Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Simon Rupar

Razvoj povezave med programoma gretl in Excel

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Ljubljana 2012

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Simon Rupar

Razvoj povezave med programoma gretl in Excel

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: prof. dr. Blaž Zupan
Somentor: doc. dr. Gaj Vidmar

Ljubljana 2012

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: 00153/2011
Datum: 01.09.2011

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

Kandidat: **SIMON RUPAR**

Naslov: **POVEZAVA GRETLA Z EXCELOM**
INTERFACING GRETL AND EXCEL

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Namen diplomskega dela je izvesti povezavo odprtokodnega večplatformskega ekonometričnega programskega paketa gretl z elektronsko preglednico Excel. Na podlagi gretlovega vmesnika uporabniškega programa (libgretl API) bi diplomant poskusil razviti dodatek za Excel, ki bi prenašal podatke v obe smeri in izvajal pilotne primere analiz. Izdelano programje bi hkrati lahko predstavljalo podlago za integracijo zmogljivosti gretla v druge programe (zlasti s področja strojnega učenja oz. podatkovne analitike), ki bi jih bilo smiselno dopolniti s široko paleto zahtevnih statističnih metod (zlasti za analizo časovnih vrst), ki jih nudi gretl.

Mentor:

prof. dr. Blaž Zupan

Dekan:

prof. dr. Nikolaj Zimic

Somentor:

doc. dr. Gaj Vidmar

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Simon Rupar, z vpisno številko **63050103**, sem avtor diplomskega dela z naslovom:

Razvoj povezave med programoma gretl in Excel

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Blaža Zupana in somentorstvom doc. dr. Gaja Vidmarja,
- 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 15. marec 2011 Podpis avtorja:

Najlepše se zahvaljujem somentorju doc. dr. Gaju Vidmarju za vso pomoč in nasvete pri izdelavi diplomskega dela in mentorju prof. dr. Blažu Zupanu za potrpežljivost. Zahvalil bi se tudi vsem razvijalcem na MSDN, ki so mi s tehničnimi nasveti pomagali pri reševanju problemov. Hvala tudi družini in prijateljem, ki so mi stali ob strani in pretrpeli mojo odsotnost.

Kazalo

Seznam uporabljenih kratic in simbolov

gretl (angl. Gnu Regression Econometrics and Time-series Library) program za ekonometrijo

COM (angl. Component Object Model) binarni vmesniški standard

API (angl. Application Programming Interface) vmesnik uporabniškega programa

CLI (angl. Common Language Infrastructure) skupna jezikovna infrastruktura

CLR (angl. Common Language Runtime) skupni jezikovni izvajalnik

.NET programsko ogrodje, razvito v podjetju Microsoft

P/INVOKE (angl. Platform Invocation Services) tehnologija uporabljena za klice vmesnika uporabniškega programa znotraj ogrodja .NET

DLL (angl. Dynamic-link library) skupna knjižnica v sistemih Windows

GDT oblika zapisa podatkov, ki jo uporablja programski paket gretl

VBA (angl. Visual Basic for Applications) integrirano razvojno okolje znotraj paketov Microsoft Office

MSND (angl. Microsoft Developer Network) socialno omrežje za razvijalce v okoljih Microsoft

GUI (angl. graphical user interface) grafični uporabniški vmesnik

IDE (angl. Integrated development environment) integrirano razvojno okolje

IL (angl.Intermediate Language) vmesni jezik

Povzetek

Glavni cilj diplomske naloge je povezava programskega paketa gretl z Microsoftovo elektronsko preglednico Excel. gretl je paket za analizo v ekonometriji. S povezavo med tem dvema aplikacijama bi razširili nabor funkcij, ki jih omogoča Excel, z ekonometričnimi oziroma zahtevnimi statističnimi analizami. Namen diplomske naloge je dokazati, da je povezava med gretlom in Excelom izvedljiva in poiskati najboljši način izvedbe povezave. S to rešitvijo bi omogočili širši množici ljudi, ki sicer uporabljajo Excel, da se lažje srečajo z ekonometrijo oziroma napredno statistiko. V nalogi sta najprej na kratko predstavljena ekonometrija in programski paket gretl. V nadaljevanju je opisano, kaj je vmesnik uporabniškega programa, kako se ga uporablja in kako se ga zgradi. Kasneje sta opisana pojma ovojnica in Microsoftovo ogrodje .NET. Podrobno je opisan postopek izdelave dodatka za Excel, preko katerega poteka povezava z gretlom. Sledi predstavitev delovanja pilotnega dodatka, na koncu pa so podane smernice za nadaljnji razvoj.

Ključne besede:

COM Interop, ovojnica, COM vmesnik, programsko ogrodje .NET, vmesnik uporabniškega programa, Win32 API, skupni jezikovni izvajalnik, skupna jezikovna infrastruktura, P/Invoke, Microsoft Excel, Excelov objektni model, dodatek za Excel

Abstract

The main goal of the thesis was to connect the gretl econometric software package with the Microsoft's Excel electronic spreadsheet. By connecting these two applications, the functionality of Excel is extended by econometric and advanced statistical analyses. The aim of the thesis was to demonstrate that a connection between gretl and Excel is possible, and to find the most feasible way to implement it. The solution would enable the wide audience of Excel users to face econometrics and advanced statistics more easily. The thesis first briefly introduces econometrics and the gretl software. Next, the concept and development of application program interface is explained. The Wrapper concept and the Microsoft .NET framework are then described. The development of the Excel add-in, which implements the connection with gretl, is described in detail. The functionality of the pilot add-in is demonstrated. In the conclusion, possibilities for further development are outlined.

Key words:

COM Interop, Wrapper, COM Interface, .NET Framework, API, Win32 API, CLR, CLI, P/Invoke, Microsoft Excel, Excel object model, Excel Add-in

Poglavje 1

Uvod

V dandanašnjem času pohitrenega ritma življenja nam dostikrat ostane malo časa za učenje novih stvari. Na poklicnem področju je problem še izrazitejši, saj je razvoj znanosti in novih tehnologij še hitrejši, zato slednje vse hitreje zastarajo, zlasti na področju računalništva oziroma informatike.

Če želimo doseči, da bo našo aplikacijo uporabljal širok krog uporabnikov, jo moramo razviti tako, da bo v osnovi enostavna in intuitivna za uporabo. Čeprav je aplikacija revolucionarna in zelo dobro dokumentirana, je uporabniki brez tega ne bodo uporabljali, saj je ne bodo znali obvladovati, za poglobljeno učenje pa nimajo časa in motivacije.

Ena izmed aplikacij, za katero je ta problematika aktualna, je ekonometrični programski paket gretl. Čeprav je zasnovan tako, da postane razmeroma enostaven za uporabo, če le pregledamo dokumentacijo, večina potencialnih uporabnikov ni pripravljena vložiti truda in pozornosti v študij dokumentacije. Praktično vsi potencialni uporabniki (tj. vsi, ki imajo opravka z analizo podatkov) pa so se že srečali oziroma redno uporabljajo elektronsko preglednico Microsoft Excel. Zato smo se odločili, da ta programa povežemo.

Glavni cilj diplomske naloge je torej rešitev povezave gretla z Excelom. Programska rešitev je izvedena z dodatkom za Microsoft Excel. Preko tega dodatka smo izbrane funkcionalnosti, ki obstajajo v paketu gretl, prenesli v Excel. S tem pokazali, kako uporabniku omogočiti uporabo funkcij ekono-

metrije oziroma zahtevne statistike znotraj Excela.

V nalogi sta najprej na kratko predstavljena ekonometrija in programski paket gretl. V nadaljevanju je opisano, kaj je vmesnik uporabniškega programa (API, angl. Applicaton programming interface), kako se ga uporablja in kako se ga zgradi. V naslednjem koraku je opisano, kaj je ovojnica (angl. Wrapper) in čemu se uporablja. V nadaljevanju je razloženo delovanje ogrodja Microsoft .NET. V tem poglavju je pojasnjeno, kaj je Interop, obrazložen je pojem nadzorovane in nenadzorovane kode (angl. managed and unmanaged code), Microsoftov skupni jezikovni vmesnik (CLI, angl. Common Language Interface) in skupni jezikovni izvajalnik (CLR, angl. Common Language Runtime). V poglavju o razvoju rešitve je opisan postopek izdelave dodatka za Excel, preko katerega poteka povezava z gretlom, s kakšnimi težavami smo se srečali pri izdelavi programskega dela naloge in kako smo te probleme rešili. Sledi predstavitev delovanja pilotnega dodatka, na koncu pa so podane smernice za nadaljnji razvoj.

Poglavje 2

Razlaga osnovnih pojmov

2.1 Ekonometrija

Ekonometrija, kot pove že etimologija izraza, združuje ekonomijo in merjenje. Njenih definicij je – tako kot učbenikov in druge strokovne in znanstvene literature na to temo – ogromno, lahko pa povzamemo, da *ekonometrija* združuje ekonomsko teorijo, matematično modeliranje, zbiranje ekonomskih podatkov in statistične metode [1].

Ekonometrija temelji na razvoju statističnih metod za ocenjevanje ekonomskih odnosov, preverjanje ekonomskih teorij in vrednotenje ekonomskih odločitev oziroma politik. Tipična uporaba ekonometrije je napovedovanje ključnih makroekonomskih količin, kot so obresti, stopnja inflacije in bruto družbeni proizvod. Seveda pa so ekonometrične metode uporabne tudi na številnih drugih področjih v okviru ekonomije (npr. v mikroekonomiji ali trženju) in izven nje (npr. v politologiji, za analizo ekonomskih vidikov izobraževanja, znanosti in zdravstva idr.) [2].

Ekonometrični modeli so temelj kvantitativnih ekonomskih analiz. V ekonometriji uporabljajo ekonomske teorije in vire, poslovne podatke, spoznanja družboslovnih znanosti in statistična orodja za odgovarjanje na vprašanje "Koliko?"v ekonomskih analizah [3].

Osnovo ekonometrije predstavljajo regresijski modeli, ki segajo od osnov-

nih (multipla linearna regresija) do zahtevnih (modeli za omejeno opazovane spremenljivke, sistemi sočasnih enačb). Posebno pomembno in obsežno področje ekonometrije je analiza oziroma napovedovanje časovnih vrst (spektralna analiza, filtri, avtoregresijski modeli itd.).

Kakršnakoli vsebinska obravnava oziroma uporaba ekonometrije seveda daleč presega namen in raven tega diplomskega dela. Ker je namen diplomskega dela iskanje programske rešitve in izdelava pilotnega programja kot dokaz koncepta, smo uporabili le peščico najpreprostejših statističnih metod, ki predstavljajo dobesedno neznaten del celotne ekonometrije. Omejili smo se na najpreprostejše opisne statistike in na najpreprostejši regresijski model (z eno odvisno in neodvisno spremenljivko). Nekaj osnovnih informacij o uporabljenem regresijskem modelu je navedenih v poglavju o razvoju rešitve.

2.2 gretl

gretl je odprtokodni programski paket za ekonometrijo, ki obsega skupne knjižnice, program ukazne vrstice in grafični uporabniški vmesnik. Ime *gretl* (ki bi, pisano z veliko začetnico, pomenilo lik iz znane pravljice Hansl und Gretl, ki sta jo zapisala brata Grimm – v slovenščini torej Metka) je akronim za **G**nu **R**egression **E**conometrics and **T**ime-series **L**ibrary. *gretl* izvira iz programa ESL (angl. Econometrics Software Library), ki ga je napisal profesor Ramu Ramanathan s kalifornijske uniferze UCSD, avtor priljubljenega ekonometričnega učbenika Introductory Econometrics with Applications. Profesor Ramanathan je program ESL izdal pod licenco GNU General Public Licence in pomagal pri začetkih razvoja gretla. *gretl* sta razvila in še vedno dnevno razvijata profesorja Allin Cotrell (iz ZDA) in Richardo Luchetti (iz Italije).

Po navedbah avtorjev (spletišče *h*ttp://gretl.sourceforge.net/) je gretl

- prijazen do uporabnika (ponuja intuitiven uporabniški vmesnik, namestitev programa in poganjanje analiz je enostavno, ponuja dva zelo

dobra vira informacij – splošno dosegljivo dokumentacijo in elektronsko poštno listo oz. novično skupino gretl-users);

- prilagodljiv (nudi veliko možnosti uporabniških nastavitev, shranjuje izpis modelov v obliki LaTeX, tabel v obogatenem besedilu ali enačb);
- podprt na različnih platformah (Windows, Linux in Mac OS X);
- odprtokoden (v skladu z licenco GNU);
- napreden (ponuja širok nabor funkcij za ocenjevanje regresijskih modelov po metodi najmanjših kvadratov, metodi največjega verjetja in metodi momentov za posamezne enačbe ali sisteme enačb, vključuje vektorsko avtoregresijo, modele za opisne in okrnjene odvisne spremenljivke itd.);
- razširljiv (uporabniki lahko pišejo lastne funkcije in procedure v skriptnem jeziku, ki podpira širok nabor matričnih funkcij, simulacije Monte Carlo itd.);
- točen (testiran je bil z več uveljavljenimi preizkusi oziroma standardnimi podatkovji – angl. benchmarks);
- pripravljen za medmrežje (lahko dostopa do podatkovnih zbirk na strežnih preko interneta, verzija za Windows vključe možnost samodejnega posodabljanja);
- mednaroden (trenutno je poleg angleščine lokaliziran v francoščino, italijanščino, španščino, poljščino, portugalščino, nemščino, baskovščino, turščino in ruščino).

gretl lahko integrirano uporabljamo z naslednjimi statističnimi oziroma matematičnimi programskimi paketi:

- X-12-ARIMA,

- TRAMO/SEATS,
- R,
- Octave in
- Ox.

gretl je napisan v programskem jeziku C; uporablja GTK za izdelavo grafičnega uporabniškega vmesnika in gnuplot za izrisovanje grafikonov. Znotraj paketa *gretl* je tudi program ukazne vrstice. *gretl* ponuja lasten, polno dokumentiran datotečni tip, zgrajen na podlagi XML. Bere lahko tudi praktično vse najpomembnejše datotečne tipe, ki se uporabljajo pri ekonometrični oziroma statistični analizi podatkov: ASCII, CSV, databank, EViews, Excel, Gnumeric, Octave, JMulTi, OpenDocument Spreadsheet, PcGive, RATS 4, SAS Export, SPSS in Stata. Zapisuje pa lahko v datotečne tipe Octave, R, CSV, JMulTi in PcGive.

Poglavje 3

Tehnologija in orodja

3.1 API - vmesnik uporabniškega programa

Dandanes si težko predstavljamo razvoj programskih rešitev brez uporabe *vmesnika uporabniškega programa* (API – angl. Application Programming Interface). Tako uporabniki kot razvijalci vsakodnevno uporabljajo programe z vmesnikom uporabniškega programa, a se tega pogosto sploh ne zavedajo, najsi bo to v okolju Windows® ali v okoljih Linux in OS X.

Vmesnik uporabniškega programa je specifikacija, ki temelji na izvorni kodi in jo lahko uporabimo kot vmesnik, ki služi za medsebojno komunikacijo različnih programskih delov. Vmesnik vsebuje specifikacijo za podprograme, strukture, razrede ali spremenljivke.

Poenostavljeno povedano je vmesnik uporabniškega programa povezava med dvema programoma. Omogoča, da dva programa komunicirata med seboj brez vednosti uporabnika ali njegovega posredovanja. Z njegovo uporabo lahko zelo pohitrimo razvoj programske rešitve, saj lahko namesto, da bi del sistema razvijali sami, uporabimo že obstoječi API. Slaba stran takega pristopa je, da postanemo odvisni od razvijalcev drugega programa oziroma njegovega vmesnika, s čimer se vse napake v tujem programu prenašajo na naše programje. Poleg tega sami ne moremo popravljati vmesnika, pač pa se moramo zanesti na njegovega avtorja oziroma izdajatelja, da bo prepoznane

težave odpravil v doglednem času.
API je lahko

- **jezikovno odvisen:** vmesnik lahko uporabljamo samo s sintakso in elementi specifičnega programskega jezika znotraj določenega operacijskega sistema (npr. vmesniki v okolju Windows – Kernel32.dll, User32.dll in kasneje opisani libgretl.dll); ali
- **jezikovno neodvisen:** vmesnik lahko uporabljamo iz več različnih programskih jezikov. To je zaželena funkcionalnost storitveno naravnanih vmesnikov, ki niso vezani na določen proces ali operacijski sistem in se lahko uporabljajo kot oddaljeni klici procedur ali spletnih storitev. Pogosto uporabljani primeri so Facebook API, Twitter API in Google Maps API.

3.1.1 Uporaba API

Preden izbrani API uporabimo, moramo ugotoviti, ali je jezikovno odvisen ali jezikovno neodvisen. Jezikovno odvisen API je v okolju Windows videti kot datoteke tipa `dll` (angl. dynamic-link library), a vse datoteke `dll` niso API-ji. Poleg tega je za izdelavo vmesnika uporabniškega programa potrebna posebna tehnologija programiranja. V okolju Windows obstaja poseben nabor vmesnikov uporabniškega programa, imenovan WinAPI. Preko tovrstnih vmesnikov lahko komuniciramo z operacijskim sistemom Windows.

Spodaj je preprost programček v programskem jeziku C# za okolje Windows, ki kliče Windows API User32.dll, in znotraj njega kliče metodo MessageBeep(). Vse, kar programček naredi, je, da vrne zvok za opozorilo:

```
// Primer preprostega klica piska sistema Windows
class Program
{
        [DllImport("User32.dll")]
        static extern Boolean MessageBeep(UInt32 beepType);
```

```
        static void Main(string[] args)
        {
                MessageBeep(0);
        }
}
```

Predstavitveni programček uporablja tehnologijo P/Invoke v povezavi s programskim jezikom C#. Ta tehnologija je podrobneje opisana v nadaljevanju. V splošnem se API razlikujejo med programskimi jeziki. Za primerjavo si oglejmo še isti programček v jeziku C++:

```
#include <Windows.h>
#include <stdio.h>

int main()
{
        MessageBeep(0);
}
```

Edina razlika je, da v primeru programskega C++ potrebujemo datoteko z "glavo"(angl. header), v kateri so že deklarirani klici, namesto da bi uporabili pristop P/Invoke.

3.1.2 Izdelava API

Ključni pojem za izdelavo API so že omenjene datoteke tipa `dll`. Gre za Microsoftovo implementacijo koncepta skupne knjižnice v okoljih Windows in OS/2. Skupna knjižnica je datoteka, ki je namenjena tako izvršilnim datotekam kot drugim skupnim knjižnicam ali objektom.

V nadaljevanju je opisan postopek izdelave preprostega vmesnika Win32 API v programskem jeziku C++, tj. izdelava vmesnika uporabniškega programa za 32-bitne sisteme Microsoft® Windows®. Najprej moramo deklarirati prevajalniku, da naj izdela Win32 API. To naredimo z dodajanjem naslednje izvorne kode v naš C++ projekt:

```
BOOL WINAPI DllMain (HINSTANCE, DWORD, LPVOID)
{
    return TRUE;
}
```

Potem lahko napišemo svojo funkcijo, katere funkcijski prototip (angl. function prototype) mora ustrezati naslednjem formatu:

```
extern "C" <tip> _declspec (dllexport) <function1>() { }
```

extern C: C++ prevajalniku deklariramo, da naj naslednjo funkcijo prevaja kot v funkcijo, napisano v jeziku C

tip: povemo, kakšen tip naj vrača funkcija; najpogostejši tipi v programskem jeziku C++ so int, double, void in bool

_declspec (dllexport): določa, da bo funkcija statična in vidna navzven;

function1: ime funkcije.

Tako lahko izdelamo funkcijo znotraj Win32 vmesnika uporabniškega programa, ki vrne seštevek dveh celoštevilskih parametrov:

```
extern "C" int _declspec (dllexport) sum(int var1,int var2)
{
        int total = var1+var2;
        return total;
}
```

3.1.3 gretl API – libgretl

Vmesnik uporabniškega programa gretl omogoča, da številne ekonometrične oziroma statistične funkcije, implementirane v okviru programskega paketa gretl, uporabimo izven gretla. gretlov vmesnik uporabniškega programa, ki je prav tako kot sam gretl napisan v programskem jeziku C, se imenuje libgretl. V tej nalogi je uporabljena verzija 1.9.6.

Vmesnik libgretl predstavlja skupni vhod za program ukazne vrstice gretlcli in grafični uporabniški vmesnik (GUI, angl. graphical user interface) programa gretl. Dokumentacija za libgretl API je javno dostopna na naslovu *http://gretl.sourceforge.net/API/new/gretl/index.html*. Čeprav je še nepopolna, nam omogoča osnovni vpogled v vmesnik. Nedokončana dokumentacija je sicer zelo oteževala razvoj naše rešitve, a projekt gretl je odprtokoden in s tem omogoča vpogled v izvorno kodo, zato lahko na podlagi izvorne kode bolje razumemo oziroma smiselno dopolnimo dokumentacijo.

Za uporabo libgretl API vmesnika je najprej (torej pred klici drugih funkcij) potrebno izvesti klic *libgretl_init()*, po koncu uporabe vmesnika libgretl pa je treba izvesti klic *libgretl_cleanup()*.

3.1.4 Excelov objektni model

Ko se odločimo za programsko rešitev, ki bo vključevala Microsoft® Excel, se moramo soočiti z Excelovim objektnim modelom (angl. Excel Object Model). Preko objektnega modela lahko nadzorujemo Excel iz drugega programa. Objektni model v splošnem predstavlja zbirko objektov in razredov, skozi katero lahko katerikoli program pregleduje in ureja specifične dele programa. V našem primeru predstavlja objektno orientiran vmesnik do Excela. Tako lahko z Excelovim objektnim modelom simuliramo Excelov uporabniški vmesnik. Z objektnim modelom dobimo dostop do Excelovih razredov, ki so spodaj našteti od najvišje ravni proti najnižji:

Excel.Application predstavlja celotno Excelovo aplikacijo. Vsebuje tako nastavitve in možnosti trenutno pognane aplikacije, kot tudi lastnosti objektov na najvišji ravni (kot sta ActiveCell in ActiveWindow).

Excel.WorkBook prestavlja posamezen delovni zvezek. Pripada zbirki Excel.WorkBooks, ki predstavlja vse odprte delovne zvezke znotraj Excelove aplikacije.

Excel.Worksheet predstavlja posamezen delovni list znotraj Excelovega delovnega zvezka.

Excel.Range prestavlja celico, vrstico, stolpec, izbrane celice ali trirazsežno območje.

Sledi primer uporabe Excelovega objektnega modela s programskim jezikom C#:

```
/// <summary>
/// Odpri novo Excelovo aplikacijo in dodaj delovni list
///  v delovni zvezek.
/// </summary>
static void ExcelInterakcija()
{
// naredi aplikacijo
Microsoft.Office.Interop.Excel.Application applikacija =
      new Microsoft.Office.Interop.Excel.Application();
applikacija.Visible = true;

// v Excelu naredi nov delovni zvezek
Microsoft.Office.Interop.Excel.Workbook delovniZvezek =
      applikacija.Workbooks.Add();

// pripravi nov delovni list
Microsoft.Office.Interop.Excel.Worksheet delovniList =
      new Microsoft.Office.Interop.Excel.Worksheet();

// izpisi stevilo delovnih listov -> v nasem primeru 3
Console.WriteLine("Stevilo delovnih listov: " +
      delovniZvezek.Worksheets.Count);

// dodaj delovni list
delovniList = delovniZvezek.Worksheets.Add();

// dodaj delovnem listu ime
```

```
delovniList.Name = "Dodani list";

// se enkrat izpisi stevilo delovnih listov
//          -> v nasem primeru 4
Console.WriteLine("Stevilo delovnih listov po dodanem
      enem listu" + delovniZvezek.Worksheets.Count);
}
```

3.2 Ovojnica

Včasih se pri razvoju kompleksnih programskih rešitev soočimo s problemom, da vmesnik uporabniškega programa ne zadošča za želeno funkcionalnost, ali pa je potreben klic funkcije vmesnika, ki ni mogoč v izbranem razvojnem okolju. V tem primeru nudi rešitev tehnologija, ki se imenuje *ovojnica* (angl. wrapper ali wrapper libraries – ovojne knjižnice).

Ovojnica s programerskega vidika predstavlja knjižnico s tanko plastjo programske rešitve, ki pretvori obstoječi knjižnični vmesnik v bolj ustrezen združljiv vmesnik. Kot smo omenili, knjižnica vsebuje zbirko podprogramov ali razredov, ki jih uporabljamo za razvijanje programske rešitve. Knjižnice prav tako vsebujejo vmesnike, preko katerih lahko uporabniki uporabljajo knjižnico za izvajanje podprogramov. Tako ravnamo iz različnih razlogov:

- da redefiniramo neustrezno arhitekturo knjižnice ali prezapleten vmesnik;
- da omogočimo delovanje skozi več različnih jezikov (v našem primeru, da lahko kličemo funkcije, napisane v programskem jeziku C, preko C++/CLI v programskem jeziku C#);
- da omogočimo skupno delovanje izvorne kode, ki sicer ne bi bilo mogoče (npr. zaradi nezdružljivih podatkovnih tipov).

3.3 Ogrodje .NET

Microsoft ogrodje .NET (angl. .NET Framework) je programsko ogrodje, ki primarno deluje na sistemih Windows®. Vključuje veliko knjižnico in podpira več programskih jezikov, kar omogoča interoperabilnost (angl. interoperability – vsak jezik lahko uporablja kodo, napisano v drugem jeziku; več o tem v nadaljevanju). *Ogrodje .NET* je zgrajeno tako, da ustreza naslednjim pogojem:

- zagotavlja dosledno objektno orientirano programsko okolje, kjer je objekt kode bodisi shranjen in izvršen lokalno, bodisi izvršen lokalno in distribuiran preko interneta, bodisi izvršen oddaljeno;
- zagotavlja okolje za izvajanje kode, ki minimizira čas razvoja programske kode in konflikte pri različicah programske kode;
- zagotavlja okolje, ki spodbuja varno izvajanje kode, vključno s kodo, ki jo ustvarijo neznani razvijalci;
- zagotavlja okolja za izvajanje kode, ki odpravlja probleme z učinkovitostjo skriptnih okolij;
- zagotavlja enako razvijalsko izkušnjo v širokem naboru različnih tipov aplikacij, kot so aplikacije v okolju Windows ali spletne aplikacije;
- omogoča komunikacijo, s katero se lahko aplikacija, izdelana v okolju .NET, integrira z aplikacijami, izdelanimi z drugimi programskimi jeziki.

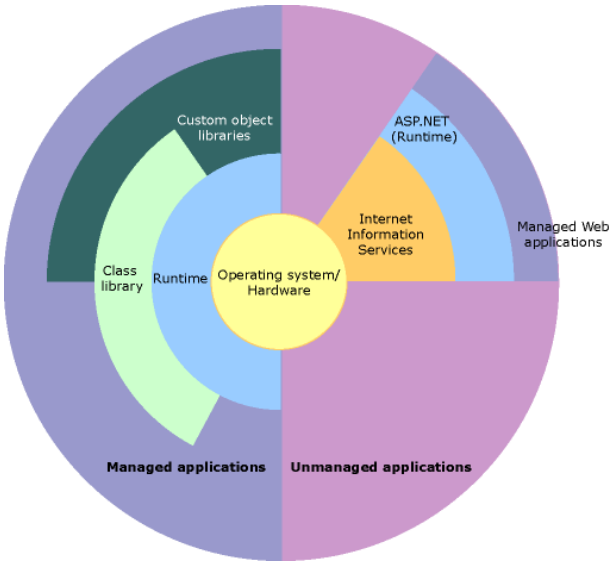
Okolje .NET ima dve glavni komponenti – skupni jezikovni izvajalnik (CLR – Common Language Runtime) in zbirko knjižnic razredov .NET Framework:

- **CLR** je osnova okolja .NET. Je del, ki skrbi za nadzorovano kodo (angl. managed code) v izvajalnem okolju, zagotavlja osnovne storitve (kot so

nadzor pomnilnika, nadzor niti in oddaljeni klici) in skrbi za točnost, varnost in robustnost izvorne kode. Koncept nadzora nad nadzorovano izvorno kodo je osnovni namen skupnega jezikovnega izvajalnika. Nenadzorovana koda (angl. unmanaged code) – kot pove že njeno ime – z vidika CLR deluje brez nadzora.

- **Zbirka knjižnic razredov .NET Framework** je celovita, objektno usmerjena zbirka ponovno uporabljivih (angl. reusable) tipov, ki se uporabljajo za razvijanje različnih aplikacij (od takih za tradicionalno ukazno vrstico do takih z grafičnim uporabniškim vmesnikom, ki temeljijo na novejših tehnologijah ASP.NET, kot so Web Forms ali XML Web Services).

Programsko ogrodje .NET v kontekstu različnih oblik in nivojev programske kode ponazarja slika 3.1.



Slika 3.1: *Zgradba programskega ogrodja .NET v kontekstu različnih oblik in nivojev programske kode (vir: http://msdn.microsoft.com/en-us/library/zw4w595w.aspx; za pojasnila glej besedilo)*

Zgodovino razvoja ogrodja .NET povzema preglednica 3.1, tehnologije, uporabljene znotraj posamezne verzije programskega ogrodja .NET, pa prikazuje slika 3.2.

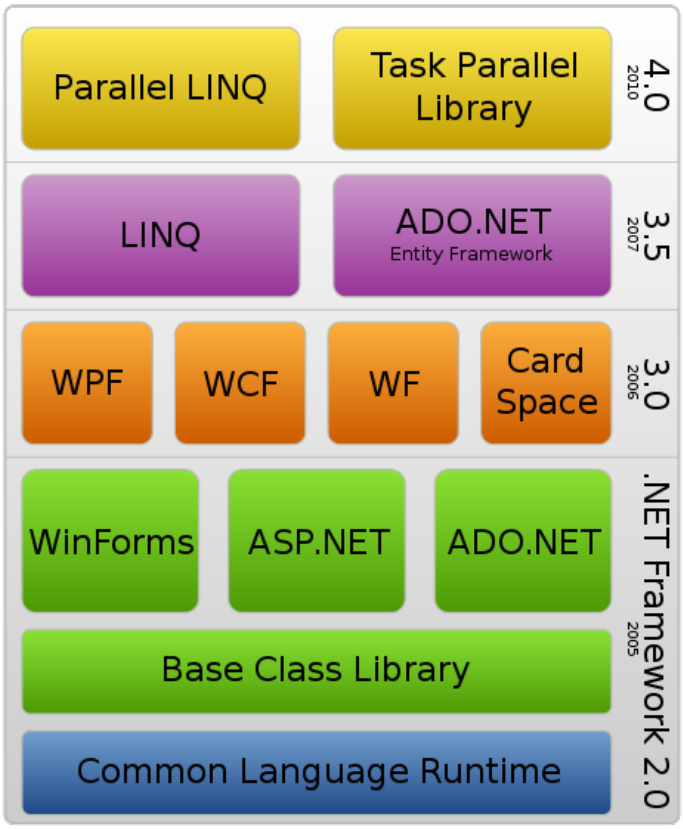
Verzija	Št. izdaje	Datum izdaje	Verzija orodja Visual Studio	Privzeta za različico okolja Windows
1.0	1.0.3705.0	13.02.2002	Visual Studio .NET 2002	Windows XP Tablet, Media Center Editions
1.1	1.1.4322.573	24.04.2003	Visual Studio .NET 2003	Windows Server 2003
2.0	2.0.50727.42	07.11.2005	Visual Studio 2005	Windows Server 2003 R2
3.0	3.0.4506.30	06.11.2006		Windows Vista, Windows Server 2008
3.5	3.5.21022.8	19.11.2007	Visual Studio 2008	Windows 7, Windows Server 2008 R2
4.0	4.0.30319.1	12.4.2010	Visual Studio 2010	Windows 7 (priporočeno)
4.5	4.5.40805			Windows 8, Windows Server 8

Tabela 3.1: *Zgodovina razvoja ogrodja .NET (vir: http://en.wikipedia.org/wiki/.NET_Framework)*

3.3.1 Nadzarovana in nenadzorovana koda (Managed vs. Unmanaged code)

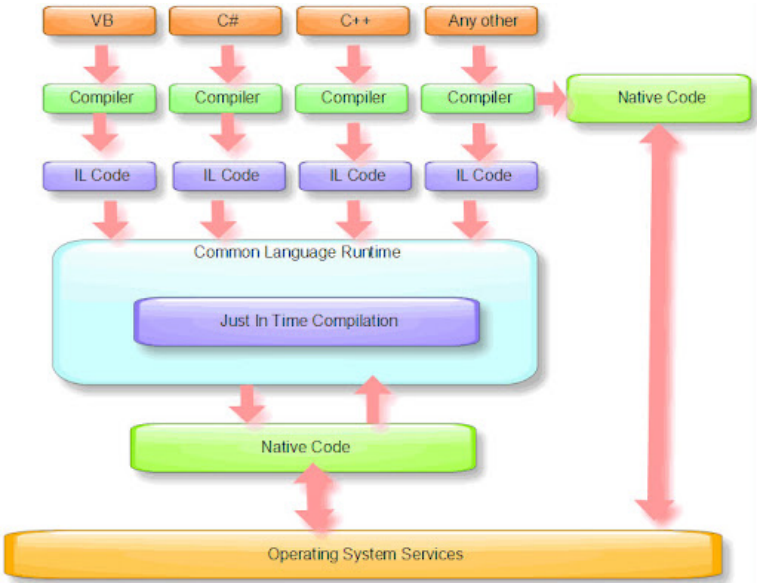
Nenadzorovana koda je Microsoftov izraz, ki označuje kodo pred izdajo Visual Studio .NET 2002. Visual Basic 6, Visual C++ 6 in tudi prevajalnik programskega jezika C proizvedejo *nenadzorovano kodo*. To so prevajalniki, ki prevajajo neposredno v strojno kodo, ki teče na strojni opremi, na kateri je bila koda prevedena, in vseh podobnih strojnih opremah enako, če imajo podobno arhitekturo. Podatke o varnosti in pomnilniku dobiva neposredno od operacijskega sistema – navadno preko vmesnika uporabniškega programa, v novejši obliki pa preko klicev vmesnika COM (ki je podrobneje predstavljen v nadaljevanju).

Nadzorovano kodo proizvedejo prevajalniki programskih jezikov Visual Basic .NET in C#. Prevajalniki pretvorijo kodo v vmesni jezik (IL, angl. Intermediate Language). IL je zapisan v datoteki, imenovani Assembly, skupaj z metapodatki, ki opisujejo razrede, metode in atribute pretvorjene iz-



Slika 3.2: *Tehnologije, uporabljene znotraj posamezne verzije programskega ogrodja .NET (vir: http://en.wikipedia.org/wiki/.NET_Framework; za pojasnila glej besedilo).*

vorne kode. Nadzorovana koda nato teče pod kontrolo skupnega jezikovnega izvajalnika (slika 3.3).



Slika 3.3: *Skupni jezikovni izvajalnik (CLR – Common Language Runtime; vir: http://www.smallworkarounds.net/2008/11/common-language-runtime-beginner-series.html; za pojasnila glej besedilo).*

3.3.2 CLI - Common Language Interface

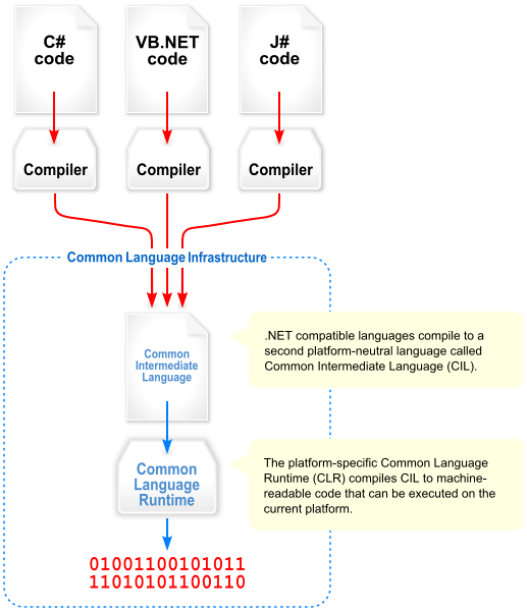
Skupna jezikovna infrastruktura (CLI, angl. Common Language Infrastructure) je odprta specifikacija, ki jo je razvil Microsoft, standardizirana po standardih ISO in ECMA. Definira izvajano kodo in izvajalno okolje, ki tvorita jedro Microsoftovega .NET Framework, pa tudi brezplačnih odprtokodnih implementacij Mono in Portable .NET. Specifikacija CLI definira okolje, ki omogoča večjemu številu višjih programskih jezikov, da jih uporabimo na različnih računalnikih, ne da bi bilo potrebno ponovno prevajanje kode za specifično računalniško arhitekturo. Delovanje CLI ponazarja slika 3.4.
Specifikacija CLI med drugim opisuje:

- The Common Type System (CTS – nabor podatkovnih tipov in operacije, ki so enaki za vse programske jezike, združljive s CTS);
- metapodatke (informacije o strukturi programa, da se lahko nanj skli-

cujemo skozi programske jezike in orodja, s čimer poenostavimo delo s kodo);

- skupne specifikacije jezika (CLS, angl. Common Language Specification – nabor osnovnih pravil, ki jih mora upoštevati vsak programski jezik znotraj CLI, če želi sodelovati z drugimi jeziki, združljivimi s CLS);

- virtualni sistem izvajanja (VES, angl. Virtual Execution System – omogoča združevanja ločeno ustvarjenih kosov kode v času izvajanja – naloži in izvaja programe, združljive s CLI, z uporabo metapodatkov).



Slika 3.4: *Skupna jezikovna infrastruktura (CLI – Common Language Infrastructure; vir: http://en.wikipedia.org/wiki/.NET_Framework; za pojasnila glej besedilo).*

3.3.3 CLR - Common Language Runtime

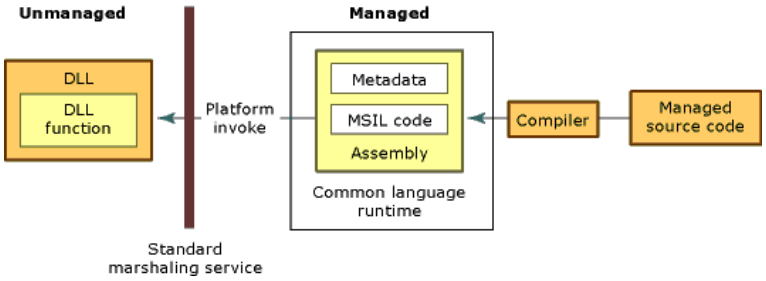
Skupni jezikovni izvajalnik (CLR, angl. Common Language Runtime) je komponenta navideznega stroja znotraj Microsoftovega ogrodja .NET, ki je odgovorna za izvajanje programov tipa .NET. V procesu just-in-time (JIT) skupni jezikovni izvajalnik prevede skupno vmesno jezikovno kodo (angl. Common Intermediate Language) v strojne ukaze, ki jih izvede procesor računalnika. Prevajalnik omogoča širok nabor programskih storitev za izvajano programsko kodo. CLR nato preveri, ali je datoteka z IL ustrezna. Potem poskrbi, da se datoteka Assembly uredi za strojno kodo za računalnik, na katerem se program poganja. Poleg tega se vsa nova strojna koda predpomni za naslednje klice metod. Ko se datoteka Assembly izvaja, se programu tudi posredujejo podatki o varnosti, nadzoru pomnilnika, nitenju ipd. - aplikacija je pri svojem izvajanju nadzorovana.

3.3.4 Interop

Interoperabilnost je po definiciji sposobnost različnih sistemov in organizacij za enotno delovanje v smeri dogovorjenih skupnih ciljev z izmenjavo informacij in znanja preko podprtih poslovnih procesov. Ožja definicija za področje informatike in računalništva pravi, da interoperabilnost pomeni zmožnost izmenjave podatkov med informacijskimi sistemi oziroma programskimi rešitvami. V našem primeru smo jo zagotovili z Microsoftovo tehnologijo Interop. Ta tehnologija nam v okviru Microsoftovega programja omogoča, da uporabljamo nenadzorovano kodo in posledično izrabljamo njene prednosti. Izvorno kodo, ki teče pod nadzorom skupnega jezikovnega izvajalnika, Microsoft imenuje nadzorovana koda.

P/Invoke

Spoznali smo, kako se koda izvaja v ogrodju .NET in kako izven njega. Pomembna pa je tudi tehnologija, ki nam omogoča klice iz ogrodja .NET v nativno okolje. Ta tehnologija se imenuje Platform Invocation Services,



Slika 3.5: *Delovanje tehnologije P/Invoke (vir: http://highoncoding.com/Articles/538_How_to_start_application_by_scheduled_time_with_Windows_mobile_.aspx; za pojasnila glej besedilo).*

skrajšano P/Invoke. P/Invoke omogoča, da nadzorovana koda kliče nenadzorovano funkcijo, ki je implementirana v datoteki tipa dll.

P/Invoke se zanaša na metapodatke, da locirajo zunanje funkcije, in razporeja njihove argumente v času izvajanja. K pojmu razporejanja (angl. marshalling) se bomo vrnili kasneje.
Ko P/invoke kliče nativno funkcijo, deluje po naslednjih korakih (slika 3.5):

- locira datoteko tipa `dll`, ki vsebuje funkcijo;
- naloži datoteko tipa `dll` v pomnilnik;
- locira naslov funkcije v pomnilniku, ga potisne na sklad in izvede razporejanje;
- izvede nenadzorovano funkcijo.

P/Invoke nam tudi vrača izjeme, ki so se zgodile v nenadzorovanem okolju, v nadzorovano. Spodaj je preprost primer uporabe tehnologije P/Invoke. Uporablja datoteko msvcrt.dll in znotraj nje vrača *putc()* in *_flushall()*:

```
// PInvokeTest.cs
using System;
using System.Runtime.InteropServices;

class PlatformInvokeTest
{
    [DllImport("msvcrt.dll")]
    public static extern int puts(string c);
    [DllImport("msvcrt.dll")]
    internal static extern int _flushall();

    public static void Main()
    {
        puts("Test");
        _flushall();
    }
}
```

Nevarni kontekst

Programski jeziki znotraj nadzorovane programske kode so precej različni od tistih v nenadzorovani kodi, kakršna sta C in nativni C++. Ena od teh razlik je opustitev kazalcev kot podatkovnih tipov. Namesto kazalcev je ponujena rešitev preko sklicev in sposobnosti, da ustvarimo predmete, ki jih upravlja zbiralec smeti (angl. garbage collector). Ta razlika skupaj s kopico drugih značilnosti naredi te programske jezike varnejše za razvijanje. Tako se izognemo celi vrsti "nevidnih"hroščev, ki nam često otežujejo razvoj v programskem jeziku C ali nativnem C++. Vendar je v določenih primerih, kot npr. pri interakciji z operacijskim sistemom, nujno uporabiti kazalce. V tem primeru deklariramo izvorno kodo kot nevarno (angl. unsafe).

Besedna "unsafe"je v programskih jezikih .NET rezervirana beseda in z njo deklariramo nevarno kodo (tj. kodo v t.i. nevarnem kontekstu – angl.

unsafe context). Sledi primer uporabe nevarne kode v programskem jeziku C#:

```
// cs_unsafe_keyword.cs
// compile with: /unsafe
using System;
class UnsafeTest {
   // unsafe method: takes pointer to int:
   unsafe static void SquarePtrParam (int* p)    {
      *p *= *p;
   }
   unsafe public static void Main()    {
      int i = 5;
      // unsafe method: uses address-of operator (&)
      SquarePtrParam (&i);
      Console.WriteLine (i);
   }
}
```

Razporejanje

Razporejanje (angl. marshalling) je postopek pretvorbe predmeta, zapisanega v pomnilniku, v podatkovni tip, primeren za hrambo ali prenos. Navadno ga uporabimo, kadar moramo prenesti podatke med različnimi deli programa ali iz enega programa v drugega. V osnovi je razporejanje implementacija oddaljenega klica (angl. Remote Procedure Call – RPC), ki je pomemben za prenos podatkov med procesi ali nitmi (angl. threads).

V Microsoftovem vmesniku COM morajo biti kazalici razporejeni, ko prehajajo meje sveta COM. V ogrodju .NET je pretvorba med nenadzorovanim in nadzorovanim tipom, kot na primer v klicih P/Invoke, primer uporabe razporejanja. Spodnji primer kaže, kako z razporejanjem prenesemo vrednosti iz IntPtr, ki je reprezentacija kazalca, v niz:

```
Intptr msgNative = NativeGretl.Methods.
        gretl_print_get_buffer(pointerPrn);
string message = "Error: " +
        Marshal.PtrToStringAnsi(msgNative);
```

Nekoliko bolj zapleten je primer, kako pretvoriti kazalec v seznam nizov:

```
/// <summary>
/// Metoda za pretvarjanje IntPtr v string.
/// </summary>
/// <param name="ptr">Kazalec na string.</param>
/// <param name="size">Velikost polja.</param>
/// <returns>Zbirka stringov.</returns>
public static List<string> MarshalIntPtrToStringList
                                (IntPtr ptr, int size)
{
    var list = new List<string>();
    for (int i = 0; i < size; i++)
    {
        var strPtr = (IntPtr)Marshal.
                    PtrToStructure(ptr, typeof(IntPtr));
        list.Add((Marshal.PtrToStringAnsi(strPtr)));
        ptr = new IntPtr(ptr.ToInt64() + IntPtr.Size);
     }
     return list;
}
```

Bločno prenosljivi objekti

Večina programskih tipov ima svojo reprezentacijo tako v nadzorovanem kot v nenadzorovanem okolju. Ti tipi ne potrebujejo posebnega obvladovanja, pretvorbe ali razporejanja. Znani so pod imenom bločno prenosljivi objekti (angl. blittable objects).

Vsi tipi in strukture, ki se vračajo z uporabno P/Invoke, morajo biti bločno prenosljivi. Bločno prenosljivi tipi so (*vir: http://msdn.microsoft.com/en-us/library/75dwhxf7.aspx*):

- System.Byte,
- System.SByte,
- System.Int16,
- System.UInt16,
- System.Int32,
- System.UInt32,
- System.Int64,
- System.UInt64,
- System.IntPtr,
- System.UIntPtr,
- System.Single,
- System.Double,
- enorazsežne tabele katerega od zgoraj navedenih tipov,
- formatirani vrednosti tipi, ki vsebuje samo bločno prenesljive objekte.

Naslednji tipi niso bločno prenesljivi (*vir: http://msdn.microsoft.com/en-us/library/75dwhxf7.aspx*):

- System.Array (pretvori se v C-jevsko polje ali SafeArray),
- System.Boolean (pretvori se v 1, 2, ali 4-bitno vrednost s true kot 1 ali -1),

- System.Char (pretvori se v Unicode ali ANSI znak),
- System.Class (pretvori se v vmesnik razreda),
- System.Object (pretvori se v variant ali vmesnik),
- System.Mdarray (pretvori se v C-jevsko polje ali SafeArray),
- System.String (pretvori se v string, ki se konča z ničelnim znakom ali BSTR),
- System.Valuetype (pretvori se v strukturo s fiksno pomnilniško zgradbo),
- System.Szarray (pretvori se v C-jevsko polje ali SafeArray).

Vmesnik COM

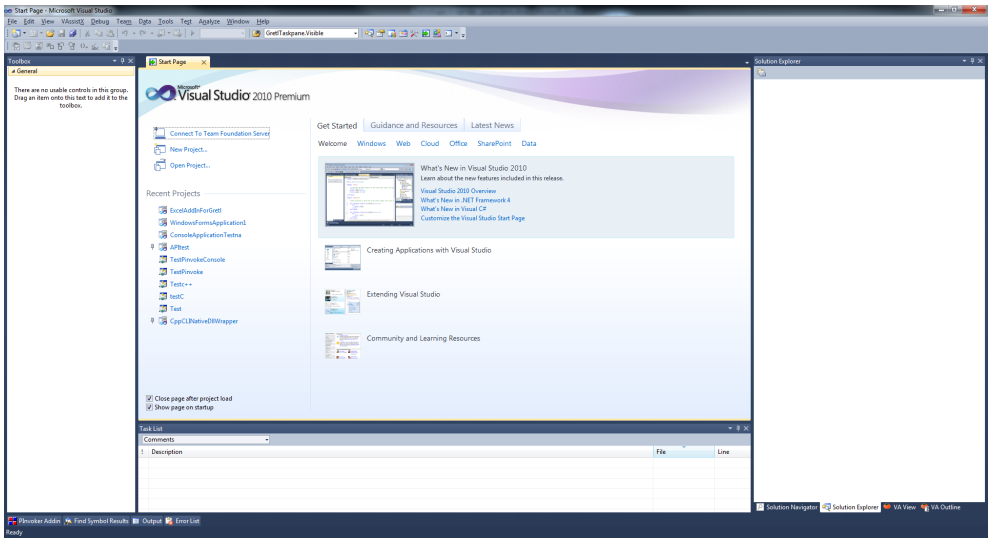
Component Object Model (COM) je binarni vmesniški standard za delo s komponentami programja. COM omogoča medprocesno komunikacijo in možnost dinamičnega ustvarjanja objektov. To tehnologijo uporabljajo razvijalci, da ustvarijo ponovno uporabljive komponente programske rešitve, povežejo te komponente med seboj in izrabljajo prednosti storitev Windows. COM se pogosto uporablja v Microsoftovih izdelkih za razvoj programske opreme kot krovni pojem, ki zajema tehnologije, kot so OLE, OLE Automation, ActiveX kontrolniki, COM + in Distributed COM (DCOM).

Microsoft ponuja vmesnike COM za veliko svoje programske opreme, npr. Direct Show, Windows Animation Manager, Windows Potrable Devices in Microsoft Active Directory (AD). Vmesnik COM je uporabljen tudi v aplikacijah, ki tvorijo pisarniški programski paket Windows Office. Tako npr. tehnologija COM/OLE omogoča Wordovim dokumentom, da dinamično povežejo podatke z Excelovimi delovnimi listi, COM Automation pa dovoli uporabnikom, da napišejo skripte, ki upravljajo eno aplikacijo iz druge.

3.4 Razvijalska orodja in pomagala

3.4.1 Micorsoft Visual Studio

Microsoft Visual Studio je integrirano razvojno okolje (IDE, angl. integrated development environment). Uporabljeno je za razvoj konzolnih aplikacij, aplikacij grafičnega uporabniškega vmesnika skupaj z aplikacijami tipa Windows Forms, spletnih strani, spletnih aplikacij in spletnih storitev tako v nadzorovani kot v nenadzorovani kodi na platformah Microsoft Windows, Windows Mobile, Windows CE, ogrodju .NET in Microsoft Silverlight.



Slika 3.6: *Začetno pozdravno okno razvojnega okolja Microsoft Visual Studio 2010, v katerem smo razvili rešitev.*

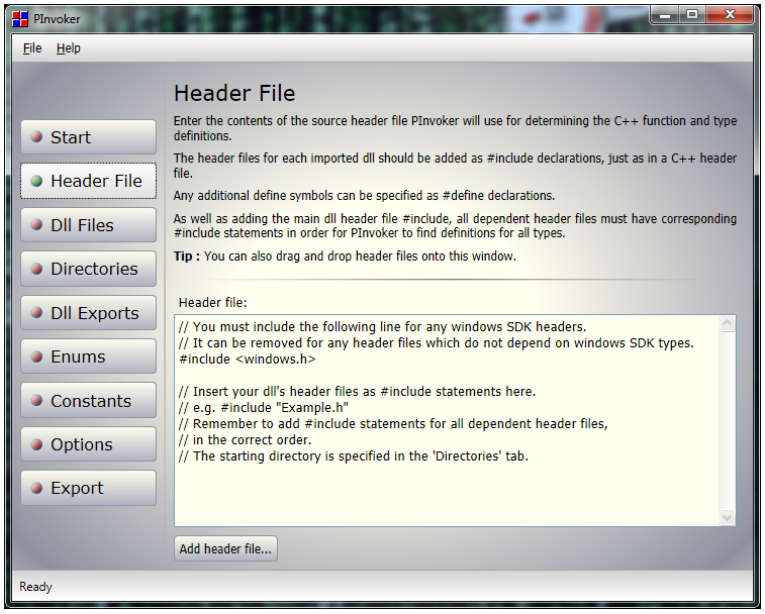
Visual Studio vključuje urejevalnik izvorne kode, podprt s tehnologijo IntelliSense in tehnologijo refaktorizacije kode (angl. code refactoring). Ostala vgrajena orodja so oblikovalec obrazcev(angl. Form Designer) za izdelavo aplikacij grafičnega uporabniškega vmesnika, oblikovalec spleta(angl. Web Designer), oblikovalec razredov (angl. Class Designer) in oblikovalec sheme podatkovne zbirke (angl. Database Schema Designer). V razvojno okolje lahko vključimo tudi vtičnike, ki lahko še izboljšajo funkcionalnost na vsaki

stopnji.

Visual Studio podpira več različnih programskih jezikov v smislu storitev programskih jezikov, ki omogočajo, da urejevalnik izvorne kode in razhroščevalnik podpreta katerikoli programski jezik, pod pogojem, da za ta programski jezik obstaja storitev jezikovne podpore. Vgrajeni programski jeziki so C/C++, VB.NET, C# in F#.

3.4.2 Pinvoker

Po definicij je Pinvoker orodje, s katerim lahko generiramo datoteke tipa `dll`, ki temeljijo na tehnologiji P/Invoke, z uporabo .NET Interop. Za izdelavo datoteke tipa `dll` potrebujemo statične datoteke tipa `dll` in datoteke z "glavo", vse pa morajo biti pisane v programskem jeziku C++. Pinvoker uvaža funkcije in spremenljivke, izvožene iz datoteke tipa `dll`, najde njihove definicije v datotekah z "glavo"in nato le-te izvozi v datoteke tipa `dll`, ki so znotraj ogrodja .NET.

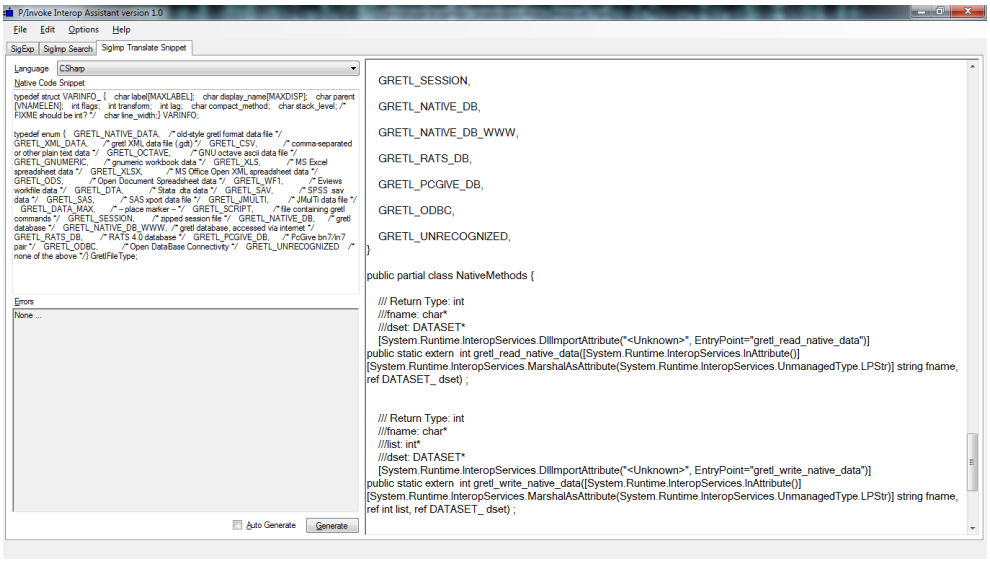


Slika 3.7: *Grafični uporabniški vmesnik orodja Pinvoker.*

PInvoker nam zgradi datoteke tipa `dll`, preko katerih lahko kličemo funkcije v izbranih datotekah tipa `dll`, pod pogojem, da so le-te pisane v programskem jeziku C++. Vse kar potrebujemo, so datoteke tipa `dll` in njihove ustrezne datoteke tipa header. Rezultat so datoteke tipa `dll`, ki so vidne znotraj ogrodja .NET.

3.4.3 P/Invoke Interop Assistant

Pri razvoju programske rešitve smo našli orodje, s katerim lahko generiramo podpise metod, ki jih lahko nato kličemo s tehnologijo P/Invoke. Orodje je brezplačno in generira kodo tako v programskem jeziku Visual Basic kot tudi v C#. Orodje ima vgrajeno tudi preverjanje napak.



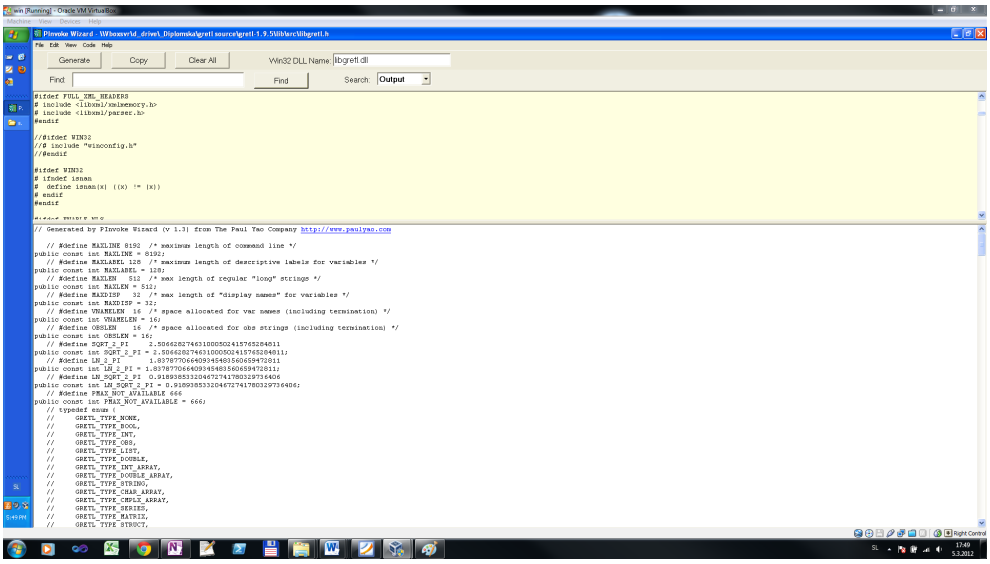
Slika 3.8: *Grafični uporabniški vmesnik orodja Pinvoker.*

Orodje deluje tako, da najprej zgoraj izberemo programski jezik, za katerega hočemo generirati podpise (v našem primeru je to programski jezik C#). Pod to izbirno polje vstavimo izvorno kodo, nad katero hočemo klicati posamezne klice. Tukaj moramo vključiti tudi vse strukture in deklaracije ostalih spremenljivk, če hočemo, da se zgradi pravilni podpis metod. Na to

nas prav tako opozori ogrodje. Na desni je polje, v katerem se zgradi izvorna koda, preko katere nato lahko kličemo metode z uporabo tehnologije P/Invoke.

3.4.4 Pinvoke Wizard

Še eno koristno orodje za generiranje izvorne kode je P/Invoke Wizard. P/Invoke Wizard je plačljivo orodje za generiranje podpisov funkcij, ki jih kličemo s tehnologijo P/invoke. Orodje v času našega dela ni bilo dosegljivo v razvojnem operacijskem okolju Window 7, kar je zahtevalo postavitev virtualnega okolja Windows XP.

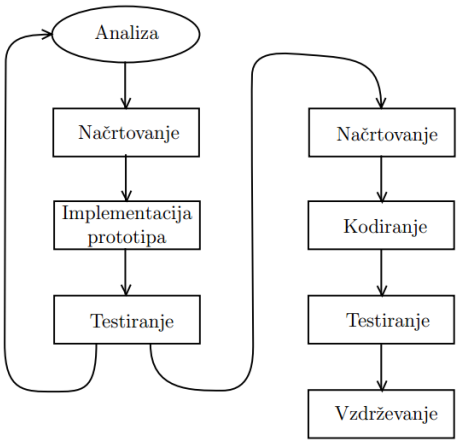


Slika 3.9: *Zaslonska slika orodja Pinvoke Wizard, pognanega znotraj virtualnega okolja Windows XP preko orodja VirtualBox.*

Poglavje 4

Razvoj rešitve

Dandanes poznamo mnogo metodologij razvoja programske opreme (angl. software development methodologies). Pod tem pojmom razumemo ogrodje, ki ga uporabimo za izdelavo strukture, načrtovanje in nadzorovanje procesa razvoja programske rešitve. Odločili smo se za razvoj z uporabo prototipov (slika 4.1) [4]. Za ta postopek smo se odločili, ker nismo mogli vnaprej popolnoma definirati zahteve oziroma problemske domene.



Slika 4.1: *Razvoj programske opreme z uporabo prototipov (vir: F.Solina, Projektno vodenje razvoja programske opreme; dosegljivo na http://eprints.fri.uni-lj.si/142/).*

Za vsako dobro programsko rešitev je potrebno, da je ta narejena čim bolj enostavno in čim bolj robustno. Kot pri vsaki programski rešitvi, je v prvem koraku najprej potrebno preučiti, kakšen bo postopek razvoja rešitve. Pri tem predstavlja nezadostno poznavanja problemske domene visoko oviro. Vprašati smo se morali:

- kako bomo izvedli povezavo med Excelom in gretlom,
- kakšen naj bi bil končni izgled povezave in
- kako bomo preverili končno delovanje?

Če zgornje korake prevedemo v programsko metodologijo, so postopek razvoja sestavljale:

- analiza problemske domene;
- načrtovanje programske rešitve;
- kodiranje in implementacija programske rešitve;
- preizkušanje.

4.1 Začetna iteracija

4.1.1 Analiza problemske domene

Eden prvih korakov je bila analiza problemske domene. V analizi poskušamo čim bolje razumeti problemsko domeno, da bi čim bolj popolno razumeli naloge in lastnosti programske rešitve. Ker je v našem primeru problemska domena povezava Excela s paketom gretl, je bil naš prvi korak namestitev teh aplikacij v razvojno okolje.

Pri Excelu smo hitro naleteli na njegov objektni model, ki drugim programom omogoča razmeroma preprosto povezavo z Excelom. Največ rešitev v povezavi z Excelom se rešuje preko danega objektnega modela, kar nam je

poenostavilo iskanje testnih primerov in pomoči na to temo. Več o Excelu in njegovem objektnem modelu je že opisano v prejšnjih poglavjih.

Na drugi strani smo imeli odprtokodni paket gretl. Hiter pregled dokumentacije nas je usmeril v uporabo vmesnika libgretl. Čeprav je dokumentacija in specifikacija vmesnika (še) okrnjena, smo razmeroma hitro dobili idejo in odgovore, kako rešiti problemsko domeno. Poleg tega obstaja tudi poštni seznam (angl. mailing list) oziroma novična skupina (angl. newsgroup), kjer je moč hitro priti do strokovnih odgovorov. Več o vmesnikih uporabniškega programa in libgretlu je prav tako že napisano v prejšnjih poglavjih.

Oba programska paketa torej vsebujeta vmesnik, preko katerega lahko dostopamo do funkcij aplikacije. Možne rešitve so bile:

- **Neposredno kopiranje kode**
  Ker je gretl odprtokodni paket, bi lahko funkcije, ki bi jih potrebovali, samo prepisali v lastno programsko rešitev. A postopek je zamuden, iz pravnega in moralnega vidika vprašljiv in vprašljiva bi bila tudi točnost in optimalnost prepisane rešitve, še zlasti, ker sodi statistična programska oprema med algoritmično in numerično najzahtevnejše programje, ki sploh obstaja.

- **Preko vmesnika COM**
  Vzeli bi obstoječi gretl paket in si naredili novo vejo za razvijanje izvorne kode. Ta koda bi vsebovala vmesnike COM, preko katerih bi dostopali do paketa gretl. Kot prednost bi dobili možnost, da katerokoli funkcionalnost, ki obstaja v paketu gretl, uporabimo za svoj namen. Tudi ta interakcija bi bila enostavna, toda ta postopek zahteva, da vse popravke in dodane funkcionalnosti, ki se naredijo oziroma uvedejo v primarnem razvoju gretla, sami implementiramo v našo rešitev. S tega vidika bi bilo potrebno vložiti veliko truda v vzdrževanje programske rešitve.

- **Preko vmesnika uporabniškega programa**
  Preko vmesnika uporabniškega programa lahko dostopamo samo do

funkcionalnosti, ki nam jih le-ta izstavi. Na ta način smo lahko omejeni glede razvoja in izrabe funkcionalnosti, po drugi strani pa zmanjšamo breme oziroma stroške vzdrževanja. Slednji so minimalni, dokler ne pride do velikih sprememb pri vmesniku uporabniškega programa.

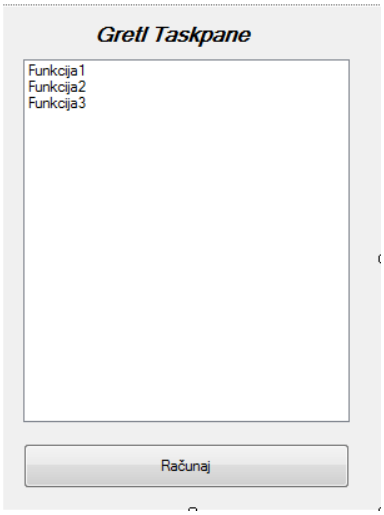
Hkrati smo ugotovili, da bi najhitreje in najlažje rešili povezavo z Excelom preko dodatka (angl. add-in). Odločili smo se za razvojno okolje Microsoft Visual Studio 2010, saj nam Microsoft s svojim razvojnim okoljem omogoča najvišjo možno podprtost za izdelavo dodatka za Excel.

4.1.2 Načrtovanje rešitve problema

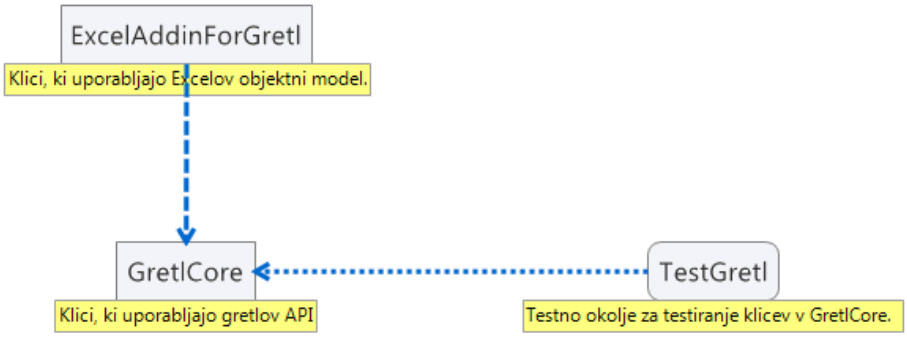
Pri načrtovanju smo se odločili za rešitev preko vmesnika uporabniškega programa. Ta rešitev nam omogoča, da se lahko osredotočimo na uporabo funkcionalnosti za uporabo ekonometrije oziroma statistike in vsaj na nivoju izdelave prototipa ne zahteva od nas poglobljenega matematičnega znanja. Poleg tega nam omogoča uporabo zadnje verzije paketa gretl brez večjih posegov v izvorno kodo rešitve, s čimer mnogo lažje sledimo razvoju paketa gretl.

Povezavo z Excelom najlažje izvedemo preko dodatka za Excel. Načrtovan je bil dodatek za Microsoft Excel 2007 (Excel 12.0) in novejši, torej za Excel, ki vsebuje trak (angl. ribbon). Dodatek naj bi vseboval podokno opravil, skozi katerega bi lahko klicali funkcije gretla. Videz začetnega grafičnega uporabniškega vmesnika prikazuje slika 4.2.

Nato smo kmalu ugotovili, da bi bilo smiselno klice tudi predhodno testirati. Prav tako smo bili mnenja, da bi se rešitev lahko ločila na dva dela: v enem delu kličemo gretlove funkcije, v drugem pa jih integriramo v Excel. Tako smo prišli do programske arhitekture na sliki 4.3.



Slika 4.2: *Prototip podokna opravil za klic funkcij gretla.*



Slika 4.3: *Arhitektura začetne programske kode.*

4.1.3 Kodiranje in implementacija programske rešitve

V iteraciji kodiranja in implementacije programske rešitve smo naredili projekt z dogovorjeno arhitekturo in poizkušali implementirati dogovorjene funkcije. Uspešno smo implementirali funkciji *gretl_read_native_data()*, ki prebere

datoteko tipa `.gdt`, in *gretl_write_native_native_data()*, ki zapiše datoteko tipa `.gdt` na lokalni podatkovni disk. Obe funkcije smo implementirali v knjižnico `GretlCore.dll`. Ti funkciji smo tudi preverili preko programa ukazne vrstice `TestGretl`. Nato smo prenesli te funkcionalnosti v dodatek za Excel. Takoj se je pojavila potreba po notranjih razredih, preko katerih bi prenašali vrednosti, namesto da uporabimo že definirane v gretlu. V veliko pomoč nam je bilo tudi Excelovo notranje programsko okolje `VBA` (Visual Basic for Application), s katerim smo reševali probleme, kot so zapis vrednosti v celico in branje vrednosti iz celic.

4.1.4 Testiranje delovanja

V testni fazi smo preizkusili delovanje branja in zapisovanja datotek tipa `.gdt` v programu ukazne vrstice `TestGretl`. Z manjšimi popravki nam je uspelo rešiti branje in zapisovanje tudi preko dodatka. Med testiranjem smo prišli do zamisli, da bi namesto, da uporabljamo shranjevanje in odpiranje preko podokna opravil, uporabili gumbe na traku Excela. Tako bi naredili rešitev bolj prijazno uporabniku, ker sledi privzeti obliki Excela. Poleg tega smo ugotovili, da bi bila rešitev bolj robustna, če bi lahko dinamično naslavljali pot, kjer se nahaja vmesnik uporabniškega programa gretl, kot pa če je ta statično zapisana v izvorni kodi.

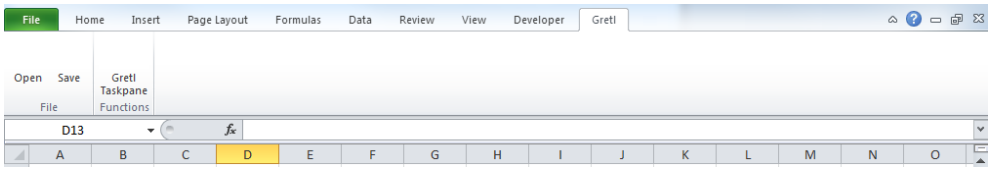
4.2 Druga iteracija

4.2.1 Načrtovanje rešitve problema

Kot smo ugotovili v testiranju v predhodni iteraciji, je bilo potrebno rešiti naslednje probleme:

- premakniti shranjevanje in odpiranje iz podokna opravil;
- rešiti naslavljanje dinamične poti gretlovega vmesnika;
- rešiti naslavljanje dinamične poti gretlovega vmesnika;

Načrtovali smo, da bi na trak dodali gumbe Shrani, Zapiši in gumb za odpiranje in zapiranje podokna opravil za gretl (slika 4.4).



Slika 4.4: *Prilagojeni Excelov trak v 2. iteraciji.*

Odločili smo se, da implementiramo naslednje funkcije za opisno statistiko:

gretl Min vrne minimum v naboru števil

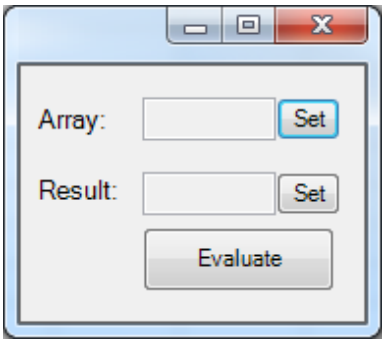
gretl Max vrne maksimum v naboru števil

gretl Sum vrne seštevek nabora števil

gretl Mean vrne aritmetično sredino nabora števil

Ker vsaka od teh funkcij zahteva nabor števil, smo se odločili, da izdelamo okno, na podlagi katerega lahko nastavljamo območje nabora vrednosti in kam bomo zapisali rezultat (slika 4.5).

Sredi predhodne iteracije smo zasledil, da sta bila izdani novi verziji gretla in vmesnika uporabniškega programa libgretl – 1.9.6 namesto 1.9.5. Zato smo se odločili, da preidemo na novo verzijo.



Slika 4.5: *Okno za računanje opisne statistike v 2 iteraciji.*

4.2.2 Kodiranje in implementacija programske rešitve

Najprej smo oblikovali trak, na katerega smo nato vezali nove funkcionalnosti. Funkcije Shrani in Zapiši smo enostavno prestavili na ta prirejeni trak. Naredili smo tudi, da se podokno opravil na začetku sicer inicializira, vendar se skrije in se mu potem nastavlja vidljivost glede na gumb Gretl Taskpane. S tem rešimo problem, da se nam ne pojavi več instanc podokna opravil.

Načrtovali smo tudi implementacijo nekaj funkcionalnosti opisne statistike. Implementacija te rešitve je bila razmeroma enostavna, ker smo že v predhodni iteraciji dobro implementirali notranje razrede, preko katerih smo potem te vrednosti prenašali iz Excela preko dodatka za Excel ExcelAddinForGretl v knjižnico `GretlCore`, ki jo je kasneje klical vmesnik uporabniškega programa gretl.

Implementacija dinamičnega naslavljanja poti do gretla pa je predstavljala problem. Metoda P/Invoke zahteva statično konstanto za določanje poti do vmesnika, preko katerega izvajamo klice. Rešitev, ki bi zahtevala, da se za vsako različno namestitev paketa gretl naredi novo izdajo rešitve, se nam je zdela zelo slaba in potratna. Problem je bil tudi, kako dobiti datotečno pot do paketa gretl.

Problem s potjo do vmesnika smo hitro rešili. V registru sistemov Windows si gretl naredi zapis do izvajalne datoteke. Ta zapis je v ključu *SOFTWARE\gretl\gretlw32.exe*. Če ga nekoliko predelamo, dobimo pot do vmesnika. Problem dinamične poti do vmesnika uporabniškega programa gretl smo nato rešili z klicanjem drugega vmesnika uporabniškega programa. Uporabili smo Kernel32.dll, ki je del vseh operacijskih sistemov Windows. Na podlagi tega vmesnika smo dinamično naložili knjižnico, ki smo jo potem lahko statično klicali.

Tudi prehod na novo verzijo vmesnika uporabniškega programa (iz verzije 1.9.5 na verzijo 1.9.6) je potekal brez večjih problemov.

4.2.3 Testiranje delovanja

V tej fazi je bilo potrebnega nekoliko več testiranja – potrebno je bilo preveriti:

- ali vse deluje po prehodu na novo verzijo vmesnika uporabniškega programa,
- preverjanje po meri izdelanega traku,
- delovanje podokna opravil,
- delovanje funkcij opisne statistike,
- dinamično nastavljanje poti do paketa gretl.

Toda večji popravki niso bili potrebni, saj smo veliko testov naredili že v fazi kodiranja in implementiranja. Veliko testov je bilo predhodno narejenih tudi iz testnega programa ukazne vrstice TestGretl in tu smo jih le ponovili.

4.3 Tretja iteracija

4.3.1 Načrtovanje rešitve problema

V tretji iteraciji smo si zadali cilj, da implementiramo analitično zmožnost, ki je Excel nima. Vse funkcije opisne statistike, ki smo jih implementirali v drugi iteracij, so namreč že vključene v Excel. Tako smo se odločili, da bi implementirali funkcionalnost iz ocen regresijskih modelov. Odločili smo se za oceno linearnega modela po metodi najmanjših absolutnih odstopanj (funkcija lad, angl. least absolute deviation). Tako oceno v statistiki imenujejo robustno, saj je v primerjavi z običajno linearno regresijo (tj. po metodi najmanjših kvadratov odstopanj, angl. least squares, ki je v Excelu že implementirana s funkcijo LINEST, v dodatku za analizo in v raztresnem diagramu kot trendna črta) manj občutljiva na odstopajoče vrednosti (osamelce, angl. outliers).

Poleg tega smo si zadali za cilj, da naredimo program lokaliziran, se pravi, da je lahko preveden v katerikoli jezik. Z tem bi izboljšali uporabniško izkušnjo in program približali potencialnim uporabnikom.

4.3.2 Kodiranje in implementacija programske rešitve

Lokalizacija povezave je razmeroma enostavna. Vse kar je potrebno je, da vse besedila, ki se pojavljajo v programu, zapišemo v eno datoteko in jo vključimo v vire projekta. Ta ena datoteka predstavlja en jezik. Če hočemo implementirati podporo za nadaljnji jezik, samo naredimo dvojnik te datoteke in prevedemo vsa besedila.

Implementacija regresijskega modela pa je bila ena od glavnih težav v razvoju projekta in je tudi vzela največ časa. Tu smo najbolj občutili negativno stran okrnjene dokumentacije in slabo definirane arhitekture vmesnika uporabniškega program. Potrebno je bila implementacija strukture MODEL, s katero bi klicali funkcijo za oceno po metodi najmanjših absolutnih odstopanj.

Po večtedenski analizi in raziskovanju, poizvedovanjih pri znancih v programerskih podjetjih, iskanju rešitev preko spletnega portala Social MSDN (ki je posvečenemu razvijalcem v Microsoftovih tehnologijah) in vprašanjih preko poštne liste in novične skupine razvijalcev paketa gretl smo končno prišli do rešitve. Potrebno je bila implementacija strukture MODEL preko uporabe nove tehnologije v povezavi s programskim jezikom C++ CLI. V naši dosedanji tehnologij P/Invoke in programskem jeziku C# to ni bilo mogoče. Po ducat napisanih programskih primerih smo vendarle prišli do ustrezne rešitve. Morali smo implementirati novo knjižnico v projekt. Knjižnica se imenuje `GretlSubCore` in se obnaša kot ovojnica (wrapper) za knjižnico `GretlCore`. Pojem ovojnice je razložen v predhodnih poglavjih.

S to knjižnico smo na podlagi izvorne kode le uspeli rešili problem slabo definirane arhitekture vmesnika uporabniškega programa. Knjižnica kliče ustrezne funkcije vmesnika uporabniškega programa in nam preko notranjih razredov prenese ustrezne objekte. V tem koraku je bila tudi potrebna izde-

lava notranjega razreda ModelClass, ki služi za prenos vrednosti.

4.3.3 Testiranje delovanja

V tej iteraciji testiranja je bilo potrebno preveriti

- ali lokalizacija deluje na več različnih okoljih in
- ali se ovojnica ustrezno obnese za računanje ocen regresijskih modelov.

Napisano je bilo nekaj testov v programu ukazne vrstice TestGretl. Rezultati smo preverili tudi preko izvedljive datoteke gretl.

Z lokalizacijo pa smo v testnem koraku imeli kar nekaj problemov. Ugotovili smo, da se lokalizacija nastavi na jezik operacijskega sistema. Tako se v primeru, da ima uporabnik Excel v slovenščini in operacijski sistem Windows v angleščini, izbere angleška lokalizacija. Problem smo lahko hitro rešili z uporabo nastavljanja jezika Excela, a je še vedno je prihajalo do napake na traku. Lokalizacija traku se namreč nastavi še preden lahko nastavimo jezik za lokalizacijo. Lokalizacijo za trak smo zato rešili tako, da pustimo, da se trak postavi, in šele nato "povozimo"besede za gumbe Traka.

4.4 Četrta iteracija

4.4.1 Načrtovanje rešitve problema

V četrti iteraciji je bil poudarek izboljšanju in optimizaciji že implementiranih funkcij. Načrtovali smo:

- implementacijo okna za pomoč;
- nastaviti vsa okna tako, da so podrejena aplikaciji Excel;
- popravek okna za opisno statistiko;
- dokončati slovensko lokalizacijo in
- popravek širine podokna opravil za gretlove funkcije.

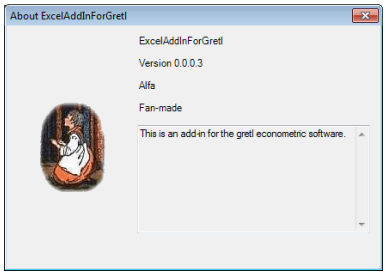
Okno za pomoč smo se odločili spremenili v okno "O programu". Tukaj bi uporabnik dobil osnovne informacije o programu. Znotraj tega okna bi bila tudi povezava na pomoč. Po našem mnenju je tak način bolj prijazen do uporabnika od ustaljenega.

Pri popravku za okno za opisno statistiko smo se prav tako odločili, da bi naredili delovanje bolj prijazno uporabniku. Okno deluje sedaj tako, da lahko uporabnik še pred klicem funkcije izbere območje, nad katerim bo izvedel funkcijo opisne statistike. Menimo, da je tak način delovanja bolj podoben navadam uporabnikov in njihovim pričakovanjem, kako naj bi program deloval.

Pri podoknu opravil za gretlove funkcije smo opazili, da lahko je pri inicializacij okno nekoliko preozko. Načrtovali smo, da bi se pri vsakem pogledu v podokno opravil avtomatsko prednastavila širina podokna opravil, kar bi bilo prav tako bolj prijazno do uporabnika.

4.4.2 Kodiranje in implementacija programske rešitve

Kot rečeno, smo najprej popravili okno za pomoč. Okno ima sedaj namembnost okna »O programu«, kjer lahko uporabniki dobijo osnovne podatke, kot so verzija in ime programa (slika 4.6). Znotraj tega programa smo nameravali izdelati tudi pomoč, a smo to funkcionalnost pustili za prihodnost.



Slika 4.6: *Okno "O programu".*

V skladu z načrtom smo vsa okna naredili kot podrejena aplikaciji Excel. Tukaj nam je ponovno prišlo prav poznavanje vmesnikov Win32. Uporabili

smo vmesnik user32.dll, preko katerega smo klicali funkcijo SetParent(). To funkcijo uporabimo, ko hočemo eno okno podrediti drugemu. To pomeni, da vsa interakcija z nadrejenim oknom vpliva tudi na podrejenega.

Popravili smo tudi okna za opisno statistiko in podokno opravil za gretlove funkcije. Ti popravki so naredili dodatek precej bolj prijazen uporabniku. Vsi popravki so bili razmeroma enostavno in hitro implementirani. Ker smo ugotovili, da prvotna slovenska lokalizacija ni bila popolna in povsem pravilna, zato so bili potrebni manjši popravki tudi v tem delu.

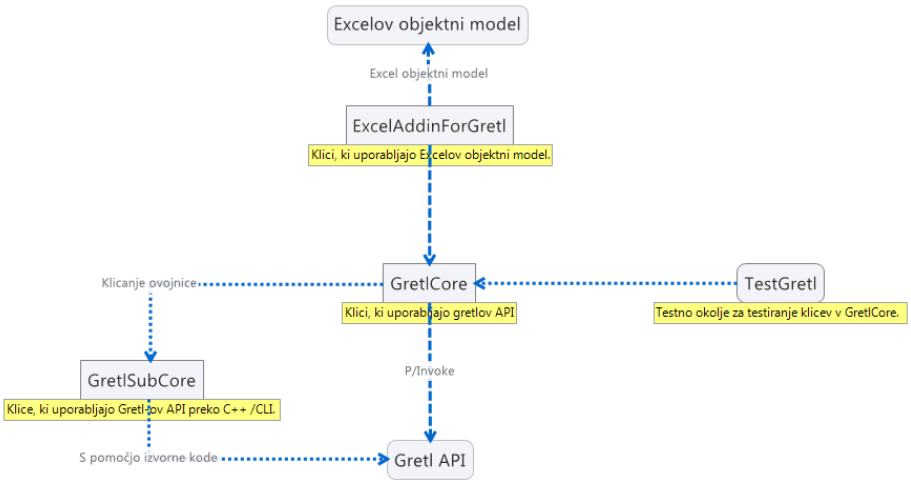
4.4.3 Testiranje delovanja

Ker smo se v tej fazi osredotočili na optimizacijo in izboljšanju uporabniške izkušnje, ni bilo potrebnega veliko testiranja. Preverili smo le, ali je osnovno delovanje ostalo enako.

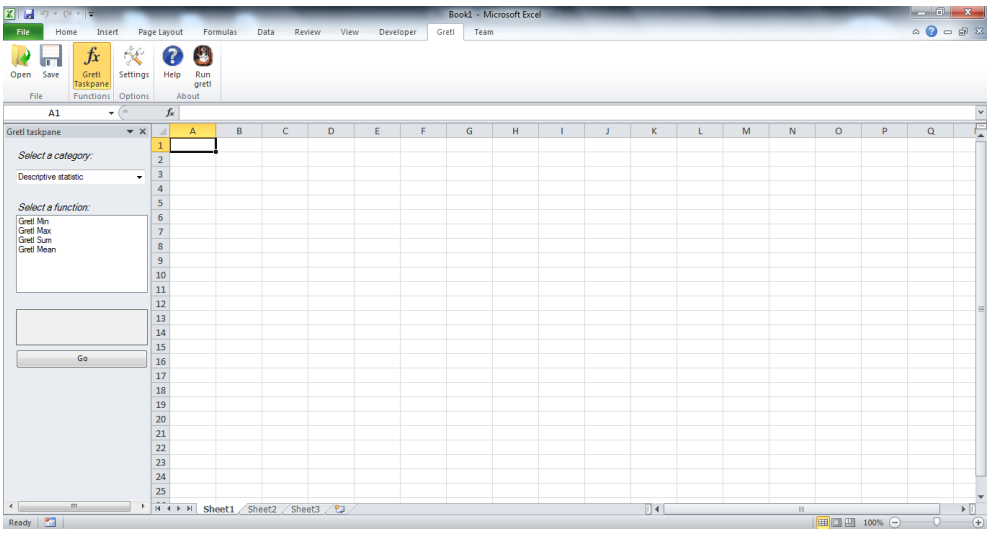
4.5 Opis končne rešitve

V tem podpoglavju podajamo končni izgled in delovanje programske rešitve. Po končni iteraciji smo prišli do arhitekture programske rešitve, ki jo prikazuje slika 4.7. Na sliki 4.8 vidimo končni uporabniški vmesnik v angleški lokalizaciji.

Kot vidimo na sliki, vsi klici, ki jih uporabljamo v rešitvi, izhajajo iz dodatka za Excel ExcelAddinForGretl. Vsa interakcija z uporabnikom gre torej skozi ta dodatek. Poenostavljeno povedano: ta dodatek je naša povezava s paketom gretl. Druga zelo pomembna knjižnica je `GretlCore`. Ta knjižnica skrbi za vso obdelavo in pravilno izvajanje klicev, ki prihajajo iz paketa gretl. Na levi lahko vidimo ovojnico `GretlSubCore`, ki služi kot pomagalo za vse tiste klice, ki jih ne moremo neposredno klicati v vmesnik uporabniškega programa gretl. Na desni vidimo program ukazne vrstice TestGretl, ki služi le za preverjanje klicev v `GretlCore`.



Slika 4.7: *Končna arhitektura programske kode.*

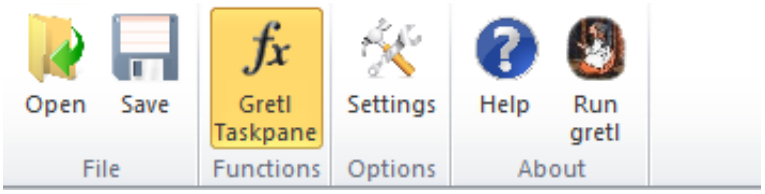


Slika 4.8: *Končni izgled grafičnega uporabniškega vmesnika povezave (angleška lokalizacija).*

4.5.1 Komponente

Prirejeni trak za Excel

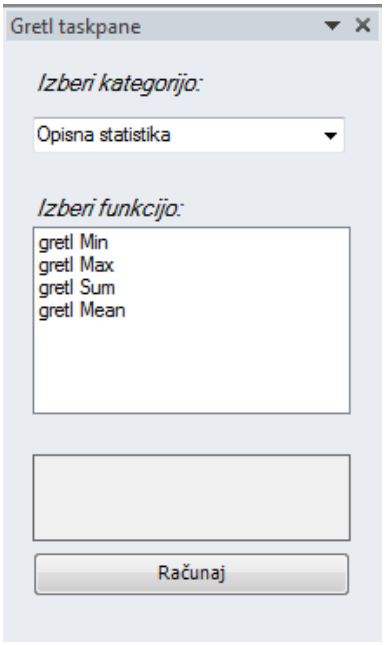
Prirejeni trak prikazuje slika 4.9. Gumba Open in Save skrbita za odpiranje in shranjevanje podatkovnih datotek tipa `.gdt`, ki jih uporablja gretl. Sledi



Slika 4.9: *Trak dodatka ExcelAddinForGretl.*

gumb Gretl Taskpane, ki pokaže podokno opravil, v katerem so definirane statistične funkcije. Še bolj levo je gumb Settings, ki odpre okno, v katerem lahko nastavljamo pravila in nastavitve za delo z dodatkom za gretl. Na koncu imamo še gumba Help in Run gretl – prvi pokaže pomoč, drugi pa požene program ukazne vrstice gretl.

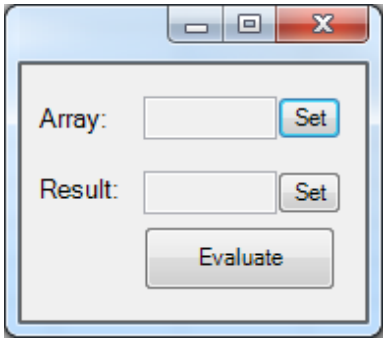
Podokono opravil



Slika 4.10: *Podokno opravil za klic funkcij znotraj dodatka ExcelAddinForGretl.*

Na sliki 4.10 je prikazano podokno opravil za gretlove funkcije. Deluje tako, da uporabnik izbere kategorijo funkcij. Trenutno sta implementirani samo "Opisna statistika"in "Ocene regresijskih modelov". Pri izbiri funkcije se nam spodaj izpiše tudi osnovna pomoč za to funkcijo. Pri kliku na gumb "Go"nam program izračuna funkcijo glede na podane parametre. V nadaljevanju so bolj podrobno razložene kategorije in okna, ki se prikažejo po kliku na ta gumb.

4.5.2 Okno Opisne statistike



Slika 4.11: *Okno za opisno statistiko.*

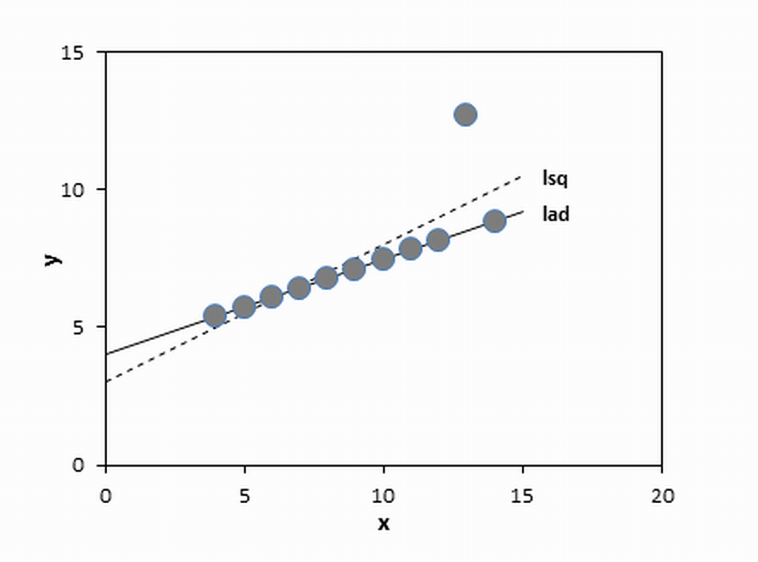
Po kliku na katerokoli izmed funkcij podokna opravil za funkcije paketa gretl se nam pokaže okno, ki je prikazano na sliki 4.11. V tem oknu nastavljamo območje, nad katerim izvajamo izbrano funkcijo in začetno celico območja, kamor bomo zapisovali rezultat izračuna. Vhodno območje se nam nastavi tudi, če ga izberemo preden pokličemo funkcijo. Po kliku na gumb *Evaluate* se izračunana vrednost zapiše v izbrano celico.

4.5.3 Okno Ocene regresijskih modelov

Po kliku na trenutno edino implementirano funkcijo ocen regresijskih modelov se po izračunu prikaže okno, v katerem z izbiro kljukic izberemo, katere

vrednosti bi radi prenesli v Excel. Po kliku na gumb *Move to workbook* se izbrane vrednosti prenesejo na nov delovni list v aktivnem delovnem zvezku.

Na tem mestu je smiselno podati primer uporabe regresije po metodi najmanjših absolutnih odstopanj. Uporabili smo znano podatkovje "Anscombov kvartet"(glej npr. http://en.wikipedia.org/wiki/Anscombe's_quartet), ki služi za ilustracijo pomena razumevanja podatkov in grafičnih prikazov oziroma učinkovito razkriva usodne napake, do katerih vodi površna in mehanska uporaba statističnih formul. Podatkovje sestavljajo štirje nizi dveh spremenljivk, med katerimi smo izbrali tretjega, ki ponazarja linearno povezavo z enim osamelcem (slika 4.12). Pri običajni linearni regresiji po metodi najmanjših kvadratov osamelec "povleče"regresijsko premico navzgor (tj. poveča ji naklon in zmanjša konstanto) v primerjavi z bolj smiselnim modelom, ki ga oceni regresija po metodi najmanjših absolutnih odstopanj (slika 4.12, preglednica 4.1).

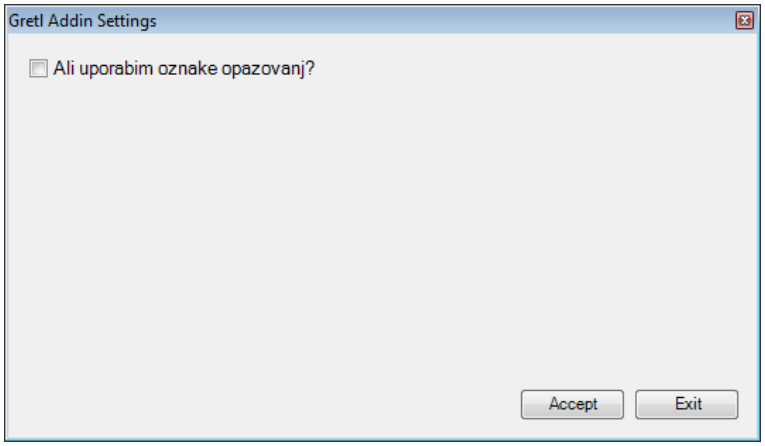


Slika 4.12: *Izbrano podatkovje (3. niz v t.i. Anscombovem kvartetu – za pojansila glej besedilo) z regresijsko premico, ocenjeno po metodi najmanjših kvadratov (lsq) in po metodi najmanjših absolutnih odstopanj (lad).*

Ocenjeni parameter		Metoda lsq	Metoda lad
konstanta	ocena	3,002	4,010
	st. napaka ocene	1,124	0,850
regresijski koeficient	ocena	0,500	0,345
	st. napaka ocene	0,118	0,120

Tabela 4.1: *Ocene parametrov za 3. podatkovni niz Anscombovega kvarteta po metodi najmanjših kvadratov (lsq) in najmanjših absolutnih odstopanj (lad; za pojasnila glej besedilo).*

Okno nastavitev



Slika 4.13: *Okno nastavitev*

V oknu za nastavitve lahko nastavljamo pravila in nastavitve, ki naj veljajo znotraj dodatka za gretl. V skladu z implementiranimi funkcijami je trenutno implementirana le ena nastavitev- "Ali uporabim oznake opazovanj?"(slika 4.13). Tu določimo, ali naj pri prenosu podatkov iz Excela v gretl uporabimo oznake vrstic. Ta nastavitev je pomembna pri shranjevanju podatkov v datoteko tipa `.gdt`: če je vklopljena, se prvi stolpec vhodnih podatkov pretvori v oznake vrstic, če je izklopljena, pa se uporabijo privzete vrednosti, ki so samo indeksi vrstic.

Poglavje 5

Sklepne ugotovitve

V okviru diplomske naloge smo načrtovali in implementirali programsko rešitev za povezavo med vodilno elektronsko preglednico Excel in ekonometričnim programskim paketom gretl. Razvili smo dodatek za Excel 2007 in Excel 2010, preko katerega lahko uporabljamo funkcije paketa gretl. Integracijo smo izvedli z uporabo Microsoftovih tehnologij.

Glavni poudarek je bil na načrtovanju najboljše možne rešitve in osnovni implementaciji izbrane rešitve. Uspeli smo narediti dobro arhitekturo s podporo za razmeroma preprosto implementacijo širokega nabora ostalih funkcij, ki jih ponuja vmesnik uporabniškega programa gretl.

Kljub uspehom smo pri razvoju naleteli na vrsto težav. Eden od največjih problemov je nastal pri implementaciji funkcij za ocene regresijskih modelov. Zaradi slabše izdelave vmesnika uporabniškega programa libgretl se je pokazalo, da je implementacija s predvideno tehnologijo neizvedljiva. Po nekaj tednih raziskovanja in iskanja odgovorov preko Microsoftovega razvojnega socialnega omrežja in drugih virov smo prišli do implementacije rešitve.

5.1 Možnosti za nadaljnje delo

Nadaljnje možnosti razširitve vidimo najprej v implementaciji gretlove funkcije, ki bi v Excel prenesla grafikon, na primer histogram (ki ga je v Excelu

zelo težko kakovostno izdelati) ali škatlasti grafikon (graf kvantilov, angl. boxplot, za katerega je v Excelu potrebnih veliko trikov ali uporaba brezplačnih ali plačljivih statističnih dodatkov). V naslednji iteracij bi bila smiselna implementacija preproste analize časovne vrste (angl. time series), kar bi bil lahko preprost univariatni avtoregresijski model ali pa postopek z grafičnim rezultatom (npr. avtokorelogram ali preiodogram). S predlaganima razširitvama bi namreč premagali oba glavna preostala izziva pri uporabi vmesnika libgretl – delo z grafikoni in delo s podatkovnim tipom časovnih vrst

Menimo, da bi bila po implementaciji teh funkcij nadaljnja razširitev nabora funkcij precej bolj preprosta, seveda pa bi še vedno ostalo veliko izzivov. Na te bi naleteli zlasti pri preverjanju ustreznosti podatkov, načrtovanju vnosnih form in procesiranju rezultatov, saj so sedaj implementirane le univariatne oziroma bivariatne metode brez preverjanja vhodnih podatkov, prava uporabnost ekonometričnih oziroma statističnih metod oziroma modelov pa se pokaže, ko imamo opravka z več spremenljivkami hkrati (s čimer se bistveno poveča tudi število predpogojev oziroma predpostavk glede podatkov).

Upamo, da bo to delo prevzel nekdo, ki ni nujno izrazito vešč programiranja oziroma najsodobnejših Microsoftovih tehnologij, a bo bistveno bolj od avtorja tega diplomskega dela razumel pojme ekonometrije in statistike, s tem pa tudi vsebino in delovanje paketa gretl.

Ne glede na to pa je implementirana rešitev ključni prvi korak na poti integracije zmogljivosti gretla v druge programe (ne le v Excel, pač pa tudi v programe s področja strojnega učenja oziroma rudarjenja podatkov, programe za prikaz podatkov oziroma vizualno analitiko ipd.), ki bi jih bilo smiselno dopolniti s široko paleto zahtevnih statističnih metod (zlasti za analizo časovnih vrst), ki jih nudi gretl. Ker gre za globalno zanimivo problematiko, je pomembno, da rešitev ponuja zelo preprosto lokalizacijo v katerikoli jezik.

Poglavje 6

Dodatek

6.1 Namestitev

Za namestitev izdelanega dodatka je potrebno imeti nameščen:

- Microsoft Excel 2007 ali Microsoft Excel 2010,
- .NET framework 4.0 in
- paket gretl (verzija 1.9.6 ali novejša).

Ko imamo nameščeno vse zgoraj navedeno programje, poženemo datoteko ExcelAddInForGretl.vsto in kmalu je vse pripravljeno za delo z dodatkom.

Slike

Literatura

[1] D.Gujarati, *Basic econometrics*, 4. izdaja. New York: McGraw-Hill, 2004, 1002 str.

[2] J.M.Woolridge, *Introductory Econometrics: a modern approach*, 4. izdaja. Cincinnati: Thomson South-Western, 805 str.

[3] R.C.Hill, W.E. Griffiths, G.C. Lim, *Principles of Econometrics*, 4. izdaja. New York: John Wiley & Sons, 2011, 785 str.

[4] F.Solina, *Projektno vodenje razvoja programske opreme*, 1. izdaja, Ljubljana: Fakulteta za računalništvo in informatiko, 1997, 212 str.

[5] MSDN forumi. (2012) Dostopno na:
http://social.msdn.microsoft.com/Forums/en-US/categories

[6] C# documentation.(2012) Dostopno na:
http://msdn.microsoft.com/en-us/library/kx37x362.aspx

[7] Visual C++ documentation. (2012) Dostopno na:
http://msdn.microsoft.com/en-us/vstudio/hh388567

[8] S.B.Lippman, Pure C++: Hello, C++/CLI, MSDN magazine (2012) Dostopno na:
http://msdn.microsoft.com/en-us/magazine/cc163681.aspx