znižanju možnosti napak pri izvajanju določenega poslovnega procesa. Zaradi omenjenih razlogov so se za avtomatizacijo poslovnih procesov odločili tudi pri enem večjih upraviteljev nepremičnin na področju Ljubljane. Avtomatizacijo poslovnih procesov so izvedli v sklopu celotne informacijske storitve, kar je pomenilo velik napredek na več področjih poslovanja.

Meni je pripadel del celotne storitve in sicer poslovni proces likvidacije masovnih faktur. Šlo je za avtomatsko obdelovanje prispelih faktur s strani naročnikov. Avtomatizacija je bila potrebna predvsem zaradi velike količine prispelih faktur, kar je pripomoglo pri uparjanju (na podlagi določenih pravil) prispelih dokumentov s podatkih v zalednih sistemih. V sklopu izdelave ni šlo le za tipično spletno aplikacijo, ampak za povezovanje več modulov v celoto. Tako je bil izdelan tudi iskalnik, ki je omogočal hitro in natančno iskanje (na podlagi določenih pogojev) po vseh prejetih dokumentih. Za procesiranje prispelih faktur je bil izdelan nadzorni modul, ki je skrbel za uvoz faktur in elektronsko obdelavo le teh.

Pri izdelavi vseh omenjenih modulov sem naletel na manjše težave povezane predvsem z nekompatibilnostjo s starejšo obstoječo različico informacijske storitve, ki se je izvajala na drugačni platformi. Nekompatibilnost se je pokazala predvsem pri migraciji podatkov med dvema podatkovnima bazama – prehod iz Oracle 9i na Microsoft SQL. Problem sem uspešno rešil s pomočjo programskega prenosa podatkov.

Prenovljena informacijska storitev je izdelana dinamično in je odprta za dodatne module, glede na naročnikove zahteve. Določena poslovna pravila so ostala nespremenjena, kar lahko v prihodnosti predstavlja ozko grlo. Z nadgradnjo le teh lahko celotna informacijska storitev in z njo tudi moj poslovni proces močno pridobi na dinamičnosti izvajanja vsakodnevnih nalog.

7. Kazalo slik

8. Viri

[1] (2008) DataGridView Control in C#. Dostopno na:
http://www.c-sharpcorner.com/uploadfile/raj1979/datagridview09052008041419am/datagridview.aspx

[2] (2011) datetime (Transact-SQL). Dostopno na:
http://msdn.microsoft.com/en-us/library/ms187819.aspx

[3] (2011) Intruducing SQL Server Management Studio. Dostopno na:
http://msdn.microsoft.com/en-us/library/ms174173%28v=SQL.90%29.aspx

[4](2011) Linking Servers. Dostopno na:
http://msdn.microsoft.com/en-us/library/ms188279.aspx

[5] (2011) Microsoft Visual Studio 2003: Dostopno na:
http://msdn.microsoft.com/en-us/library/6d72zczx%28v=vs.71%29.aspx

[6] (2011) Microsoft Visual Studio 2008. Dostopno na:
http://www.microsoft.com/slovenija/msdn/vs2008/pro/novosti.mspx

[7] (2011) Oracle/PLSQL: Data Types. Dostopno na:
http://techonthenet.com/oracle/datatypes.php

[8] (2008) SQL® Navigator Tricks and Tips. Dostopno na:
http://www.quest.com/sql_navigator/nav_tips/connection/managing_oracle_connections.asp

[9](2010) Ultimus Adaptive BPM Suite. Dostopno na:
http://www.ultimus.com/adaptive-bpm-suite-software-modules/

[10] (2000) Users guide for Microsoft Visio 2000. Dostopno na:
http://demon.eltc.ru/man/EntGuide.pdf

[11] (2011) Ultimus Online Training. Dostopno na:
http://training.ultimus.com/