Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Bojan Benčina

Zajemanje dokumentov v finančnih aplikacijah

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: prof. dr. Franc Jager

Ljubljana 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00543 / 2013
Datum: 15.9.2013

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

Kandidat: **BOJAN BENČINA**

Naslov: **ZAJEMANJE DOKUMENTOV V FINANČNIH APLIKACIJAH**
CAPTURING DOCUMENTS IN FINANCIAL APPLICATIONS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Razvijte sistem za zajemanje vhodnih dokumentov s pomočjo naprav za zajemanje slik. Sistem naj podpira čim večje število zajemnih naprav in naj omogoča shranjevanje dokumentov v šifrirani obliki na datotečni sistem ali na strežnik SQL. Sistem naj omogoča tudi urejanje zajetih dokumentov v smislu ogleda, brisanja, dodajanja, vrivanja, ažuriranja, izvažanja in pošiljanja dokumentov.

Mentor:
prof. dr. Franc Jager

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Bojan Benčina, z vpisno številko **63010183**, sem avtor diplomskega dela z naslovom:

Zajemanje dokumentov v finančnih aplikacijah

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Franca Jagra
- 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 21. septembra 2013 Podpis avtorja:

Za pomoč pri pisanju naloge se zahvaljujem svojemu mentorju prof. dr. Francu Jagru. Hvala tudi staršem, ki so mi študij omogočili, sestrama, ki sta me pri njem podpirali in moji dragi ženi, ki mi je bila v oporo in pomoč pri pisanju te naloge.

Kazalo

Seznam uporabljenih kratic

VFP9 (ang. Visual FoxPro 9) Podatkovno osredotočen in objektno orientiran programski jezik.

SQL (ang. Structured Query Language) Standardnim poizvedbeni jezik za izvajanje operacija na relacijski podatkovni bazi.

ODBC (ang. Open Database Connectivity) Orodje za povezavo na podatkovno bazo.

CRUD (ang. Create, Read, Update and Delete) Osnovne operacije obdelave podatkov.

DLL (ang. Dynamic-Link Library) Dinamično povezovalna knjižnica z izvršljivimi rutinami.

CLR (ang. Common Language Runtime) Komponenta ogrodja .NET za izvajanje programov.

WPF (ang. Windows Presentation Foundation) Programski model, za razvijanje grafičnih aplikacij v ogrodju .NET.

LINQ (ang. Language Integrated Query) Komponenta ogrodja .NET, ki omogoča pisanje poizvedb v objektno orientirani kodi.

XML (ang. eXtensible Markup Language) Označevalni jezik za prenos podatkov.

HTML (ang. HyperText Markup Language) Jezik za pripravo spletnih strani.

XHTML (ang. Extensible HyperText Markup Language) Označevalni jezik iz družine jezikov XML.

WSDL (ang. Web Services Description Language) Jezik iz družine XML za

opis mrežnih servisov.

WAP (ang. Wireless Application Protocol) Protokol za pridobivanje podatkov preko mobilnih naprav.

RSS (ang. Rich Site Summary) Tehnologija za obveščanje o spremembah na spletnih straneh, ki se pogosto osvežujejo.

DTD (ang. Document Type Definition) Množica pravil, ki definirajo dokumente napisane v označevalnem jeziku.

NTFS (ang. New Technology File System) Vrsta datotečnega sistema, ki ga je razvil Microsoft.

BLOB (ang. Binary Large OBject) Zbirka binarnih podatkov shranjenih kot en zapis v podatkovni bazi.

LGPL (ang. Lesser General Public License) Tip licence programske kode.

WIA (ang. Windows Image Acquisition) Platforma sistema Windows za komunikacijo z zajemnimi napravami.

STI (ang. Still Image Architecture) Nizko nivojna abstrakcija strojne opreme.

DDI (ang. Device Driver Model and Interface) Gonilnik zajemne naprave in njegov vmesnik.

API (ang. Application Programming Interface) Aplikacijski programski vmesnik.

COM (ang. Component Object Model) Binarni standard za med procesno komunikacijo in ustvarjanje objektov.

BMP (ang. BitMaP image file) Format slike v rastrski obliki.

TIFF (ang. Tagged Image File Format) Format slike v izboljšani rastrski obliki.

RGB (ang. Red Green Blue) Seštevalni barvni model, ki z mešanjem rdeče, zelene in modre barve ustvarja široko paleto barv.

CMYK (ang. Cyan, Magenta, Yellow, Key) Odštevalni barvni model, ki se uporablja predvsem pri tiskanju.

LZW (ang. Lempel-Ziv-Welch) Metoda stiskanja datotek TIFF.

CCITT4 Metoda stiskanja datotek TIFF, posebej primerna za črno bele slike.

PDF (ang. Portable Document Format) Format datotek, za prikaz dokumentov na različnih operacijskih sistemih in napravah.
DPI (ang. Dots Per Inch) Enota za merjenje gostote pik, ki jih lahko spravimo v okvir enega inča, to je 2.54 cm.

Seznam slik

Seznam programske kode

Povzetek

Z vse večjim razmahom elektronskega poslovanja je na področju finančnih aplikacij prišlo do potrebe po elektronskem zajemu dokumentov. Tema diplomske naloge je razvoj orodij za zajemanje vhodnih dokumentov, s pomočjo naprav za zajemanje slik, s parametri kot jih določi uporabnik ter za urejanje zajetih dokumentov v smislu ogleda dokumentov, njihovega brisanja, dodajanja, vrivanja, ažuriranja, izvažanja ter pošiljanja dokumentov. Razvili smo sistem za shranjevanje dokumentov v šifrirani obliki na datotečni sistem ali na strežnik SQL. Naš cilj je bil podpreti čim večje število zajemnih naprav in razviti čim bolj enostaven ter hiter program. Pri razvoju smo uporabili različna orodja kot so: *Visual FoxPro 9*, C#, SQL, protokole in navodila TWAIN ter WIA in objektno orientirane tehnike programiranja.

Ključne besede: Zajem vhodnih dokumentov, zajemne naprave, sistem shranjevanja dokumentov, urejanje zajetih dokumentov, objektno programiranje

Abstract

Growing expansion of e-commerce in financial applications leads to the need for electronic capturing of documents. Topic of diploma thesis is development of tools for capturing input documents using devices to capture images with the parameters specified by the user, and editing of the captured images in the sense of viewing the documents, their deleting, adding, inserting, updating, exporting and sending the documents. We developed a document storage system to store the documents on a file system or on SQL server in an encrypted form. Our goal was to support as high number of capturing devices as possible and development of as simple and fast program as possible. During the development, we used different tools like: Visual FoxPro 9, C#, SQL, protocols and guidelines TWAIN and WIA, and object oriented techniques of programming.

Key words: Capturing input documents, capture devices, storage system for documents,editing of captured documents , object oriented programing

Poglavje 1

Uvod

Kljub dobi informacijske tehnologije se pri vsakodnevnem poslovanju podjetji še vedno pojavlja velika količina dokumentov v papirni obliki. Iskanje dokumentov v takšni obliki je časovno zelo potratno in po nepotrebnem povečuje kompleksnost in stroške poslovanja, hkrati pa povzroča slabo voljo zaposlenih in njihovih nadrejenih.

Cilj te diplomske naloge je bil pripraviti prijazno rešitev za uporabnike Gradovih aplikacij in hkrati olajšati delo programerjem, ki bi radi v svoje aplikacije dodali možnost zajema slike. Rešitev omogoča zajem poljubnih dokumentov na enostaven način, hiter dostop do zajetih vsebin in s tem optimizacijo poslovnih procesov, ki so lahko dolgotrajni in mukotrpni.

Z elektronsko hrambo dokumentov dosežemo večji nadzor in centralizacijo podatkov, saj ležijo na enem mestu in so popolnoma dostopni iz aplikacij ne glede na uporabnikovo lokacijo. Za dostop je potreben le internetni dostop in Gradova e-aplikacija. Elektronska oblika računa omogoča lažje ohranjanje prvotne oblike dokumentov, saj niso izpostavljeni zunanjim vplivom.

Ko govorimo o vhodnih dokumentih se osredotočamo najprej na prejete račune in celotno vhodno pošto. Pri prejetih računih gre za pomemben napredek, saj njihovo hranjenje v elektronki obliki omogoča uporabnikom likvidacijo na daljavo. To dodatno pospešuje poslovanje, razbremenjuje uporabnike in povzroča zadovoljstvo pri izdajateljih računov, saj se plačila hitreje izve-

dejo. Vendar tu nismo rekli stop. Sistem vključuje zajem poljubnih evidenc, ki si jih nastavijo uporabniki sami. Primer so razna potrdila v programu za vodenje kadrovskih evidenc, skladiščni dokumenti, dokumenti osnovnih sredstev, logotipi, žigi ...

Gradove aplikacije so medsebojno povezljive programske rešitve za vodenje poslovanja. Delujejo v okolju Microsoft Windows (XP,Vista,7,8). Njihov aplikativni del je napisan v programskem jeziku Microsoft visual foxpro 9, podatkovni del teče na strežniku Microsoft SQL. Gradove e-aplikacije so namenjene opravljanju poslovanja na daljavo preko spletnih servisov. Gradove i-aplikacije predstavljajo napredek na področju mobilnih in tabličnih naprav saj omogočajo poslovanje preko pametnih naprav. Na vseh platformah je možen ogled zajete slike iz osnovnega sistema.

V diplomski nalogi smo pripravili rešitev, ki v aplikativnem delu omogoča uporabniku preprosto nastavljanje parametrov za zajem in hrambo dokumentov. Uporabnik lahko nastavi imenik in način shranjevanja, ločljivost zajema slike, kakovost, format slike (A4), paketni zajem (za naprave s podajalcem), obojestranski zajem in brisanje praznih strani. V izbranih delih programov, običajno je to pri zajemu podatkov, se uporabniku ponudi gumb "zajem slike", s katerim sproži zajem dokumentov. Uporabniku ni potrebno zapustiti aplikacije ali prekiniti z vnosom podatkov, saj se zajem slike dogaja v ločenem procesu, kar omogoča hkraten vnos preko uporabniškega vmesnika in zajemne naprave. Po končanem procesu skeniranja se glede na potrditev ali preklic vnosa, dokument shrani na pred tem nastavljeno lokacijo in se poveže s trenutnim podatkom. Uporabnik nato lahko preko gumba za ogled slike dobi rezultat zajema na zaslon. Na tem mestu dokument lahko izvozi v drugačno obliko (PDF) ali pošlje po elektronski pošti.

Za komunikacijo z zajemno napravo in delo s sliko smo pripravili programski dodatek Dynamic-Link Library (DLL), ki ga naložimo v aplikativnem delu in s parametrom kličemo njegovo funkcijo. Knjižnica je napisana v programskem jeziku C# s podporo ogrodja .NET 2.0. Odločitev za ogrodje .NET 2.0 smo sprejeli zaradi razširjenosti Gradovih aplikacij pri uporabni-

kih, ki na operacijskem sistemu nimajo naloženih novejših različic ogrodja. Proces namestitve novih verzij je časovno potraten, zapleten in zahteva administratorska pooblastila, ki jih običajni uporabniki nimajo.

Pri pripravi programske knjižnice in razredov uporabniškega vmesnika je bil cilj pripraviti objektno orientirano rešitev, ki programerjem omogoča enostavno implementacijo storitve v svojo aplikacijo. Tako se vse krmiljenje knjižnice odvija le z nastavljanjem lastnosti objektov, kar pomeni, da razvijalec ne potrebuje globokega poznavanja problematike zajemnih naprav in ostalih procesov, ki se odvijajo na nižjem nivoju.

Dokumente lahko shranjujemo na dva načina, na datotečni sistem, kar je s stališča delovanja programa hitreje in bolj enostavno, vendar izpostavljeno zunanjim vplivom (spremembam izven aplikacije). Drug način je shranjevanje na strežnik SQL, ki nudi večjo varnost. Za zagotavljanje najvišjega nivoja varnosti smo uporabnikom ponudili možnost shranjevanja dokumentov v šifrirani obliki.

V nadaljnjih poglavjih diplomske naloge smo podrobno predstavili potek dela. V poglavju številka dva smo opisali orodja in protokole ter tehnike programiranja, ki smo jih uporabili. V tretjem poglavju smo predstavili programsko kodo knjižnice za zajem slike in urejanja dokumentov, uporabniškega vmesnika in prikazali, kako sliko shranimo na strežnik SQL s sistemom FILESTREAM.

Četrto poglavje služi predstavitvi končne rešitve in primerom praktične uporabe. V njem smo prikazali izgled uporabniškega vmesnika. V petem poglavju podamo zaključne misli in se ozremo v prihodnost k morebitnim izboljšavam.

Poglavje 2

Predstavitev uporabljenih orodji in protokolov

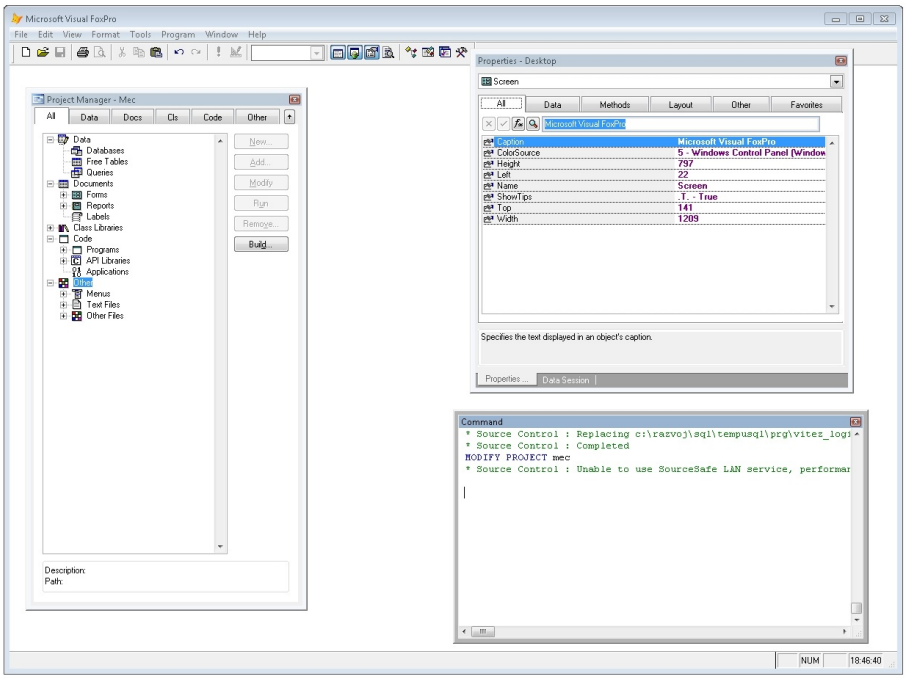
2.1 Orodje za izdelavo uporabniškega vmesnik

Izbor programskega jezika za razvoj uporabniškega vmesnika ni bil težak. Naloga je namenjena izboljšanju Gradovih programov, zato smo se morali podrediti programskemu jeziku, v katerem so aplikacije razvite. Gre za razvojno okolje *Visual FoxPro 9* (VFP9), prikazano na sliki 2.1.

VFP9 je programsko orodje za hitro razvijanje 32 bitnih aplikacij. Gre za objektno orientirani programski jezik, ki razvijalcem nudi vse možnosti priprave sistema za upravljanje podatkovnih zbirk. Ena glavnih prednosti, ki so bile pomembne pri razvoju Gradovih programov, je možnost priprave aplikacij po sistemu klient – strežnik. Pri tem gre za to, da VFP9 uporabimo za razvoj uporabniškega vmesnika. Podatkovni del pa razvijemo ločeno, v našem primeru je to strežnik Microsoft SQL. Za povezavo med obema deloma aplikacije smo uporabili gonilnik *Open Database Connectivity* (ODBC) [1, 2].

Med večje prednosti spada še močno orodje za izdelavo poročil, kar smo v nalogi izkoristili pri pripravi ogleda slike.

VFP9 ima tudi svoje slabosti. Največja, na katero smo naleteli pri pri-



Slika 2.1: Razvojno okolje VFP9.

pravi naloge, je podpora le ene programske niti. To in sama zastarelost orodja je botrovalo odločitvi uporabe jezika C# pri pripravi knjižnice, ki opravlja komunikacijo z zajemno napravo in omogoča urejanje slike.

2.2 Orodje za izdelavo programske knjižnice

Za izdelavo knjižnice DLL za zajem in urejanje slike smo uporabili orodje Microsoft Visual C#. Gre za programski jezik primeren za izdelavo široke palete aplikacij. Je del Microsoftovega ogrodja .NET. Izhaja iz programskih jezikov C in C++ in se zelo približa Javi. Je moderen, varen in objektno orientiran. Primeren je za izdelavo tradicionalnih Windows programov, XML spletnih servisov, podatkovnih aplikacij, sistemov klient – strežnik, skratka močno orodje [3]. Za razvoj smo uporabili okolje Microsoft Visual Studio 2012.

2.2.1 Objektno programiranje

Kot objektno usmerjeni programski jezik C# podpira enkapsulacijo, dedovanje in polimorfizem. Enkapsulacija pomeni, da je skupina sorodnih metod, lastnosti in ostalega obravnavana kot ena enota oziroma objekt. Dedovanje je možnost ustvarjanja novega razreda na podlagi obstoječega. Polimorfizem je izmenična uporaba več razredov, čeprav vsak posamezen razred implementira lastnosti in metode na drugačen način [3, 4].

Pri objektnem programiranju se srečujemo z izrazi kot so razred in objekt. Razred si lahko predstavljamo kot načrt, s pomočjo katerega ustvarimo objekt. V C# je na voljo tako imenovana lažja različica razredov oziroma struktura. Uporablja se za ustvarjanje velikega števila objektov na sistemu, ki ima na voljo malo spomina [4].

Razred sestavljajo lastnosti, metode in dogodki. Lastnosti vsebujejo informacije, ki jih najdemo na objektu. Metode predstavljajo akcije, ki jih objekt izvaja. Razred lahko vsebuje več metod z enakim imenom in različnimi parametri ali tipi parametrov (overloads). Dogodki služijo komuniciranju med objekti in razredi [4].

Objekti vsebujejo konstruktorje in destruktorje. Pri prvih se koda izvede na začetku kreiranja objektov, pred vsemi ostalimi metodami in sicer samo enkrat. Za destruktorje velja, da se izvedejo v trenutku uničenja objekta. Destruktor je samo eden [4]. Primer kreiranja razreda je prikazan v programski kodi 2.1.

Programska koda 2.1 : Primer kreiranja razreda v programskem jeziku C#

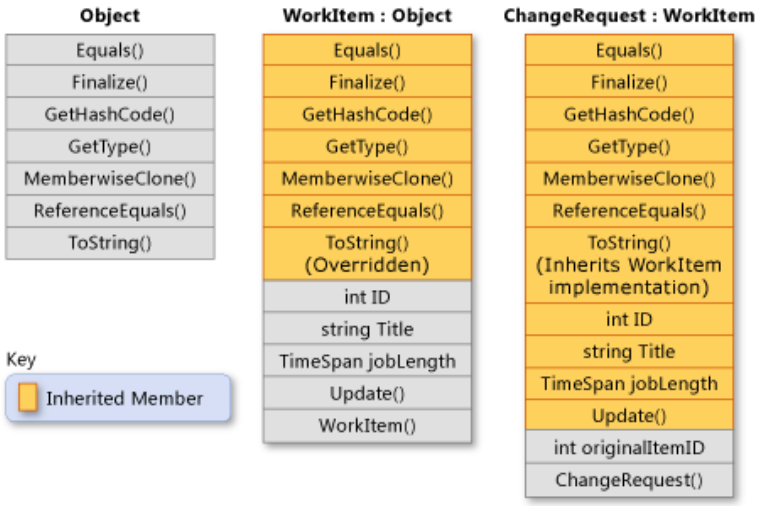
```
1  public class CRazred1
2  {
3      public string lJavna; //lastnost
4      public CRazred1() //konstruktor
5      {
6
7      }
8      private int mTest(string sampleParam) //Definicija metode
9      {
10
11     }
```

```
12      CRazred1 oObjekt1 = new CRazred1 ();//kreiranje objekta
13      oObjekt1.lJavna = "Priredimo lastnost!";
14  }
```

V programskem jeziku C# poznamo naslednje ravni dostopa do lastnosti oziroma metod:

- public - dostop je dovoljen katerikoli kodi, ne glede na to, v katerem sklopu se izvaja
- private - dostop je dovoljen samo kodi znotraj istega razreda
- protected - dostop je dovoljen samo kodi znotraj istega ali izpeljanega razreda
- internal - dostop je dovoljen samo kodi istega sklopa
- protected internal - dostop je dovoljen kodi iz istega sklopa ali kodi na izpeljanem razredu v poljubnem sklopu

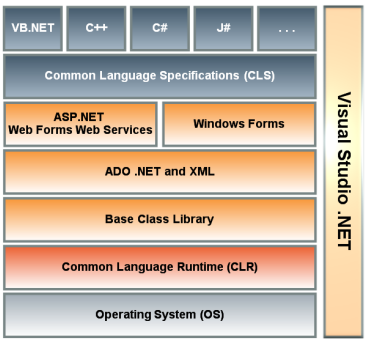
Ena največjih prednosti objektnega programiranja je dedovanje. To omogoča ustvarjanje novih razredov na podlagi drugih razredov. Obstoječe metode in lastnosti lahko uporabljamo, razširjamo, spreminjamo ali dodajamo nove. Pri dedovanju ločimo osnovne in izpeljane razrede. Načeloma lahko izhajamo iz vseh razredov, razen tistih, ki jim v njihovi definicij dodamo kontrolno besedo *sealed* [5]. Za lažjo predstavo smo primer dedovanja prikazali na sliki 2.2.



Slika 2.2: Primer dedovanja.

2.2.2 Ogrodje .NET

Ogrodje .NET je bilo predstavljeno leta 2000 kot nov način ustvarjanja tradicionalnih Windows programov, ki poleg tega omogoča še hitro in enostavno izdelavo spletnih aplikacije in servisov [6]. Na sliki 2.3 je prikazana zgradba ogrodja .NET.



Slika 2.3: Zgradba ogrodja .NET.

Glavni cilji razvoja takšnega ogrodja so bili:

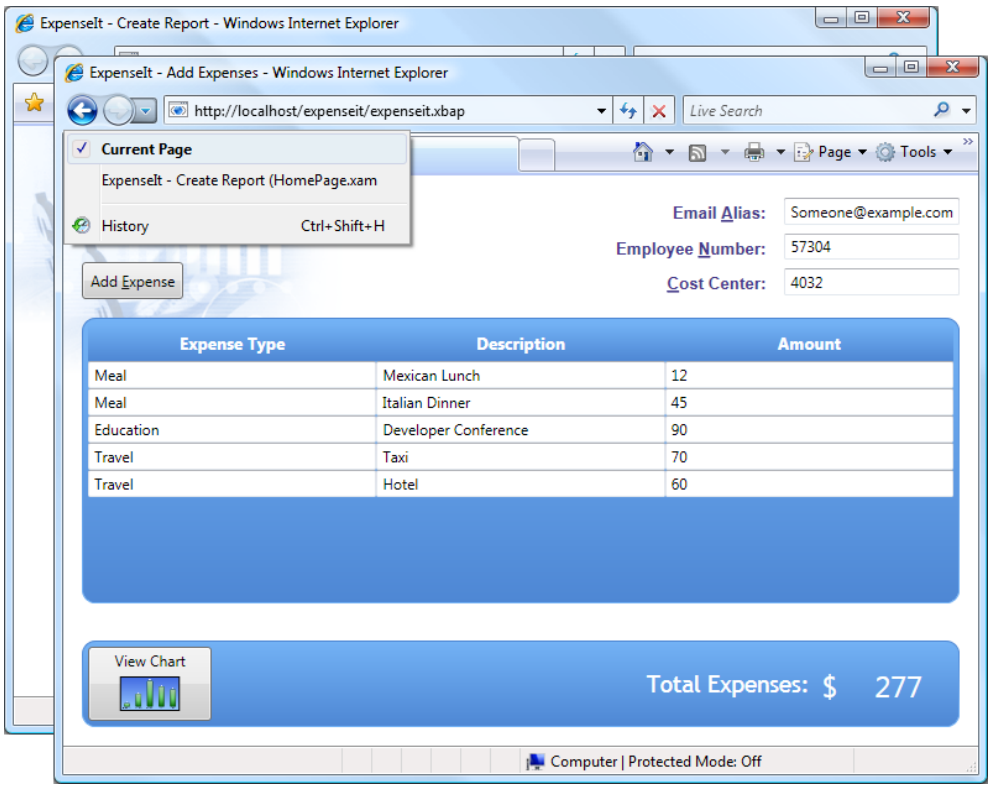
- omogočiti objektno orientirano programiranje ne glede na lokacijo programske kode (lokalno, oddaljeno)
- zagotoviti okolje za izvajanje kode, ki bo zmanjšalo težave z nameščanjem novih verzij programske opreme
- ustvariti okolje, ki bo omogočalo varno izvajanje programske kode
- omogočiti razvoj široke palete aplikacij (namizne, spletne ...)
- uveljaviti standarde za razvoj programske opreme [8]

Ogrodje .NET je sestavljeno iz dveh komponent in sicer, Common Language Runtime (CLR) in knjižnice razredov. CLR je neke vrste agent, ki v času izvajanja skrbi za programsko kodo. Pri tem opravlja naloge, kot so upravljanje spomina, spremljanje programskih niti in uveljavljanje varnostnih nastavitev. Omogoča delo z upravljano in ne-upravljano kodo.

Pojem upravljana koda je izraz za programsko kodo, napisano v programskem jeziku, ki je del ogrodja .NET (C#, Visual Basic) in z njo upravlja CLR. Neupravljana koda je napisana v programskem jeziku, ki ni del ogrodja .NET in CLR ne upravlja z njo. Ta prednost nam je prišla pri izdelavi diplomske naloge zelo prav, saj smo uporabili velik del neupravljane kode za komuniciranje z zajemno napravo. CLR torej omogoča izvajanje programske kode ne glede na njen izvorni jezik (C#, Visual Basic, C++, C) [9].

Knjižnica razredov je objektno orientirana zbirka tipov, ki jih lahko uporabimo za razvoj različnih aplikacij (osnovni Windows programi, zahtevna grafična orodja in spletna okolja). Največja prednost programske knjižnice je hitrejše razvijanje aplikacij. Programerjem namreč ni potrebno skrbeti za velike količine programske kode, ki bi jo morali razviti na nizkem nivoju, namesto tega uporabijo že pripravljene rešitve. Kot primer vzemimo programerja, ki piše podatkovno aplikacijo. Pred razvojem ogrodja .NET je moral sam poskrbeti za kodo, ki se povezuje na bazo, z novim okoljem uporabi obstoječi objekt, sam pa se posveti reševanju ostalih problemov [8].

Naj naštejemo nekatere prednosti in novosti ogrodja .NET, ki so se razvile iz osnovne različice. Leta 2005 je izšla verzija 2.0, ki je s seboj prinesla velik nabor programskih knjižnic za razvijalce. Različica 3.0 je nastala leta 2008. Njena največja prednost je odlična podpora Windows Presentation Foundation (WPF). To je močno okolje za razvoj grafičnih aplikacij, primer vidimo na sliki 2.4 [7].



Slika 2.4: Spletna aplikacija razvita v okolju WPF.

Sledila je verzija 3.5, ki je s seboj prinesla jezik Language Integrated Query (LINQ). Gre za možnost pisanja poizvedb s kodo, ki je precej podobna objektno orientirani. V času pisanja te diplomske naloge sta bili na trgu različici 4.0 in 4.5.

Naša diplomska naloga je bila razvita z različico 2.0, saj pokriva starejše sisteme, ki nimajo nameščenih novih verzij ogrodja .NET.

2.2.3 XML

Za komunikacijo med uporabniškim vmesnikom in DLL knjižnico smo uporabili parametre v jeziku XML. Kratica pomeni razširljiv označevalni jezik. XML opisuje množico podatkovnih objektov (datoteke, zapisi v relacijski podatkovni bazi) in hkrati do neke mere določa obnašanje programov, ki tak dokument obdelujejo [10].

Jezik XML je soroden HTML-ju, saj na podoben način uporablja zapise z oznakami, vendar značke niso vnaprej definirane. XML nam dovoljuje samostojno določanje oznak, ki imajo pomenski smisel za nas. Zato velja, da je njegova oblika razumljiva tako človeku kot tudi računalniku [11]. Razvit je bil za prenos podatkov in ne za njihovo prikazovanje.

V tem trenutku je najmočnejše orodje za izmenjavo podatkov med različnimi aplikacijami, saj je shranjen v preprosti tekstovni obliki, kar omogoča lažjo komunikacijo med programi, ki so sicer pisani v različnih programskih jezikih. Takšno shranjevanje omogoča lažje prehajanje med različnimi operacijskimi sistemi brez izgube podatkov [12].

Poleg tega, da v jeziku XML določamo svoje značke, lahko z uporabo specifikacije določimo, katere oznake so dovoljene in kakšna je veljavna struktura dokumenta. S shemo Document Type Definition (DTD) ali XML določimo pomene naših podatkov. Na drugi strani aplikacija uporabi tako imenovani potrjevalni razčlenjevalnik XML, ki naš dokument ovrednoti v primerjavi z enim ali več dokumenti sheme XML. Dokument je potrjen, če so pravila, navedena v shemi, upoštevana. Na sliki 2.5 vidimo enostaven primer sheme DTD, na sliki 2.6 pa XML, ki tej shemi ustreza.

```
<!ELEMENT people_list (person)*>
<!ELEMENT person (name, birthdate?, gender?, socialsecuritynumber?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT birthdate (#PCDATA)>
<!ELEMENT gender (#PCDATA)>
<!ELEMENT socialsecuritynumber (#PCDATA)>
```

Slika 2.5: Primer enostavne sheme DTD.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE people_list SYSTEM "example.dtd">
<people_list>
  <person>
    <name>Fred Bloggs</name>
    <birthdate>2008-11-27</birthdate>
    <gender>Male</gender>
  </person>
</people_list>
```

Slika 2.6: XML, ki ustreza shemi prikazani na sliki 2.5.

Dobro sestavljen XML dokument mora upoštevati naslednja pravila:

- imeti mora korenski element
- elementi morajo vsebovati zaključni zaznamek
- zaznamki upoštevajo velikost črk
- elementi morajo biti pravilno gnezdeni
- vrednosti atributov morajo biti v navednicah [13]

Na osnovi jezika XML je bilo razvitih veliko novih internetnih jezikov. Nekateri med njimi so:

- Extensible HyperText Markup Language (XHTML)
- Web Services Description Language (WSDL)
- Wireless Application Protocol (WAP)
- Rich Site Summary (RSS)

2.3 Strežnik Microsoft SQL

V diplomski nalogi smo za shranjevanje slik uporabili strežnik Miscrosoft SQL in sicer od različice 2005 naprej. To je sistem za upravljanje relacijske podatkovne baze. Njegov namen je shranjevanje podatkov ter skrb za izvajanje poizvedb nad njimi.

2.3.1 Relacijska podatkovna baza

Relacijska podatkovna baza shranjuje podatke v tabelah. Tabele so sestavljene iz stolpcev, ki shranjujejo določen tip podatkov. Običajno imajo ključe, to je en ali več stolpcev, ki enoznačno opišejo vrstico tabele. Za izboljšanje dostopnih časov do podatkov se na tabelah nahajajo indeksi sestavljeni iz enega ali več stolpcev. Tabele so med seboj povezane preko ključev, temu pravimo relacije [14].

Najbolj pogosta uporaba relacijske podatkovne baze je implementacija modela Create, Read, Update and Delete (CRUD), to so osnovne operacije, ki jih lahko izvajamo na tabelah. CRUD operacije v jeziku SQL vidimo v programski kodi 2.2.

Programska koda 2.2 : Primer osnovnih CRUD operacij v jeziku SQL

```
1
2   use diplomska_naloga
3
4   INSERT INTO [podjetje_01].[podatki] (sifra,naziv)
5   VALUES('00001','Testni podatek')
6
7   SELECT p.*
8   FROM [podjetje_01].[podatki] p
9   where p.sifra='00001'
10
11  UPDATE [podjetje_01].[podatki]
12  SET naziv='Testni podatek 2'
13  where sifra='00001'
14
15  DELETE FROM [podjetje_01].[podatki]
16  where sifra='00001'
```

Napredna uporaba vključuje:

- shranjevanje objektov
- implementacija obnašanja aplikacije znotraj podatkovne baze - tukaj govorimo o pisanju procedur, funkcij in sprožilcev, ki ležijo na podatkovni bazi in definirajo obnašanje programa

- izvajanje sočasnih zahtev
- upravljanje transakcij - transakcija je skupina akcij, ki se izvajajo zaporedno. Potrdijo se samo takrat, ko se vse operacije izvedejo uspešno, drugače jih sistem zavrne.
- uveljavljanje referenčne integritete - preverjanje konsistentnosti podatkov glede na relacijo

Najboljše orodje za opravljanje poizvedb na relacijski podatkovni bazi je jezik SQL.

2.3.2 Jezik SQL

SQL je kratica za standardnim poizvedbeni jezik. Z njim lahko na relacijski podatkovni bazi opravljamo poizvedbe, operacije CRUD, ustvarjamo nove podatkovne baze in tabele, postavljamo indekse, izdelujemo procedure, funkcije, sprožilce in poglede. Skratka vse, kar nam podatkovna baza omogoča.

2.3.3 FILESTREAM

Kot možnost shranjevanja slik na podatkovno bazo smo uporabnikom, ki imajo nameščeno različico strežnika SQL 2008 ali novejšo, dali na voljo možnost shranjevanja s sistemom FILESTREAM.

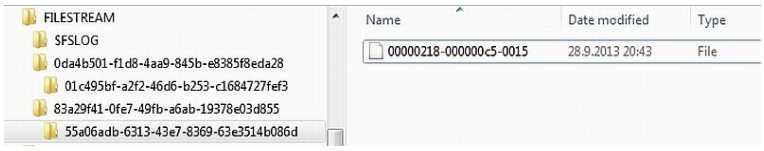
FILESTREAM združuje podatkovno bazo SQL z datotečnim sistemom NTFS tako, da shranjuje zapise tipa varbinary(MAX) oziroma Binary Large OBject (BLOB), kot datoteke na datotečni sistem. Uporablja predpomnilnik sistema NT za predpomnenje datotek in s tem zmanjša obremenitev podatkovne baze, kar posledično pomeni, da ima baza na voljo več spomina za izvajanje poizvedb. Za razliko od običajnega tipa varbinary(MAX), ki ima omejitev velikosti datoteke postavljeno na 2 GB, so podatki z načinom FILESTREAM omejeni le z velikostjo datotečnega sistema [15].

Tak sistem naj bi uporabljali, kadar imamo opravka z datotekami večjimi

od 1 MB. Pri manjših datotekah je običajno hitrejši navaden zapis tipa varbinary(MAX) oziroma image.

Za delovanje sistema moramo pri instalaciji strežnika SQL omogočiti njegovo uporabo. Na tabeli ga implementiramo tako, da označimo stolpec tipa varbinary(MAX), v našem primeru image kot FILESTREAM. S tem strežnik samodejno shranjuje podatke na datotečni sistem in ne več v tabelo.

Podatki tipa FILESTREAM so shranjeni v posebnih datotečnih skupinah, ki namesto datotek hranijo sistemske direktorije. Te mape oziroma tako imenovani podatkovni zabojniki so vmesniki med datotečnim sistemom in podatkovno bazo. Primer takšne strukture vidimo na sliki 2.7 [15].



Slika 2.7: Datotečna struktura FILESTREAM.

Pri uporabi shranjevanja podatkov na takšen način je potrebno upoštevati naslednja pravila:

- tabele, na katerih se nahajajo takšni podatki, morajo imeti enoznačen identifikacijski ključ vrstice, ki ne sme omogočati vrednosti NULL
- gnezdenje takšnih podatkov ni možno
- kadar uporabljamo preklopno grozdenje morajo biti datotečne skupine v skupni rabi

Ker je način FILESTREAM omogočen neposredno na tabeli, nam ni potrebno skrbeti za ločeno zaščito podatkov, saj se podatki ščitijo skupaj z relacijsko podatkovno bazo, lahko pa jih iz zaščite izključimo [15].

2.4 Protokoli za komuniciranje s skenerjem

Eden izmed ciljev diplomske naloge je bil podpreti veliko število skenerjev. Zato smo uporabili dva najpogostejša protokola za komunikacijo z zajemnimi napravami in sicer TWAIN in WIA. V tem poglavju smo ju podrobneje opisali.

2.4.1 TWAIN

TWAIN (ne gre za uradno kratico, vendar je najbolj znana Technology Without An Interesting Name) je standardni protokol in aplikacijski programski vmesnik, ki skrbi za komunikacijo med napravo za zajem slike in aplikacijo. Potreba po takšnem protokolu izhaja iz časov začetkov zajema slik, ko so proizvajalci programske in strojne opreme določali svoje vmesnike za zajem slike. S časoma je postalo jasno, da ta rešitev ni dobra, saj je zahtevala od razvijalcev programske opreme, da napišejo gonilnike za vsako napravo, ki jo želijo podpreti, prav tako ni smiselno, da razvijalci strojne opreme pripravljajo rešitve za posamezne aplikacije. In kar je najpomembnejše, takšne težave obremenijo uporabnike, ki se morajo po nepotrebnem ukvarjati z namestitvami gonilnikov za različne aplikacije in naprave.

Rešitev je prišla v obliki odprtega industrijskega vmesnika, ki se izvaja znotraj aplikacije in neposredno zajema sliko iz zunanje naprave. Na takšen način lahko razvijalci programske opreme uporabijo standardne klice za komunikacijo z zajemnimi napravami, medtem ko proizvajalci strojne opreme pripravijo samo en gonilnik, ki sledi navodilom protokola. Tako so bile zadovoljene potrebe vseh vpletenih. Na eni strani veliko časa in denarja prihranijo proizvajalci strojne in razvijalci programske opreme, saj razvijajo le eno rešitev in posledično tudi uporabniki, ki imajo za eno napravo nameščen le en gonilnik in uporabniški vmesnik [16].

Osnovni cilji razvoja TWAIN protokola so bili in ostajajo:

- podpora velikega števila operacijskih sistemov

- podpreti čim več zajemnih naprav (skenerji, kamere) tako za osebno kot profesionalno rabo
- zagotoviti dobro definiran vmesnik, ki ga razvijalci lahko vključijo v svoje rešitve
- slediti mora razvoju, tako da podpira nove možnosti zajemnih naprav
- biti mora združljiv s starejšimi verzijami
- biti mora enostaven za uporabo
- je razvit kot javni protokol z odprto kodno licenco Lesser General Public License (LGPL), saj s tem spodbuja zunanje razvijalce k izboljšavam sistema [16]

TWAIN zagotavlja preprosto metodologijo za univerzalno povezovanje aplikacij in zajemnih naprav, ki upoštevajo njegova navodila. Komunikacija poteka preko štirih plasti:

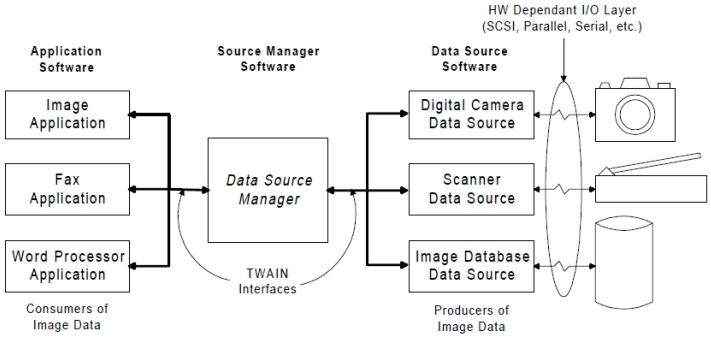
- aplikacijska
- protokolna
- zajemna
- napravna

Komponente za zajem komunicirajo preko teh plasti in vključujejo aplikacijo, upravitelja virov, vir ter fizično zajemno napravo. Aplikacija z upraviteljem virov komunicira z gonilnikom, ki predstavlja fizično napravo.

Strojni del vmesnika TWAIN arhitekture je vir, katerega namen je pridobivanje podatkov iz zajemne naprave in posredovanje aplikaciji. Pripravijo ga proizvajalci strojne opreme in lahko upravlja tako lokalne kot mrežno nameščene, fizične in navidezne zajemne naprave.

Centralni del TWAIN arhitekture je upravitelj virov, ki ustvarja in nadzoruje komunikacijo med aplikacijami in viri. Uporabnikom omogoča izbiro

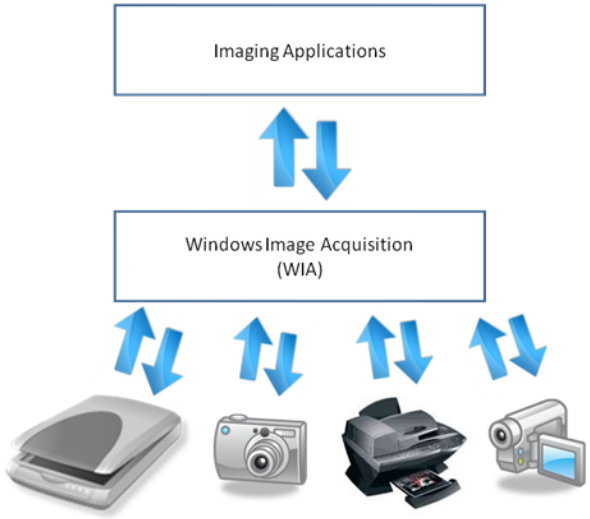
poljubnega vira in njegovo upravljanje. Aplikacija vedno komunicira z zajemno napravo preko upravitelja virov. Na operacijskem sistemu Windows do njega pridemo preko knjižnice DLL [16]. Arhitekturo protokola TWAIN prikazuje slika 2.8.



Slika 2.8: Arhitektura TWAIN.

2.4.2 WIA

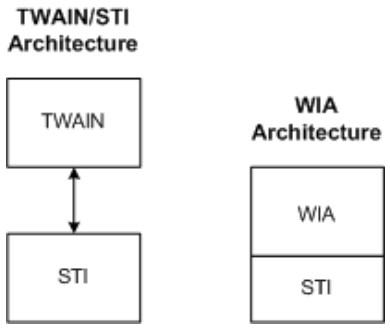
WIA je platforma sistema Windows za komunikacijo z zajemnimi napravami. Prvič se je pojavila v operacijskih sistemih Windows Me in Windows XP. Na sliki 2.9 vidimo način povezovanja zajemnih naprav in aplikacij z arhitekturo WIA.



Slika 2.9: Arhitektura WIA.

Pred tem je bila edina rešitev uporaba TWAIN protokola, ki je komuniciral z nizko nivojno abstrakcijo strojne opreme sistema Windows imenovano Still Image Architecture (STI). WIA je arhitektura, ki je nadgradila sistem STI, kar pomeni, da TWAIN ni več potreben, še vedno pa je podprt. Na sliki 2.10 vidimo arhitekturi WIA in TWAIN ter njuno povezanost na STI [18].

WIA nudi ogrodje, ki napravam omogoča, da operacijskemu sistemu predstavijo svoje lastnosti, aplikacijam pa nudi sklicevanje nanje. Vsebuje protokol za zajem podatkov, Device Driver Model and Interface (DDI), Application Programming Interface (API) in WIA servis, ki teče na operacijskem sistemu [18].



Slika 2.10: Arhitekture WIA in TWAIN v relaciji s STI.

Uporabniški vmesnik WIA nudi aplikacijam:

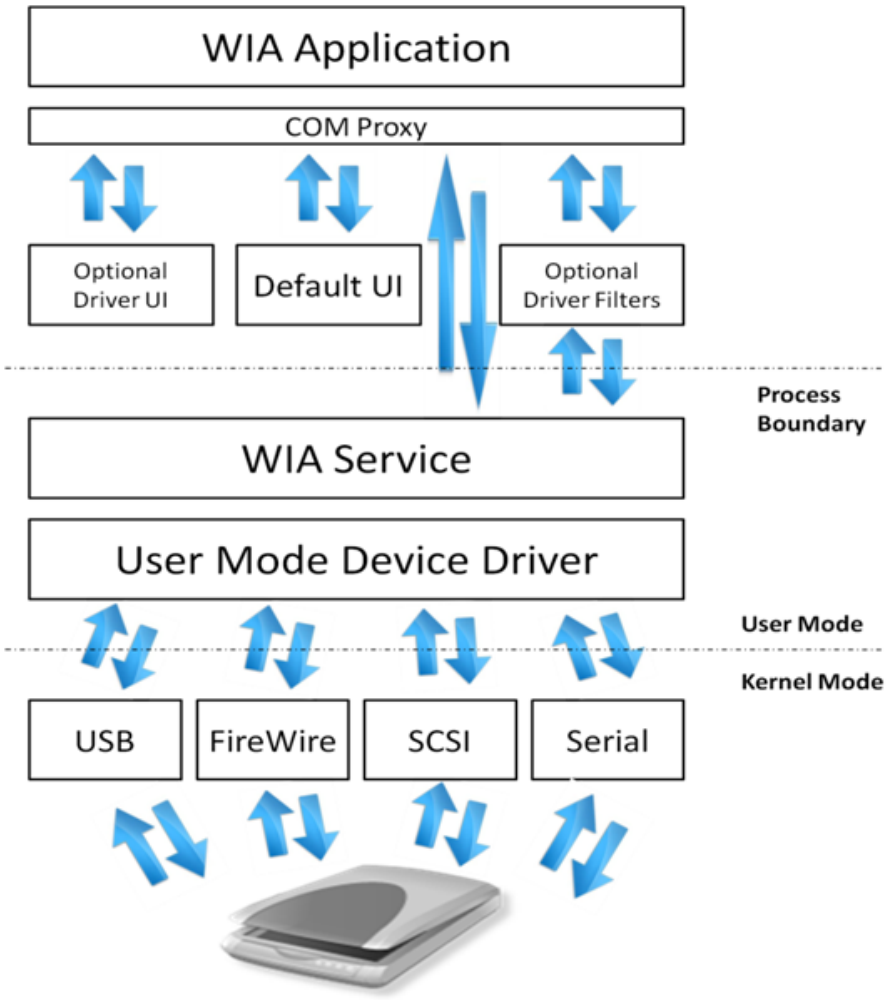
- robustno in stabilno okolje
- izbor kompatibilnih naprav
- sočasno povezovanje na več naprav
- pridobivanje lastnosti naprave
- zajem slik
- obveščanje o dogodkih [19]

WIA DDI je zastavljen tako, da lahko razvijalci strojne opreme s čim manj kode razvijajo samostojne rešitve. To je doseženo z:

- zagotovitvijo standardne servisne knjižnice, ki izvaja večino operacij gonilnika
- podpiranjem velikega števila naprav z enim WIA gonilnikom
- omogočanje napravam, da imajo svoje uporabniške vmesnike in hkrati implementirajo WIA uporabniški vmesnik [19]

WIA ima nabor gonilnikov s katerimi opravlja komunikacijo z napravami priključenimi na računalnik preko USB, serijskega porta, SCSI ali vodila

FireWIre. Njegov podsistem vključuje kompatibilnostno plast, ki aplikacijam narejenih po navodilih TWAIN protokola, omogoča uporabo gonilnikov WIA. Gonilniki WIA so sestavljeni iz uporabniškega vmesnika in jedra gonilnika, ki sta naložena v dva različna procesna prostora in sicer uporabniški vmesnik v aplikacijo, jedro pa v servis, ki teče na operacijskem sistemu. Primer sestave in delovanja je predstavljen na sliki 2.11 [18].



Slika 2.11: Delovanje arhitekture WIA.

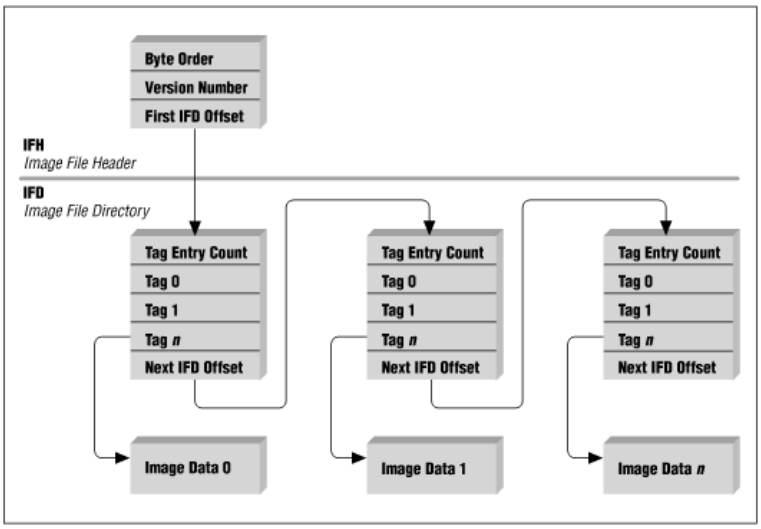
2.4.3 TWAIN ali WIA

Oba protokola imata kar nekaj skupnih točk. Oba lahko zajameta sliko, ugotavljata kakšne lastnosti zajemna naprava podpira in jih nastavljata. Bolj nas zanimajo njune razlike, ki tehtnico prevesijo na stran protokola TWAIN. Njegove prednosti so:

- uporaba proizvajalčevega dialoga za zajem slike - WIA ima na voljo le splošen vmesnik, to običajno pomeni, da ima TWAIN na voljo več možnosti
- shranjevanje nastavitev
- nastavljanje lastnostni posameznih strani pri obojestranskem zajemu
- uporaba tudi tistih možnosti zajemne naprave, ki jih osnovna specifikacija ne podpira
- več možnosti prenosa podatkov (izvorni, v spomin in preko datoteke) [20]

2.5 Format shranjenih dokumentov

V diplomski nalogi smo za shranjevanje zajetih dokumentov izbrali *Tagged Image File Format* (TIFF). Gre za format datotek, ki je bil razvit za potrebe skeniranja z namenom združiti različne formate datotek posameznih proizvajalcev pod eno streho. Je neodvisen od operacijskega sistema, njegovo prvo okolje je bil operacijski sistem Macintosh. Podpira večino barvnih prostorov (RGB, CMYK, ...). Lahko je shranjen brez stiskanja ali z različnimi metodami kompresije (LZW,CITT4 ...). Pomemben faktor pri izboru formata TIFF je bila možnost shranjevanja večstranskih dokumentov v eno datoteko, tako imenovani večstranski TIFF [21]. Logično strukturo datoteke TIFF smo prikazali na sliki 2.12.



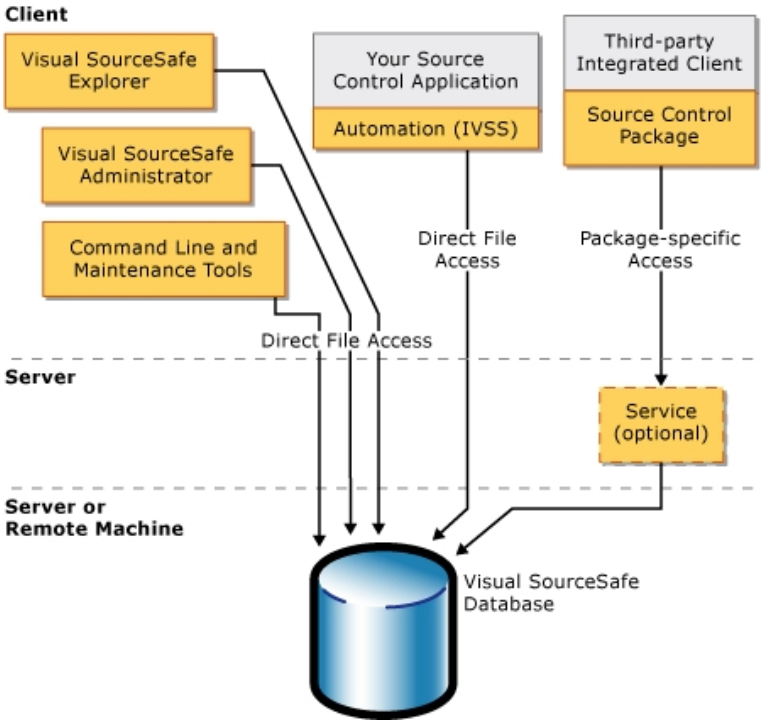
Slika 2.12: Logična struktura datoteke TIFF.

2.6 Vodenje verzij programske kode

Ker so vsa orodja, ki smo jih uporabili pri našem delu od proizvajalca Microsoft, smo za vodenje verzij programske kode uporabili rešitev *Microsoft Visual SourceSafe* (VSS).

Vodenje verzij je pri razvoju aplikacij pomembno predvsem s stališča zaščite programske kode, saj nam omogoča vpogled v programsko kodo preteklih verzij ter beleženj popravkov sprememb, odkrivanje in odpravljanje morebitnih pomanjkljivosti. Pomembno je tudi s stališča sočasnega razvoja aplikacij, saj mora omogočiti deljenje datotek med projekti, hkrati pa mora skrbeti za pravilno združevanje programske kode datotek, ki jih sočasno dopolnjuje več razvijalcev.

VSS deluje po principu klient - strežnik. Programsko koda se hrani centralno na strežniku, med tem ko poseben modul teče na računalnikih razvijalce in jim omogoča dostop do baze s kodo. Z njim lahko razvijalci svoje delo prijavljajo, odjavljajo, razveljavljajo spremembe, spremljajo zgodovino verzij, delijo svojo izvorno kodo ter ustvarjajo nove veje datotek [22]. Arhitekturo VSS smo prikazali na sliki 2.13.



Slika 2.13: Arhitekura VSS.

Poglavje 3

Izvedba projekta

Diplomsko nalogo smo izvedli v dveh korakih. Najprej smo razvili knjižnico DLL za komunikacijo z zajemno napravo, skeniranje in urejanje slike. Nato smo pripravili uporabniški vmesnik, v katerega smo implementirali novo nastali DLL.

3.1 Razvoj knjižnice za zajem in shranjevanje slike

Kot je bilo opisano v prejšnjem poglavju, smo programsko knjižnico za zajem slike razvili z orodjem Microsoft Visual Studio 2012 in sicer v programskem jeziku C#. Najprej smo definirali imenski prostor in osnovni razred. Odločili smo se, da bo knjižnica delovala tudi kot *Component Object Model* (COM) objekt, kar je vidno v programski kodi 3.1. Zaradi varnosti smo navzven odprli samo eno metodo, ki sprejema parameter tipa XML, ta pa določa vrsto nadaljnjih operacij in lastnosti sproženih procesov.

Programska koda 3.1 : Ustvarjanje imenskega prostora in glavnega razreda

```
1  namespace grad_skeniranje
2  {
3      [ClassInterface(ClassInterfaceType.AutoDual)]
4      [ProgId("grad_skeniranje.CZajem_slike")]
5      [ComVisible(true)]
```

```
6       public class CZajem_slike
7       {
8
9               public string sprozi_operacijo(string lps_parameter)
10               {
11
12                   XmlDocument lpx_parametri = new XmlDocument();
13
14                   lpx_parametri.LoadXml(lps_parameter);
15
16                   cls_skeniranje skeniranje = new cls_skeniranje();
17                   string ls_vrni = "";
18                   skeniranje.proces_skeniranja(lpx_parametri, ref ls_vrni)
                         ;
19
20                   Stream str = Properties.Resources.Speech_On;
21                   SoundPlayer snd = new SoundPlayer(str);
22                   snd.Play();
23
24
25                   lpx_parametri = null;
26                   skeniranje = null;
27
28
29                   return ls_vrni;
30               }
31      }
32  }
```

Program mora ugotoviti kaj uporabnik od njega zahteva, zato smo najprej prebrali vsebino parametra XML. Ker vhodne parametre potrebujemo v različnih delih programa, smo shranili njihovo vsebino v objekt. Na ta način smo si olajšali delo, saj je bil objekt s tako pripravljenimi parametri viden skozi celoten imenski prostor, kar smo dosegli z uporabo kontrolne besede *public*. To prikazuje programska koda 3.2.

Programska koda 3.2 : Branje XML parametrov v objekt

```
1   namespace Skeniranje
2   {
3       internal class Parametri_skeniranja
4       {
5           public int i_izberi_scanner = 0;
6           public int i_stevilo_DPI = 150;
```

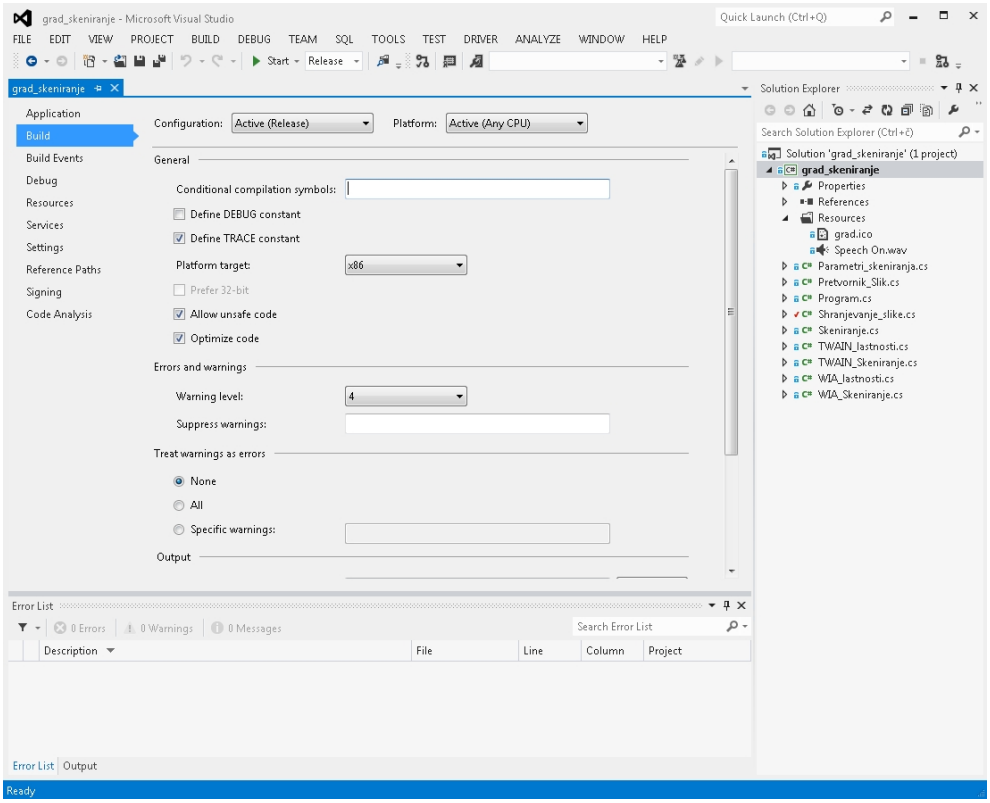
```
7           public int i_barve = 0;
8           public int f_nastavi_parametre(XmlDocument lpx_parameter)
9           {
10              try
11              {
12                  XmlNode xml_vozlisce = lpx_parameter.SelectSingleNode("/
                        parametri/izberi_scanner");
13                  if (xml_vozlisce != null)
14                      i_izberi_scanner = Int32.Parse(xml_vozlisce.
                            FirstChild.Value);
15
16                  xml_vozlisce = lpx_parameter.SelectSingleNode("/
                        parametri/stevilo_DPI");
17                  if (xml_vozlisce != null)
18                      i_stevilo_DPI = Int32.Parse(xml_vozlisce.FirstChild.
                            Value);
19          ...
20                  return 1;
21              }
22              catch
23              {
24                  MessageBox.Show("Napaka pri nastavljanju parametrov!", "
                        Napaka!", MessageBoxButtons.OK, MessageBoxIcon.Error
                        );
25                  return -1;
26              }
```

Naslednji korak je ugotavljanje in izvedba uporabnikove zahteve. S parametrom lahko sproži naslednje operacije:

- zajem slike s protokolom TWAIN ali WIA
- napredni zajem slike (možnost skeniranja s pomočjo skenerjevega uporabniškega vmesnika)
- brisanje, zamenjava, vrivanje in štetje strani
- shranjevanje posameznih strani dokumenta v formatu BMP
- izbor privzete zajemne naprave

3.1.1 Zajem slike s protokolom TWAIN

Za delovanje zajema slike s protokolom TWAIN smo morali najprej naložiti določene COM objekte. Zato je bilo potrebno aplikaciji dovoliti izvajanje tako imenovane nevarne kode. To smo storili v nastavitvah projekta, kar smo prikazali na sliki 3.1.



Slika 3.1: Omogočanje nevarne kode.

V začetni definicij smo uvozili metode, ki jih uporabljamo za krmiljenje zajemne naprave. Uvoz metod je prikazan v programski kodi 3.3.

Programska koda 3.3 : Uvoz metod iz knjižnice DLL

```
1
2       internal static class NativeMethods
3       {
4
5         [DllImport("twain_32.dll", EntryPoint = "#1")]
6         internal static extern TwainReturnCode DSuserif([In, Out]
              TwainIdentity origin, [In, Out] TwainIdentity dest,
              TwainDataGroups dg, TwainDataArgumentType dat, TwainMessage
              msg, TwainUserInterface guif);
7         ...
8     }
```

Po pripravi definicij smo se lotili nastavljanja začetnega okolja. Tu smo naleteli na prvo težavo. Ugotovili smo, da je za nadziranje skenerja potrebno ustvariti novo okno. Za očeta tega okna smo določili glavno okno uporabniškega vmesnika in s tem dosegli, da se zajem slike izvaja znotraj okna aplikacije. Ročaj aplikativnega okna smo prenesli v metodo preko parametra. Primer ustvarjanja novega okna z očetom je prikazan v kodi 3.4.

Programska koda 3.4 : Ustvarjanje okna

```
1
2         if (o_param_skeniranje.HandleFoxOkno != 0)
3               {
4                   glavnoOkno=(IntPtr)o_param_skeniranje.HandleFoxOkno;
5               }
6               else
7               {
8                   glavnoOkno=NativeMethods.GetDesktopWindow();
9               }
10
11              hNovoOkno = NativeMethods.CreateWindowEx(0,"STATIC",
12                      "Acquire Proxy",
13                      WindowStyles.WS_POPUPWINDOW,
14                      (int)WindowStyles.CW_USEDEFAULT, (int)WindowStyles.
                            CW_USEDEFAULT,
15                       (int)WindowStyles.CW_USEDEFAULT, (int)WindowStyles.
                             CW_USEDEFAULT,
```

```
16                      glavnoOkno,
17                      System.IntPtr.Zero,
18                      hInstance,
19                      System.IntPtr.Zero);
```

Ustvarjanju okolja sledi izbor in odpiranje zajemne naprave. To vidimo v programski kodi 3.5. V osnovnem načinu zajema slike se uporablja privzeta naprava, v naprednem pa jo izbere uporabnik.

Programska koda 3.5 : Izbor naprave.

```
1
2           if (o_param_skeniranje.i_izberi_scanner == 1 ||
                o_param_skeniranje.i_nacin_zajema== 1)
3           {
4              naprava = SelectSource(appid);
5               if (o_param_skeniranje.i_izberi_scanner == 1)
6                   return ;
7           }
8           else
9           {
10              naprava = GetSource(appid);
11          }
12          if (naprava.Id == IntPtr.Zero || naprava == null)
13          {
14              return;
15          }
```

Nato pričnemo z nastavljanjem lastnosti poteka zajema in izgleda slike. Nastaviti poizkušamo:

- velikost strani
- barvno globino
- ločljivost
- zajem slik iz podajalca
- obojestranski zajem
- odstranjevanje praznih strani

Veliko zajemnih naprav ne podpira vseh možnosti, zato najprej sprožimo poizvedbo podpore posamezni lastnosti in jo šele v primeru pozitivnega odgovora nastavimo. V programski kodi 3.6 sta prikazani funkciji, ki opravljata branje in nastavljanje posameznih lastnosti naprave.

Programska koda 3.6 : Funkciji za branje in nastavljanje lastnosti naprave.

```
1
2           //funkcija, ki nastavlja vrednost po vrednost
3           private static void f_nastavi_eno_vrednost(TwainIdentity appid,
                TwainIdentity naprava, TwainCapabilityType TwTip, TwainType
                ntiptwain, short ln_vrednost)
4           {
5               TwainCapability ptCap = new TwainCapability(TwTip,ntiptwain,
                     ln_vrednost);
6               var rc = NativeMethods.DScap(appid, naprava, TwainDataGroups
                    .Control, TwainDataArgumentType.Capability, TwainMessage
                    .Set, ptCap);
7           }
8
9           //preberem vrednost lastnosti iz skenerja
10          private static TwainReturnCode f_beri_eno_vrednost(TwainIdentity
                 appid, TwainIdentity naprava, TwainCapabilityType TwTip,
                TwainType ntiptwain)
11          {
12              TwainCapability ptCap = new TwainCapability(TwTip);
13             return NativeMethods.DScap(appid, naprava, TwainDataGroups.
                   Control, TwainDataArgumentType.Capability, TwainMessage.
                   Get, ptCap);
14          }
```

Na podlagi testiranja različnih zajemnih naprav smo ugotovili, da ima možnost odstranjevanja praznih strani drugačen naslov lastnosti glede na proizvajalca programske opreme. Do tega pride zaradi različnega časovnega razvoja naprav in knjižnice TWAIN. Proizvajalci namreč lahko določeno možnost razvijejo prej kot jo TWAIN podpre. Razširljivost knjižnice TWAIN poskrbi, da se lahko uporabljajo tudi funkcije, ki jih formalno še ne podpira in sicer tako, da proizvajalec za vpis lastnosti uporabi v naprej pripravljene naslove knjižnice TWAIN. Tako smo iz proizvajalčeve dokumentacije ugotovili pravi naslov in v programu pripravili kodo, ki iz zajemne naprave prebere

proizvajalca in nato določi ustrezen naslov lastnosti, ki jo želimo nastaviti. To smo prikazali v programski kodi 3.7.

Programska koda 3.7 : Branje proizvajalca in nastavljanje lastnosti.

```
1      if (naprava.Manufacturer.ToLower().Contains("canon"))
2      {
3          if (f_beri_eno_vrednost(appid, naprava, TwainCapabilityType.
               AutoDiscardBlankPagesCannon, TwainType.Bool) ==
               TwainReturnCode.Success)
4          {
5
6              f_nastavi_eno_vrednost(appid, naprava, TwainCapabilityType.
                   DuplexEnabled, TwainType.Bool, 0);
7
8              f_nastavi_eno_vrednost(appid, naprava, TwainCapabilityType.
                   AutoDiscardBlankPagesCannon, TwainType.Bool, 1);
9              f_nastavi_eno_vrednost(appid, naprava, TwainCapabilityType.
                   AutoDiscardBlankPagesCannonTrashold, TwainType.Int32,
                   10);
10
11             o_param_skeniranje.i_brisi_prazne = 0;
12         }
13
14     }
```

Po nastavljanju lastnosti je potrebno napravo omogočiti. Tu se proces naprednega zajema ponovno loči od osnovnega. Pri naprednem skeniranju v trenutku omogočanja pokličemo vmesnik zajemne naprave s katerim opravljamo skeniranje. Ta možnost je koristna takrat, ko želimo uporabiti način skeniranja, ki ga osnovne nastavitve ne podpirajo (predogled, rotacija, višja kakovost ...). Za uspešno komunikacijo med skenerjevim vmesnikom in našim programom je bilo potrebno ustvariti komunikacijsko zanko, ki interpretira sporočila vmesnika. To smo prikazali v programski kodi 3.8.

Programska koda 3.8 : Komunikacijska zanka.

```
1    Application.AddMessageFilter(this);
2      Application.Run();
```

Omogočanju naprave sledi proces zajema slike, ki je enak za napredni in

osnovni zajem. Naprej ustvarimo prazen seznam strani tipa BMP. Nato preberemo lastnosti slike ter sprožimo proces skeniranja, ki nam vrne dokument tipa BMP, ki ga dodamo v seznam. Če uporabljamo zajemno napravo, ki uporablja podajalec in je uporabnik izbral paketni ali obojestranski zajem, ponavljamo zgornji proces toliko časa, dokler so v podajalcu dokumenti. Navzgor smo ga omejili z 15500 stranmi, kar mora zadostovati potrebam vseh uporabnikov. Zanko zajema slike smo prikazali v programski kodi 3.9.

Programska koda 3.9 : Proces zajema slike.

```
1
2       var pictures = new List<Bitmap>(); //naredim seznam strani BMP
3
4       TwainReturnCode rc;
5       IntPtr hbitmap;
6       var pxfr = new TwainPendingXfers(); //dokumenti v podajalcu
7       int li_stevec = 0;
8       pxfr.Count = 15500; //najvecje stevilo dokumentov
9
10      if (o_param_skeniranje.i_paketno == 0)
11          pxfr.Count = 1;
12      do
13      {
14         rc = NativeMethods.DSiinf(appid,
15                                   scanner,
16                                   TwainDataGroups.Image,
17                                   TwainDataArgumentType.ImageInfo,
18                                   TwainMessage.Get,
19                                   iinf);
20
21        rc = NativeMethods.DSixfer(appid,
22                                    scanner,
23                                    TwainDataGroups.Image,
24                                    TwainDataArgumentType.ImageNativeXfer,
25                                    TwainMessage.Get,
26                                    ref hbitmap);
27
28        rc = NativeMethods.DSpxfer(appid,
29                                    scanner,
30                                    TwainDataGroups.Control,
31                                    TwainDataArgumentType.PendingXfers,
32                                    TwainMessage.EndXfer,
33                                    pxfr);
34        if (hbitmap != IntPtr.Zero)
35        {
```

```
36              var bmp = TwainBitmapConvertor.ToBitmap(hbitmap);
37              pictures.Add(bmp);
38          }
39
40      } while (pxfr.Count != 0 )
```

Po zaključenem procesu skeniranja napravo onemogočimo in zapremo, sliko pa pošljemo v proces shranjevanja.

3.1.2 Zajem slike s protokolom WIA

Uporaba protokolom WIA je implementirana v ogrodje .NET, zato je za njegovo uporabo dovolj, da v glavi programa navedemo imenski prostor WIA. Postopek zajema pričnemo z izborom naprave. V primerjavi s protokolom TWAIN pride v tej točki do prve razlike, saj je pri protokolom WIA potrebno posebej shraniti zadnjo uporabljeno napravo. To storimo s pisanjem v bazo SQL preko uporabniškega vmesnika, nato pa podamo identifikacijsko oznako zadnje uporabljene naprave kot parameter skeniranja. V programski kodi 3.10 vidimo način izbora zajemne naprave.

Programska koda 3.10 : Izbor zajemne naprave s protokolom WIA.

```
1       if (o_param_skeniranje.i_izberi_scanner == 1)
2       {
3           naprava = dialog.ShowSelectDevice(WiaDeviceType.
                ScannerDeviceType, false, false);
4       }
5       else
6       {
7           for(int i = 1;i<deviceManager.DeviceInfos.Count;i++)
8           {
9               if (string.Compare(deviceManager.DeviceInfos[i].DeviceID.
                    ToString(), o_param_skeniranje.ls_id_naprave) == 0)
10              {
11                  naprava = deviceManager.DeviceInfos[i].Connect();
12              }
13          }
14
15          if(naprava==null)
16              naprava = deviceManager.DeviceInfos[1].Connect();
17
```

```
18      }
```

Sledi nastavljanje lastnosti zajema. V programski kodi 3.11 prikazujemo zanko sprehoda po lastnostih naprave.

Programska koda 3.11 : Sprehod po lastnostih naprave.

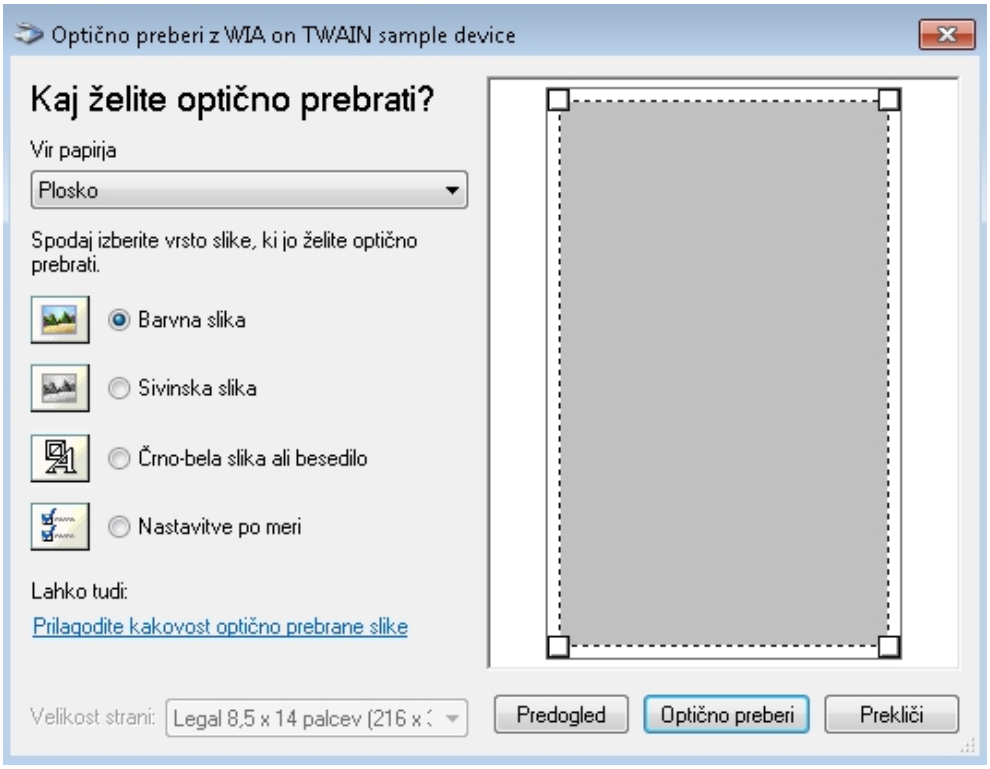
```
1       foreach (Property item in naprava.Items[1].Properties)
2       {
3           switch (item.PropertyID)
4           {
5               case 3097:
6                   SetProperty(item, 0);
7                   break;
8            ...
9            }
10      }
```

Temu sledi zajem slike. Ponovno ločimo napredni in osnovni zajem, s to razliko, da je pri naprednem uporabljen uporabniški vmesnik operacijskega sistema Windows. Implementacija naprednega zajema je bolj enostavna kot pri protokolu TWAIN, saj ni potrebno ustvarjati dodatne komunikacijske zanke. Slaba stran je manjše število možnosti, ki jih lahko pri takšnem zajemu nastavimo. Klic naprednega zajema slike je prikazan v programski kodi 3.12, uporabniško okno, ki ga uporablja WIA pa smo prikazali na sliki 3.2.

Programska koda 3.12 : Koda za zajem slike.

```
1     skeniranaslika = dialog.ShowAcquireImage(
2         WiaDeviceType.ScannerDeviceType,
3         WiaImageIntent.UnspecifiedIntent,
4         WiaImageBias.MaximizeQuality,
5         FormatID.wiaFormatTIFF,
6         true, false, false);
```

Paketni in obojestranski zajem opravljamo v zanki, ki preverja število strani v podajalcu in kliče metodo za zajem slike. Kot rezultat vsakega klica



Slika 3.2: Napredni zajem s sistemom WIA.

zajema dobimo sliko, tokrat v formatu TIFF, ki jo nato neposredno dodamo v dokument. Za osvežitev spomina, metoda s TWAIN protokolom pripravi seznam strani tipa BMP, ki jih nato shranimo v končni TIFF. S tem je proces zajema slike zaključen zato napravo ponovno sprostimo.

3.2 Operacije na dokumentih

V tem razdelku smo opisali proces shranjevanja posameznih strani tipa BMP v večstransko sliko tipa TIFF. Posvetili smo se še opisu operacij, ki jih na slikah lahko izvaja uporabnik.

Pri shranjevanju slike je potrebno najprej nastaviti ustrezne kodirnike. Pri tem gre za določanje ustreznega formata, oblike, barvne globine in načina stiskanja dokumenta. V diplomski nalogi smo uporabili format datoteke TIFF v večstranski obliki. Barvno globino določimo glede na uporabnikov izbor, za črno bele dokumente uporabimo 2, za sivinske 8 in za barvne 24 bitov. Za stiskanje smo pri črno belih dokumentih uporabili metodo CCITT4, za vse ostale LZW. Primer nastavljanja kodirnikov je prikazan v programski kodi 3.13.

Programska koda 3.13 : Nastavljanje kodirnikov slike.

```
1       encoderInfo = GetEncoderInfo("image/tiff");
2
3       encoderParams = new EncoderParameters(3);
4
5       encoderParams.Param[0] = new EncoderParameter(Encoder.SaveFlag, (
            long)EncoderValue.MultiFrame);
6       encoderParams.Param[1] = new EncoderParameter(Encoder.Compression, (
            long)EncoderValue.CompressionLZW);
7
8       long ln_globina = 24L;
9       switch (o_param_skeniranje.i_barve)
10          {
11          case 0:
12              {
13                  encoderParams.Param[1] = new EncoderParameter(Encoder.
                        Compression, (long)EncoderValue.CompressionCCITT4);
14                  ln_globina = 1L;
15                  break;
```

```
16              }
17          ...
18          }
19      encoderParams.Param[2] = new EncoderParameter(Encoder.ColorDepth,
            ln_globina);
```

Tu smo naleteli na zaplet z velikostjo datotek. Ugotovili smo, da nastavljanje barvne globine kodirnika slike ne zmanjša. Rezultat je sicer bila slika želene barvne kakovosti, vendar je fizično vedno uporabila 32 bitno globino. To je pomeni tudi do desetkrat večje slike. Razlika je bila vidna predvsem pri črno belih dokumentih, ki uporabljajo način stiskanja CCITT4, saj se stiskanje takšne datoteke ni pravilno izvedlo. Zato smo za vsako barvno globino pripravili pretvornik slik v pravilno obliko, primer pretvorbe iz 32 v 24 bitno globino vidimo v programski kodi 3.14. Podobne metode smo uporabili tudi pri črno beli in sivinski barvni globini.

Programska koda 3.14 : Pretvornik slike iz 32 v 24 bitno globino.

```
1   internal Bitmap SpremeniV24bpp(Image slika)
2   {
3       var bmp = new Bitmap(slika.Width, slika.Height, System.Drawing.
            Imaging.PixelFormat.Format24bppRgb);
4       using (var gr = Graphics.FromImage(bmp))
5           gr.DrawImage(slika, new Rectangle(0, 0, slika.Width, slika.
                Height));
6       return bmp;
7   }
```

Po nastavljanju kodirnikov in pretvorbe je dodajanje strani v TIFF dokaj preprosto. Prikazujemo ga v programski kodi 3.15. Ponovno gre za sprehod po datoteki. Če ne obstaja, jo ustvarimo, nato pa ji dodajamo posamezne strani. Na takšen način izvajamo tudi operacije brisanja, zamenjave, vrivanja in dodajanja strani. Pri teh operacijah dobimo kot vhodni parameter številko strani, na kateri želimo izvesti akcijo. Nato v zanki naredimo sprehod po dokumentu in operacijo izvedemo. Pri brisanju stran preprosto izločimo in vse strani, ki ji sledijo, premaknemo za eno nazaj. Pri vrivanju delamo

premik strani naprej.

Programska koda 3.15 : Shranjevanje strani v dokument

```
1       if (!File.Exists(zacasno_ime))
2       {
3           nov_dokument = s_stran;
4           nov_dokument.Save(zacasno_ime, encoderInfo, encoderParams);
5           encoderParams.Param[0] =   new EncoderParameter(Encoder.SaveFlag
                ,(long)EncoderValue.FrameDimensionPage);
6        }
7        else
8        {
9
10           nov_dokument.SaveAdd(s_stran, encoderParams);
11       }
```

3.2.1 Brisanje praznih strani

V diplomski nalogi smo podprli brisanje praznih strani na dva načina. Boljši način je nastavljanje lastnosti zajemne naprave tako, da takšne strani v procesu skeniranja samodejno izloči glede na podano toleranco. Težava je v tem, da nekatere zajemne naprave podpirajo obojestranski zajem slike, vendar praznih strani ne znajo izločati. Na tak način prihaja do velikega števila nepotrebnih podatkov, ki zasedajo prostor in upočasnjujejo delovanje sistema, saj povečujejo čas prenosa posamezne datoteke. Za takšne naprave smo razvili programsko rešitev, ki glede na nastavitev praga ugotavlja ali je stran prazna in jo v primeru pozitivnega odgovora izloči iz shranjevalnega procesa.

Funkcija deluje tako, da se sprehodi po vseh bitih v sliki in računa njihovo svetlost. Stran je izločena, ko je odstotek svetlih delov večji od podane tolerance. Ker sprehod po vseh bitih predstavlja veliko časovno obremenitev, smo pripravili funkcijo, ki nam vrne tabelo vseh bitov, saj je sprehod po takšni tabeli precej hitrejši.

3.3 Uporabniški vmesnik

Uporabniški vmesnik smo razvili v programskem jeziku VFP9. Pri tem smo uporabili že razvite razrede Gradovih aplikacij. Najprej smo pripravili grafično okolje, v katerem uporabnik lahko nastavlja parametre zajema slike. Vrednosti parametrov shranjujemo na strežnik SQL. Pripravili smo razred, ki naloži programsko knjižnico za zajem slike in kliče njeno vhodno funkcijo.

3.3.1 Razred za klic skeniranja

Za povezavo uporabniškega vmesnika in knjižnice za zajem slike smo v programskem jeziku VFP9 pripravili poseben razred. Za to smo se odločili, ker smo predvideli uporabo zajema slike na različnih mestih. V programski kodi 3.16 prikazujemo definicijo razreda za skeniranje, napisano v programskem jeziku VFP9.

Programska koda 3.16 : Definicija razreda v programskem jeziku VFP9.

```
1  DEFINE CLASS c_skeniranje as osnovni_custom OF sql_razredi.prg
2    _c_dll='Grad_Skeniranje.dll'
3    _c_funkcija='sprozi_operacijo'
4    _c_razred_skernianje='grad_skeniranje.CZajem_slike'
5    ...
6    FUNCTION Init
7      this.n_barvna_globina=o_setup.preberi_num('si_barvna_globina',2,.T
           .)
8    ...
9    ENDFUNCTION
10 ENDDEFINE
```

Kot vidimo, razred sestavljajo različne lastnosti, ki določajo njegovo obnašanje. Nekatere izmed njih so fiksne in jih uporabniki ne morejo nastavljati. To so naziv programske knjižnice za zajem slike, njen imenski prostor, ime vhodne funkcije in druge. Prednost takšnega dela je, da morebitne spremembe nazivov ali celotne knjižnice izvedemo samo na enem mestu.

V kodi 3.16 vidimo tudi funkcijo init, ki je konstruktor razreda. V njej preberemo vse parametre, ki jih je uporabnik določil v nastavitvah. Vse potrebne parametre iz objekta pretvorimo v obliko XML in kličemo funkcijo navedeno na lastnostih razreda, v našem primeru je to vedno 'sprozi_operacijo'.

Programski jezik VFP9 ni del ogrodja .NET, zato je potrebno knjižnice pisane v takšnem okolju posebej naložit. Pred nalaganjem je potrebno najprej preveriti obstoj ustrezne različice ogrodja .NET.

Ko smo poizkušali knjižnico naložiti v uporabniški vmesnik smo naleteli na težavo. Večina aplikacij, ki uporabljalo našo knjižnico se namreč nahaja na mreži, ogrodje .NET pa vsebuje določene varnostne omejitve, ki preprečujejo izvajanje programov preko omrežja. Uporabnikov nismo želeli obremenjevati z nastavljanjem varnostnih politik, zato smo morali pripraviti drugačno rešitev.

Problem smo rešili s kopiranjem DLL-ja na lokalni disk v mapo aplikacijskih podatkov. Pri tem smo morali poskrbeti, da bo uporabnik imel vedno na voljo zadnjo verzijo, hkrati pa nismo želeli vsakokratnega kopiranja. V ta namen pred kopiranjem preverjamo verzijo datoteke DLL in prenašamo samo novejše različice, to smo prikazali v programski kodi 3.17.

Programska koda 3.17 : Preverjanje verzij in prenos na lokalni disk

```
1  LOCAL lo_fscript,lc_verzij1,lc_verzija2
2  lo_fscript=CREATEOBJECT('Scripting.FileSystemObject')
3  lc_verzij1=lo_fscript.GetFileVersion(oproda_sql_dll_path)
4  lc_verzij2=lo_fscript.GetFileVersion(app_data_dll_path)
5
6  IF  lc_verzij1>lc_verzij2
7    TRY
8      lo_fscript.CopyFile(oproda_sql_dll_path,app_data_dll_path,.T.)
9    CATCH
10     n_napaka=1
11   ENDTRY
12 ENDIF
13 IF FILE(app_data_dll_path)
14   n_napaka=0
15 ELSE
16   n_napaka=1
17 ENDIF
```

Lokalno knjižnico naložimo brez težav in pokličemo funkcijo. Ta postopek prikazuje koda 3.18. Uporabnik na zaslon dobi samo morebitne napake.

Programska koda 3.18 : Klic funkcije za zajem slike

```
1  lpc_rezultat=''
2  LOCAL lc_napaka,lo_sporocilo,lo_napaka,ln_Vrni
3  lc_napaka=''
4  lo_sporocilo=this.CreateClrInstanceFrom(FULLPATH(this.
       c_pot_dll_za_klic),lpc_dll,@lc_napaka)
5  IF ISNULL(lo_sporocilo)
6    o_dialogi.napaka(lc_napaka)
7    RETURN -1
8  ENDIF
9  lpc_function='lo_sporocilo.'+lpc_function+"(lpc_xml)"
10 ln_Vrni=0
11 IF l_brez_try==.f.
12   TRY
13     lpc_rezultat=&lpc_function
14     this.UnloadClrDll()
15     ln_Vrni=1
16   CATCH TO lo_napaka
17     ln_Vrni=-1
18   ENDTRY
19 ELSE
20   lpc_rezultat=&lpc_function
21   this.UnloadClrDll()
22   ln_Vrni=1
23 ENDIF
```

Po pripravi razreda smo se lotili grafičnega dela uporabniškega vmesnika. Pripravili smo gumb za zajem slike, ki naloži pripravljen razred in izvede skeniranje. V programski kodi 3.19 smo prikazali ustvarjanje objekta iz razreda in klic funkcije.

Programska koda 3.19 : Ustvarjanje razreda v programskem jeziku VFP9

```
1  this.objekt_sken=NEWOBJECT('c_skeniranje','skeniranje_logika.prg')
2  this.objekt_sken.izvedi_scan(.T.)>0
3  this.objekt_sken=NULL
```

Gumb smo pripravili kot razred. Cilj je bil olajšanje dela programerjem, ki bodo uporabili možnost zajema slike v svojih aplikacijah. Tako lahko z minimalnim trudom dosežejo delovanje skeniranja. Potrebno je le prenesti gumb na vnosno masko ter določiti pot in ime s katerim naj se slika shrani, tabelo na strežniku SQL in ime ključnega polja. Vse ostalo sistem opravi sam. S tem smo izpolnili cilj objektnega programiranja, saj naš objekt deluje po sistemu črne škatle.

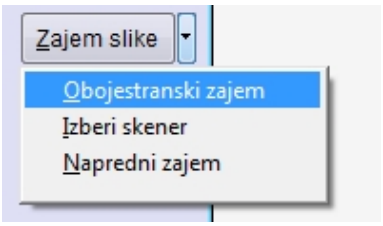
Proces zajema slike preko gumba je zastavljen tako, da lahko sproži proces skeniraja, hkrati pa uporabniku dovoli nadaljnje delo. Do takšne potrebe je prišlo, ker je pri uporabnikih veliko število počasnih zajemnih naprav. S tem, ko uporabniku omogočimo nadaljevanje vnosa, smo ta problem izločili, saj vnos meta podatkov običajno traja dlje kot zajem dokumenta. Da pa uporabnik ve, kdaj je bil proces zajema zaključen ga opozorimo z zvočnim signalom.

Pri pripravi tega sistema smo imeli težave z enonitnim programskim jezikom VFP9. Pripraviti smo morali dodaten program, ki naloži knjižnico za zajem slike in opravi skeniranje s parametri, ki mu jih posreduje uporabniški vmesnik. Po končanem procesu zajema mora sliko prenesti na pravo lokacijo, ali pa jo preprosto izbrisati, če je prišlo do uporabnikovega preklica operacije. Dodaten program smo napisali v programskem jeziku VFP9 in v njem uporabili isti razred za klic zajema slike kot v uporabniškem vmesniku. Sliko na datotečni sistem oziroma strežnik shranjuje tisti program, ki kasneje zaključi operacijo. Primer, če uporabnik vnese podatke pred koncem zajema slike in pritisne potrditve, potem dodatni program dobi informacijo o primarnem ključu podatka in on opravi shranjevanje, če pa se je proces skeniranja zaključil pred potrditvijo, potem datoteko shranjuje uporabniški vmesnik, saj se proces zajema ne obstaja več. S tem smo preprečili morebitno izgubo dokumenta, ki bi se lahko pripetila zaradi težava bodisi pri vnosu bodisi v procesu skeniranja. V programski kodi 3.20 smo prikazali funkcijo za zagon programa v svoji procesni niti.

Programska koda 3.20 : Zagon program v svoji procesni niti

```
1  FUNCTION zazeni
2    PARAMETER s_program,s_commandline,lp_skrij_okno
3
4    LOCAL n_vrnjena_vrednost
5
6    DECLARE INTEGER ShellExecute IN SHELL32.dll INTEGER nWinHandle,
         STRING cOperation, STRING cFileName, STRING cParameters, STRING
         cDirectory, INTEGER nShowWindow
7    DECLARE INTEGER FindWindow IN WIN32API STRING cNull, STRING cWinName
8
9    IF !lp_skrij_okno
10     n_vrnjena_vrednost=ShellExecute(FindWindow( 0, _SCREEN.caption), "
           Open",s_program,s_commandline, "", 1)
11   ELSE
12     n_vrnjena_vrednost=ShellExecute(FindWindow( 0, _SCREEN.caption), "
           Open",s_program,s_commandline, "", 0)
13   ENDIF
14
15   CLEAR DLLS FindWindow
16   CLEAR DLLS ShellExecute
17
18   RETURN n_vrnjena_vrednost
19 ENDFUNC
```

Na gumb smo dodali tudi dodaten meni, ki uporabnikom omogoča izbor nekaterih naprednih možnosti zajema brez proženja nastavitev. Gumb s pripadajočim menijem vidimo na sliki 3.3.



Slika 3.3: Gumb za zajem slike.

Za prikaz slike ter njen izvoz in pošiljanje po elektronski pošti smo uporabili programske knjižnice, ki so standardni del Gradovih aplikacij. Izgled uporabniškega vmesnika smo predstavili v poglavju 4.

3.4 Shranjevanje in branje slik s strežnika

Kot rečeno, lahko slike shranjujemo na datotečni sistem ali na strežnik SQL. Pri tem omogočamo uporabo sistema FILESTREAM. Po uspešno končanem zajemu slika leži na datotečnem sistemu. V tej točki je potrebno sliko, glede na željo uporabnika, najprej spremeniti v šifriran dokument in nato po potrebi prenesti na strežnik. Šifriranje smo izvedli z zato pripravljeno knjižnico. Nato smo se lotili prenosa na strežnik.

Uporaba sistema FILESTREAM je postala na voljo z različico Microsoftovega strežnika SQL 2008. Ker Gradove aplikacije podpirajo strežnike z verzijo 2005, smo pripravili rešitev, ki preverja nastavitve strežnika in nato omogoči sistema FILESTREAM in doda potrebna polja na tabele.

Najprej preverimo ali je možnost FILESTREAM na voljo. To smo prikazali v programski kodi 3.21.

Programska koda 3.21 : Preverjanje možnosti FILESTREAM

```
1  SELECT COUNT(*) as fs_obstaja
2  FROM sys.configurations
3  WHERE description LIKE '%filestream access level%'
```

Kadar je FILESTREAM na podatkovni bazi omogočen, preverimo ali na ciljni tabeli že obstaja stolpec takšnega tipa. Če ga ni, to je običajno samo ob prvem poizkusu shranjevanja, ga ustvarimo. Prikaz ustvarjanja smo pripravili v programski kodi 3.22, v njej lahko vidimo tudi prepis starih podatkov na nov sistem.

Programska koda 3.22 : Prikaz dodajanja stolpca tipa FILESTREAM v tabelo

```
1  declare @ln_stevilo_stolpcev int
2  set @ln_stevilo_stolpcev=(select count(*) as obstaja from sys.columns
       where Name = N'<<lpc_stolpec>>'
3    and Object_ID = Object_ID(N'<<this.c_tabela_fs>>'))
4
5    if @ln_stevilo_stolpcev>0
6      return
7
8  DECLARE @ln_st_identity_zapisov int
```

```
9
10    select * into #tmp_slike
11    from <<this.c_tabela_fs>>
12
13    SET @ln_st_identity_zapisov =
14    (
15      select count(*) from sys.tables t
16      inner join sys.schemas s on s.schema_id=t.schema_id
17      inner join sys.columns c on t.object_id=c.object_id
18      where t.name='<<this.c_tabela>>' and s.name='<<this.c_shema>>' and c
            .is_identity=1
19    )
20
21    delete from <<this.c_tabela_fs>>
22    ALTER TABLE <<this.c_tabela_fs>>
23    ADD <<this.c_polje_v_tabeli>> VARBINARY(MAX) FILESTREAM DEFAULT(0x)
24
25    IF @ln_st_identity_zapisov>0
26      SET IDENTITY_INSERT <<this.c_tabela_fs>> ON
27
28    insert into <<this.c_tabela_fs>>
29    (<<lc_stolpci>>,<<this.c_polje_v_tabeli>>)
30    select *,convert(varbinary(max),<<this.c_staro_ime>>)
31    from #tmp_slike
32
33    UPDATE <<this.c_tabela_fs>>
34    SET <<this.c_staro_ime>>=''
35
36    drop table #tmp_Slike
37
38    IF @ln_st_identity_zapisov>0
39      SET IDENTITY_INSERT <<this.c_tabela_fs>> OFF
```

Po pripravi okolja lahko sprožimo dejanski prenos slike na strežnik. Najprej smo pretvorili datoteko v binarno obliko in jo nato poslali na strežnik SQL. To smo prikazali v programski kodi 3.23.

Programska koda 3.23 : Pretvorba slike v binarno obliko in shranjevanje na strežnik

```
1    c_slika=CREATEBINARY(FILETOSTR(lpc_pot_do_slike))
2
3    TEXT TO pizraz TEXTMERGE PRETEXT 7 NOSHOW
4      UPDATE <<this._sql_tabela>>
5      SET <<this._polje_slike>>= ?c_slika
6      WHERE <<this._polje_kljuca>>=>>this._vrednost_kljuca>>
```

```
7       IF @@ROWCOUNT = 0
8       BEGIN
9         INSERT INTO <<this._sql_tabela>>(<<this._polje_slike>>,<<this.
              _polje_kljuca>>)
10        VALUES (?c_slika,<<this._vrednost_kljuca>>)
11      END
12    ENDTEXT
13
14    IF SQLEXEC(o_connobj.rocaj,pizraz)<1
15      RETURN -1
16    ENDIF
```

Poleg shranjevanja dokumentov smo morali pripraviti tudi njihovo branje. Pred samim procesom skeniranja je namreč potrebno sliko shraniti na datotečni sistem, saj nad njo lahko izvajamo različne operacije (brisanje, dodajanje, vrivanje strani). Za branje smo uporabili programsko kodo prikazano v primeru 3.24. Pri branju smo morali najprej omogočiti zapis binarnih podatkov in jo šele nato shraniti na datotečni sistem.

Programska koda 3.24 : Branje datotek iz strežnika in shranjevanje na disk

```
1   CursorSetProp('MapBinary',.T.,0)
2   IF SQLExec(o_connobj.rocaJ,'select '+this._polje_slike+' as slika from
        '+this._sql_tabela+' WHERE '+this._polje_kljuca+'='+this.
        _vrednost_kljuca,'cur_tmp_slike')<1&&$neprevajaj$
3     RETURN -1
4   ENDIF
5   IF !EOF() AND !ISNULL(slika) AND !EMPTY(slika)
6     STRTOFILE(slika,this.c_tmp_pot_do_slike)
7   ENDIF
```

S tem smo zaključili opis programske kode in izvedbe projekta. V naslednjem poglavju smo prikazali končni rezultat našega dela, vključno z nekaterimi primeri praktične uporabe naše rešitve.

Poglavje 4

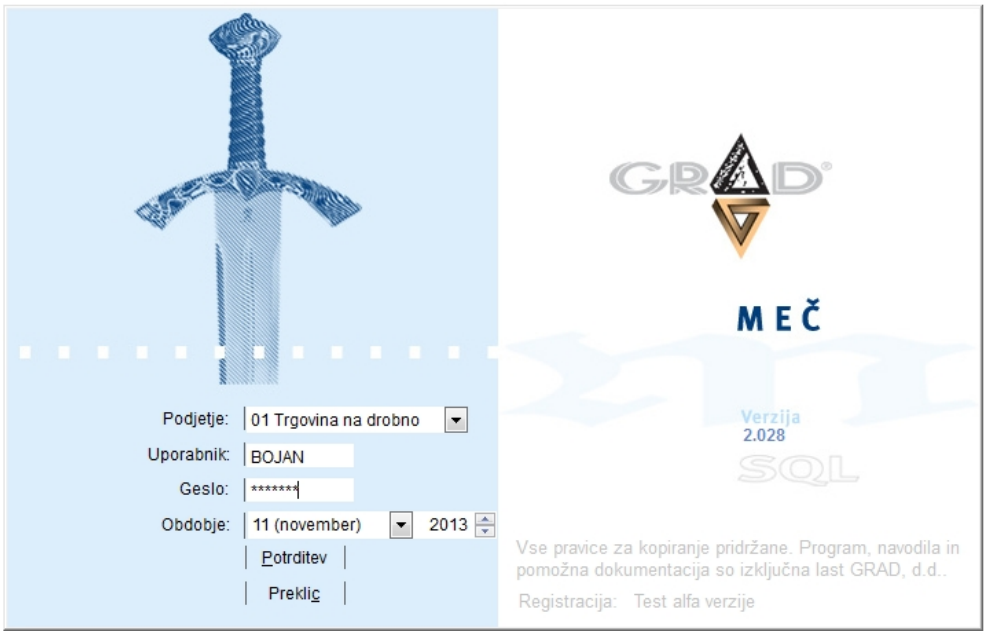
Končna rešitev in primeri uporabe

V tem poglavju smo prikazali izgled uporabniškega vmesnika in nekatere primere uporabe naših rešitev.

4.1 Programi Grad

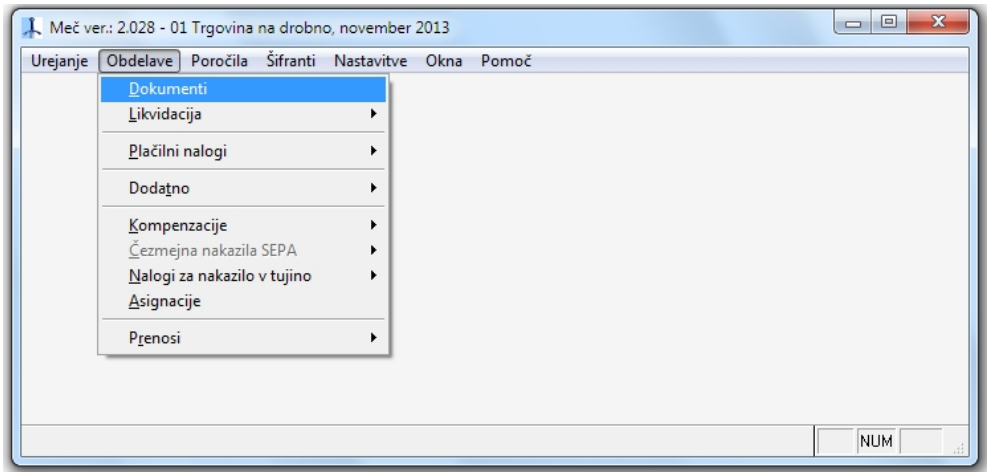
Za začetek smo najprej predstavili osnovni okvir Gradovih programov. V posamezen program vstopimo preko prijavne maske, na kateri izberemo podjetje prijave, obdobje v katerega se prijavljamo, vpišemo uporabniško ime in geslo, ki nam je bilo dodeljeno znotraj sistema Grad in pritisnemo potrditev. Primer prijavne maske Gradove aplikacije smo prikazali na sliki 4.1.

Z uporabniškim imenom in šifro podjetja se programu predstavimo in povemo, katere nastavitve naj uporablja. Nekatere nastavitve se razlikujejo od uporabnika do uporabnika, glede na njegovo zajemno napravo in individualne želje izgleda dokumenta. Druge pa veljajo na nivoju celotnega podjetja, to so tiste, ki določajo imena zajetih dokumentov in njihovo lokacijo. Nastavitve smo podrobneje predstavili v naslednjih poglavjih.



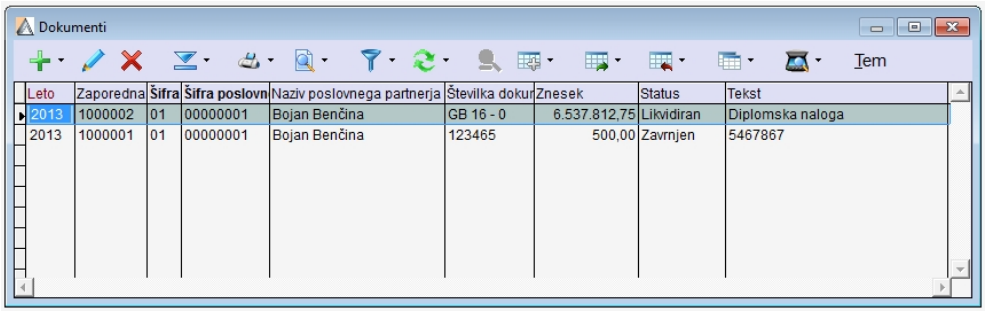
Slika 4.1: Prijavna maska Gradove aplikacije.

Po vstopu v program na zaslon dobimo programski meni, preko katerega kličemo posamezne operacije. Osnovno okno z menijem smo prikazali na sliki 4.2.



Slika 4.2: Osnovno okno Gradove aplikacije.

Za prikaz podatkov programi uporabljajo podatkovne mreže. Vsaka ima svojo orodno vrstico preko katere izvajamo operacije nad podatki. Slika 4.3 prikazuje podatkovno mrežo z orodno vrstico.



Slika 4.3: Podatkovna mreža v Gradovi aplikaciji.

Orodna vrstica podatkovne mreže omogoča naslednje operacije:

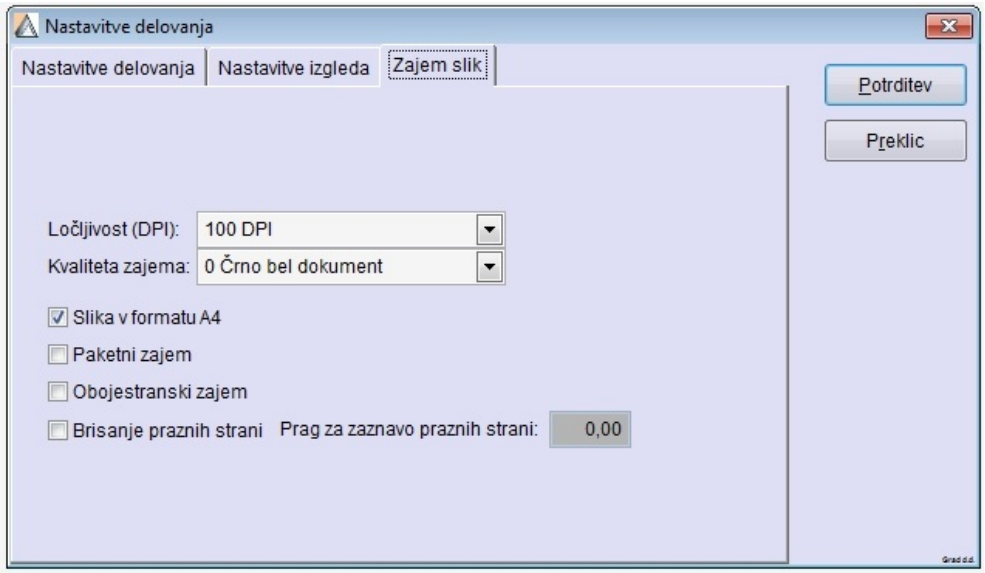
- Vnos, popravljanje, brisanje
- iskanje
- skok na konec oziroma začetek

- tiskanje in ogled
- iskanje in osvežitev podatkov
- prikaz neaktivnih zapisov
- prikaz povezanih podatkov
- ogled in urejanje zajete slike

Gradovi programi omogočajo še veliko drugih operacij in možnosti, ki pa jih za razumevanje te diplomske naloge ni potrebno podrobneje poznati.

4.2 Izgled skupnih delov

Kot zapisano, smo v diplomski nalogi uporabnikom omogočili krmiljenje zajemne naprave, poteka skeniranja in izgleda slike preko nastavitev. Vidimo jih na sliki 4.4.



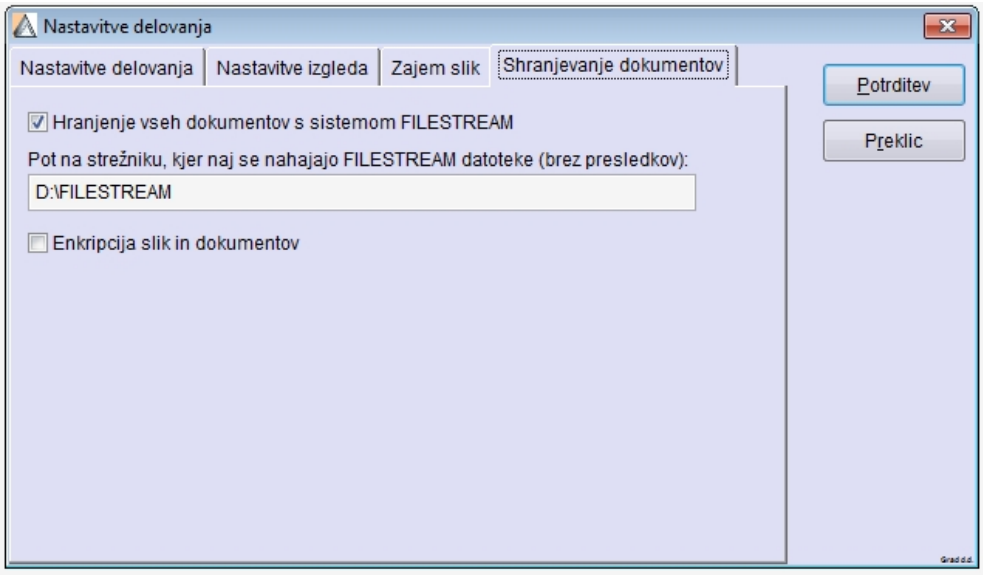
Slika 4.4: Nastavitve zajema slik.

Gre za nastavitve, ki so vezane na uporabnika aplikacije. Shranjujejo se na strežnik SQL in so odvisne od podjetja prijave in uporabniškega imena.

Nastaviti je mogoče:

- ločljivost - na voljo so možnosti 100, 150, 200 in 300 DPI. Višjih ločljivosti nismo podprli, saj bi lahko preveč upočasnile sistem, še vedno pa do njih lahko pridemo preko možnosti naprednega skeniranja
- kvaliteta zajema - gre za barvno globino, podprli smo črno bele, sivinske in barvne dokumente
- slika v formatu A4 - zajemni napravi pove, da skenira dokument velikosti formata A4. Ta možnost je uporabna predvsem pri skenerjih s podajalcem, saj lahko pride do prekrivanja med stranmi in s tem do napačnih dimenzij.
- paketni zajem - če ima naprava podajalec, potem skenira z njega in sicer toliko časa, dokler so v njem dokumenti
- obojestranski zajem
- brisanje praznih strani in toleranca - najprej s pomočjo zajemne naprave poizkusimo odstraniti prazne strani v procesu skeniranja, če naprava tega ne podpira, to storimo programsko in sicer glede na podani prag tolerance

Na tak način ni bilo mogoče dodati nastavitev načina shranjevanja slik. Vsi uporabniki morajo uporabljati enotno in vsem dostopno lokacijo. Tako na sliki 4.5 vidimo nastavitve načina shranjevanja, ki so vezane na podjetje prijave uporabnika.

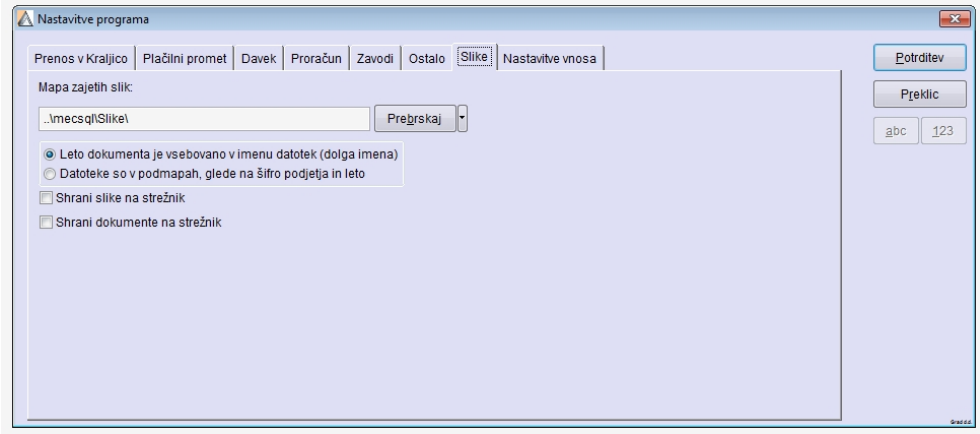


Slika 4.5: Nastavitve načina shranjevanja slik.

Na nivoju podjetja tako določimo:

- način hranjenja dokumentov s sistemom FILESTREAM
- mapo za shranjevanje datotek
- šifriranje dokumentov

Kje ležijo slike posameznih aplikacij, določimo v posebnih nastavitvah vsakega programa. Pri tem velja, da vsaka aplikacija shranjuje dokumente na svojem mestu v datotečnem sistemu oziroma v svoji tabeli na strežniku SQL. To nam omogoča omejevanje dostopov do določenih dokumentov, ki jih lahko vidijo samo izbrani uporabniki. Slaba lastnost takšnega shranjevanja je drobljenje podatkov po sistemu, kar otežuje pisanje kode za dostop iz drugih aplikacij. Prevladala je miselnost, da je pomembneje ugoditi zahtevam uporabnikov kot razvijalcev, zato so dokumenti ostali ločeni in hkrati dostopni iz različnih aplikacij. Na sliki 4.6 vidimo nastavitve lokacij in imen datotek v programu za vodenje prejetih računov.



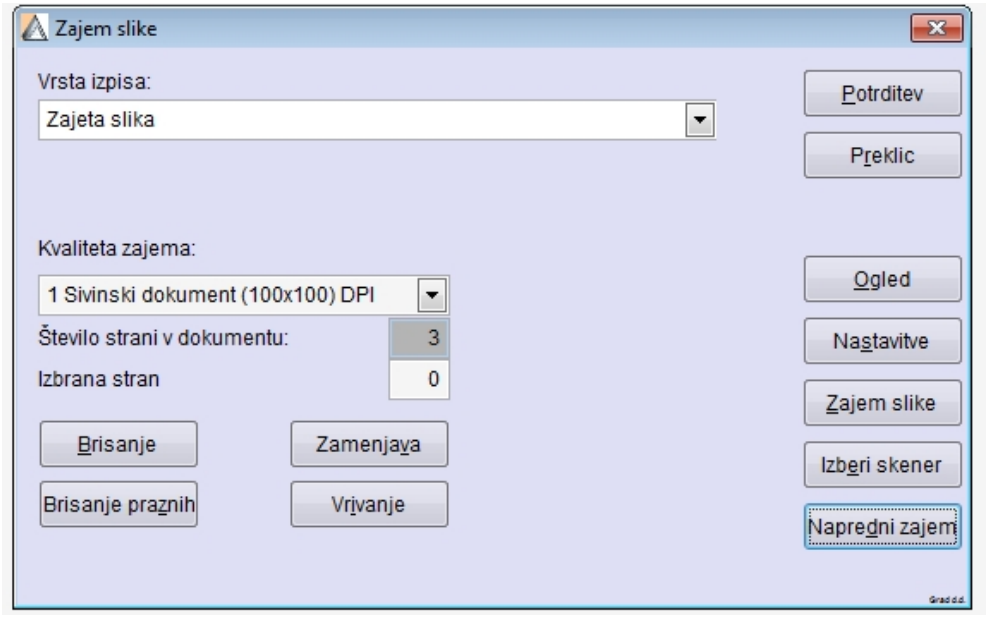
Slika 4.6: Nastavitve načina shranjevanja slik v programu za vodenje prejetih računov.

Za urejanje in zajem dokumentov smo pripravili poseben obrazec, ki je prikazan na sliki 4.7. Tu smo uporabniku dali možnost uporabe nekaterih prednastavljenih kakovosti in barvnih globin, ki smo jih določili na podlagi testiranja. Pri tem nas je vodilo načelo velikosti zajete slike in kakovosti njenega izgleda. Iskali smo prvo berljivo kakovost in pri tem ugotovili, da je bolje nekoliko dvigniti kakovost, saj vsi vhodni dokumenti niso v idealnem stanju. Program si izbrano nastavitev zapomni in jo uporabi v vseh ostalih točkah zajema.

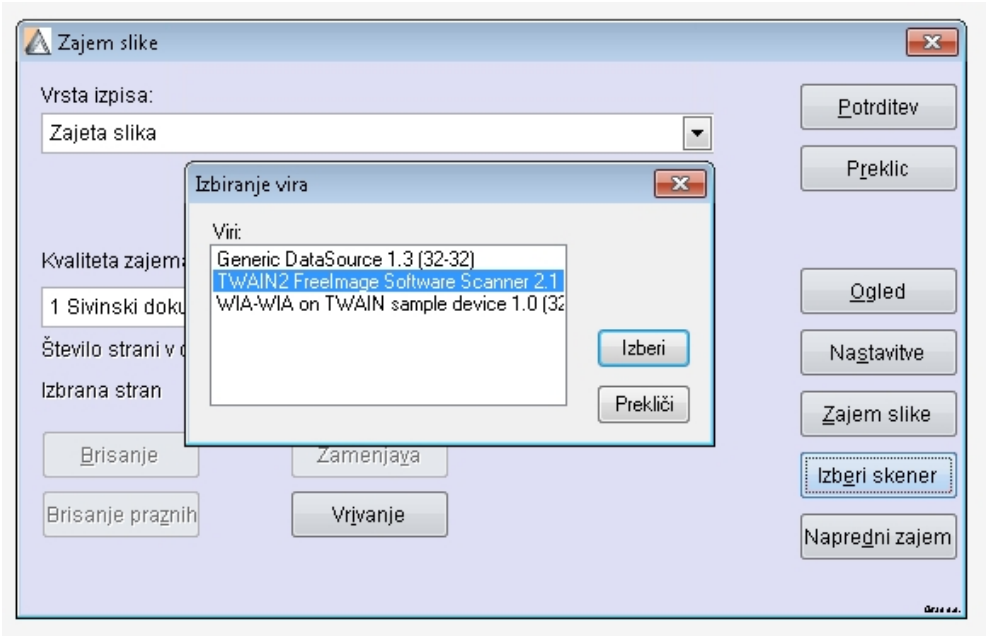
Na sliki 4.7 med drugim vidimo tudi gumb za napredni zajem slike, ki sproži skeniranje preko uporabniškega vmesnika zajemne naprave. Poleg operacij nad zajetim dokumentom smo omogočili še izbiro privzetega skenerja, kar smo prikazali na sliki 4.8 in ogled dokumenta. Prikaz ogleda znotraj Gradove aplikacije vidimo na sliki 4.9.

V orodni vrstici imamo na voljo možnosti sprehoda po straneh, izbor poljubne povečave, tiskanje dokumenta ter izvoz v obliko PDF in pošiljanje po elektronski pošti.

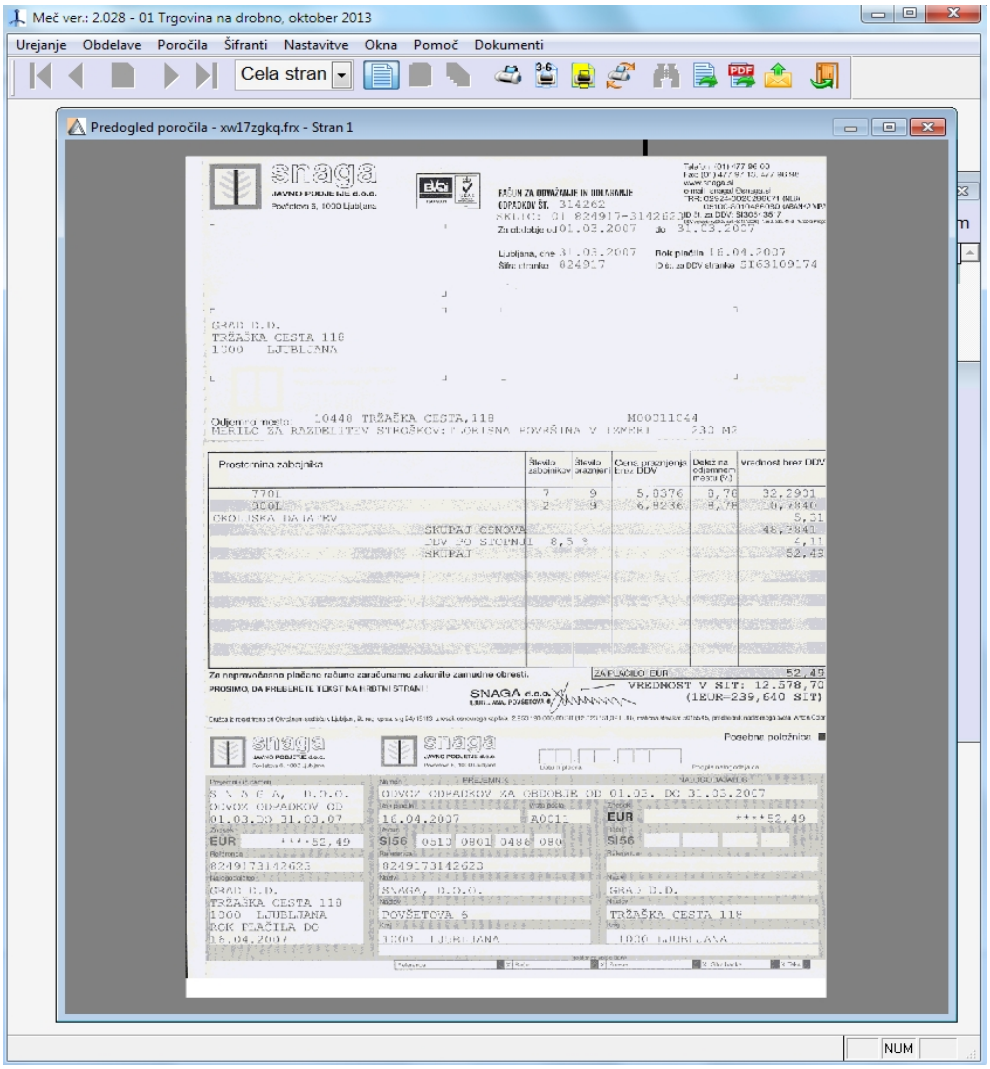
Vse zajete slike dobimo na zaslon preko gumba v orodni vrstici podatkovne mreže. Odpreti pa je možno tudi povezano okno, ki interaktivno pri-



Slika 4.7: Urejanje in zajem slike.

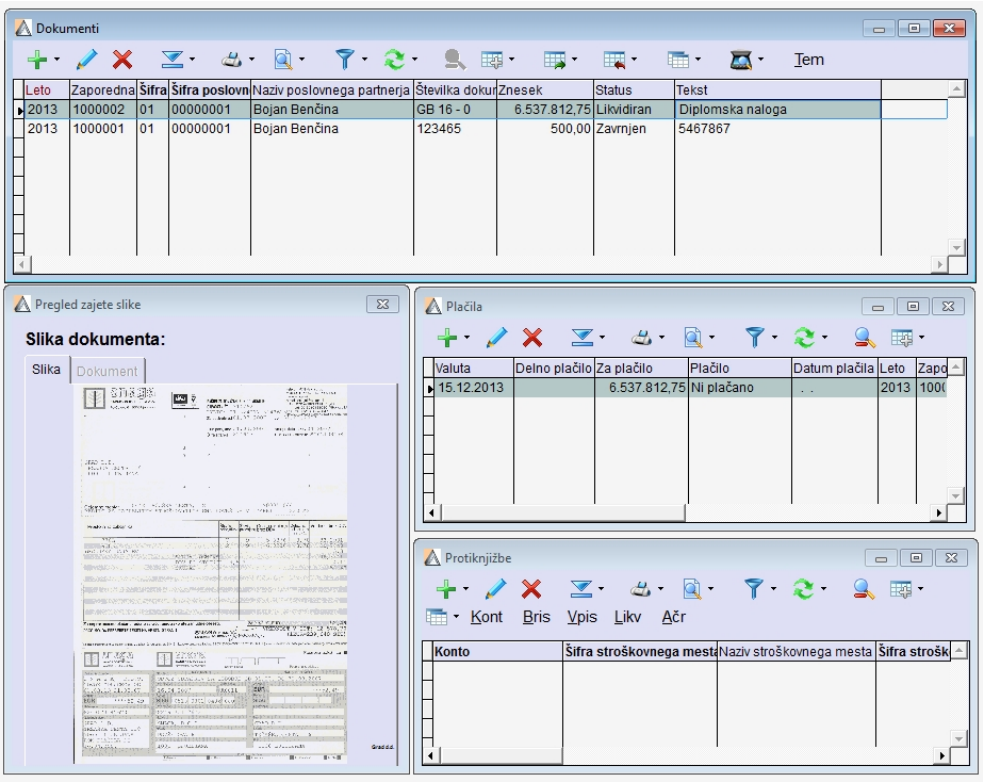


Slika 4.8: Izbor privzete zajemne naprave.



Slika 4.9: Ogled zajetega dokumenta.

kazuje sliko, glede na našo pozicijo v podatkovni mreži. Primer podatkovne mreže z gumbom za ogled zajetega dokumenta in povezanim oknom smo prikazali na sliki 4.10.



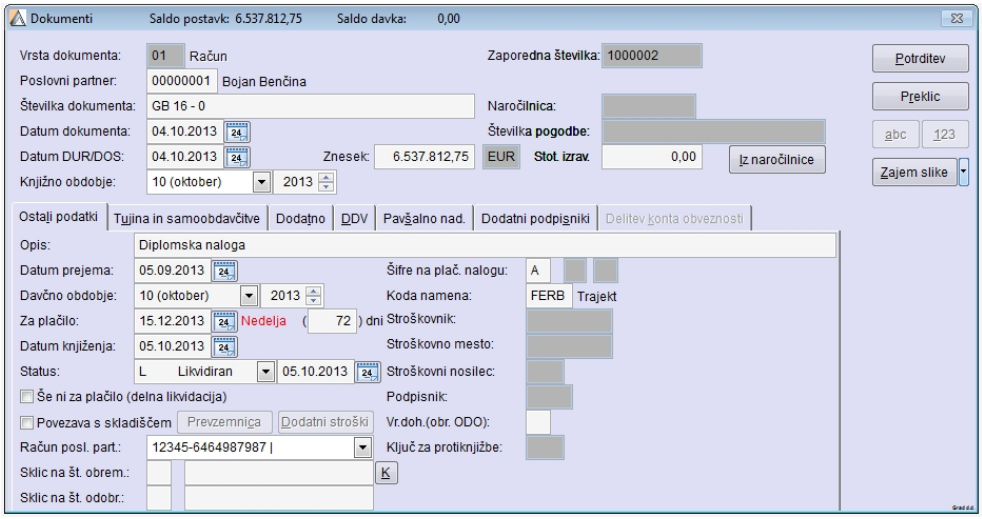
Slika 4.10: Podatkovna mreža in povezano okno s sliko.

Pri uporabi interaktivnega okna smo sistem razvili tako, da se slika prikaže šele po določenem časovnem intervalu. S tem smo povečali odzivnost programa, saj je prikazovanje množice slik počasno in moti običajne delovne procese.

4.3 Primeri uporabe

V praktični uporabi se je potreba po zajemu slike pojavila na kar nekaj mestih v različnih aplikacijah. Najprej je bila implementirana v program za

vodenje prejetih računov. Na sliki 4.11 vidimo masko vnosa prejetega računa in gumba za zajem slike.



Slika 4.11: Vnos prejetega računa z gumbom zajem slike.

Našo rešitev smo dodali še v:

- program za vpis prejete pošte, kjer se zajemajo podatki vseh vhodnih dokumentov
- program za vodenje osnovnih sredstev, kjer skeniramo račune, navodila, garancijske liste in ostale dokumente
- v program za vodenje kadrovske evidence, kjer skeniramo razna potrdila, spričevala in v nekaterih primerih celo slike zaposlenih

Na šifrantih se rešitev uporablja kot zajem poljubnih dokumentov, ki jih uporabnik določi sam, glede na svoje potrebe. To pomeni, da se lahko na istem mestu pri dveh različnih uporabnikih isto polje uporablja v dva različna namena in hrani različne dokumente.

Skeniranje slike se uporablja tudi pri zajemu internih dokumentov kot so logotipi, žigi in podpisi pooblaščenih oseb. Ti se uporabljajo na izpisih, kar

pomeni, da si lahko vsak uporabnik nastavi svojo obliko in izgled določenih poročil.

Ker je celoten sistem med seboj povezan, smo pripravili rešitve, ki omogočajo prikazovanje podatkov v različnih aplikacijah. Tako lahko v programu za vodenje prejetih računov prikažemo sliko pogodbe, ki je račun povzročila.

Poglavje 5

Zaključek

V dobi vse večje uporabe elektronske hrambe dokumentov smo v diplomski nalogi pripravili rešitev, ki uporabnikom aplikacij Grad omogoča preprost in hiter zajem vhodnih dokumentov različnih tipov.

Naredili smo knjižnice in nastavitve, ki omogočajo uporabnikom opravljanje operacij na zajetih dokumentih ter njihov izvoz in pošiljanje po elektronski pošti. Držali smo se načela preprostosti uporabe, kar smo zagotovili z uporabo tehnologij, ki so dostopne širokim množicam uporabnikov, brez nameščanja dodatnih gonilnikov ali aplikacij.

Ker se programi sistema Grad razvijajo naprej smo naše rešitve pripravili po načelu objektnega programiranja, ki razvijalcem aplikacij omogoča enostavno implementacijo naše rešitve v svoje aplikacije. Zato ne potrebujejo dodatnega poznavanja zajemnih naprav, saj svoje potrebe izrazijo preko predvidenih lastnosti na objektih, ki smo jih pripravili v okviru diplomske naloge.

Trudili smo se razviti razširljivo rešitev, predvsem v delu razvoja knjižnice za zajem dokumentov. V ta namen smo ločili programski jezik uporabniškega vmesnika, ki je nekoliko zastarel, od jezika, s katerim smo ustvarili DLL. C# je namreč moderen jezik, ki ima pred seboj svetlo prihodnost, česar ne moremo trditi za programski jezik VFP9. S tem smo dosegli, da bo ob morebitni zamenjavi programskega jezika naša knjižnica še vedno delovala,

na novo bo potrebno napisati le uporabniški vmesnik in zaslonke maske.

Naredili smo preprost uporabniški vmesnik. Uporabnikov nismo preobremenili z velikim številom nastavitev in gumbov. Za lažjo in hitrejšo uporabo smo pripravili nekatere prednastavljene možnosti in menijske bližnjice, katerih namen je bil zmanjšati potrebno število klikov za dosego njihovega cilja.

Za hitrejše izvajanje vnosa podatkov smo pripravili rešitev, ki skeniranje sproži v svojem procesu. S tem smo omogočili uporabniku nemoteno nadaljevanje dela.

Poskrbeli smo, da dokumente lahko hitro najdemo in prikažemo na zaslonu ter s tem pripomogli k optimizaciji poslovanja podjetji.

Za varnost shranjenih dokumentov smo poskrbeli z implementacijo možnosti njihovega šifriranja in ločenih lokacij. Za čim manj zasedenega prostora pa s pravilnim izborom formata stiskanja datotek TIFF ter s pripravo metode, ki poišče prazne strani v dokumentu in jih izbriše.

Pri našem delu smo naleteli na marsikatero težavo. Rešili smo jih s pomočjo svetovnega spleta, kjer smo bolj ali manj hitro našli pravilne odgovore. V nekaterih točkah smo si lahko pomagali z dokumentacijo proizvajalcev. Tu velja omeniti predvsem dobro dokumentiran protokol TWAIN. Ta nam je olajšal marsikatero težavo, ki se je pojavila pri komunikaciji z zajemno napravo. Predvsem so bile težave povezane z naprednim zajemom, ki je potreboval svojo komunikacijsko zanko in s shranjevanjem slik, ki jih je bilo potrebno najprej pretvoriti v pravilno barvno globino.

Na področju uporabniškega vmesnika smo morali biti inovativni predvsem v proženju skeniranja v svojem procesu. Težave nam je povzročal enonitni programski jezik VFP9. Zato smo pripravili svojo rešitev, ki ustvari nov proces in kasneje, v primeru potrditve vnosa podatkov, sliko zapiše na strežnik SQL, skratka opravlja komunikacijo s svojim nadrejenim procesom.

Naše rešitve smo implementirali v različne Gradove aplikacije, kjer že služijo svojemu namenu, to je izboljšana uporabniška izkušnja, zmanjšanje časa potrebnega za iskanja podatkov, omogočanje ogleda povezanih dokumentov na enem mestu in s tem pohitritev ter izboljšanje njihovega poslova-

nja.

Pri pripravljanju diplomske naloge smo spoznali kar nekaj tehnologij, ki jih pred tem nismo poznali ali uporabljali. Tu imamo v mislih način shranjevanja datotek na strežnik SQL s sistemom FILESTREAM. Novost je bila tudi delo v programskem okolju Microsoft Visual Studio 2012, ki ga ocenjujemo kot uspešno, saj smo zastavljene cilje dosegli.

5.1 Izboljšave

Naš pogled je že usmerjen k izboljšanju osnovne rešitve. V prihodnosti načrtujemo dopolnitev programske knjižnice, ki bo omogočalo optično branje čim večjega števila podatkov v času zajema slike. Osnovni cilj pri tem je zmanjšanje števila napak, ki se pojavljajo pri vnosu dokumentov. Naš prvi cilj na tem področju je pripraviti sistem vnosa prejetega računa, ki bo najprej optično prebral dokument in iz njega izluščil podatke o datumih, davkih in zneskih. Nato se bo preko davčne številke povezal s šifrantom poslovnih partnerjev in izbral pravilno šifro za vnos. Podatki, ki se bodo na vnosni ekran prenesli bodo barvno označeni, tako bo uporabnik te vnose samo preveril, ostale, ki jih iz dokumenta v papirni obliki ni bilo mogoče prebrati, pa vnesel ročno.

Z vse večjim razmahom pametnih mobilnih in tabličnih naprav si želimo napredka tudi na tem področju. Nekatere Gradove aplikacije že tečejo na takšnih napravah. Zato bo potrebno pripraviti novo knjižnico, ki bo tekla na drugačnih operacijskih sistemih kot obstoječa. Cilj je mobilna aplikacija, ki zaposlenemu na službenemu potovanju oziroma na oddaljeni lokaciji omogoča slikanje prejetega računa. S tem bo že povzročil vnos, ter sprožil postopek likvidacije. Na tak način bo oseba, ki skrbi za prejete račune v podjetju opravile potrebne preglede, ga dokončno odobrila ali zavrnila ter izvedla plačilo. S takšnim načinom dela bomo še pohitrili in porazdelil zajem podatkov. Velika prednost bo hitrejše izvajanje plačil in s tem takojšen prihranek, saj bomo avtomatsko zmanjšali možnost zamude roka plačila ter

se s tem izognili plačevanju nepotrebnih zamudnih obresti. Vse to bo seveda mogoče le, če bodo naprave omogočale dovolj dobro kakovost zajema dokumentov.

Vsekakor se bomo tudi v prihodnje trudili slediti razvoju novih tehnologij. Pri pripravi programskih rešitev pa bo naše vodilo zadovoljstvo uporabnikov.

Literatura

[1] Visual FoxPro. Dostopno na:
http://msdn.microsoft.com/en-us/library/ms950411.aspx

[2] Visual FoxPro Developer Center. Dostopno na:
http://msdn.microsoft.com/en-US/vfoxpro/

[3] Introduction to the C# Language and the .NET Framework. Dostopno na:
http://msdn.microsoft.com/en-us/library/vstudio/z1zx9t92.aspx

[4] Object-Oriented Programming (C# and Visual Basic). Dostopno na:
http://msdn.microsoft.com/en-us/library/vstudio/dd460654.aspx

[5] Inheritance (C# Programming Guide). Dostopno na:
http://msdn.microsoft.com/en-us/library/ms173149.aspx

[6] Master the fundamentals of C# 3.0 by Jesse Liberty, Brian MacDonald. Dostopno na:
http://msdn.microsoft.com/en-us/library/orm-9780596521066-01-01.aspx

[7] Introduction to WPF. Dostopno na:
http://msdn.microsoft.com/en-us/library/aa970268.aspx

[8] .NET Framework Conceptual Overview. Dostopno na:
http://msdn.microsoft.com/en-us/library/zw4w595w(v=vs.80).aspx

[9] An Overview of Managed/Unmanaged Code Interoperability. Dostopno na:
http://msdn.microsoft.com/en-us/library/ms973872.aspx#manunman_mancode

[10] Extensible Markup Language (XML) 1.0. Dostopno na:
http://www.xml.com/axml/testaxml.htm

[11] Bart A. DePetrillo, "Razumeti Microsoft .NET", Pasadena, 2002.

[12] Introduction to XML. Dostopno na:
http://www.w3schools.com/xml/xml_whatis.asp

[13] Introduction to XML. Dostopno na:
http://www.w3schools.com/xml/xml_dtd.asp

[14] Relational Databases 101: Looking at the Whole Picture. Dostopno na:
http://www.agiledata.org/essays/relationalDatabases.html

[15] FILESTREAM Overview. Dostopno na:
http://technet.microsoft.com/en-us/library/bb933993(v=sql.105).aspx

[16] The Origin of TWAIN. Dostopno na:
http://www.twain.org/images/docs/TWAIN_The_Origin_of_TWAIN.pdf

[17] TWAIN Linking Images With Applications. Dostopno na:
http://www.twain.org/images/docs/TWAIN_Linking_Images_With_Applications.pdf

[18] Windows Image Acquisition (WIA). Dostopno na:
http://msdn.microsoft.com/en-us/library/windows/desktop/ms630368(v=vs.85).aspx

[19] About Windows Image Acquisition. Dostopno na:
http://msdn.microsoft.com/en-us/library/windows/desktop/ms630343(v=vs.85).aspx

[20] What's the similarities/differences between TWAIN and WIA? Dostopno na:
http://kb.dynamsoft.com/questions/739/What%27s+the+similarities%7B47%7Ddifferences+between+TWAIN+and+WIA%3F

[21] TIFF - Tag Image File Format. Dostopno na:
http://www.scantips.com/basics9t.html

[22] How Visual SourceSafe Works. Dostopno na:
http://msdn.microsoft.com/en-US/library/wa77s8c0(v=vs.80).aspx