UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Rok Resman

Program MS Excel kot uporabniški grafični vmesnik

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

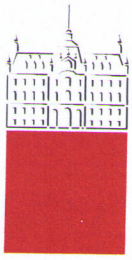
Mentor: doc. dr. Boštjan Slivnik

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

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



Št. naloge: 00516/2010
Datum: 05.04.2010

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

Kandidat: **ROK RESMAN**

Naslov: **PROGRAM MS EXCEL KOT UPORABNIŠKI GRAFIČNI VMESNIK**

MS EXCEL AS A GRAPHICAL USER INTERFACE FOR APPLICATION PROGRAMS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

Tematika naloge:

Raziščite možnosti za uporabo programa MS Excel kot enotnega grafičnega uporabniškega vmesnika, s katerim bi poenotili in poenostavili delo z uporabniškimi programi za računalnika manj vešče uporabnike. Natančno opišite možnosti za obojestransko povezavo uporabniškega programa in programa MS Excel. Napišite primer uporabniškega programa, ki uporablja program MS Excel kot uporabniški grafični vmesnik za dostop do podatkovnih baz shranjenih v sistemu MySQL.

Mentor:
B. Slivnik
doc. dr. Boštjan Slivnik

Dekan:
Franc Solina
prof. dr. Franc Solina

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

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

Št. naloge: 00516/2010
Datum: 05.04.2010

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

Kandidat: **ROK RESMAN**

Naslov: **PROGRAM MS EXCEL KOT UPORABNIŠKI GRAFIČNI VMESNIK**

MS EXCEL AS A GRAPHICAL USER INTERFACE FOR APPLICATION PROGRAMS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

Tematika naloge:

Raziščite možnosti za uporabo programa MS Excel kot enotnega grafičnega uporabniškega vmesnika, s katerim bi poenotili in poenostavili delo z uporabniškimi programi za računalnika manj vešče uporabnike. Natančno opišite možnosti za obojestransko povezavo uporabniškega programa in programa MS Excel. Napišite primer uporabniškega programa, ki uporablja program MS Excel kot uporabniški grafični vmesnik za dostop do podatkovnih baz shranjenih v sistemu MySQL.

Mentor:
B. Slivnik
doc. dr. Boštjan Slivnik

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
Franc Solina
prof. dr. Franc Solina

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a _____Rok Resman_____ ,

z vpisno številko _____63040145_____ ,

sem avtor/-ica diplomskega dela s naslovom:

Program MS Excel kot uporabniški grafični vmesnik_____

_____

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom(naziv, ime in priimek) doc. dr. Boštjan Slivnik_____
  in somentorstvom (naziv, ime in priimek) _____/_____
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov.,angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

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

Zahvala

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

Prav tako gre zahvala moji družini, punci in vsem tistim, ki so mi stali ob strani v času študija.

Kazalo

Povzetek

Pojav grafičnih uporabniških vmesnikov je zelo razširil krog uporabnikov računalnikov in drugih elektronskih naprav. Zaradi širjenja kroga uporabnikov računalnikov in vse večje vpletenosti računalnikov v vsakdanje življenje so računalnike začeli uporabljati tudi računalnika manj vešči uporabniki. Ker tem uporabnikom učenje novih programov pomeni dodaten stres, se je porodila ideja o uporabi enotnega grafičnega uporabniškega vmesnika.

Osrednja tema je uporaba aplikacije Microsoft Excel kot grafični uporabniški vmesnik programa in implementacija le tega v preprosto aplikacijo. Predstavljena je komunikacijska pot med predstavitveno aplikacijo in aplikacijo Microsoft Excel na osnovi OLE Automation, ki omogoča aplikacijam, da uporabljajo program Microsoft Excel kot en skupen grafični uporabniški vmesnik. Hkrati so opisane tudi ostale možnosti komuniciranja med različnimi aplikacijami.

Ključne besede:

Grafični uporabniški vmesnik, Microsoft Excel, OLE Automation, razvojno okolje Delphi

Abstract

The phenomenon of Graphical User Interface expanded the circle of users of personal computers and other electronic devices. Therefore, and also because of the role computers play in every day life nowadays, more and more less skillful users use computers on a daily basis. Process of learning how to use new programs is additional stress for them, so a new idea of using unified graphical user interface came to life.

A communication based on OLE Automation between Microsoft Excel application and other applications is described. It is demonstrated how this communication can be used so that all applications could use Microsoft Excel as a unified graphical user interface. Other types of communications between different applications are described as well.

Key words:

Graphical user interface, Microsoft Excel, OLE Automation, Delphi IDE

1. Uvod

Pojav prvih operacijskih sistemov v 80-letih, ki so bili namenjeni namiznim računalnikom je pomenil začetek hitrega razvoja računalništva po vsem svetu. Pred tem so z računalnikom upravljali predvsem znanstveniki, poleg njih pa tudi peščica najbogatejših ljudi. Še hitreje se je računalništvo začelo razvijati po prihodu operacijskega sistema Windows 95, ki je nekako začrtal pot razvoja grafičnih uporabniških vmesnikov (v nadaljevanju GUI vmesnik iz angleške besede Graphical User Interface) programov. Od tedaj so si GUI vmesniki pri vseh programih ne glede na operacijski sistem dokaj podobni. Ti novi operacijski sistemi in programi so poenostavili delo z računalniki in s tem poskrbeli, da ima v današnjem času že skoraj vsaka družina v razvitem delu sveta vsaj en namizni računalnik, poleg tega pa še kak prenosni računalnik.

Z razširitvijo kroga uporabnikov računalnika se je pojavili nov tip uporabnika. Če so bili znanstveniki vešči uporabniki, so sedaj prišli računalnika nevešči uporabniki. Vešči uporabniki si sami nameščajo operacijske sisteme, različne programe, vedo kaj je varna uporaba interneta, kaj so sestavni deli računalnika in znajo odpraviti večji del tako napak na programski opremi (angl. software) kot napak na strojni opremi (angl. hardware) računalnika. Na drugi strani so manj vešči uporabniki, za katere je sestavni del računalnika zaslon, tipkovnica in miška. Tak uporabnik ne loči med Windowsi 98 in Windowsi XP, če bodo le funkcije, katere uporablja, tam kjer naj bi bile.

Taki uporabniki seveda ne menjajo programa, ki so ga vajeni, saj bi jim učenje uporabe novega programa vzelo preveč časa ali pa jih računalništvo niti ne zanima, poleg tega pa bi bilo to zanj tudi stresno. Že sama nadgradnja programa na novejšo različico je včasih težavna, ko se spremeni vizualna podoba programa, kot se je npr. spremenila pri prehodu iz pisarniškega paketa Microsoft Office 2000 na Microsoft Office 2007, pa čeprav se je sama uporaba programa poenostavila. Tudi večina vladnih in nevladnih organizacij redko nadgradi svojo programsko opremo na najnovejšo različico. Tipičen primer je ohranjanje operacijskega sistema Microsoft Windows in pisarniškega paketa Microsoft Office na več let stari verziji, kaj šele da bi celotni paket aplikacij nadomestili z alternativnimi odprtokodnimi rešitvami. V primeru kombinacije Microsoft Windows in Microsoft Office imamo povsem nadomestljivo in tudi po uporabnosti precej podobno alternativo v kombinaciji operacijskega sistem Linux Ubuntu, ter pisarniškega paketa OpenOffice.

Prav zaradi manj veščih uporabnikov smo prišli do ideje, da bi za programe s podobno funkcionalnostjo lahko uporabljali enoten GUI vmesnik, za katerim pa bi se skrivali različni algoritmi. Seveda je uporaba enotnega GUI vmesnika pri vseh tipih funkcionalnosti vprašljiva. Težko bi napisali tak GUI vmesnik, ki bi na primer zadoščal potrebam različnim programom za 3D modeliranje. Lahko pa bi tak enoten GUI vmesnik uporabili recimo za programe, ki urejajo podatke.

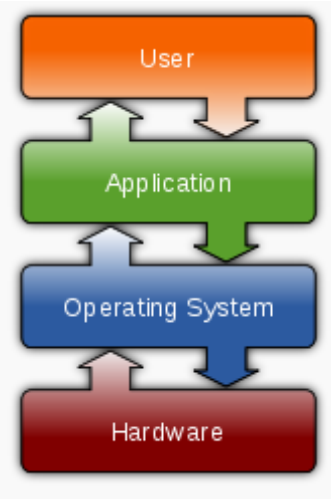
V okviru diplomske nalogo smo si zadali cilj, da pripravimo čisto enostaven program, ki bo uporabil Microsoft Excel kot GUI vmesnik našega programa. Program mora biti zmožen komunicirati z uporabnikom ter shraniti vnešene podatke v podatkovno bazo MySQL. Poleg tega, da omogočimo branje vtipkanih podatkov, moramo med delovanjem programa uporabniku servirati svoje podatke.

Za komunikacijo z Microsoft Excel programom bomo uporabili OLE Automation, za povezavo s podatkovno bazo MySQL pa DLL knjižnico. Vse skupaj bo napisano večnitno v programskem jeziku Borland Delphi 7.

Predenj smo se lotili pisanja naše aplikacije je bilo potrebno preučiti osnove GUI vmesnikov ter možne alternative povezovanja med aplikacijami. GUI vmesnike bomo spoznali v poglavju 2, vrste in načine povezovanja med programi pa v poglavju 3.

2. Grafični uporabniški vmesniki

Grafični uporabniški vmesnik ali GUI (iz angl. Graphical User Interface) je vmesnik za komuniciranje med elektronsko napravo in uporabnikom. Ta elektronska naprava je lahko računalnik, MP3 predvajalnik, GSM telefon ali celo pečica. Za komuniciranje z uporabnikom se namesto teksta uporabljajo slike in podobe(1).



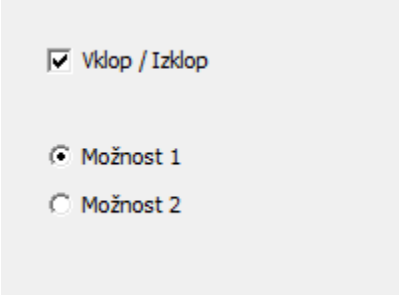
Slika 1: Prikaz nivojev pri delu z računalnikom

2.1. Ikone, komponente in njihovo razumevanje

Ključnega pomena pri razvijanju GUI vmesnika je izbira čim bolj preprostih in intuitivnih ikon, ki uporabniku takoj povedo kaj predstavljajo. Podobo znaka X lahko povežemo s tem da se bo nekaj zaprlo ali pa da je nekaj označeno, puščico pa lahko povežemo s tem da se bo nekaj premaknilo v smer kamor kaže puščica ali pa da bomo odšli na novo stran. Seveda je povezava z določeno akcijo, ki se izvede na računalniku odvisna od situacije. Če se pri tem omejimo na X, je recimo le ta v kvadratku v zgornjem desnem kotu programa povezan z zaprtjem le tega. To je enako ne glede na operacijski sistem, ki ga uporabljamo (slika 2).

Slika 2: Zapiranje programa

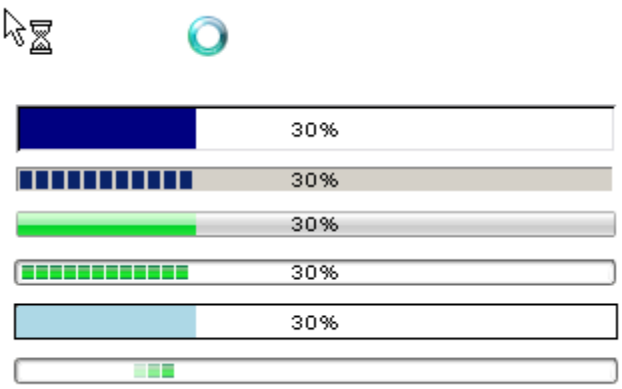
Seveda pa se lahko X uporabi tudi za druge akcije, kot je recimo izbira. X za izbiro večkrat zamenja tudi kljukica, medtem ko okrogli izbirniki povedo, da izbiramo med več možnostmi (slika 3).



Slika 3: Izbiranje

Prav tako, kot je poenotena uporaba različnih komponent, so si v različnih operacijskih sistemih zelo podobne tudi ikone standardnih funkcij. Podoba koša pove, da s to akcijo želimo nekaj izbrisati, podoba zemlje najverjetneje pomeni internet, podoba mape pa pove da gre za imenik na disku, itn. Čeprav so si te podobe nekoliko različne, pa jih uporabnik kaj kmalu poveže z akcijo, ki jim pripada.

Poleg ikon uporabniki takoj prepoznajo razna sporočila o delovanju sistema oziroma programa. Simbol peščene ure na miškinem kazalcu, vrteča ura ali pa lestvica napredka, nam sporočajo da se nekaj trenutno izvaja in naj počakamo (slika 4).



Slika 4: Program sporoča da je zaseden

2.2. Navigacija po sistemu

Po operacijskem sistemu oziroma njegovih aplikacijah se lahko sprehajamo z miško ali tipkovnico. Vsak uporabnik ve, da bo premik miške v levo povzročil premik miškinega kazalca v levo ter da bo pritisk na desni gumb prikazal morebiten spustni meni, itn. Vendar pa bo marsikateri, računalnika manj vešči uporabnik, imel težavo pri enojnem in dvojnem kliku. Največ težav se pojavlja pri uporabi internetnega brskalnika, ko na straneh na povezavo dvokliknejo, čeprav je za sledenje tej potreben le en klik.

Poleg uporabe miške, kot navigacijskega sredstva, je tudi uporaba tipkovnice enaka na vseh operacijskih sistemih. Vemo, da nas pritisk na tipko TAB premakne na naslednjo komponento, pritisk na ENTER nam potrdi izbiro, pritisk na tipko F1 nam nudi pomoč, itn.

2.3. Sestavni deli programa

Skoraj vse novejše aplikacije imajo nekaj komponent, ki se pojavljajo ne glede na vrsto funkcije, ki jo ta aplikacija opravlja. Seveda te komponente niso v vseh aplikacijah na istem koncu.

2.3.1. Glavna vrstica programa

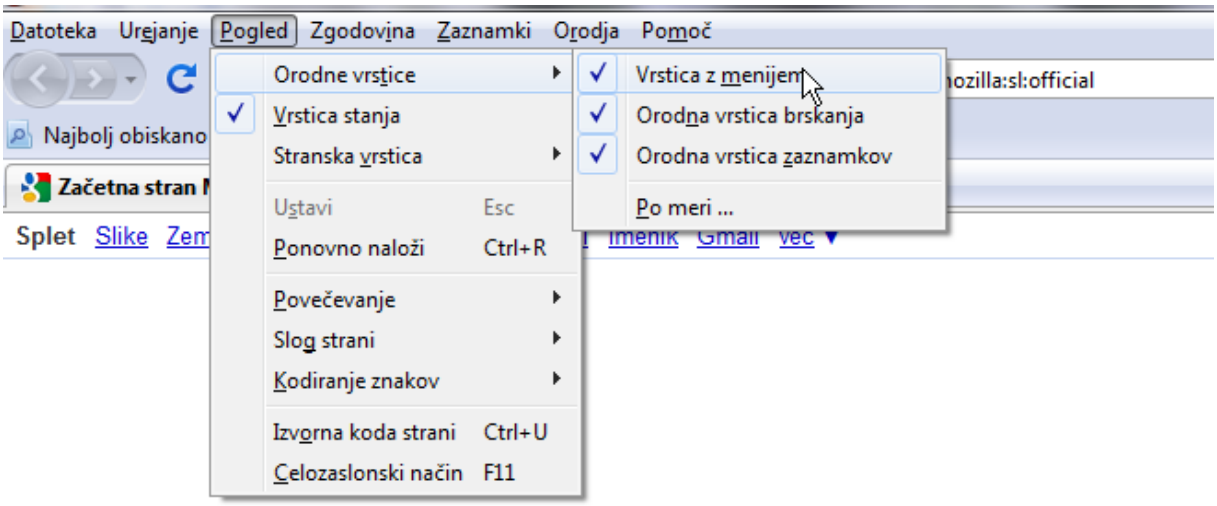
V glavni vrstici aplikacije imamo vedno napisano ime aplikacije, ki je odprta, poleg tega pa skoraj vedno izpisano tudi ime trenutno odprtega projekta oziroma dokumenta. V primeru brskalnika je to lahko odprta internetna stran. V tej vrstici najdemo tudi tri standardne gumbe - za minimizacijo aplikacije, spremembo velikosti aplikacije ter zapiranje aplikacije (slika 5).

Slika 5: Glavna vrstica programa

2.3.2. Vrstica z menijem

Vrstica z menijem je še en tipična komponenta. Preko nje so po navadi dostopne skoraj vse funkcije, ki jih aplikacija omogoča (slika 6). Razvrščene so po sklopih, med katerimi najdemo tudi nekaj standardnih:

- Datoteka
- Urejanje
- Pogled
- Orodja
- Pomoč



Slika 6: Primer vrstice z menijem

2.3.3. Orodna vrstica

Orodna vrstica je vrstica, preko katere imamo izpostavljene funkcionalnosti aplikacije, ki jih večkrat uporabljamo (slika 7). Izpostavljenost teh funkcionalnosti nam olajšuje delo z aplikacijo in omogoča hitrejše delo.

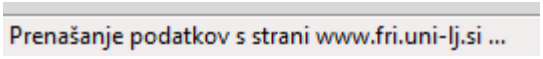
Slika 7: Orodna vrstica

2.3.4. Področje dela

Področje dela se najbolj razlikuje med posameznimi aplikacijami. V tem delu najdemo podatke s katerimi upravljamo, najsi bo to slika ki jo urejamo, list papirja na katerega pišemo ali pa internetna stran po kateri brskamo.

2.3.5. Statusna vrstica

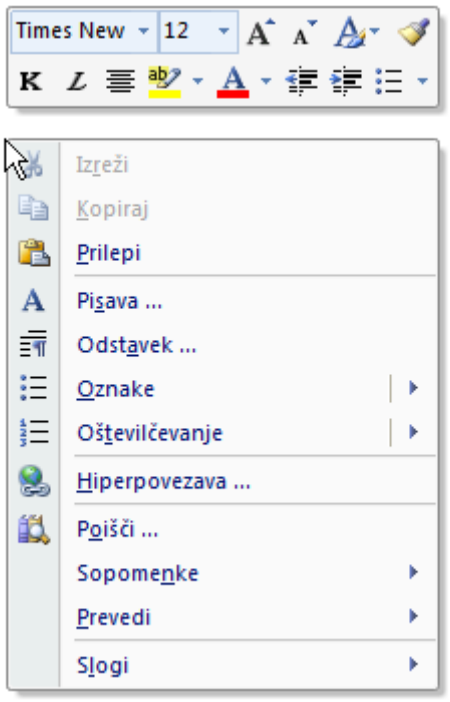
V statusni vrstici so izpisani podatki o trenutnem delovanju aplikacije in ostale informacije, za katere je razvijalec menil, da bodo uporabnika utegnile zanimati. Tipično je v primeru urejevalnika besedila v statusni vrstici izpisana trenutna stran in skupno število strani, v primeru internetnega brskalnika pa je izpisano trenutno napredovanje odpiranja internetne strani (slika 8).



Slika 8: Primer informacije v statusni vrstici

2.3.6. Pojavni meni

Pojavni meni je meni, ki se nam pokaže ob desnem kliku v področju dela (slika 9). V tem meniju so tipično le funkcionalnosti, ki so nam omogočene glede na lokacijo, kjer smo uporabili desni klik in tudi glede na izbrane elemente (označena vrstica v drevesu, označena beseda v tekstu).



Slika 9: Primer pojavnega menija v programu Microsoft Word

3. Povezovanje med različnimi aplikacijami

Vedno bolj se v računalništvu uveljavlja povezovanje med različnimi aplikacijami zaradi hitrejšega in bolj učinkovitega dela. Povezovanje je lahko znotraj operacijskega sistema (na enem računalniku) ali med posameznimi računalniki. Za povezovanje med različnimi aplikacijami poznamo več različnih načinov. V primeru, da se aplikaciji povezujeta iz različnih računalnikov, je potrebno vedeti preko katerih vrat se bosta med seboj pogovarjali. Če gre za povezovanje znotraj enega računalnika, pa se lahko aplikaciji povezujeta tudi neposredno, recimo preko DLL knjižnic.

3.1. Znani načini povezovanja med aplikacijami

Kot smo že omenili, nam je na voljo ogromno število različnih načinov povezovanja, tako so med najbolj znanimi povezovalnimi metodami uporaba mrežnih vtičnikov, SOAP protokola, uporaba DLL knjižnic, itd. Povezovanje med aplikacijami z uporabo teh tehnologij nas ne omejuje na en posamezni programski jezik, kar je tudi osnovna ideja za uporabo teh komunikacijskih poti.

3.1.1. Mrežni vtičnik (angl. Network Socket)

Mrežni vtičnik je vmesnik med različnimi aplikacijami, ki predstavlja mehanizem za zagotavljanje dostavljanja in oddajanja paketov preko TCP/IP protokola. (2) Aplikaciji, ki se med seboj povežeta preko mrežnega vtičnika, se povežeta na točno določen IP naslov in vrata. Namesto zunanjega IP naslova se lahko povežemo tudi lokalno.

Preden si aplikaciji začneta izmenjevati podatke se med seboj prepoznata in vzpostavita stalno povezavo. Mrežni vtičniki omogočajo tudi varno povezavo (angl. secure connection). Za še bolj varni prenos podatkov lahko uporabimo tudi elektronske certifikate.

Ko je povezava vzpostavljena, lahko aplikaciji komunicirata dvosmerno. Možnih načinov mrežnih vtičnikov je več, med bolj uporabljenimi pa sta TCP in UDP.

Primer povezave dveh aplikacij preko mrežnih vtičnikov si bomo ogledali na primeru klienta in strežnika. Aplikacija je napisana v JAVI in smo jo dobili na internetnem naslovu http://www.kodejava.org/examples/216.html. Kot bomo videli iz primera je za enostavno izmenjavo nekaj besed med klientom in strežnikom potrebno napisati kar 3 razrede in skoraj 100 vrstic kode. Še bolj kompleksno vse skupaj postane ko želimo vzpostaviti stalno povezavo med klientom in strežnikom.

3.1.1.1. Strežnik

V našem primeru je strežnik sestavljen iz dveh razredov. Prvi razred je ServerSocketExample (slika 10).

```
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.lang.ClassNotFoundException;
import java.lang.Runnable;
import java.lang.Thread;
import java.net.ServerSocket;
import java.net.Socket;

public class ServerSocketExample {
    private ServerSocket server;
    private int port = 7777;

    public ServerSocketExample() {
        try {
            server = new ServerSocket(port);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        ServerSocketExample example = new ServerSocketExample();
        example.handleConnection();
    }

    public void handleConnection() {
        System.out.println("Waiting for client message...");

        //
        // The server do a loop here to accept all connection initiated by the
        // client application.
        //
        while (true) {
            try {
                Socket socket = server.accept();
                new ConnectionHandler(socket);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Slika 10: Razred ServerSocketExample

Razred ServerSocketExample v svojem konstruktorju kreira objekt tipa ServerSocket, ki posluša na vratih 7777. Ko je objekt kreiran se pokliče funkcija handleConnection, ki čaka na novo povezavo, v našem primeru klienta. Ko dobimo novega klienta kreiramo nov razred tipa ConnectionHandler, ki je svoja nit in dejansko izvrši kratek pogovor s klientom (slika 11).

```
class ConnectionHandler implements Runnable {
    private Socket socket;

    public ConnectionHandler(Socket socket) {
        this.socket = socket;

        Thread t = new Thread(this);
        t.start();
    }

    public void run() {
        try
        {
            //
            // Read a message sent by client application
            //
            ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());
            String message = (String) ois.readObject();
            System.out.println("Message Received: " + message);

            //
            // Send a response information to the client application
            //
            ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
            oos.writeObject("Hi...");

            ois.close();
            oos.close();
            socket.close();

            System.out.println("Waiting for client message...");
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Slika 11: Razred ConnectionHandler

3.1.1.2. Klient

Klient je predstavljen kot razred ClientSocketExample, ki kreira objekt tipa Socket (slika 12). Le ta se poveže na lokalni naslov na vrata 7777 in ko je povezava vzpostavljena pošlje sporočilo. Po poslanem sporočilu čaka na sporočilo, ki ga bo vrnil strežnik. Vse skupaj se konča z zaprtjem povezave do strežnika.

```java
package socket.simple;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.lang.ClassNotFoundException;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;

public class ClientSocketExample {
    public static void main(String[] args) {
        try {
            //
            // Create a connection to the server socket on the server application
            //
            InetAddress host = InetAddress.getLocalHost();
            Socket socket = new Socket(host.getHostName(), 7777);

            //
            // Send a message to the client application
            //
            ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
            oos.writeObject("Hello There...");

            //
            // Read and display the response message sent by server application
            //
            ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());
            String message = (String) ois.readObject();
            System.out.println("Message: " + message);

            ois.close();
            oos.close();
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Slika 12: Razred ClientSocketExample

3.1.2. SOAP protokol

SOAP ali Simple Object Access Protocol je protokol za izmenjevanje strukturiranih podatkov pri izvajanju spletnih storitev (angl. Web Service). Kot format sporočanja med klientom in strežnikom uporablja XML (angl. eXtensible Markup Language), prenos podatkov pa skoraj po pravilu poteka preko protokola HTTP (angl. HiperText Transfer Protocol). Možen pa je prenos podatkov tudi preko drugih protokolov. (3)

Kot prednosti protokola SOAP lahko omenimo, da je dovolj robusten, da omogoča prenos preko različnih protokolov. Še bolj pomembno je, da protokol SOAP lahko komunicira preko protokola HTTP in HTTPS. Ker komunikacija preko protokolov HTTP in HTTPS poteka

preko vrat 80 oz. 443, ki jih požarni zidovi ne preverjajo nimamo bojazni, da poslani paketi nebi prišli na svoj cilj.

Protokol SOAP je zelo učinkovit pri prenašanju relativno majhnih sporočil. Z velikostjo podatkov pa se sama komunikacija precej upočasni, ker je procesiranje XML datotek zelo potratno glede procesorske moči ter porabe delovnega pomnilnika (angl. RAM). Slaba stran uporabe protokola SOAP je tudi to, da omogoča samo podatke na zahtevo (angl. PULLING). Čeprav protokol nima dobre podpore v vseh programskih jezikih, je dobro podprt s strani večine najbolj znanih npr. Java, Delphi, PHP, .NET.

Kot primer uporabe SOAP protokola si bomo pogledali enostaven primer PHP aplikacije, ki se poveže na WebService, pokliče eno od njenih funkcij in na zaslon izpiše rezultat.

3.1.2.1. Strežnik

Strežnik, je enostavna PHP datoteka v kateri je potrebno kreirati objekt tipa SoapServer (slika 13). Temu objektu nato dodamo funkcijo, ki jo bo imel strežnik izpostavljeno in pokličemo. Pri PHP SOAP protokolu ni potrebno strežnika predhodno pognati, saj se zažene na zahtevno klienta.

```php
<?php
function HelloWorld(){
  return "HelloWorld";
}

$server = new SoapServer(null, array('uri' => "http://localhost/server.php",
                                     'encoding' => "UTF-8"));
$server->addFunction("HelloWorld");
$server->handle();


?>
```

Slika 13: SOAP strežnik

3.1.2.2. Klient

Za SOAP klienta potrebujemo še manj vrstic kode kot za strežnik. Vse kar moramo storiti je, da kreiramo nov objekt tipa ter pokličemo želeno funkcijo (slika 14).

```php
<?php
$client = new SoapClient(null, array('trace' => 1,
                                     'location'=> 'http://localhost/server.php',
                                     'uri' => 'http://localhost/server.php'));
echo $client->HelloWorld();
?>
```

Slika 14: Klient

3.1.3. DLL

DLL (angl. Dynamic-link library) je Microsoftova implementacija koncepta skupne rabe knjižnic v operacijskih sistemih Microsoft Windows in OS2. Po navadi imajo te knjižnice končnico DLL, lahko pa tudi OCX. OCX se uporablja za knjižnice, ki vsebujejo kontrolnike ActiveX. Uporabljajo pa enak datotečni format kot EXE datoteke in lahko vsebujejo tako kodo kot tudi podatke. (4)

Uporaba DLL kot knjižnico je zelo pogosta saj pomaga programom lažje komunicirati z drugimi programi in storitvami, tako da opisuje njihove zunanje funkcije.

Primer: Za komuniciranje z MySQL podatkovno bazo potrebujemo mysqllib.dll.

3.1.4. OLE Automation

Microsoft je za svoj operacijski sistem Windows razvil mehanizem za komunikacijo med procesi, ki temelji na modelu COM (angl. Component Object Model). Sprva je bil razvit predvsem za skriptne programske jezike, kot je recimo Visual Basic, sedaj pa se uporablja pri vseh programskih jezikih namenjenim razvijanju aplikacij za operacijski sistem Microsoft Windows.(5) Model COM razširja DLL v objektno orientirano programiranje, pri čemer omogoča dostopanje do objektov drugih procesov ali do objektov procesov na drugih napravah.

OLE Automation omogoča infrastrukturo, kjer klient, imenovan »automation controller«, lahko dostopa in manipulira z lastnostmi oziroma metodami druge aplikacije. Na drugi strani imamo aplikacijo, ki omogoča dostop zunanjemu svetu do svojih metod in lastnosti in je neke vrste strežnik.

Kot smo rekli je bil COM sprva namenjen skriptnim jezikom, zato nam v času programiranja prevajalnik ne zazna sintaktičnih napak zaradi napačnega imena metode, kot tudi ne zaradi napačnega števila parametrov oziroma njihovega tipa. Na te sintaktične napake naletimo šele med samim izvajanjem aplikacije. Ta način programiranja je poznan pod imenom pozno povezovanje (angl. late binding).

Uporaba OLE Automation v načinu poznega povezovanja je le ena od možnosti. Drugi način je uporaba zgodnjega povezovanja oz. povezovanje v času prevajanja (angl. compile-time early binding). Oba načina imata svoje prednosti in slabosti. Prednost uporabe poznega povezovanja je neobčutljivost na verzijo knjižnice. Na drugi strani je prednost zgodnjega povezovanja hitrejše delovanja, saj se pravilnost funkcij in njihovih parametrov preverja med prevajanjem.

Ker pa je programiranje v načinu poznega povezovanja precej oteženo, če že ne celo nemogoče, si programerji v večini primerov izberejo način zgodnjega povezovanja. To naredijo tako, da v aplikacijo dodajo tako imenovane tipske knjižnice (angl. type library), ki nam posredujejo meta podatke o razredih, vmesnikih in drugih lastnostih, ki so nam posredovane s strani objektov.

Ker je OLE Automation omogočena v prav vseh aplikacijah napisanih s strani Microsofta in tudi nekaterih drugih aplikacijah, ki niso njihov produkt, je podprta tudi s strani skoraj vseh programskih jezikov namenjenim Windows okolju. Med njimi lahko najdemo:

- C
- C++
- Visaul Basic
- Delphi
- Microsoft .NET
- APL
- Java
- JScript in VBScript
- Perl
- PHP
- Python
- Ruby

3.1.5. MS Windows Messages

Za razliko od aplikacij pisanih za MS-DOS, so aplikacije pisane za operacijski sistem vodene s pomočjo dogodkov (angl. event driven), zato čakajo, da jim sistem pošlje vhodne podatke. Operacijski sistem sporoča vhodne podatke vsem oknom v aplikacijah, zato ima vsako okno funkcijo imenovano »window procedure«, ki jo operacijski sistem pokliče kadarkoli ima kakšno sporočilo za to okno. (6)

Vhodni podatki so sporočeni v obliki sporočil (angl. message). Sporočila lahko generira tako sistem kot tudi aplikacije. Sporočila se generirajo pri vsakem dogodku v sistem, kot so npr. tipkanje, premikanje miške, klik na katero od komponent, sprememba velikosti okna ali pisave,… itn. Aplikacije lahko pošljejo sporočilo sama sebi, to je uporabno recimo za komunikacijo med nitmi, ali pa za sporočanje drugi aplikaciji.

Sporočilo je poslano s štirimi parametri. Prvi parameter je ciljno okno (angl. window handle) preko katerega operacijski sistem ugotovi kateremu oknu je sporočilo namenjeno. Drugi parameter je identifikator sporočila (angl. message identifire), ki je konstanta s katero določimo namen sporočila. Preko te konstante, okno ve, kako ravnati s tem sporočilom.

Ostala sta nam še dva parametra, ki se imenujeta sporočilna parametra (angl. message parameters). Preko teh dveh parametrov lahko sporočimo numerično vrednost (angl. integer), kazalec na objekt (angl. pointer), itn. Glede na identifikator sporočila tudi vemo kako ravnati s sporočilnima parametroma, in kakšnega tipa sta.

Poznamo dve vrsti sporočil, prva so sistemska sporočila (anl. System-Defined Messages), druga pa so aplikacijska sporočila (Application-Defined Messages).

Operacijski sistem pošilja sistemska sporočila, ko želi komunicirati s posamezno aplikacijo ali pa ji le želi sporočiti nek vhodni podatek. Poleg sistema lahko tudi aplikacije pošiljajo ta sporočila. Primer uporabe pošiljanja sistemskih sporočil s strani aplikacije je, ko želimo programsko spremeniti velikost odprtega okna ali pa, ko želimo ponovno izrisati okno. Vsako sistemsko sporočilo ima svojo konstanto v obliki simbola. Simbol WM_PAINT nam pove, da želimo ponovno izrisati okno. V tem primeru sta prvi dve črki WM predpona (ang. Prefix), ki nam pove kategorijo sistemskega sporočila. Ta predpona pove, da je to splošno okno (angl. General Window), PAINT pa je akcija, ki jo želimo izvesti.

Predpona	Kategorija
ABM	Application desktop toolbar
BM	Button control
CB	Combo box control
CBEM	Extended combo box control
CDM	Common dialog box
DBT	Device
DL	Drag list box
DM	Default push button control
DTM	Date and time picker control
EM	Edit control
HDM	Header control
HKM	Hot key control
IPM	IP address control
LB	List box control
LVM	List view control
MCM	Month calendar control
PBM	Progress bar
PGM	Pager control
PSM	Property sheet
RB	Rebar control
SB	Status bar window
SBM	Scroll bar control
STM	Static control
TB	Toolbar
TBM	Trackbar
TCM	Tab control
TTM	Tooltip control
TVM	Tree-view control
UDM	Up-down control
WM	General window

Tabela 1: Seznam kategorij

Poleg že standardnih konstant določenih s strani sistema, lahko aplikacije uporabljajo tudi sporočila, ki jih definiramo sami. Ta sporočila definiramo tako, da prištejemo neko svojo konstanto sistemsko definirani konstanti WM_USER ali WM_APP. Če se želita dve aplikaciji sporazumevati preko aplikacijskih sporočil morata obe poznati vrsto sporočila in kako ravnati s tem sporočilom.

3.2. Možnosti in omejitve

Po tem, ko smo spoznali nekaj vrst povezovanja med aplikacijami, lahko rečemo da ima vsak način povezovanja tako pozitivne kot negativne strani. V našem primeru bomo uporabili 2 vrsti komunikacije med aplikacijami (OLE Automation in sporočila), ter povezovanje s podatkovno bazo (DLL knjižnica).

Komunikacija s podatkovno bazo poteka po načinu podatki na zahtevo (angl. pulling). To dejansko ni dvosmerna komunikacija, saj nam podatkovna baza, ki predstavlja neke vrste strežnik, le odgovarja na naša SQL vprašanja. To SQL vprašanje se lahko nanaša na vstavljanje nove vrstice, posodobitev obstoječe vrstice ali pa samo na pridobitev podatkov.

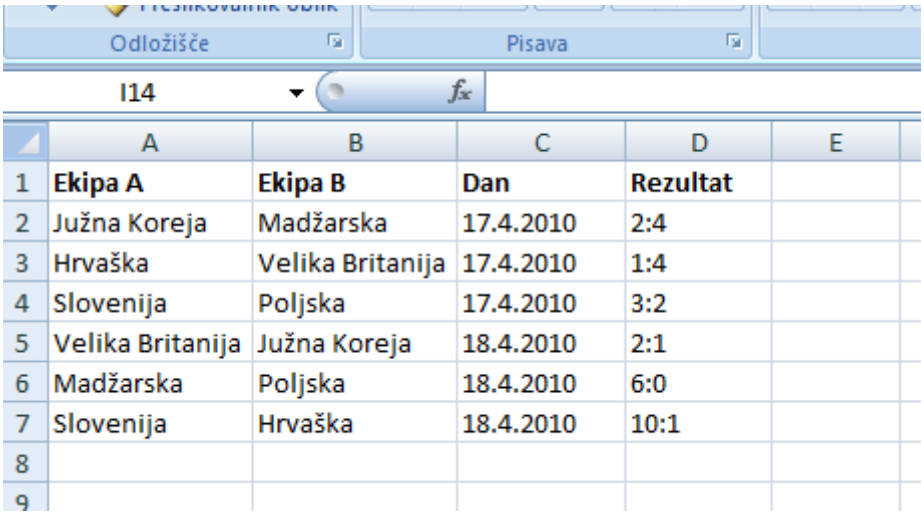
Komunikacija preko OLE Automation je zelo podobna komunikacija s podatkovno bazo MySQL. Razlika med prvo in drugo je le v tem, da pri podatkovni bazi MySQL uporabljamo SQL vprašanja, pri OLE Automation pa objekte in metode.

Ker nam Microsoft Excel ne nudi sporočanja o spremembah, ki so se zgodile v odprti datoteki se moramo za ugotavljanje sprememb upreti na druge mehanizme. V našem primeru bomo uporabili sistemska sporočila, ki jih operacijski sistem sporoča ob vsaki spremembi v določenem imeniku. Ker nam sistemska sporočila ne podajo točne informacije kaj se je spremenilo, bomo ob zaznani spremembi sprožili algoritem, ki bo to ugotovil.

4. Primer: opis aplikacije CollectionEditor

4.1. Kaj je CollectionEditor

CollectionEditor je aplikacija, ki nam bo predstavila uporabo OLE Automation kot komunikacijsko pot med našo aplikacijo in aplikacijo Microsoft Excel. Predstavila nam jo bo v obliki urejevalnika zbirk. Posamezna zbirka predstavlja eno tabelo v podatkovni bazi MySQL, paket zbirk pa je shema v podatkovni bazi MySQL. Zbirka je v aplikaciji CollectionEditor predstavljena kot dvodimenzionalna tabela, ki ima kot prvo vrstico imena stolpcev v tabeli podatkovne baze MySQL. Tip podatkov je zaradi lažjega programiranja zaklenjen na besedilo (angl. string).



	A	B	C	D	E
1	**Ekipa A**	**Ekipa B**	**Dan**	**Rezultat**	
2	Južna Koreja	Madžarska	17.4.2010	2:4	
3	Hrvaška	Velika Britanija	17.4.2010	1:4	
4	Slovenija	Poljska	17.4.2010	3:2	
5	Velika Britanija	Južna Koreja	18.4.2010	2:1	
6	Madžarska	Poljska	18.4.2010	6:0	
7	Slovenija	Hrvaška	18.4.2010	10:1	
8					
9					

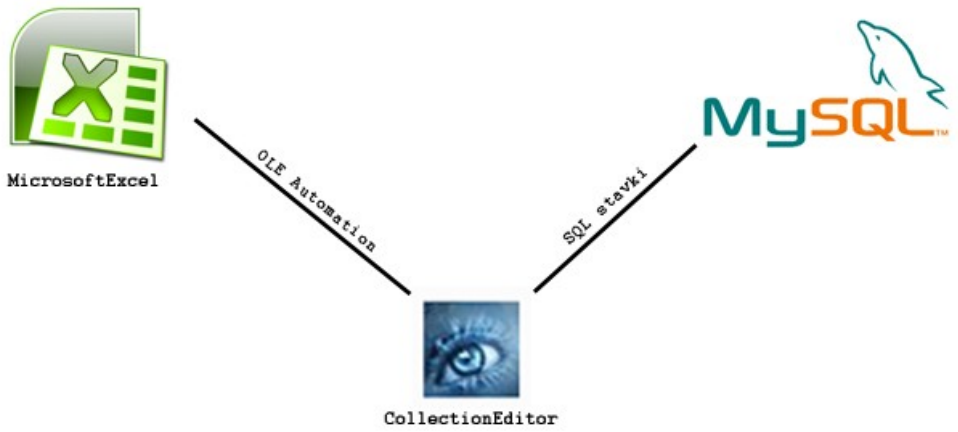
Slika 15: Primer zbirke

CollectionEditor teče v treh nitih, ki med seboj v času delovanja komunicirajo preko sporočilnega sistema. Prva nit aplikacije je tudi glavna nit. To je nit na katero ima uporabnik neposreden vpliv. Ta nit vedno čaka na uporabnikove ukaze in glede na njih tudi sproži določene akcije. Ta nit upravlja z oknom kjer lahko dodajamo oziroma odpiramo obstoječe zbirke.

Druga nit aplikacije je sistem za zaznavanje sprememb v imeniku (angl. File Monitor) našega diska. Nit deluje v neskončni zanki, ki se konča z zaprtjem aplikacije. Ko zaznamo novo sporočilo, preko sporočilnega sistema sporočimo glavni niti, da je prišlo do sprememb v imeniku. Glavna nit nato pregleda kaj se je spremenilo in na spremembo tudi reagira, druga nit pa ponovno čaka na novo sporočilo.

Da bi dokazali zmožnost hkratnega spreminjanja enega delovnega lista v aplikaciji Microsoft Excel s strani uporabnika na eni strani in aplikacije na drugi strani smo uvedli tretjo nit. V tej niti imamo ponovno neskončno zanko, ki vsakih 30 sekund prvi niti sporoči, naj ponovno naloži glavo zbirke. Le ta pa posodobitev izvede na vseh trenutno odprtih zbirkah.

Tako lahko rečemo, da je CollectionEditor vmesnik med aplikacijo MS Excel ter podatkovno bazo MySQL (slika 16).



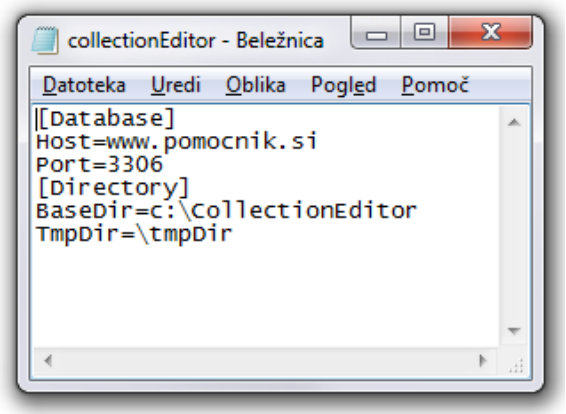
Slika 16: Shema povezave Collection Exe in povezanih programov

4.2. Implemetacija

4.2.1. Nastavitve CollectionEditor

Kot vsaka druga aplikacij, tudi CollectionEditor potrebuje določene nastavitve. V aplikaciji CollectionEditor imamo na voljo dva sklopa nastavitev. Prvi sklop nastavitev je namenjen nastavitvam podatkovne baze. Omogočeno nam je nastavljanje lokacije podatkovne baze, ter vrat na katerih ta podatkovna baze posluša. Drugi sklop nastavitev se nanaša na imenik kamor bomo shranjevali trenutno odprte zbirke.

Za nastavitve se v večini primerov uporabljata dva načina. Prvi način je uporaba registra, drugi pa uporaba INI datotek. Oba načina sta v programskem jeziku Delphi dobro podprta, odločili pa smo se za uporabo knjižnice za urejanje INI datotek. INI datoteka je enostavna tekstovna datoteka z končnico .INI, v kateri so nastavitve napisane vsaka v svoji vrstici (slika 17).



Slika 17: Nastavitve programa CollectionEditor

Branje in pisanje v INI datoteko poteka preko objekta TIniFile, ki ga kreiramo kot spremenljivko v funkciji ali razredu. Pomembno je da spremenljivko kreiramo s podano potko do naše INI datoteke. Primer uporabe objekta TIniFile v aplikaciji CollectionEditor najdemo v funkciji Initialize razreda DBHandler (slika 18).

```
function TDBHandler.Initialize: integer;
var
  appINI : TIniFile;
  hostString, portString : string;
begin

 // appINI := TIniFile.Create('collectionEye.ini') ;
  appINI:=TIniFile.Create(ChangeFileExt( Application.Exename,'.ini'));
  if not Assigned(appINI) then begin
    ShowMessage('Nastavitve ne obstajajo!');
    Result := mrCancel;
  end;
  hostString := appINI.ReadString('Database','Host','') ;
  portString := appINI.ReadString('Database','Port','') ;
  if ((hostString = '') OR (portString = '')) then begin
```

Slika 18: Branje iz *.INI datotek

4.2.2. Delo s podatkovno bazo MySQL

Delo s podatkovnimi bazami je eno najpomembnejših in tudi eden najbolj ranljivih delov vsakega programa. Priporočena je uporaba že uveljavljenih knjižnic, ki pa so v večini primerov plačljive. V našem primeru smo zato napisali svojo knjižnico imenovano DBManagerHandler, ki uporablja standardne Delphi komponente za delo z bazami (slika 19).

```
type TDBManagerHandler = class
  private
    m_con:  TZConnection;

    procedure DestroyDB();
  public
    constructor Create();
    destructor Destroy();

    function  Connect(host, port, username, password, dbname: string): boolean;
    procedure Disconnect();

    function  CreateQuery(): TZQuery;
    procedure DestroyQuery(var data: TZQuery);
  end;
```

Slika 19: Inicializacija TZConnection ter funkcij

V knjižnici so:

- spremenljivka **m_con**

To je spremenljivka objekta TZConnection, ki drži aktivno povezavo na podatkovno bazo.

- procedura **DestroyDB()**

Procedura DestroyDB pokliče funkcijo Disconnect objekta TZConnection in nato sprosti pomnilnik, ki ga je zasedala spremenljivka m_con.

- konstruktor **Create()**

Konstruktor Create pokliče dedovani kontruktor, ki naredi instanco razreda.

- destruktor **Destroy()**

Destruktor Destroy pokliče dedovani destruktor, ki uniči instanco razreda DBManagerHandler.

- funkcija **Connect**

Funkcija Connect izvede povezavo na podatkovno bazo preko spremenljivke m_con. Za uspešno povezavo potrebuje lokacijo podatkovne baze, vrata na katerih podatkovna baza posluša, uporabniško ime, geslo ter shemo na katero se bomo priklopili.

Če je povezava uspešna vrne TRUE, drugače pa FALSE.

Na sliki 20 si funkcijo Connect tudi lahko ogledamo.

```
{-------------------------------------------------------------------------------}

function  TDBManagerHandler.Connect(host, port, username, password, dbname: string): boolean;
begin
  DestroyDB();

  try
    m_con  := TZConnection.Create(nil);


    m_con.User := username;
    m_con.Password := password;
    m_con.Database := dbname;
    m_con.Name := 'm_con';
    m_con.AutoCommit := true;
    m_con.Protocol := 'mysql-5';
    m_con.HostName := host;
    m_con.Port := StrToInt(port);
    m_con.Connect();

   if (m_con.Connected = true) then begin
      Result := true;
    end else begin
      FreeAndNil(m_con);
      Result := false;
    end;
  except
    on E: Exception do begin
      FreeAndNil(m_con);
      Result := false;
    end;
  end;
end;
```

Slika 20: Funkcija Connect

- procedura **Disconnect**

Procedura Disconnect pokliče privatno proceduro DestroyDB.

- funkcija **CreateQuery()**

Funkcija CreateQuery naredi novo instanco objekta TZQuery. Temu objektu poda tudi povezavo preko spremenljivke m_con, in nato to spremenljivko vrne klicoči funkciji.

- procedura **DestroyQuery()**

Procedura DestroyQuery uniči instanco objekta TZQuery in sprosti pomnilnik, ki ga je ta objekt zasedal.

Knjižnica DBManagerHandler je namenjena najosnovnejšem delu s podatkovno bazo (priklop na bazo, izdelava SQL poizvedb,…) zato je potrebno napisati tudi razred, ki bo to knjižnico uporabljal na nivoju SQL poizvedb. V ta namen je bil napisan razred DBHandler, ki razširja knjižnico DBManagerHandler (slika 21).

```
type TDBHandler = class(TDBManagerHandler)
  private
```

Slika 21: TDBHandler razširja knjižnico TDBManagerHandlerja

V tem razredu se nahajajo vse potrebne funkcije za dodajanje, urejanje in branje zapisov na podatkovni bazi. Poleg tega vsebuje tudi funkcijo za dodajanje novih tabel imenovano CreateNewTable (slika 22).

```
procedure TDBHandler.CreateNewTable(tableName: string);
var
  sql: string;
  data: TZQuery;
begin
  try
    data := CreateQuery();
    sql := 'CREATE TABLE ' + tableName + ' ( ' +
           '`Id` int(10) unsigned NOT NULL AUTO_INCREMENT, ' +
           'PRIMARY KEY (`Id`) ' +
           ') ENGINE=InnoDB DEFAULT CHARSET=latin1;';
    data.SQL.Add(sql);
    data.ExecSQL;
  finally
    DestroyQuery(data);
  end;
end;
```

Slika 22: Funckija CreateNewTable

Funkcija kliče dedovano funkcijo CreateQuery. Funkcija CreatQuery vrne objekt TZQuery, kateremu dodamo SQL vprašanje. Na koncu moramo še izvesti dejansko poizvedbo na podatkovno bazo. To poizvedbo lahko izvedemo s klicem funkcije ExecSQL ali funkcije Open. Katero funkcijo bomo uporabili je odvisno od vrste SQL vprašanja. Funkcijo ExecSQL uporabimo za SQL vprašanja dodajanja in urejanja zapisov, funkcijo Open pa za SQL vprašanja pridobitve podatkov.

4.2.3. Delo z Excelom

Bistvo naše aplikacije je seveda prikaz dela z aplikacijo Microsoft Excel preko protokola OLE Automation. Kot smo ugotovili imamo dve možnosti, pozno ali zgodnje povezovanje. Za pozno povezovanje je dovolj, da uporabimo knjižnico OLEVariant, ki je standardna Delphi knjižnica. Prav tako je standardna Delphi knjižnica tudi ExcelXP, katero uporabimo ko želimo uporabljati zgodnje povezovanje.

Delovanje OLE Varianta si bomo ogledali skozi implementacijo v aplikaciji CollectionEditor in sicer v vrstnem redu, kot se pojavijo med samim delovanjem. Tako je prva funkcionalnost, ki jo srečamo, povezovanje z aplikacijo Microsoft Excel ter kreiranje nove datoteke. Povezavo z aplikacijo Microsoft Excel ustvarimo s pomočjo objekta TExcelApplication, poleg tega pa potrebujemo tudi objekta TExcelWorkBook ter TExcelWorkSheet. Ker bomo držali povezavo z aplikacijo Microsoft Excel za vsako posamezno datoteko do konca izvajanja aplikacije CollectionEditor, oziroma do zaprtja Microsoft Excel datoteke, moramo vse tri objekte držati v listi. Zaradi lažjega hranjenja in posledično pridobivanja aktualne povezave z Microsoft Excel aplikacijo smo vse tri zgoraj naštete objekte združili v objekt imenovan TCEExcelObject(slika 23).

```
type
  TCEExcelObject = class (TObject)
  private
    m_collection : string;
    m_ExcelApp   : TExcelApplication;
    m_WorkBook   : TExcelWorkBook ;
    m_workSheet  : TExcelWorkSheet;
  public
    property collection : string            read m_collection write m_collection;
    property ExcelApp   : TExcelApplication read m_ExcelApp   write m_ExcelApp;
    property WorkBook   : TExcelWorkBook    read m_WorkBook   write m_WorkBook;
    property workSheet  : TExcelWorkSheet   read m_workSheet  write m_workSheet;
  end;
```

Slika 23: TCEExcelObject

Da ustvarimo aktivno povezavo z aplikacijo Microsoft Excel, najprej kreiramo novo instanco TCEExcelObject, nato pa kreiramo še lastnost tega objekta z imenom ExcelApp, ki je tipa TExcelApplication. Tej lastnosti nato določimo še tip povezave (angl. ConnectKind), ki je lahko ckNewInstance, ckRunningOrNew ali ckRunningInstance (Tabela 2).

Tip povezave	Akcija
ckNewInstance	Naredimo nov proces.
ckRunningOrNew	Če že obstaja proces se priklopi nanj, drugače kreira svojega
ckRunningInstance	Če že obstaja proces se priklopi nanj, v nasprotnem primeru javi napako.

Tabela 2: Načini povezave do MS Excel

Ko se povežemo z aplikacijo Microsoft Excel moramo ustvariti še novo instanco delovnega zvezka, ki je tipa TExcelWorkBook, le tega pa nato povežemo z aktivnim delovnim zvezkom naše instance aplikacije Microsoft Excel. Zadnji je delovni list tipa TExcelWorkSheet, ki ga priklopimo na prvi delovni list aktivnega delovnega zvezka. Vso to funkcionalnost imamo zajeto v funkciji NewFile (slika 24).

```
procedure TExcelHandler.NewFile(collection : string);
var
  startCell, endCell : String;
  i : integer;
begin
 try
   m_Excel := TCEExcelObject.Create;

   m_Excel.collection := collection;

   m_Excel.ExcelApp := TExcelApplication.Create(nil);
   m_Excel.ExcelApp.ConnectKind := ckNewInstance;
   m_Excel.ExcelApp.Connect;

   m_Excel.ExcelApp.Workbooks.Add(Null,0);
   m_Excel.WorkBook := TExcelWorkbook.Create(nil);
   m_Excel.WorkBook.ConnectTo(m_Excel.ExcelApp.Workbooks[1]);
   m_Excel.WorkSheet := TExcelWorksheet.Create(nil);
   m_Excel.WorkSheet.ConnectTo(m_Excel.WorkBook.Worksheets[1] as _Worksheet);
   m_Excel.WorkSheet.Cells.ClearFormats();

   m_fileName := KernelHandler.GetTmpDir() + collection + KernelHandler.GetExcelExtenstion;

   startCell := Char(64+1)+IntToStr(1);
   endCell := Char(64+EXCEL_MAX_COLUMN)+IntToStr(EXCEL_MAX_ROWS);
   m_Excel.WorkSheet.Range[startCell,endCell].Cells.NumberFormat := '@';

   CellStyle();

   m_step := 2;
   m_cell := 1;
   m_row := 1;

 except
   //
 end;
end;

{-------------------------------------------------------------------------------}
```

Slika 24: Funkcija NewFile

Podatke dodajamo v datoteko preko našega delovnega zvezka. Pri dodajanju je pomembno, da vemo v katero celico želimo dati posamezen podatek. OLE Automation nam omogoča, da dodajamo podatke v obliki več dimenzionalnega polja (angl. array) za izbrano skupino (angl. Range) celic. Skupino celic A1, A2, A3, B1, B2, B3 določimo tako, da podamo A1 kot začetno in B3 kot končno celico. Če je začetna in končna celica ista, pomeni da želimo urejati le tisto celico.

Ko imamo izbrano skupino celic lahko nad njo izvajamo različne akcije, kot je recimo spreminjanje tipa podatkov, formatiranje pisave, itn. Poleg vseh drugih funkcije je za nas pomembno dodajanje in branje podatkov. Vrednost posamezne celice je dostopna preko lastnosti Value2. Na sliki 25 si lahko ogledamo primer dodajanja vrednosti v eno celico.

```
m_Excel.WorkSheet.Range[startCell,startCell].Value2 := value;
```

Slika 25: Vstavljanje vrednosti

Ker je Value2 tipa OLEEnum, sprejme kateri koli tip podatkov. Če izberemo le eno celico, lahko podamo podatke tipa integer, string, boolean,, itn, drugače pa je potrebno podati polje vrednosti.

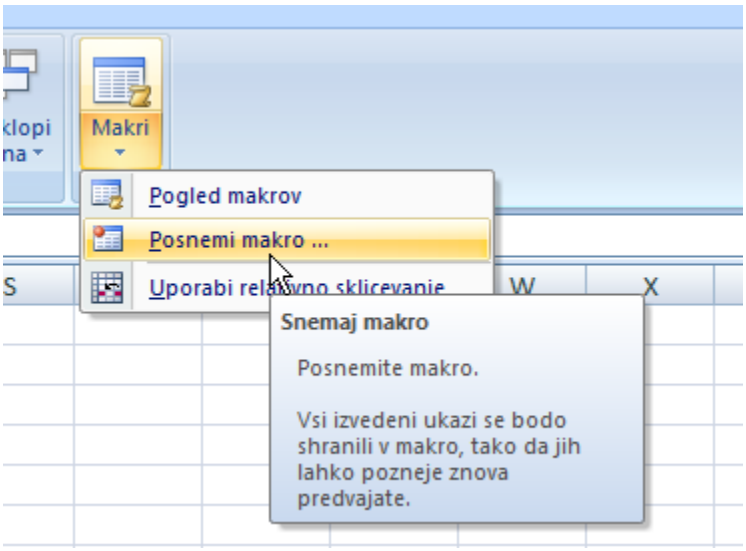
Podatke shranjene v datoteki Microsoft Excel moramo sedaj še shraniti v naš imenik na disku, katerega bomo nadzorovali za spremembe. Funkcionalnost, ki nam omogoča shranjevanje trenutno kreiranega Microsoft Excel dokumenta, se nahaja v funkciji EndEditing (slika 26).

```
procedure TExcelHandler.EndEditing(save : boolean = true);
var
  fileName : OleVariant;
begin
  try
    m_Excel.WorkSheet.Columns.AutoFit;
    if (save) then begin
      if (FileExists(m_fileName)) then begin
        KernelHandler.IOHandler.RemoveReadOnlyFlag(m_fileName);
      end;
      fileName := m_fileName;

      m_Excel.WorkBook.SaveAs(fileName,xlNormal, '', '', false, false,
                    xlNochange,xlUserResolution,False,EmptyParam,EmptyParam,
                    EmptyParam, LOCALE_USER_DEFAULT);

    end;
    m_Excel.ExcelApp.DisplayAlerts[LOCALE_USER_DEFAULT] := false;

    m_Excel.ExcelApp.Visible[LOCALE_USER_DEFAULT] := true;
    m_applicationList.AddNode().data := m_Excel;
    m_step := 1;
  except
    on E: Exception do begin
      m_step := 1;
    end;
  end;
end;
```

Slika 26: Funkcija EndEditing

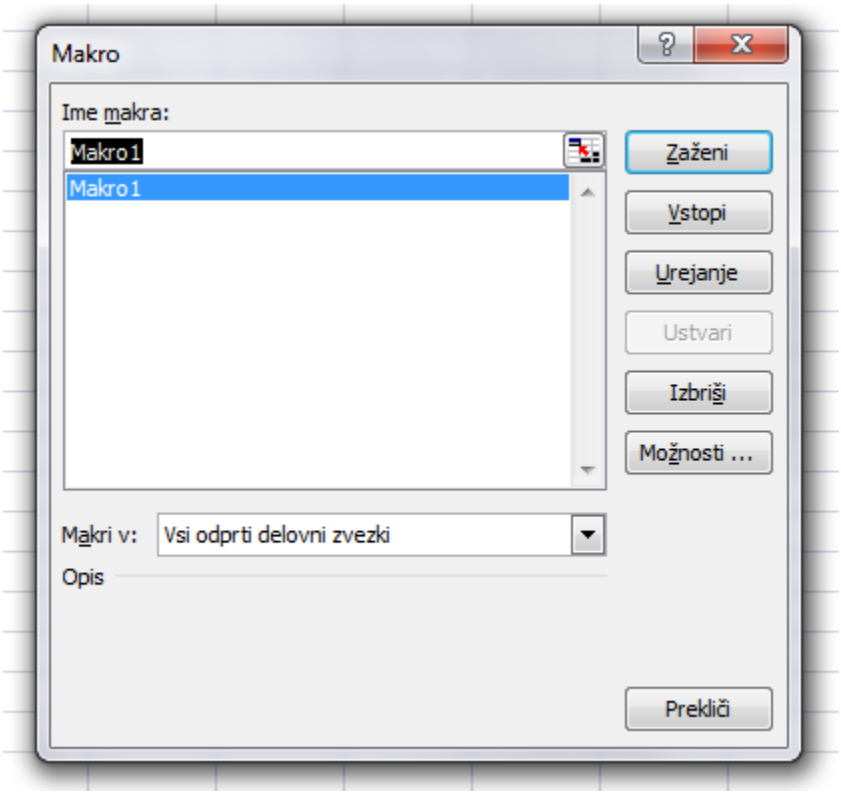
Čeprav imamo vse objekte in funkcije dostopne v času programiranja, pa nam včasih zmanjka idej, kako lahko preko OLE Automation izvedemo določeno funkcionalnost, ki jo Microsoft Excel omogoča. K sreči paket aplikacij Microsoft Office pozna makroje, kateri uporabnikom omogočajo, da si določene akcije, ki jih vedno izvajajo, posnamejo in jih namesto njih izvede aplikacija. Ta postopek se zapiše v obliki Visual Basic kode. Poglejmo si, kako bi to lahko uporabili, če recimo ne bi vedeli kako se spremeni velikost pisave v celici.

Najprej si odpremo Microsoft Excel, vpišemo neko vrednost v polje A1, potem pa v razdelku ogled posnamemo nov makro (slika 27).



Slika 27: Snemanje novega makroja

Sedaj spremenimo velikost te celice in ustavimo snemanje makroja preko istega spustnega menija, kot smo njegovo snemanje začeli. Iz tega menija izberemo »Pogled makrojev« in prikazal se nam bo seznam posnetih makrojev (slika 28).



Slika 28: Seznam makrojev

V tem oknu izberemo naš makro ter kliknemo na gumb Urejanje, ki nam izpiše posneto akcijo v obliki Visual Basic kode. Ta koda je precej podobna Delphi kodi, vendar jo je še vedno potrebno prevesti. Originalna Visual Basic koda je na sliki 29, prevedena Delphi koda pa na sliki 30.

```
Sub Makro1()
'
' Makro1 Makro
'

'
    Range("A1").Select
    Selection.Font.Size = 12
End Sub
```

Slika 29: Koda v programskem jeziku Visual Basic

```
WorkSheet.Range[startCell,startCell].Cells.Font.Size := 12;
```

Slika 30: Prevedena koda v Delphi programski jezik

4.2.4. Zaznavanje s sprememb

Ker ima OLE Automation pomanjkljivost, ki nam ne omogoča sporočanja o spremembah na dokumentu MS Excel, smo razvili mehanizem za ugotavljanje le teh.

Ta mehanizem je bil razvit tako, da izkorišča sistemska sporočila, ki jih operacijski sistem Microsoft Windows sporoča ob različnih dogodkih v sistemu. Tako je mogoče čakati na točno določeno sistemsko sporočilo, ki je bilo poslano iz točno določenega imenika na disku.

Mehanizem za ugotavljanje sprememb na dokumentih mora biti v svoji nit. Razlog za to je neizvajanje kode v času čakanja novega sporočila. V niti izvajamo zanko, znotraj katere čakamo na sporočilo. V aplikaciji CollectionEditor je ta neskončna zanka v funkciji Execute razreda FileMonitor (slika 31).

```
procedure TFileMonitor.Execute;
begin
  while Waiting do begin  //do zaključka delovanja programa
    dwResult := WaitForSingleObject(NotificationHandle,INFINITE); //čakanje na sporočilo
    if (dwResult = WAIT_OBJECT_0) then begin
      if (Assigned(CallBack) AND (SendMessage)) then begin
        CallBack;
        SendMessage := false;
      end else begin
        SendMessage := true;
      end;

      FindNextChangeNotification(NotificationHandle)
    end;
  end;

  FindCloseChangeNotification(NotificationHandle);
end;
```

Slika 31: Funkcija Execute, ki čaka na spremembe v imeniku

Ko zaznamo sporočilo pokličemo povratno funkcijo (angl. CallBack function), nato pa ponovno čakamo na novo sporočilo. Ta povratna funkcija uporabi aplikacijska sporočila, da o spremembi obvesti glavno nit aplikacije. V aplikaciji CollectionEditor je povratna funkcija FileWasChanged (slika 32).

```
procedure TCollectionHandler.FileWasChanged();
begin
  Sleep(500);
  PostMessage(MainForm.Handle,FORM_MESSAGE_FILE_WAS_CHANGED,0,0);
end;
```

Slika 32: Pošiljanje Windows sporočila prvi niti

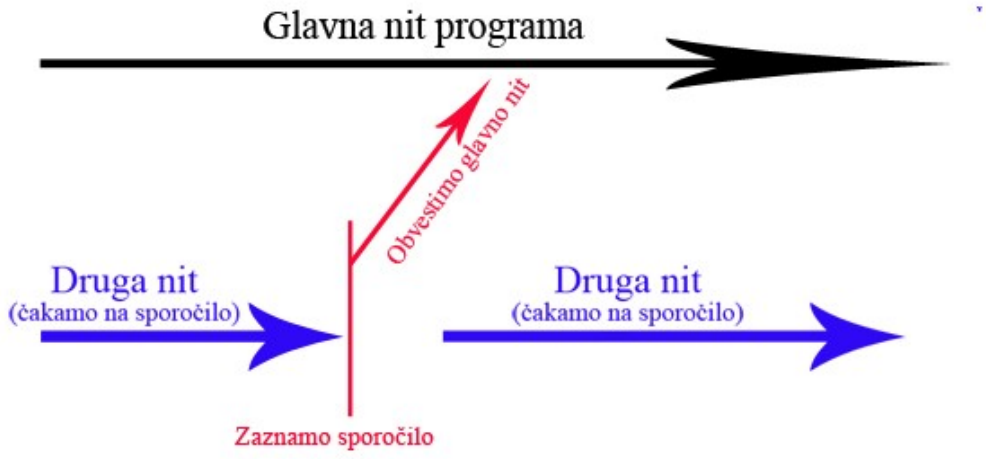
Če nit, kateri je sporočilo namenjeno, nima razvitega sistema za lovljenje tega tipa sporočil, le tega tudi ne zazna. Da nit lovi določeno sporočilo, moramo za to sporočilo napisati posebno proceduro, ki ima kot način sprožanja nastavljeno to sporočilo (slika 33).

```
procedure onFileChange(var Msg: TMessage); message FORM_MESSAGE_FILE_WAS_CHANGED;
```

Slika 33: Inicializacija funkcije, ki čaka na sporočilo

Sedaj v glavni niti vemo, da se je nekaj spremenilo v imeniku, v katerem hranimo odprte zbirke. Ne vemo pa, katera zbirka se je spremenila, zato imamo v posebni listi shranjene celotne zbirke, kjer imamo zapisan začetni objekt posamezne zbirke, ki smo jo izrisali, zadnjo spremembo datoteke ter povezavo do aplikacije Microsoft Excel. Za vsako zbirko nato sprožimo preverjanje. Če se je njena datoteka od zadnjega preverjanja spremenila potem posodobimo objekt, popravimo podatke na bazi in zapišemo nov čas spremembe.

Postopek življenja posamezne niti in komunikacijo med njima je vidna na sliki 34.



Slika 34: Dodana druga nit programu

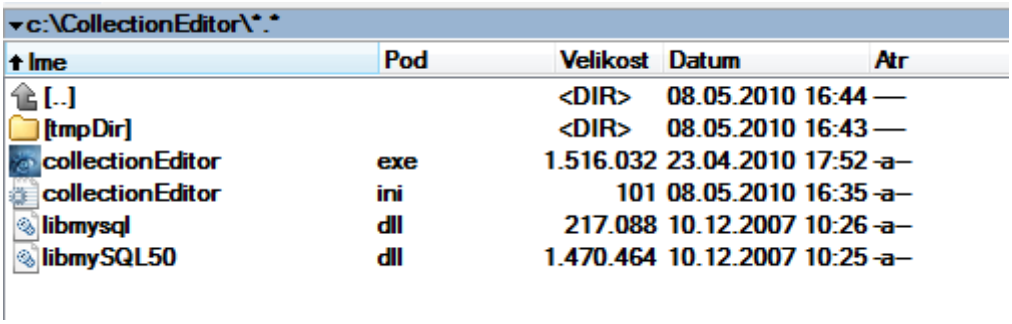
4.3. Uporaba

Preden začnemo uporabljati aplikacijo CollectionEditor je potrebno program namestiti. Za namestitev so potrebne 4 datoteke (slika 35):

libmysql.dll in **libmySQL50.dll** – DLL datoteki za delo z MySQL podatkovno bazo

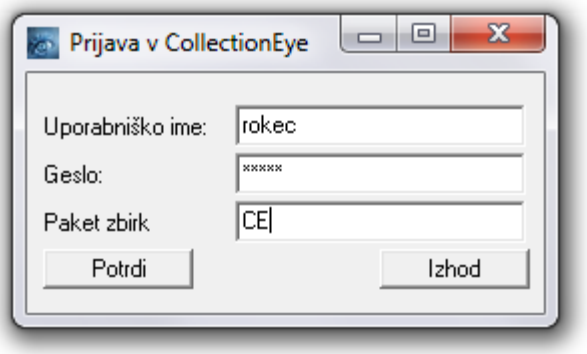
collectionEditor.ini – nastavitvena datoteka za aplikacijo CollectionEditor

collectionEditor.exe – zagonska datoteka aplikacije



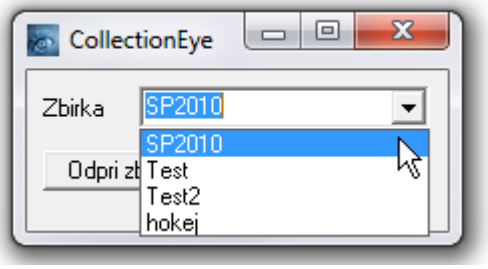
Slika 35: Datoteke programa CollectionEditor

Za začetek dela zaženemo datoteko collectionEditor.exe, ta pa nam odpre prijavno okno aplikacije, v katerega moramo vpisati uporabniško ime, geslo in ime paketa zbirke, ki je v podatkovni bazi prikazan kot shema (slika 36).



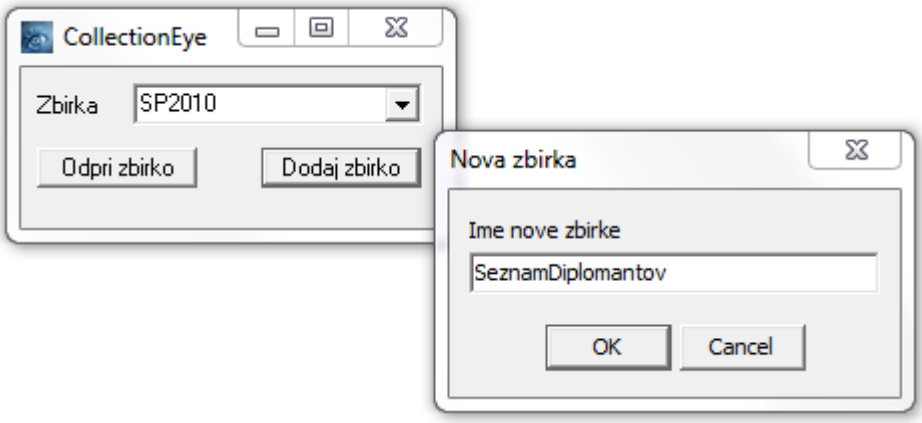
Slika 36: Prijavno okno

Uporabniško ime, geslo in pravice za posamezno shemo so standardni uporabniki MySQL podatkovne baze, z uporabo katerih se lahko na podatkovno bazo povežemo tudi preko drugih urejevalnikov. Če je prijava uspešna, se nam odpre novo okno, v katerem imamo preko spustnega menija prikazane že vnešene zbirke (slika 37).



Slika 37: Že obstoječe zbirke

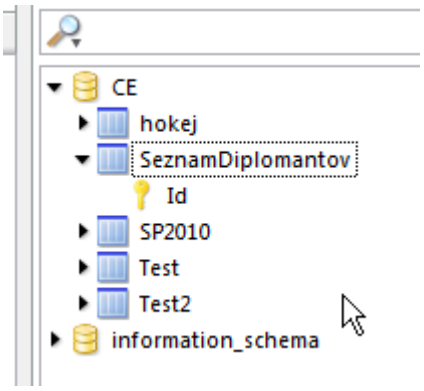
Ker pa želimo prikazati celoten postopek, od izdelave do popravljanja zbirke s z aplikacijo CollectionEditor, bomo zbirko najprej ustvarili. To naredimo tako, da pritisnemo na gumb »Dodaj zbirko« in potem v pogovornem oknu vnesemo želeno ime zbirke (slika 38).



Slika 38: Dodajanje nove zbirke

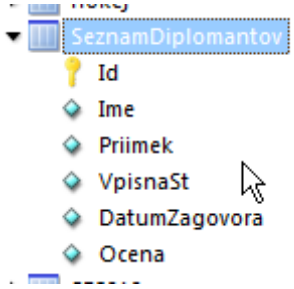
Po potrditvi lahko iz seznama zbirk izberemo našo zbirko in jo s klikom na gumb »Odpri zbirko«, tudi odpremo. Odpre se nam prazna Microsoft Excel datoteka, v katero moramo sedaj v prvo vrsto vpisati imena stolpcev, saj naša zbirko trenutno pozna samo polje Id, ki pa

ga kot uporabnik ne vidimo. Če zbirko pogledamo iz aplikacije MySQL Query Browser vidimo strukturo naše tabele (Slika 39).



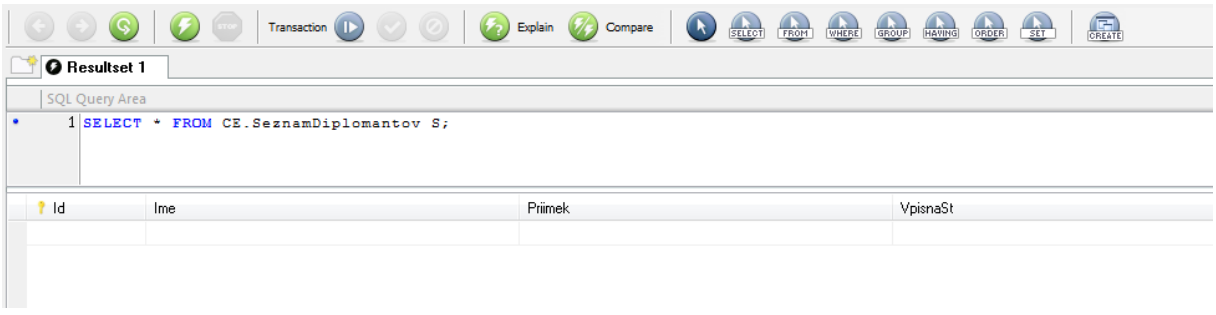
Slika 39: Struktura tabele SeznamDiplomantov v programu MySql Query Browser

Sedaj dodamo 5 polj: Ime, Priimek, VpisnaSt, DatumZagovora, Ocena in nato shranimo datoteko preko kombinacije tipk Ctrl+S. To moramo storiti v roku 30 sekund, drugače nam tretja nit (angl. thread) programa ponovno naloži prejšnjo glavo zbirke. Če gremo sedaj ponovno v aplikacije MySQL Query Browser in pogledamo strukturo naše tabele, lahko opazimo naša nova polja (slika 40).



Slika 40: Struktura tabele po tem ko smo dodali stolpce

Do sedaj smo urejali samo strukturo tabele, podatkov pa še nismo dodali. O tem se lahko prav tako prepričamo preko aplikacije MySQL Query Browser (slika 41).



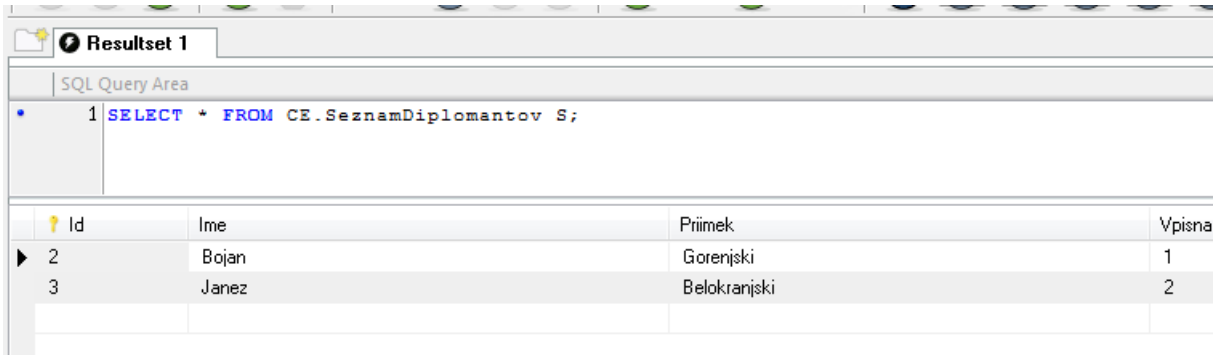
Slika 41: V tabeli še nimamo podatkov

Z drugo vrstico Microsoft Excel datoteke se začne podatkovni del zbirke. Če sedaj v Microsoft Excelu dodamo 2 namišljeni osebi (slika 42) in spremembo ponovno shranimo, bomo te podatke lahko videli v tabeli naše podatkovne baze (slika 43).

E10

	A	B	C	D	E
1	**Ime**	**Priimek**	**VpisnaSt**	**DatumZagovora**	**Ocena**
2	Bojan	Gorenjski	1	1.1.2010	7
3	Janez	Belokranjski	2	7.8.2010	/
4					

Slika 42: Podatki v programu Microsoft Excel



Slika 43: Podatki v programi MySQL Query Browser

Podatke lahko sedaj poljubno urejamo, dodajamo nove stolpce, stolpce odstranjujemo, itd.

4.4. Možne nadgradnje

Aplikacija CollectionEditor je za samo predstavitev delovanja OLE Automation povsem dobra, manjka pa ji kar nekaj funkcionalnosti, da bi dosegla raven na kateri bi jo lahko redno uporabljali. Prva in največja pomanjkljivost aplikacije je, da aplikacija ne zazna sprememb na podatkovni bazi. Ta problem je pri uporabi podatkovne baze MySQL težje odpraviti, saj ta ne omogoča mehanizma za sporočanje o spremembah. Za tako funkcionalnost bi morali preiti na kako drugo podatkovno bazo.

5. Zaključek

Uporaba enotnega GUI vmesnika v teoriji zveni zelo dobro, vendar se to ne odraža v praksi. Za kaj takega bi bilo potrebno resnično dobro načrtovanje samega vmesnika, ki bi moral biti deloma tudi nastavljiv.

Uporaba aplikacije Microsoft Excel kot enotnega GUI vmesnika ni ravno dobra, saj zahteva kar nekaj programerskega znanja. Čas, ki ga porabimo za razvoj uporabe aplikacije Microsoft Excel kot GUI vmesnika, pa bi pri razvoju lastnega GUI vmesnika zmanjšali. Poleg tega ima uporabnik na voljo prav vse Microsoft Excel funkcionalnost, ki pa jih naša aplikacija sploh ne podpira.

Zato menim, da se je bolje držati ustaljenih poti programiranja. Seveda je potrebno paziti da naredimo čim bolj preprost in za uporabnika intuitivni grafični uporabniški vmesnik.

Celotna izvorna koda progama, ki je bil razvit kot prototip aplikacije, ki uporablja program Microsoft Excel kot grafični uporabniški vmesnik, je na priloženem CDju.

Kazalo slik

Kazalo tabel

Viri in literatura

(1) (2010) Graphical User Interface. Dostopno na http://en.wikipedia.org/wiki/Graphical_user_interface
(2) (2010) Internet socket. Dostopno na http://en.wikipedia.org/wiki/Internet_socket
(3) (2010) SOAP. Dostopno na http://en.wikipedia.org/wiki/SOAP
(4) (2010) Dynamic-link lybrary. Dostopno na: http://en.wikipedia.org/wiki/Dynamic-link_library
(5) (2010) OLE Automation. Dostopno na http://en.wikipedia.org/wiki/OLE_Automation
(6) (2010) About Messages and Message Queues. Dostopno na http://msdn.microsoft.com/en-us/library/ms644927%28VS.85%29.aspx