UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Peter Kirič

RAZVOJ REŠITVE ZA UREJANJE PREDLOG XSLT Z UPORABO SISTEMA WINDOWS PRESENTATION FOUNDATION

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Zoran Bosnić

Ljubljana 2012

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani **Peter Kirič**,

z vpisno številko **63010057**,

sem avtor diplomskega dela z naslovom:

Razvoj rešitve za urejanje predlog XSLT z uporabo sistema Windows Presentation Foundation

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Zorana Bosnića,
- 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:

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: 00244/2012
Datum: 02.04.2012

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

Kandidat: **PETER KIRIČ**

Naslov: **RAZVOJ REŠITVE ZA UREJANJE PREDLOG XSLT Z UPORABO SISTEMA WINDOWS PRESENTATION FOUNDATION**

DEVELOPMENT OF A XLST TEMPLATE EDITING SOLUTION USING THE WINDOWS PRESENTATION FOUNDATION SYSTEM

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Windows Presentation Foundation (WPF) je sodobni sistem za razvoj grafičnih uporabniških vmesnikov za aplikacije v operacijskem sistemu Windows. Kandidat naj v diplomski nalogi razdela arhitekturo in delovanje tega sistema. Na njegovi osnovi naj izdela večnivojsko aplikacijo, ki funkcionalno skrbi za hranjenje razpoložljivih XLST dokumentov v podatkovni bazi SQL. Vmesnik aplikacije naj bo razvit z WPF in naj demonstrira poglavitne funkcionalnosti tega ogrodja.

Mentor:

doc. dr. Zoran Bosnić

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Zahvala

Ob zaključku študija, ki je trajal dlje, kot je bilo pričakovano, se iskreno zahvaljujem svojemu mentorju doc. dr. Zoranu Bosniću za potrpežljivost, pomoč in vodenje pri izdelavi diplomskega dela. Zahvala gre tudi sodelavcem v podjetju B2, d. o. o., predvsem mojemu nadrejenemu Tomažu Mlakarju, ki so mi omogočili izvedbo diplomskega dela. Predvsem pa se zahvaljujem družini in dekletu za vso podporo, tako v slabem kot v dobrem, skozi celoten študij.

Kazalo vsebine

Kazalo slik

Kazalo tabel

Seznam uporabljenih kratic in simbolov

AOM - Automation object model

API - Application programming interface

CSS - Cascading style sheets

DTE - Development tools environment

HTML - Hyper text markup language

IDE - Integrated development environment

RAD - Rapid application development

SDK - Software development kit

SQL - Structured query language

TFS - Team foundation server

WPF - Windows presentation foundation

XAML - Extensible application markup language

XML - Extensible markup language

XSLT - Extensible stylesheet language transformations

Povzetek

Diplomsko delo opisuje izdelavo razširitve za Visual Studio 2010, s katero je mogoče enostavneje urejati predloge XSLT, shranjene znotraj poizvedbe SQL. Razširitev poleg urejanja predlog skrbi za ustrezno komunikacijo s podatkovnim strežnikom Microsoft SQL, kot tudi s strežnikom TFS.

V diplomski nalogi predstavimo različne možnosti za razširitev razvojnega orodja Visual Studio 2010. Za razvoj grafičnega vmesnika razširitve je bil uporabljen nov programski vmesnik WPF, ki se uporablja za razvoj namiznih aplikacij z ogrodjem .NET. Ta nam omogoča izdelavo bolj bogatih in očesu prijaznih aplikacij, kar bo s pridom izkoriščala tudi naša razširitev. Sledi še kratek opis strežnika Team Foundation Server in modela za hiter razvoj aplikacij (RAD), ki sta bila uporabljena pri izdelavi razširitve.

Rezultat diplomskega dela je delujoča razširitev, ki se uporablja v slovenskem podjetju B2 d.o.o. za lažje urejanje predlog XSLT. Pravilno urejena predloga je zelo pomembna, saj se na podlagi le te prikazuje vsebina spletnega portala www.spletno-ucenje.com. Razširitev tako razvijalcem omogoča lažje in hitrejše delo.

Ključne besede

Visual Studio 2010, razširitev, WPF, .NET, TFS, Microsoft SQL podatkovni strežnik, RAD

Abstract

The thesis focuses on the development of Visual Studio 2010 extension, which will allow easier editing of XSLT templates stored in SQL database. In addition to editing, the extension must also ensure proper communication with Microsoft SQL database server as well as with Team Foundation Server.

In this thesis we present Visual Studio 2010 as a development tool and the various options for extending it. For the development of graphical user interface, a new programming interface named WPF was used that is intended for .NET development of desktop applications. It allows us to produce richer and visually appealing applications that the extension will take advantage of. Then, we follow up with a brief description of the TFS and a model for rapid application development (RAD), which was used in the development of the extension.

The result of the thesis is a functional extension for Visual Studio 2010, which is used in Slovenian company B2 d.o.o. for easier editing of XSLT templates. Properly edited template is very important because it affects the rendering of content of the web portal www.spletno-ucenje.com. So, the extension makes developers work a little bit easier and faster.

Keywords

Visual Studio 2010, extension, WPF, .NET, TFS, Microsoft SQL database server, RAD

1. Uvod

Živimo v času, ko vedno več ljudi iz dneva v dan uporablja računalnike. Ne glede na to, ali ga uporabljajo pri delu, študiju ali pa mogoče le za zabavo, se že od pojava prvih računalnikov srečamo s pojmom aplikacije. Dandanes si uporabo računalnika težko predstavljamo brez aplikacij. Že ob zagonu računalnika se zažene operacijski sistem, ki je aplikacija, namenjena upravljanju računalnika. Tako, kot si težko predstavljamo računalnik brez aplikacij, si tudi ne moremo zamisliti aplikacije brez razvijalca. Naloga razvijalca je seveda razvoj aplikacij, pri čemer si pomaga z raznoraznimi orodji. Zahtev po aplikacijah je vse več in temu primerno so se skozi čas spremenila tudi orodja. Iz preprostih urejevalnikov besedila so se razvila integrirana razvojna okolja (IDE), ki lahko poleg naprednejšega urejevalnika kode vsebujejo tudi urejevalnik grafičnega vmesnika, razhroščevalnik, prevajalnik itd. Namen takšnih orodij je olajšati delo razvijalcev in s tem med drugim povečati njihovo produktivnost. Kljub temu, da sodobna orodja vsebujejo ogromno funkcionalnosti, se pri delu razvijalca mnogokrat pojavi potreba po zelo specifični funkcionalnosti. To je lahko le zelo preprosta naloga (npr.: manipulacija besedila v urejevalniku), ki jo mora razvijalec večkrat izvesti in jo želi avtomatizirati, lahko pa imamo v mislih kompleksnejšo funkcionalnost, ki jo je potrebno vgraditi v samo orodje. Potreba po takšni rešitvi se je pojavila tudi pri naročniku in tako je nastal praktičen primer, ki je predstavljen v tem diplomskem delu.

V prvem delu diplomske naloge bomo predstavili uporabljena orodja s teoretičnega stališča ter tehnologije in metodologijo razvoja, ki so bili uporabljeni za razvoj rešitve. Tako bo najprej na kratko predstavljeno razvojno okolje Visual Studio 2010. Nekaj besed bo namenjenih osnovnim lastnostim, poleg tega pa bo predstavljen tudi koncept razširitve orodja in različni načini, ki so nam za to na voljo. Nato sledi opis nove tehnologije WPF, uporabljene pri gradnji grafičnega uporabniškega vmesnika, in pregled lastnosti strežnika TFS. Teoretičen del naloge zaključimo s pregledom metodologije za hiter razvoj aplikacij, ki bo poskrbela, da bo razvoj razširitve ostal na pravi poti.

Drugi del naloge bo namenjen prikazu razvite razširitve. Najprej bomo pogledali zahteve, ki naj bi jih aplikacija izpolnjevala, in nato nadaljevali s prikazom poteka razvoja. Ker uporabljena metodologija razvoja narekuje, da naj razvoj poteka modularno, bomo večje module opisali in jih dodatno podkrepili z diagrami poteka. Poglaviten del tega poglavja pa bo posvečen opisu razširitve in njenih funkcionalnosti.

V zadnjem delu diplomskega dela bodo sledile še sklepne ugotovitve, kjer bomo ovrednotili rezultate razvoja in opisali morebitne težave, na katere bomo med razvojem naleteli. Navedli bomo tudi morebitne ideje in izboljšave, ki bi jih lahko implementirali v nadaljnje različice razširitve.

2. Uporaba Windows Presentation Foundation (WPF) pri izdelavi razširitve za Microsoft Visual Studio 2010

2.1. Microsoft Visual Studio 2010

Microsoft Visual Studio je integrirano razvojno okolje (Integrated Development Environment – IDE). IDE je aplikacija, orodje, namenjeno razvijalcem programske opreme, ki je navadno sestavljena iz urejevalnika kode, prevajalnika, razhroščevalnika in urejevalnika grafičnega vmesnika. Namen IDE-ja je olajšati delo razvijalcev in s tem med drugim povečati njihovo produktivnost [1].

Microsoft je prvi Visual Studio javnosti predstavil leta 1997 in od takrat izdal še kar nekaj različic. Trenutno aktualna različica je Visual Studio 2010, za katerega lahko rečemo, da je eno izmed najboljših razvojnih orodji na trgu. V tem poglavju bomo predstavili samo nekaj osnovnih lastnosti, saj bi podroben pregled celotnega orodja močno presegel obseg in namen te diplomske naloge.

2.1.1. Navigacija po Visual Studiu 2010

Ko kreiramo naš prvi projekt oziroma odpremo že obstoječo rešitev, opazimo, da je razporeditev relativno generična: zbirka orodij na levi, raziskovalec rešitve na desni strani in koda oziroma naše delovno področje na sredini.

Menijska vrstica je del večine namiznih aplikacij in to orodje seveda ni izjema. Meniji so zelo intuitivni in možnosti za izbiro so na pričakovanih mestih. Posamezni deli menija se prikažejo/skrijejo glede na naš položaj v aplikaciji, funkcionalnosti, ki smo jih izbrali ob namestitvi in naš privzeti programski jezik.

Pod menijem se nahaja orodna vrstica, v kateri se nahajajo funkcije, ki jih pogosto uporabljamo in so podmnožica tistih, dostopnih iz menijev. Orodne vrstice so odvisne od konteksta, kar pomeni, da se prikazujejo in skrivajo glede na trenutno aktivnost, ki jo izvajamo znotraj orodja. Ker je vseh orodnih vrstic blizu 30 in jih je, kot rezultat tega, večina namenjenih zelo specifični uporabi, lahko uporabnik sam izbira, katere naj bodo prikazane in katere skrite.

Delovno okolje v Visual Studiu 2010 sestavljajo okna – privzeto so to okno z zbirko orodij, raziskovalec rešitve in glavno okno, v katerem urejamo kodo, pišemo poizvedbe nad podatkovno bazo, oblikujemo spletno stran in še mnogo drugih stvari. Poleg privzetih lahko prikažemo še vrsto drugih oken s specifično vsebino in namenom, kot je na primer okno z lastnostmi, okno za iskanje, okno za pregled podatkovnih strežnikov in še mnogo drugih.

Visual Studio 2010 nam torej omogoča prikaz mnogo oken, kar bi lahko povzročilo nepreglednost, celo neuporabnost orodja, saj bi se vsa ta okna zmanjšala našo delovno površino na razmeroma neuporabno velikost. Da do tega ne pride, imamo na voljo izvrstno upravljanje oken, ki prepreči gnečo oziroma prenatrpanost našega delovnega okolja.

Poljubno okno lahko zasidramo (dock) na različna "lepljiva" mesta znotraj orodja. Tipično jih lahko zasidramo na zgornjo, spodnjo, levo ali desno stran. Še več, poljubno okno lahko zasidramo tudi drug drugemu. Na primer, okno z lastnostmi lahko zasidramo na desno stran pod okno raziskovalca rešitve ali ga dodamo kot dodaten zavihek le-temu. Kot pomoč se nam med vlečenjem posameznega okna, ki ga želimo zasidrati, prikaže ikona na posameznem robu. Prav tako se pokaže ikona nad posameznim oknom, v katerega želimo zasidrati novo okno.

Okna, ki jih uporabljamo, ni nujno, da so zasidrana, lahko so tudi plavajoča. Takšno okno lahko postavimo na poljubno mesto na našem namizju, kar omogoča uporabnikom, ki pri delu uporabljajo več monitorjev, da posamezno okno prenesejo na drug monitor in s tem izkoristijo celotno delovno površino, ki jo imajo na voljo [2].

Kadar koli ustvarimo ali odpremo aplikacijo oziroma lahko tudi samo eno datoteko, Visual Studio 2010 uporabi koncept rešitve, da vse skupaj poveže v neko celoto. Tipično je rešitev sestavljena iz enega ali več projektov, ki vsebujejo posamezne elemente povezane z njimi. Rešitev s pripadajočimi projekti in elementi je predstavljena na priročen vizualni način, ki smo ga vajeni iz raziskovalca – drevesna struktura. Raziskovalec rešitve uporabljamo za navigacijo po številnih elementih znotraj naše rešitve. Prav tako lahko rešitvi dodajamo nove projekte, elemente ali reference do knjižnic, ki jih naša aplikacija uporablja [3].

Videti je, da gre z vsako novo različico in vsakim novim orodjem pri programiranju manj in manj za pisanje kode in več za "povleci – spusti" ter konfiguriranje. Mnoga orodja, kontrole in bogati oblikovalci nas po eni strani osvobodijo pisanja ponavljajoče se kode, a zahtevajo našo pozornosti v obliki vzdrževanja. To tipično opravimo preko nastavljanja dobesedno stotine lastnosti, ki usklajeno definirajo in vplivajo na našo aplikacijo. Lastnosti so predstavljene tabelarično, vsaka vrstica predstavlja eno lastnost, stolpca pa predstavljata ime lastnosti s pripadajočo vrednostjo [2].

Ko pišemo kodo, je urejevalnik pred nami na sredini in predstavlja drobovje naše aplikacije. Poskrbi za ustrezne zamike in presledke, ki naredijo našo kodo bolj čisto in berljivo. Zagotovi *IntellilSense* in samostojno dokončevanje stavkov, kar nam prihrani iskanje objektov knjižnic in rezerviranih besed v dokumentaciji oziroma drugih virih. Kodo grupira v bloke; rezervirane besede in komentarje barvno označi; podčrta napake v kodi; prikaže nov del kode v primerjavi s predhodno prevedeno kodo.

Kljub temu, da je Visual Studio orodje za razvijanje in razširjanje drugih aplikacij, imamo možnost razširljivosti tudi tega. Samoumevno je, da vgrajene funkcionalnosti ne bodo zadovoljile potrebe razvijalcev, ki potrebujejo kaj bolj specifičnega.Pri razvoju Visual Studia so vsekakor imeli v mislih tudi razširljivost. Visual Studio vsebuje svoj programski vmesnik (API - Application Programming Interface), ki razvijalcem omogoča nadzor nad mnogimi deli orodja in se imenuje Visual Studio avtomatizacijski objektni model (AOM - Automation Object Model). Razumevanje njegovih zmožnosti nam omogoča programiranje in nadzor orodja samega s pisanjem kode v obliki enega izmed načinov razširitve:

- Makroji so najlažji način avtomatizacije Visual Studia. Nanje lahko gledamo kot na nekakšen skriptni jezik orodja. Makroji so najbolj primerni za hitre avtomatizacijo nalog, kot na primer manipulacija besedila v urejevalniku kode ali avtomatizacija ponavljajoče se naloge v orodju. Makroji imajo relativno omejene zmožnosti in so primerni za preprostejšo avtomatizacijo nalog. Napisani so lahko zgolj v Visual Basic jeziku – drugi jeziki niso podprti. Razvijalci lahko makroje delijo med seboj, ampak za to je potrebna delitev datotek makro projekta, kar vključuje tudi izvorno kodo.
- Dodatki (Add-in) so zmogljivejši od makrojev, čeprav oba uporabljata avtomatizacijski model Visual Studia in omogočajo ustvarjanje novih orodnih oken in čarovnikov. Integracija novih funkcionalnosti v orodje samo je brezhibno. Dodatki so prevedeni projekti, ki jih lahko pišemo v poljubnem .NET jeziku, celo Visual C++, kar nam omogoča delitev dodatka z drugimi razvijalci brez posredovanja izvorne kode same.

- VSPackages so del razvojnega kompleta programske opreme za Visual Studio (Visual Studio SDK), ki ga prenesemo in namestimo ločeno, in zagotavlja še večjo zmogljivost kot dodatki. VSPackages nam omogočajo dostop do jedra internih vmesnikov v Visual Studiu in so zato idealni za integracijo naših lastnih urejevalnikov, oken po meri ali celo programskega jezika.
- Sestavni deli MEF-a (Managed Extensibility Framework) omogočajo razširitev novega urejevalnika v Visual Studiu 2010, ki temelji na WPF-ju. Urejevalniku lahko spremenimo izgled in vedenje. Torej, če želimo dodati funkcionalnosti urejevalniku kode, potem je to najboljša izbira [3].

2.1.1.1. AOM - Avtomatizacijski objektni model (Automation Object Model)

AOM je strukturiran razred knjižnic s korenskim objektom imenovanim `DTE` (ali `DTE2`), kar je okrajšava za Development Tools Environment – orodja za razvoj okolja. S sklicem knjižnice, s katero implementiramo objekt `DTE/DTE2`, nam primerek tega korenskega objekta omogoča dostop do komponent orodja preko njegovih pripadnikov in podrejenih razredov.

Na splošno lahko gledamo na razrede objektnega modela, kot da so organizirani v skupine, ki so v direktni komunikaciji s temi koncepti orodja:

- Rešitve in projekti
- Okna in ukazne vrstice (orodne in menijske vrstice)
- Dokumenti
- Ukazi
- Razhroščevalnik
- Dogodki

Vsi objekti v teh skupinah se dotikajo različnega dela orodja. Dostop do vsakega objekta je tipično omogočen preko objekta `DTE2` na korenskem nivoju [2].

Objekt `DTE/DTE2` predstavlja vrh programskega vmesnika. Lahko bi rekli, da predstavlja sam Visual Studio, z objekti, ki se preslikajo na razne sestavne dele orodja.Lastnosti `DTE` uporabimo za pridobitev reference specifičnega objekta orodja (ali zbirke objektov). Metode na objektih se uporabljajo za izvršitev ukazov v orodju, zagon čarovnikov ali za zapiranje orodja.

V tabeli 1 so predstavljene glavne lastnosti in metode, definirane v objektu `DTE2`.

Tabela 1: Lastnosti in metode DTE2 objekta za dostop do IDE-ja

Kategorija	Lastnost	Opis
Ukazi	`Commands`	Vrne zbirko ukaznih objektov; na splošno je ukaz akcija, ki se lahko izvede znotraj orodja, kot na primer odpiranje ali shranjevanje datoteke.
Razhroščevalnik	`Debugger`	Vrne objekt razhroščevalnika.
Dokumenti	`ActiveDocument`	Vrne objekt tipa `Document`, ki predstavlja trenutno aktiven dokument.
Dokumenti	`Documents`	Vrne zbirko objektov tipa `Document`, ki predstavljajo vse odprte dokumente.
Dogodki	`Events`	Vrne objekt tipa `Events`, ki rokuje z obveščanjem dogodkov.
Rešitve in projekti	`ActiveSolutionProjects`	Vrne zbirko objektov tipa Project, ki predstavljajo

		projekte, ki so trenutno izbrani znotraj raziskovalca rešitve
Rešitve in projekti	`Solution`	Vrne objekt tipa `Solution` za trenutno odprto rešitev.
Okna in ukazne vrstice	`ActiveWindow`	Vrne objekt tipa `Window`, ki predstavlja okno znotraj orodja, ki ima fokus.
Okna in ukazne vrstice	`CommandBars`	Vrne zbirko objektov tipa `CommandBar`, ki predstavljajo vse orodne in menijske vrstice
Okna in ukazne vrstice	`MainWindow`	Vrne objekt tipa `Window`, ki predstavlja okno samega orodja.
Okna in ukazne vrstice	`StatusBar`	Vrne objekt tipa `StatusBar`, ki predstavlja statusno vrstico Visual Studia.
Okna in ukazne vrstice	`ToolWindows`	Vrne instanco objekta tipa `ToolWindow`, ki omogoča dostop do par najpomembnejših orodnih oken: okno za ukaze, seznam z napakami, raziskovalec rešitve, seznam opravil in okno z zbirko orodij.
Okna in ukazne vrstice	`WindowConfigurations`	Vrne zbirko objektov tipa `WindowConfiguration`; ti objekti predstavljajo razne postavitve oken v uporabi v Visual Studiu.
Kategorija	Metoda	Opis
Ukazi	`ExecuteCommand`	Izvede ukaz v okolju.
-	`LaunchWizard`	Zažene opredeljen čarovnik z danimi parametri.
-	`Quit`	Zapre Visual Studio.

* Način sklicevanja in instanciranja DTE objekta se lahko malenkostno razlikuje glede na to ali pišemo makro ali dodatek.

2.2. WPF

2.2.1. Kaj je WPF?

WPF je programski vmesnik za gradnjo grafičnega vmesnika za namizne aplikacije z ogrodjem.NET.

Je WPF kratica za Windows Presentation Fundation: zbirka knjižnic .NET in pripadajočih podpornih orodij. Njegov namen je ponuditi enoten programski vmesnik za ustvarjenje bogatih in sofisticiranih uporabniških vmesnikov za okolje Windows.

WPF združuje dobre lastnosti spletnega razvijanja, kot so slogi in označevalni jezik za deklarativni uporabniški vmesnik, z dobrimi stvarmi iz bogatih internetnih aplikacij, kot je prilagodljiva vektorska grafika, animacije in podpora za multimedijo. Vse te dobre stvari pa so povezane z dobrimi starimi lastnostmi tradicionalnega razvoja namiznih aplikacij – kot na primer močna integracija z operacijskim sistemom in povezovanje podatkov s podatkovnimi viri. V WPF so vsi ti koncepti močnejši in enotni. Ampak tudi vse to ne zajame celoten obseg WPF, saj imamo tu še povsem druge plati WPF, kot so na primer podpora za risanje v 3D, napredna tipografija, prenosljivi dokumenti podobni PDF itd.

Neposredni predhodnik WPF je Windows Forms, ki je bil na voljo kot grafični programski vmesnik razvijalcem okolja .NET. Windows Forms nam je ponudi nekakšen ovoj za dostop do tradicionalnih funkcionalnosti grafičnega programskega vmesnika Windows okolja. WPF se temeljno razlikuje od svojega predhodnika v tem, da gradi vmesnik na podlagi vmesnika DirectX. DirectX je bil privzeto usmerjen predvsem v razvoj iger in multimedije, kar nam omogoča, da lahko z WPF ustvarimo precej

elegantne vizualne trike, ki jih je bilo z Windows Forms praktično nemogoče izvesti. Prav tako pa to pomeni, da bo WPF izkoristil strojno pohitritev, če je le-ta na voljo.

Značilnosti WPF:

- Deklarativen grafični vmesnik

WPF nam omogoča, da zgradimo grafični vmesnik s pomočjo označevalnega jezika, imenovanega XAML, katerega koncepti so podobni jeziku HTML. Za razliko od jezika HTML je XAML bogatejši jezik z manj nejasnostmi. XAML nudi skupen medij za interakcijo z oblikovalci.

- Inteligentna postavitev

Organizacija številnih komponent aplikacije na zaslonu je lahko zapletena, množica različnih možnosti prikaza uporabniku pa vse skupaj zapletejo še bolj. WPF zagotavlja razširljiv sistem za vizualno organizacijo elementov uporabniškega vmesnika. Odvisno od definicije postavitve lahko inteligentno prilagodi in spremeni velikost.

- Prilagodljiva grafika

Grafika v WPF temelji na vektorski v nasprotju z rastrsko grafiko. Vektorska grafika je po naravi prilagodljiva, s povečevanjem oziroma pomanjševanjem na izgublja kvalitete in tipično zavzame manj prostora za shranjevanje. WPF kljub vsemu ponuja veliko podpore za rastrsko grafiko, ampak vektorji so več kot primerni za gradnjo uporabniškega vmesnika.

- Predloge

WPF naredi ponovno uporabo elementov uporabniškega vmesnika zelo enostavno. Poznamo dve vrsti predlog: predloga za kontrole in podatkovna predloga. Predloga kontrole nam omogoča dodatno opredelitev podobe kontrole, kot na primer potreba po modri barvi ozadja in rdeči obrobi vseh spustnih seznamov v aplikaciji. Predloge kontrol tudi olajšajo delo oblikovalcem, ki lahko zagotovijo določen videz kontrol preko predlog, ne da bi s tem vplivali na sam razvojni proces.

Podatkovne predloge so podobne, le da namesto definicije videza kontrole definirajo prikaz določenih tipov podatkov. Če uporabimo povezavo takšne podatkovne predloge s katerim od elementov uporabniškega vmesnika, kot je na primer seznam, bo WPF uporabil ustrezno podatkovno predlogo. V praksi opazimo, da so podatkovne predloge zelo priročne, ko imamo opravka s seznami oziroma zbirkami podatkov.

- Povezovanje s podatki

Ko govorimo o vezavi v WPF, seveda najprej pomislimo na koncept povezovanja s podatki, katerega poznamo že iz Windows in Web Forms. Ta je postal priljubljen in je vsekakor prikazal svojo uporabnost. Čeprav ima WPF precej funkcionalnosti povezovanja s podatki (vsekakor veliko več od predhodnikov), nam poleg tega omogoča deklarativno vezavo drugih reči, kot so ukazi, animacije in dogodki. Gumb lahko, na primer, deklarativno vežemo na ukaz za lepljenje.

- Oblikovanje

WPF zablesti, ko želimo lep izgled uporabniškega vmesnika. Omogoča nam, da na primer nastavimo ozadje vnosnega polja rdeče ali nastavimo gumbu debelo modro obrobo. Določanje vrednosti osnovnih lastnosti se od klasičnega načina, ki ga v .NET okolju poznamo že od prej (Web ali Windows Forms), v bistvu ne razlikuje. Do razlike pride, ko nastavljamo kompleksnejše (sestavljene) lastnosti. Obrobo lahko, na primer, sestavimo iz »čopiča«, ki predstavlja gradient. Tudi vsebina gumba ni omejena le na tekst ali sliko – za vsebino lahko vstavimo kar celotne elemente, ki vsebujejo kup svojih podrejenih elementov, ki imajo lahko definirane poljubne stile. Stili v WPF so podobni CSS-u v HTML, le da so ti stili bogatejši in vsebujejo manj dvoumnosti. Obsegajo vse pričakovane vizualne karakteristike, kot na primer velikost robov, pozicija, barva itd. Stile lahko tudi preprosto ponovno uporabimo in jih združimo še s predlogami.

- Prožilci

Tako predloge kot stili v WPF podpirajo idejo prožilcev. To nam omogoča, da na primer, spremenimo barvo ozadja gumba, ko je miškin kazalec postavljen nad gumb. Prožilci nam torej omogočajo, da deklarativno ravnamo s spremembami stanja. Poleg tega pa so uporabni tudi pri sprožanju začetka animacij.

- Animacija

Ogrodje za animacije v WPF je impresivno in veliko bolj uporabno, kot bi si mislili. Večino lastnosti v WPF lahko animiramo in podpirajo izhode za časovne trakove, ključne slike in interpolacije.

- 3D

WPF poleg vsega omogoča tudi nekaj osnovnega 3D modeliranja in animacije. Osnovno zato, ker WPF ni bil nikoli predviden za gradnjo visoko zmogljivih 3D aplikacij. Ne glede na to so 3D funkcionalnosti močne in jih je preprosto vključiti v kateri koli uporabniški vmesnik [4].

V nadaljevanju bomo na kratko predstavili samo nekaj osnovnih in pomembnih lastnosti in konceptov v WPF, saj bi podrobnejši pregled vseh funkcionalnosti WPF močno presegel obseg (in namen) te diplomske naloge.

2.2.2. XAML

XAML je okrajšava za Extensible Application Markup Langugage in pomeni razširljiv aplikacijski označevalni jezik. Je jezik, ki so ga razvili pri Microsoftu, temelji na XML in je relativno preprost programski jezik, namenjen gradnji in inicializaciji objektov.NET.

Vloga, ki si jo predstavljamo, da jo XAML igra v povezavi s WPF, je velikokrat napačna, zato se moramo najprej zavedati, da lahko WPF in XAML uporabljamo neodvisno enega od drugega. Čeprav je bil XAML prvotno razvit za WPF, se uporablja tudi v drugih tehnologijah, kot na primer Windows Workflow Fundation. Poleg tega je uporaba XAML z WPF opcijska. Vse, kar lahko storimo z XAML, lahko prav tako storimo v našem najljubšem .NET jeziku (omeniti velja, da obratno ne drži) [5].

2.2.2.1. XAML sintaksa

V splošnem velja, da element v XAML predstavlja primerek objekta, atributi pa so lastnosti tega objekta. Naslednji kos kode predstavlja preprosto stran, ki vsebuje gumb:

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Button x:Name="blueButton"
            Width="100"
            Height="40"
            Background="Blue"
            Content="Click Me" />
</Page>
```

Korenski element se ujema z instanco objekta tipa `Page` oziroma, bolj natančno, objekta tipa `System.Window.Controls.Page`. Objekt tipa `Page` in vsi ostali objekti v imenskem prostoru `System.Windows.Controls` so WPF kontrola.

Element `Button` ustreza instanci razreda `System.Windows.Controls.Button`, atributi pa posledično predstavljajo lastnosti instance projekta. Torej, nastavljamo lastnosti širina (`Width`), višina (`Height`), ozadje (`Background`) in vsebina (`Content`).

Imamo pa tudi atribut `x:Name`, ki je v nasprotju s pravili, saj to ni lastnost razreda `Button`. Namesto tega je to poseben atribut, ki nudi edinstven identifikator za objekt, kar nam omogoči dostop do objekta iz kode. Enak rezultat bi dobili, če bi ustvarili spremenljivko tipa `Button` z imenom `blueButton`. Element `Button` v prejšnjem XAML primeru je ekvivalenten naslednji C# kodi:

```
Button blueButton = new Button();
blueButton.Width = 100;
blueButton.Height = 40;
blueButton.Content = "Click Me";
blueButton.Background = new SolidColorBrush(Colors.Blue);
```

Včasih moramo v naših oznakah določiti vrednost, katero je težavno izraziti v XAML, ali pa so izven obsega XAML procesorja. XAML ima zato funkcionalnost, ki nam omogoča ravnanje s takšnimi nerodnimi situacijami in se imenuje razširitve oznake (markup extensions). Primer uporabe prikazuje naslednji izrezek kode:

```
<Button Background="{StaticResource ResourceKey=mojaBarva}" Content="Click Me" />
```

Razširitve oznake so opredeljene z zavitimi oklepaji. Prva beseda v razširitvi pove WPF, za kateri tip razširitve gre. Imenu opcijsko sledi seznam poimenovanih parametrov [4].

V nekaterih primerih imamo lahko več kot en parameter. V teh primerih moramo pare ime/vrednost ločiti z vejicami:

```
{ImeRazširitve Parameter1=Vrednost1, Parameter2=Vrednost2, Parameter3=Vrednost3}
```

2.2.3. Novi koncepti v WPF

2.2.3.1. Logična in vizualna drevesa

XAML je idealen za predstavitev uporabniškega vmesnika zaradi njegove naravne hierarhije. V WPF so uporabniški vmesniki zgrajeni iz drevesa objektov, poznanih kot logična drevesa. Koncept logičnih dreves v WPF je precej enostaven a je vseeno zelo pomemben, ker je skoraj vsak vidik WPF (lastnosti, dogodki, viri itd.) povezan z njimi. Vrednost lastnosti se lahko, na primer, avtomatsko širi navzdol po podrejenih elementih, sproženi dogodki pa lahko potujejo gor ali dol po drevesu.

Podoben koncept logičnim drevesom so vizualna drevesa. V bistvu so vizualna drevesa razširitev logičnih dreves, kjer so vozlišča razčlenjena na osnovne vizualne komponente. Namesto da pustimo vsak element kot "črno škatlo", vizualno drevo izpostavi podrobnosti vizualne implementacije [5].

2.2.3.2. Odvisne lastnosti

WPF predstavi nov tip lastnosti imenovanih odvisne lastnosti, ki se uporabljajo po vsej platformi za omogočanje stilov, avtomatske vezave podatkov, animacije in mnogo več. Odvisna lastnost je skozi čas povezana z več ponudniki za določanje svojih vrednosti. Ti ponudniki so lahko na primer animacija s stalno spreminjajočo se vrednostjo določene lastnosti elementa, nadrejen element, katerega vrednost lastnosti se prenaša na njegove podrejene elemente itd. Verjetno najpomembnejša funkcionalnost odvisnih lastnosti pa je njena vgrajena možnost obveščanja o spremembah.

Kadarkoli se spremeni vrednost odvisne lastnosti, WPF avtomatično sproži vrsto akcij, odvisno od metapodatkov lastnosti. Te akcije so lahko ponovni izris ustreznih elementov, posodobitev trenutne postavitve, osvežitev podatkovnih povezav in še mnogo več. Ena najbolj zanimivih funkcionalnosti vgrajenih obvestil o spremembi so prožilci lastnosti, ki nam omogočijo izvajanje naših prilagojenih akcij, ko se spremeni vrednost lastnosti, brez pisanja proceduralne kode [5].

2.2.3.3. Preusmerjeni dogodki

Tako kot WPF doda več infrastrukture preprostim .NET lastnostim, doda infrastrukturo tudi dogodkom.NET. Preusmerjeni dogodki so dogodki, namenjeni za uporabo z drevesom elementov. Sprožen preusmerjen dogodek lahko potuje gor ali dol po vizualnem ali logičnem drevesu, kjer se enostavno in konsistentno sproži na vsakem elementu brez potrebe po prilagojeni kodi. Preusmerjeni dogodki pomagajo večini aplikacij, da ne skrbijo za podrobnosti vizualnega drevesa, prav tako pa so kritični za uspeh sestave WPF elementov.

Vsak registriran preusmerjen dogodek izbere eno izmed strategij preusmeritve – to je način, smer potovanja sprožitve dogodka skozi drevo elementov. Te strategije so izpostavljene preko vrednosti enumeracije `RoutingStrategy`:

- Tunneling – Dogodek se najprej sproži v korenu in nato na vsakem naslednjem elementu drevesa, dokler ni dosežen izvorni element (ali dokler ne določimo, da je bil dogodek že obdelan).
- Bubbling – Dogodek se najprej sproži na izvornem elementu in nato na vsakem naslednjem elementu, dokler ne dosežemo korena drevesa (ali dokler ne določimo, da je bil dogodek že obdelan).
- Direct – Dogodek se sproži le na izvornem elementu. To vedenje je enako vedenju običajnega .NET dogodka, s to razliko, da lahko takšen dogodek kljub temu sodeluje v mehanizmih, specifičnih za preusmerjene dogodke, kot na primer sprožilcih [5].

2.2.3.4. Ukazi

WPF nudi vgrajeno podporo za ukaze, ki so bolj abstraktna in šibko povezana verzija dogodkov. Za razliko od dogodkov, ki so vezani na podrobnosti specifične akcije uporabnika (na primer klik na gumb), ukazi predstavljajo akcije, ki so neodvisne od uporabniškega vmesnika. Tipični primeri ukazov so izreži, kopiraj in prilepi.

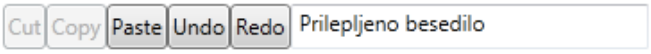
Seveda bi lahko te ukaze dokaj dobro obdelali tudi z dogodki, ampak na srečo je podpora za ukaze v WPF oblikovana tako, da je izdelava takšnih scenarijev zelo preprosta. Podpora za ukaze zmanjša količino kode, ki jo moramo napisati (v nekaterih primerih celo eliminiramo vso proceduralno kodo) in nam ponudi večjo fleksibilnost pri spreminjanju uporabniškega vmesnika brez sprememb logike kode v ozadju. Moč ukazov v veliki meri prihaja od naslednjih treh lastnosti:

- številni ukazi, definirani v WPF,
- ukazi imajo avtomatično podporo uporabnikovih ukazov (na primer bližnjice na tipkovnici),
- nekatere kontrole WPF imajo vgrajeno povezavo svojega obnašanja z ukazi.

Kot že omenjeno, vsebujejo nekatere kontrole v WPF svoje vezave ukazov. Najpreprostejši primer je kontrola `TextBox`, ki ima svoje vgrajene vezave ukazov za izreži, kopiraj in prilepi, ki delujejo na odložišče, kot tudi ukaza za razveljavi in ponovi. To ne pomeni le, da se kontrola `TextBox` odziva na standardne bližnjice na tipkovnici, ampak da je preprosto vključiti tudi druge elemente v te akcije. Naslednja XAML koda demonstrira moč teh vgrajenih vezav ukazov, rezultat te kode pa je predstavljen na Sliki 1:

```
<StackPanel Orientation="Horizontal" Height="25">
  <Button Command="Cut" CommandTarget="{Binding ElementName=textBox}"
    Content="{Binding RelativeSource={RelativeSource Self}, Path=Command.Text}"/>
  <Button Command="Copy" CommandTarget="{Binding ElementName=textBox}"
    Content="{Binding RelativeSource={RelativeSource Self}, Path=Command.Text}"/>
  <Button Command="Paste" CommandTarget="{Binding ElementName=textBox}"
    Content="{Binding RelativeSource={RelativeSource Self}, Path=Command.Text}"/>
  <Button Command="Undo" CommandTarget="{Binding ElementName=textBox}"
    Content="{Binding RelativeSource={RelativeSource Self}, Path=Command.Text}"/>
  <Button Command="Redo" CommandTarget="{Binding ElementName=textBox}"
    Content="{Binding RelativeSource={RelativeSource Self}, Path=Command.Text}"/>
  <TextBox x:Name="textBox" Width="200"/>
</StackPanel>
```

Prva dva gumba sta avtomatično onemogočena, če v vnosnem polju besedilo ni označeno, in sta avtomatično omogočena, ko označimo besedilo. Podobno se gumb prilepi avtomatično omogoči, ko imamo v odlagališču tekstovno vsebino, oziroma onemogoči v nasprotnem primeru.



Slika 1: Pet gumbov deluje kot pričakovano brez proceduralne kode, kar je posledica vgrajenih vezav vnosnega polja

Gumb in vnosno polje se drug drugega ne zavedata direktno, vendar pa lahko preko ukazov dosežemo interakcijo med njima. Prav zato je nabor vgrajenih ukazov v WPF tako pomemben. Bolj kot postanejo vgrajeni ukazi v WPF kontrolah standardizirani, bolj brezhibna (in deklarativna) lahko postane interakcija med kontrolami, ki se direktno ne zavedajo drug drugega [5].

2.2.4. Povezovanje s podatki

Povezovanje s podatki je sredstvo za povezavo elementov uporabniškega vmesnika z osnovnimi podatki za katere se aplikacija zanima. Pri povezovanju s podatki ne pišemo serije ukazov, ki naj se izvedejo, temveč opišemo relacijo, ki jo imajo podatki z elementi uporabniškega vmesnika.

Ključ vezave podatkov je objekt tipa `System.Windows.Data.Binding`, ki "zlepi" dve lastnosti in omogoči komunikacijo med njima. Ko enkrat nastavimo `Binding`, ta potem skrbi za vso sinhronizacijo v preostanku življenja aplikacije.

2.2.4.1. Uporaba Binding v XAML

Za lažjo predstavo tega koncepta predpostavimo, da imamo aplikacijo za pregledovanje slik, ki med drugim vsebuje tudi element tipa `TreeView`, v katerem je predstavljena struktura našega trdega diska, podobno kot v raziskovalcu. Dodati želimo element tipa `TextBox`, ki bo prikazal trenutno izbrano mapo v drevesu. WPF vsebuje razširitev oznake, ki nam omogoča deklarativno uporabo objekta `Binding`. Za uporabo vezave v XAML želeno lastnost direktno nastavimo na primerek objekta `Binding` in nato nadaljujemo s standardno sintakso razširitve oznake in nastavimo njene lastnosti. Naslednji košček kode predstavlja implementacijo vezave podatkov v XAML:

```
<TextBlock  x:Name="currentFolder"
            Text="{Binding ElementName=treeView, Path=SelectedItem.Header}"
            Background="AliceBlue" FontSize="16" />
```

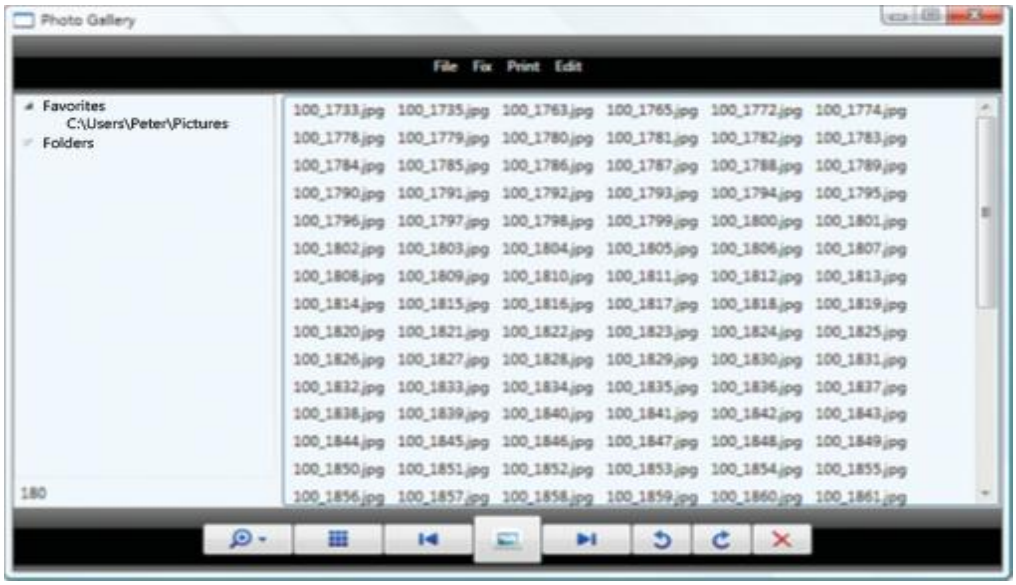
Povezovanje s podatki se sklicuje na dve lastnosti: `ElementName` in `Path`. Lastnost `ElementName` določa element (v našem primeru) v XAML, imenovan `treeView`, in se uporabi kot vir podatkov. Lastnost `Path` pa določa, kje se nahaja specifičen podatek, ki nas zanima v povezavi z virom podatkov. V našem primeru je to lastnost `SelectedItem.Header` vira podatkov (element `TreeView`).

2.2.4.2. Vezava na zbirko

Vezava posameznih .NET lastnosti je sicer uporabna, ampak bistvo vezave podatkov je seveda v povezavi kontrol in zbirk. Če nadaljujemo z nadgradnjo naše vzorčne aplikacije za pregledovanje slik, bi na primer želeli povezati kontrolo `ListBox` z zbirko `photos`,ki vsebuje osnovne podatke o posamezni sliki (ime, pot …). Vezavo zbirke in kontrole lahko definiramo na zelo preprost način:

```
<ListBox x:Name="pictureBox"
        ItemsSource="{Binding Source={StaticResource photos}}" ... >
        ...
</ListBox>
```

Če nam privzeta predstavitev zbirke `photos` (upodobitev metode `ToString`) ne ustreza, jo lahko prilagodimo našim potrebam. En način za izboljšavo je uporaba lastnosti `DisplayMemberPath`. Če jo nastavimo na primerno pot lastnosti, se bo pripadajoča vrednost lastnosti izpisala za vsak element. Ker je v našem primeru zbirka sestavljena iz objektov tipa `Photos`, lahko za prikaz uporabimo na primer ime datoteke. Na Sliki 2 lahko vidimo, kakšen bi bil rezultat naše vzorčne aplikacije.

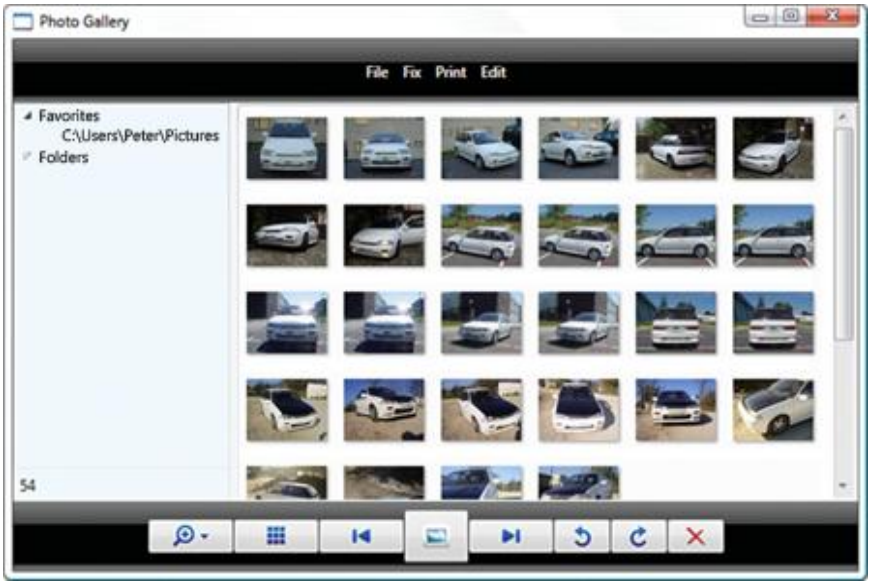


Slika 2: S pomočjo lastnosti `DisplayMemberPath` lahko preprosto prilagodimo prikaz elementov podatkovno vezane zbirke

Realizacija vezave podatkov je relativno preprosta, ko sta vir in ciljna lastnost kompatibilnega tipa podatkov in nam zadostuje privzet način prikaza podatkov. Ampak večkrat kot ne potrebujemo nekaj prilagoditve. Če pogledamo našo vzorčno aplikacijo, je očitno, da želimo v našem seznamu prikazati slike in ne le imena slik v surovi obliki. To lahko pri vezavi podatkov v WPF dosežemo s podatkovnimi predlogami. To so deli uporabniškega vmesnika, ki jih želimo uporabiti na .NET objektih, ko se le-ti izrisujejo [5].

S pomočjo podatkovnih predlog lahko dopolnimo našo aplikacijo za pregledovanje slik in nadomestimo imena slik v seznamu s seznamom slik samih. Vse, kar je potrebno storiti, je nastaviti podatkovno predlogo in določiti vir, kar seveda storimo s pomočjo podatkovne vezave. Naslednji košček kode prikaže potrebne dopolnitve v XAML, rezultat pa je viden na Sliki 3:

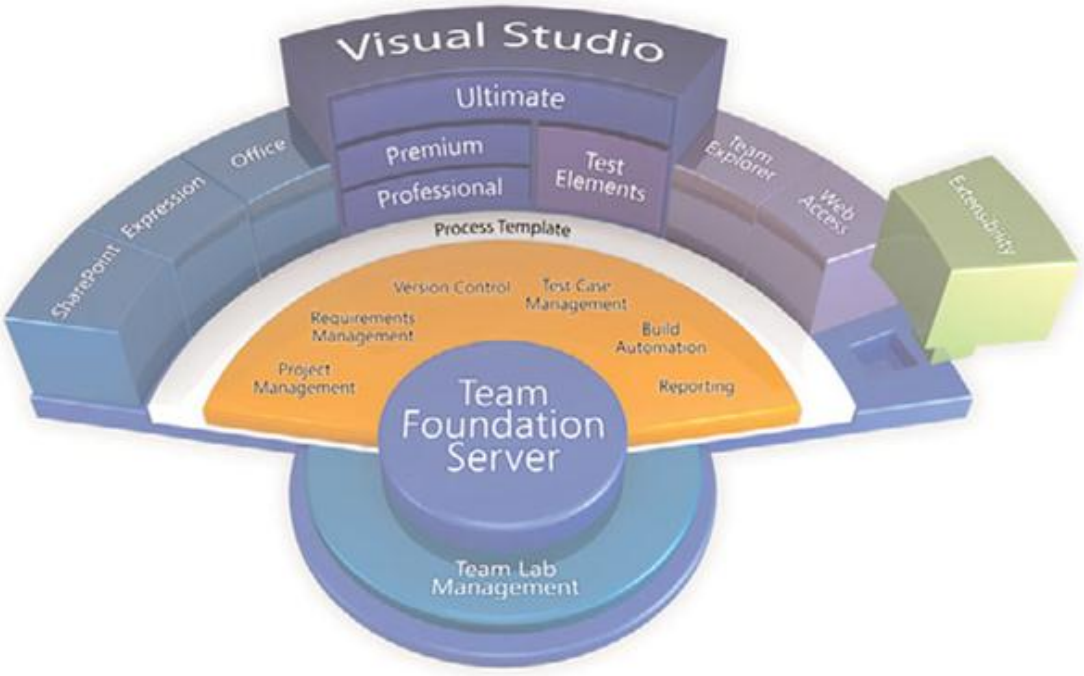
```
<ListBox x:Name="pictureBox"
         ItemsSource="{Binding Source={StaticResource photos}}" …>
    <ListBox.ItemTemplate>
        <DataTemplate>
            <Image Source="{Binding Path=FullPath}" Height="35"/>
        </DataTemplate>
    </ListBox.ItemTemplate>
    …
</ListBox>
```



Slika 3: S pomočjo podatkovne predloge dobimo želene rezultate – prikaz slike za vsak element v seznamu.

2.3. Team Foundation Server (TFS)

TFS je centralno vozlišče, ki zagotavlja integrirano upravljanje življenjskega cikla aplikacije (ALM Application Lifecycle Management) okoli številnih Microsoftovih razvojnih orodij. Prva verzija TFS-ja je izšla leta 2005 in je vključevala nadzor nad viri, centraliziran sistem upravljanja s projekti, avtomatizacija gradnje in poročanje. Microsoft je nadaljeval z nadgradnjo te različice in izdal TFS 2010. Na Sliki 6 je predstavljen TFS v povezavi s številnimi ostalimi produkti, ki obkrožajo prakse AML.



Slika 4: Team Foundation Server deluje kot središče, ki zagotavlja AML storitve mnogim razvojnim produktom, ki jih ponuja Microsoft [2].

Kot lahko vidimo na Sliki 4, predstavlja TFS središče razvoja in koordinacije življenjskega cikla aplikacije. Storitve, ki jih zagotavlja TFS:

- Usmerjanje in predloge procesov – v TFS sta privzeto vključeni dve predlogi: Microsoft Solutions Framework (MSF) za CMMI (Capability Maturity Model Integration) 5.0 in MSF za Agile 5.0. Obe nudita zbirko delovnih nalogov, potek dela in poročanja glede na specifično metodologijo. Obe prav tako nudita usmerjanje ekipe za izvajanje ključnih aktivnosti na projektu (kot na primer upravljanje zahtev ali avtomatizacija grajenja).
- Upravljanje s projekti – TFS omogoča vodjem projektov definiranje njihovih projektov v smislu iteracij in funkcionalnih področij. Nudi delovne naloge, ki jih uporabimo za definicijo, dodeljevanje in sledenje dela na projektu. Delovni nalog je lahko opravilo na projektu, zahteva, hrošč, testni scenarij in tako dalje. V bistvu delovni nalog predstavlja generično enoto dela, ki ga opravimo na projektu. Seveda imamo možnost delovne naloge po želji prilagoditi s statusi, novimi polji in z njimi povezati poslovna pravila. Delovni nalogi igrajo osrednjo vlogo pri zagotavljanju komunikacije in poročanja v ekipi.
- Upravljanje zahtev – TFS nudi specifične delovne naloge za upravljanje zahtev. V bistvu, TFS 2010 predstavi koncept hierarhičnih delovnih nalogov, kar pomeni, da imamo možnost izdelave podrejenega delovnega naloga. Tako je v TFS omogočeno bogato poročanje z združevanjem podatkov podrejenih delovnih nalogov.
- Upravljanje testnih primerov – TFS in Test Professional omogoča upravljanje delovnih nalogov specifičnih planiranju testov in testnih primerov. Definiramo lahko zbirko testnih primerov za dano zahtevo, vsak posamezni testni primer pa lahko definira korake potrebne za izvedbo testa vzporedno s pričakovanimi rezultati.
- Nadzor različic – Nadzor virov vključuje funkcije kot so na primer nabor sprememb, odlaganje sprememb na "polico", avtomatična pravila grajenja, možnost povezovanja delovnih nalogov s spremembo vira, vzporedno razvijanje, možnost razvejanja, kontrolne točke in še mnogo drugih.
- Avtomatizacija gradnje – Orodja za grajenje v TFS nam omogočajo avtomatsko in načrtovano gradnjo ter gradnjo projektov na zahtevo. Na podlagi gradenj se pripravijo poročila, dokumentacija, izvedejo se avtomatični testi in analiza pokritosti kode s testi.
- Poročanje – TFS zagotavlja bogato zbirko poročil za sledenje statistik in celotnega stanja našega projekta. Poročila vključujejo poročila, zgrajena z SQL Reporting Services kot tudi novo zbirko Excelovih poročil za neposredno delo s podatki.
- Sodelovanje in nadzorna plošča – TFS vključuje SharePoint stran za vsak projekt. Ta stran omogoča poln dostop za upravljanje in ustvarjanje delovnih nalogov izven okolij Visual Studio ali Office. Poleg tega vsebuje tudi nadzorno ploščo s poročili za hitro analizo, kaj se dogaja na projektih, kot tudi posameznih elementih prijavljenega uporabnika.
- Spletni dostop – TFS vsebuje tudi spletni dostop za delo s delovnimi nalogami preko spleta (izven okolja SharePoint). Ta verzija prav tako vsebuje dostop do nadzora virov preko brskalnika.
- Integracija z ostalimi razvojnimi okolji – TFS je dosegljiv iz okolij Visual Studio, Office, SharePoint in s spleta. Poleg tega obstaja tudi Team Exploer Everywhere za dostop do TFS funkcionalnosti z uporabo drugih razvojnih orodij, ki tečejo na operacijskem sistemu Windows. To vključuje tudi razvojno okolje Eclipse in MacOS [2].

2.4. RAD – Rapid Application Development

RAD je izraz, ki ga je predstavil James Martin leta 1991. Opisuje metodologijo razvoja programske opreme, ki vključuje kratke ponovitve in se delno opira na izdelavo prototipov za doseganje predpisanih specifikacij. V zadnjih letih se kratica uporablja v širšem pomenu z vključitvijo vrste tehnik, katerih namen je pospešitev razvoja aplikacij. RAD se pogosto uporablja v primerih, kjer nas časovne omejitve prisilijo k pristopu, v katerem je hitrejši razvoj aplikacije bolj pomemben od funkcionalnosti ali zmogljivosti. Ta metodologija je nastala kot odgovor na neagilne procese razvoja programske opreme. Iteracije in izdelava prototipov ustvarjajo priložnosti za izpolnjevanje spreminjajočih se zahtev v razvojni fazi, hkrati pa odkrivanje takšnih, ki nisi bila specificirana v začetni analizi zahtev. Uspešna RAD strategija je usmerjena k pridobivanju naslednjih prednosti:

- hitrejši razvoj aplikacij,
- izboljšana kakovost (pri RAD metodologiji je kakovost opredeljena kot stopnja, do katere aplikacija izpolnjuje potrebe uporabnikov, kot tudi z nizkimi stroški vzdrževanja z dostavljeno aplikacijo).

RAD metodologija seveda ni primerna za vse projekte. Najbolje se obnese za projekte, katerih obseg je majhen ali pa se lahko delo razdeli v obvladljive kose. Prav tako morajo biti projektne ekipe majhne (1-6 ljudi) in morajo imeti izkušnje z vsemi tehnologijami, ki bodo pri razvoju uporabljene.

2.4.1. Izdelava prototipov

Eden od ključnih vidikov metodologije RAD je gradnja prototipov za hiter začetek in zasnovo uporabnikovih zahtev. Cilj izdelave prototipov je zgraditi produkt, ki bo imel manj funkcionalnosti od končne različice v čim krajšem času, po možnosti v samo nekaj dneh. Začeten prototip služi kot dokaz koncepta, še pomembneje, služi kot osnova in orodje za natančnejše določanje nadaljnjih zahtev.

2.4.2. Iteracije

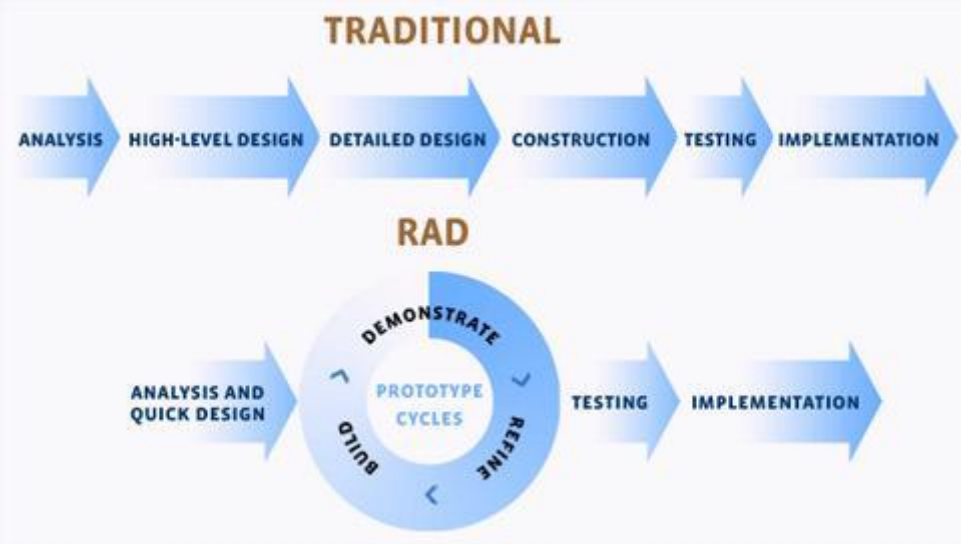
Uporaba iteracij pri razvoju aplikacij pomeni izdajo z vse več funkcionalnostmi v kratkih ciklih razvoja. Ob vsaki izdaji sledi razgovor s stranko, kjer se pregledajo zahteve za naslednjo iteracijo. Te iteracije, kot že rečeno, trajajo od enega dneva do par tednov, odvisno od trenutno uporabljenega tipa metodologije RAD na projektu.

2.4.3. Časovni okvir

Določanje časovnega okvirja je tesno povezano z izbiro funkcionalnosti, ki bodo vključene v naslednji izdaji, da lahko zaključimo z izdajo trenutne različice v najkrajšem času. Pomembnosti tega koraka ne gre podcenjevati. Brez ustreznega časovnega okvirja lahko dolžina iteracij naraste, kar pa zmanjša prednosti metodologije RAD.

2.4.4. RAD proti slapovnemu modelu

Slapovni ali zaporedni model smatramo kot tradicionalni način razvoja aplikacij, kjer razvoj enakomerno teče navzdol skozi različne faze razvojnega procesa. Problem pri večini takšnih modelov je zanašanje le na metodološko fazo analize za identifikacijo vseh kritičnih zahtev aplikacije. Obstaja veliko dokazov, da projekti, ki uporabljajo slapovni razvojni cikel, pogosto ne zagotovijo uporabnega končnega produkta. Produkti pogosto presežejo roke, ne izpolnjujejo vseh zahtev ali pa se zahteve spremenijo med dolgotrajno fazo razvoja. Na Sliki 5 lahko je predstavljen grafični prikaz razlike med tradicionalnim modelom razvoja in modelom RAD. Vidimo lahko, da korak ponavljanja reši problem povezanega z nefleksibilnim tradicionalnim pristopom razvoja programske opreme.



Slika 5: Tradicionalni razvojni model v primerjavi z RAD modelom

2.4.5. Tveganja

Ker model RAD temelji na iterativnem in inkrementalnem procesu, obstajajo določena tveganja pri uporabi le-tega. Po začetku projekta se pogosto pojavi hitro povečanje potrebnih funkcij v produktu. To je najpogostejši vir stroškov in prekoračitve časovnega plana, ki lahko ogrozijo ali celo prekinejo projekt in produkte. Da se temu izognemo, je potrebno sodelujoče projektno vodenje in pripravljenost tako stranke kot tudi ekipe za vodenje projektov, da se držijo ciklov, ki jih narekuje model RAD.

Pri razvoju po modelu RAD pa se lahko srečamo tudi z nasprotnim tveganjem. Zaradi zastavljenih časovnih okvirjev se različne funkcionalnosti nenehno prestavljajo na kasnejše različice v prid hitrejšega razvoja. Tako se lahko zgodi, da izdelamo aplikacijo, ki vsebuje manj funkcionalnosti, kot bi jih vsebovala aplikacija razvita po načelih tradicionalnih metod [6].

3. Zasnova razširitve okolja Visual Studio za urejanje predlog XSLT

V tem poglavju bomo združili tehnologije, predstavljene v prejšnjih poglavjih, in predstavili razširitev za Visual Studio 2010, ki je v pomoč razvijalcem pri urejanju predlog XSLT (EXtensible Stylesheet Language). XSLT je, na kratko, jezik, ki preoblikuje dokument XML v dokument XHTML oziroma preoblikuje en dokument XML v drugega. Posamezna predloga XSLT se v našem primeru nahaja znotraj poizvedbe SQL za vstavljanje, ki je shranjena v datoteki s končnico .sql. Visual Studio ima prilagojen urejevalnik datotek XSLT, ki razvijalcem olajša delo s funkcijo samodokončevanja, barvanjem kode, samodejnim oblikovanjem kode itd. Žal te funkcionalnosti ne moremo neposredno izkoristiti, ker Visual Studio naše datoteke ne prepozna kot XSLT, saj se le-ta namreč skriva znotraj poizvedbe SQL. Poleg tega pa mora razširitev izkoristiti že obstoječi način prevajanja besedila, ki poišče vse besede in besedne zveze primerne za prevod, in ga dopolniti z novo razvitim načinom prevajanja. Ta narekuje, da se prevedeno besedilo zamenja s šiframi, ki jih vodimo v podatkovni bazi projekta. To nam omogoča, da se za iste besede uporabi ista šifra in seveda hkrati tudi isti prevodi, kar pomeni manj oziroma nič ponavljajočih se podatkov in manj dela s prevajanjem in vzdrževanjem prevedenega besedila. Če bi besedilo le nadomestili s šiframi, bi bilo iskanje in zamenjava šifre znotraj predloge pri njeni dejanski uporabi precej težavno, če ne celo nemogoče. Zaradi tega obstaja pravilo, ki šifro zakodira v posebno zaporedje znakov: #S#P999#E#, kjer namesto števila 999 vstavimo šifro besedila. Takšna koda nam precej olajša iskanje in zamenjavo le teh, ko je to potrebno. Posledično postane urejanje predlog bolj težavno, saj se namesto besedila na njegovem mestu nahajajo kode, ki uporabniku ne povedo nič o samem besedilu. Razširitev naj bi ravno zaradi tega omogočala tudi bolj prijazen prikaz prevedenega besedila v poljubnem jeziku (če le-ta seveda obstaja) poleg kode, ki je bila generirana ob prevajanju.

Kaj vse mora razširitev omogočati pa bomo spoznali v nadaljevanju.

3.1. Zahteve

Kljub temu, da za projekt ni bilo podanih strogih smernic in zahtev ter so se med samimi cikli razvoja porodile tako nove ideje kot zahteve, so bile podane naslednje ključne smernice, ki jih je razširitev morala izpolnjevati:

- izluščiti vsebino XSLT iz datoteke, ki vsebuje predlogo XSLT znotraj poizvedbe SQL in jo prikazati v urejevalniku XSLT okolja Visual Studio.
- omogočiti iskanje želene datoteke,
- ker se datoteke nahajajo v TFS, je potrebna avtomatska rezervacija datoteke pred urejanjem in sprostitev le te, ko končamo z urejanjem,
- shranjevanje urejene vsebine XSLT v izvorno datoteko in zagon poizvedbe SQL v kateri se vsebina XSLT nahaja,
- iskanje besedila za prevajanje in pridobitev že obstoječe kode prevoda oziroma generiranje nove,
- uporabniku prijazen prikaz prevedenega besedila v poljubnem obstoječem jeziku,
- uporabniški vmesnik razširitve se mora nahajati v oknu, ki ga je mogoče pripeti na poljubno mesto znotraj Visual Studia.

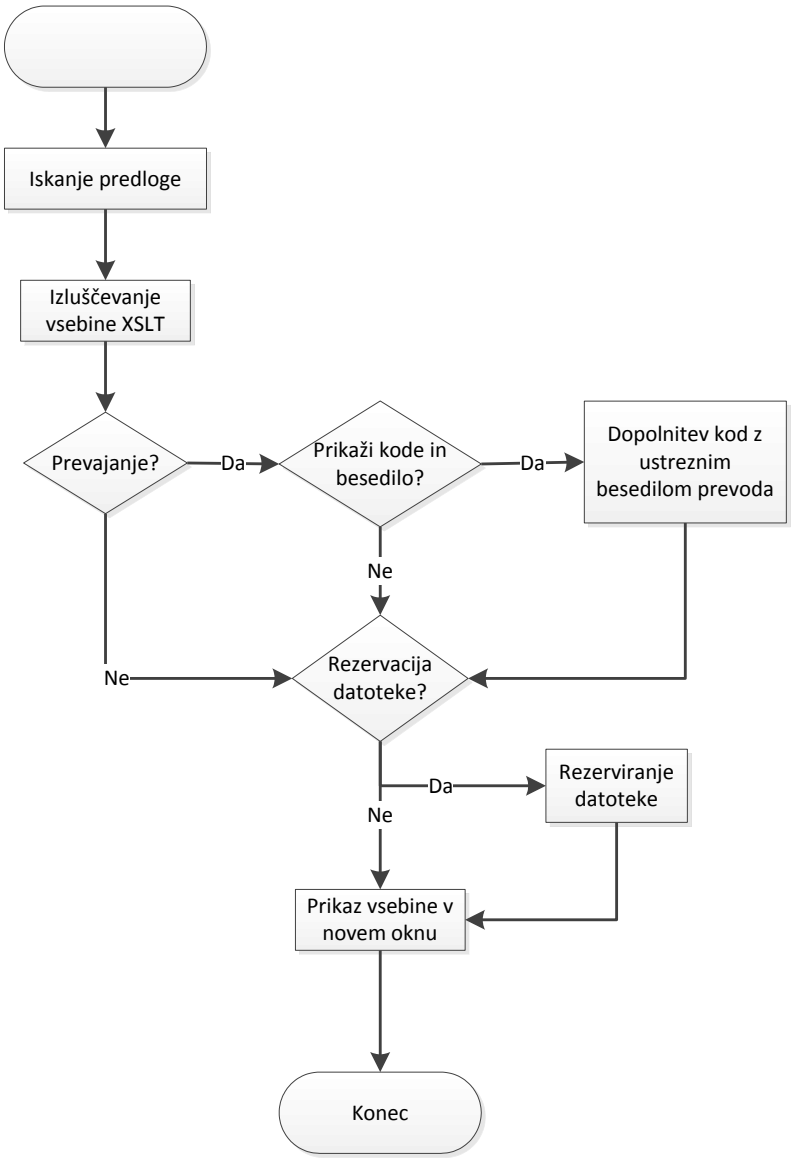
3.2. Razvoj razširitve

Tako kot projekt ni tipične narave za naročnika, tako tudi sam razvoj razširitve ni potekal po tradicionalni poti, ki navadno narekuje podrobno analizo in specifikacije zahtev. Kot že omenjeno, se je uporabljala metodologija hitrega razvoja aplikacij, zato je razvoj potekal modularno. Pripravila se je

okvirna analiza posameznega večjega modula, sledil je razvoj in testiranje le-tega. Že med samim razvojem in kasneje testiranjem so se porodile nove ideje in zahteve, ki so bile dodane v naslednjih ciklih razvoja. V nadaljevanju bodo na kratko predstavljeni glavni moduli, ki so bili razviti.

3.2.1. Iskanje in odpiranje predloge

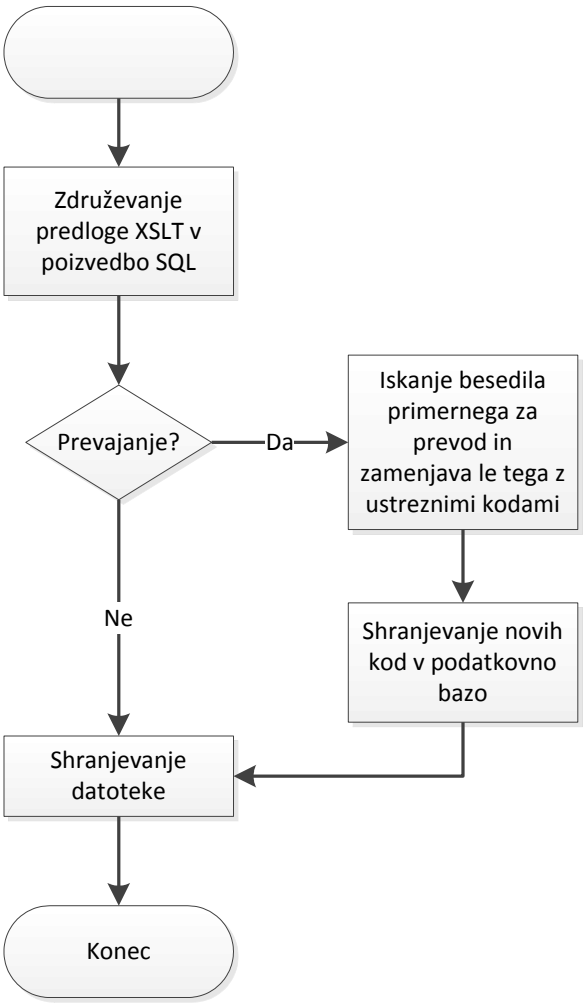
Prvi je na vrsti modul za iskanje in odpiranje predloge. Ta je najpomembnejši in predstavlja jedro razširitve, ki ji bodo dodani še ostali moduli. Na Sliki 6 je predstavljen diagram poteka, ki v grobem predstavlja funkcionalnosti in potek tega modula. Najprej je potrebno realizirati iskanje predloge. Datotek s predlogami je več kot 200 in se nahajajo v vnaprej določenih mapah znotraj svojega projekta. Ko uporabnik najde želeno predlogo, sledi odpiranje predloge in izluščevanjevsebine XSLT iz datoteke. Na prikaz rezultata v urejevalniku lahko vplivamo še zizbiro možnosti prevajanja. Odločimo se lahko za prikaz kod prevodov vključno z besedilom ali brez. Ne glede na to, ali izberemo možnost prevajanja ali ne, lahko datoteko odpremo v načinu samo za branje ali načinu za urejanje, pri katerem se izvede še rezervacija datoteke iz strežnika TFS. Zadnji korak je le še prikaz izluščene vsebine predloge v urejevalniku kode, v novem oknu Visual Studia.



Slika 6: Diagram poteka iskanja in odpiranja predloge

3.2.2. Shranjevanje predloge

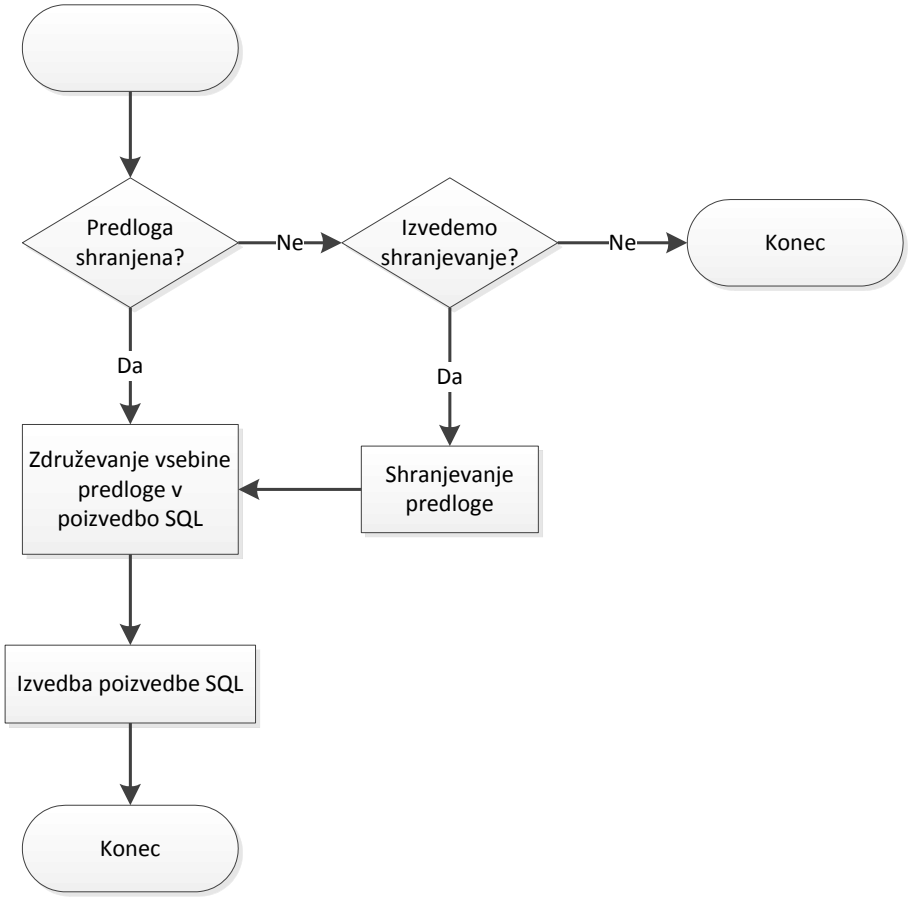
Ko končamo z urejanjem, je potrebno spremembe tudi shraniti. Proces shranjevanja je prikazan z diagramom poteka na Sliki 7. Kot prvo moramo novo vsebino predloge XSLT obdelati, da bo primerna za vključitev v poizvedbo SQL, kar je nato končna vsebina datoteke, ki jo shranjujemo. Če gre za predlogo, pri kateri želimo izvesti še prevajanje tekstov, najprej poiščemo takšna besedila znotraj predloge, ki so primerna za prevod. Ta besedila zamenjamo s kodami, ki jih dodamo v podatkovno bazo, če le-ta še ne obstajajo. Za konec ostane le še shranjevanje nove vsebine v originalno datoteko.



Slika 7: Diagram poteka shranjevanja predloge

3.2.3. Izvajanje poizvedbe SQL nad podatkovno bazo

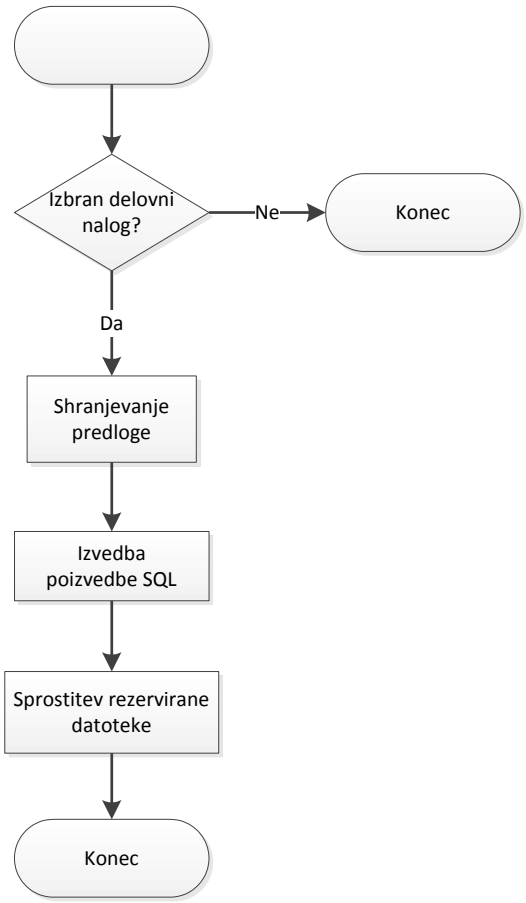
Datoteke, ki hranijo vsebino predloge so, kot že omenjeno, poizvedbe SQL. Vsebina predlog se torej hrani v tabeli v podatkovni bazi, z datotekami pa poskrbimo, da lahko vsebino urejamo na enem mestu, da se vodi zgodovina sprememb v strežniku TFS itd. Zagotoviti je torej potrebno, da je vsebina v datoteki enaka vsebini na podatkovnem strežniku, zato mora razširitev omogočati tudi izvajanje te poizvedbe SQL nad podatkovno bazo. Proces izvajanja poizvedbe SQL nad podatkovno bazo je prikazan z diagramom poteka na Sliki 8. Preden dejansko izvedemo poizvedbo, moramo zagotoviti, da je datoteka predhodno. Shranjevanje datoteke izvedemo po že opisanem postopku, kjer kot rezultat shranjevanja, med drugim dobimo tudi ustrezno poizvedbo SQL, ki jo samo še izvedemo nad podatkovno bazo.



Slika 8: Diagram poteka izvajanja poizvedbe SQL

3.2.4. Sprostitev rezerviranih datotek v strežnik TFS

Vse datoteke, ki vsebujejo vsebino predloge, se nahajajo v strežniku TFS. Ob vsaki spremembi se te datoteke rezervirajo in dokler so v takšnem stanju, se spremembe shranjujejo le lokalno. Ko zaključimo s spreminjanjem, moramo datoteko, ki sedaj vključuje tudi naše spremembe, sprostiti nazaj na strežnik TFS. Pri naročniku velja pravilo, da je potrebno izbrati delovni nalog, v okviru katerega je bila izvedena sprememba in sprostitev datoteke na strežnik TFS, povezati s tem delavnim nalogom. Tega pravila se mora držati tudi naša razširitev, zato pred sprostitvijo datoteke najprej preverimo, če je izbran delavni nalog, in z izvajanjem nadaljujemo le v primeru, če je le-ta izbran. Po izbiri delavnega naloga pred samim procesom sprostitve datoteke na strežnik TFS še enkrat preventivno izvedemo shranjevanje predloge in izvedemo poizvedbo SQL nad podatkovno bazo. S tem zagotovimo povsem ažurno stanje vsebine datoteke na strežniku TFS in podatkov v podatkovni bazi. Proces sprostitve datoteke v strežnik TFS je prikazan z diagramom poteka na Sliki 9.



Slika 9: Diagram poteka sprostitve rezerviranih datotek v strežnik TFS

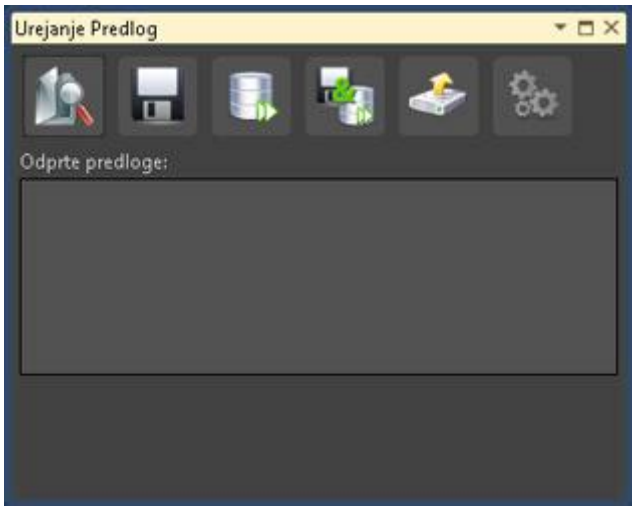
3.3. Razširitev za urejanje predlog

Glede na podane zahteve, zastavljene cilje in grobe analize potrebnih modulov je po več iteracijah razvoja nastala delujoča razširitev za Visual Studio. V naslednjih poglavjih bo predstavljen uporabniški vmesnik in osnovne funkcionalnosti razširitve.

3.3.1. Uporabniški vmesnik

Ko razširitev namestimo se nam v Visual Studiu odpre okno, prikazano na Sliki 10. Kot za vsako drugo okno je tudi za našo razširitev značilno, da jo lahko zasidramo na poljubno mesto znotraj Visual Studia, lahko jo seveda tudi pripnemo ali jo celo uporabljamo kot plavajoče okno.

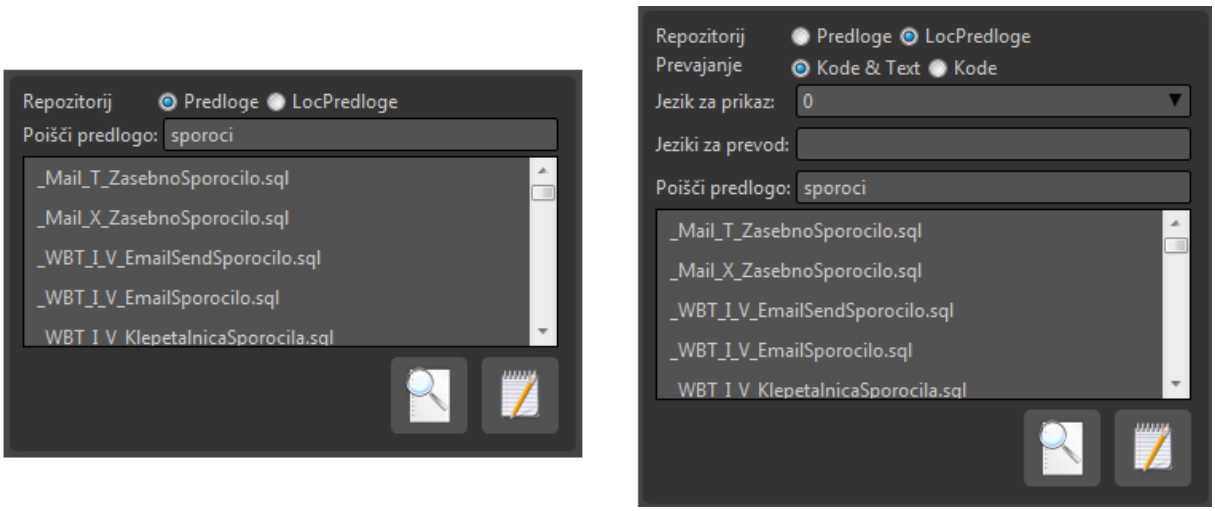
Uporabniški vmesnik razširitve, prikazan na Sliki 10, je relativno preprost in je sestavljen iz dveh delov. V zgornjem delu se nahaja šest gumbov, od katerih vsak sproži svojo akcijo ali ponudi dodatne možnosti, pod gumbi pa se nahaja seznam trenutno odprtih predlog, ki je ob zagonu razširitve prazen. Če pogledamo gumbe od leve proti desni, se za njimi skrivajo naslednje aktivnosti: iskanje in odpiranje ter rezervacija (check-out) predloge,shranjevanje predloge v datoteko, izvedba poizvedbe SQL na podatkovnem strežniku, shranjevanje in hkrati izvajanje poizvedbe SQL, sprostitev (check-in) predloge na TFS ob vezavi na delovni nalog, pregled in spreminjanje nastavitev (povezava na TFS in podatkovni strežnik).



Slika 10: Uporabniški vmesnik razširitve za urejanje predlog.

3.3.2. Iskanje, odpiranje, urejanje in prikaz predlog

Gumb za iskanje odpre dodaten sklop polj, ki omogočajo iskanje in definirajo način prikaza predlog. Ker se v projektu, za katerega je bila razvita razširitev, še ni povsem uveljavil nov način prevajanja besedila, je bila potrebna tudi podpora za stare predloge. Na Sliki 11 lahko vidimo uporabniški vmesnik za iskanje tako starih predlog (levo), kot tudi predlog, v katerih bo uporabljeno prevajanje (desno).



Slika 11: Uporabniški vmesnik za iskanje predlog brez prevajanja na levi strani in vmesnik za iskanje predlog s prevajanjem na desni strani.

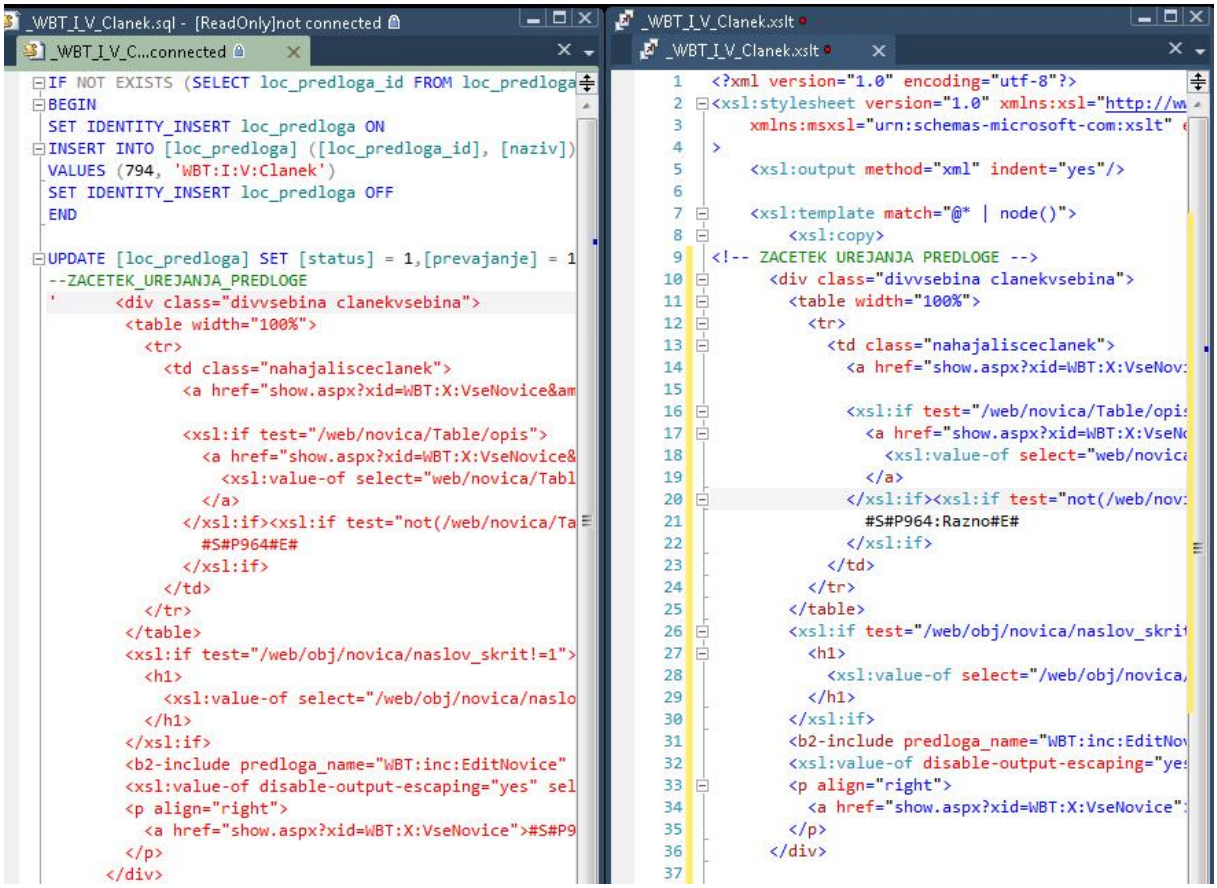
V primeru, da za polje "Repozitorij" izberemo možnost "LocPredloga", se prikažejo še dodatne možnosti, ki vplivajo na prikaz predloge v urejevalniku in prevajanje besedila. S poljem "Prevajanje" izberemo način prikaza prevedenega besedila. Možnost "Kode & Text" prikaže prevedeno besedilo kot kodo z dodanim dejanskim tekstom, ki se skriva za to kodo, možnost "Kode" pa prikaže le kode brez pripadajočega teksta. Naslednji dve polji sta povezani z jezikom. V prvega, "Jezik za prikaz", vpišemo id jezika, v katerega želimo, da se prevede besedilo poleg kode (če smo seveda izbrali to možnost) v urejevalniku. Uporabnik ima možnost izbire jezika tudi s spustnega seznama. V drugo polje, "Jeziki za prevod", pa vpišemo id-je jezikov (ločene z vejico), v katere želimo, da se prevedejo

besedila znotraj predloge XSLT. V kolikor pustimo to polje prazno, se besedila, ki so primerna za prevajanje, kljub temu zamenjajo s kodami, kar nam omogoča preprosto dodajanje dodatnih jezikov v prihodnosti poleg privzetega, če se pojavi potreba po tem.

Sledi vnosno polje, v katerega vpišemo iskalni niz, ki se upošteva pri iskanju želene predloge. Pod vnosnim poljem se nahaja seznam vseh predlog, ki ustrezajo iskalnemu nizu. Na začetku so to vse predloge, ki pa se ustrezno filtrirajo glede na vsak vpisan znak. Iskanje ne razlikuje velikih in malih črk, poleg tega pa metoda za iskanje primerja iskalni niz tudi znotraj imena predloge in ne le od začetka.

Pod seznamom vseh predlog, ki ustrezajo vnesenemu iskalnemu nizu, se nahajata dva gumba. Klik na levi gumb le odpre predlogo izbrano v seznamu, desni pa to predlogo rezervira in nam omogoči urejanje. Bolj podrobno bodo te akcije predstavljene v nadaljevanju.

Ko s pomočjo iskalnika najdemo ustrezno predlogo, ki jo želimo odpreti, imamo na voljo, kot že omenjeno, dva gumba. Oba v osnovi izvedeta enako akcijo, le da se pri kliku na gumb za urejanje predloge izvede še dodaten klic do strežnika TFS in rezervacija datoteke, ki vsebuje predlogo. Na Sliki 12 je na levi strani odprta izvorna datoteka, na desni strani pa rezultat, ki ga dobimo, če predlogo odpremo z našo razširitvijo.



Slika 12: Izvorna datoteka, ki vsebuje predlogo, na levi strani in ista datoteka odprta s pomočjo razširitve na desni strani.

Kot lahko vidimo, vsebuje izvorna datoteka le preprosto poizvedbo SQL, v kateri je definirana XSLT vsebina predloge. Čeprav so vse datoteke s poizvedbami strukturirane na isti način, sta poizvedbi dodana komentarja "--ZACETEK_UREJANJA_PREDLOGE" in "--KONEC_UREJANJA_PREDLOGE".

Komentarja označujeta začetek oziroma konec vsebine predloge XSLT in sta bila dodana za lažje razčlenjevanje ter predvsem za zmanjšanje morebitnih napak pri razčlenjevanju.

Rezultat odpiranja predloge je nov začasni dokument XSLT z istim imenom kot izvorna datoteka (razlikuje se le končnica datoteke). Ker poizvedba SQL ne vsebuje celotne strukture datoteke XSLT, temveč le osrednji, vsebinski del, lahko tudi tu opazimo podobna komentarja kot v izvorni datoteki, ki označujeta začetek oziroma konec vsebine XSLT, vsebovane v poizvedbi SQL izvorne datoteke. Pred oziroma za komentarjem je avtomatično dodana vsebina, ki je potrebna za zagotovitev pravilne strukture dokumenta. Celotna in pravilna struktura je potrebna, če želimo izkoristiti vse dodatne funkcionalnosti, ki nam jih ponuja urejevalnik XSLT dokumentov znotraj Visual Studia. Med pomembnejše in najbolj priročne lahko štejemo funkcijo samodokončevanja, barvanja teksta, kar nudi večjo preglednost in nenazadnje tudi avtomatično preverjanje pravilnosti strukture, kjer so nepravilnosti, kot je na primer manjkajoči zaključni element, podčrtane rdeče.

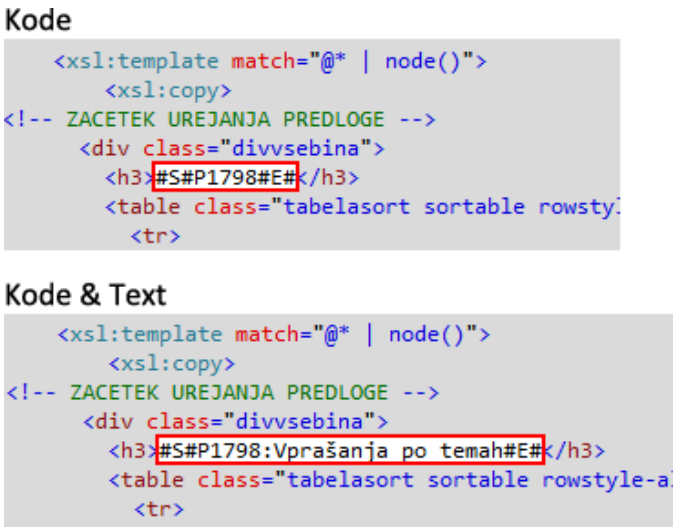
Za vsako predlogo, ki jo odpremo, ustvarimo nov razred `Predloga`, v katerem hranimo vse lastnosti predloge, ki jo potrebujemo. To so lahko na primer ime datoteke, ki je hkrati tudi ključ, po katerem iščemo že odprte predloge, celoten objekt tipa `FileInfo`, ki hrani vse lastnosti izvorne datoteke (pot, ime, končnica, mapa, v kateri je datoteka vsebovana, itd.), vsebina predloge, ali je datoteka shranjena, rezervirana itd. Poleg razreda `Predloga` imamo še razred `PredlogaBL`, katerega namen pa je manipulacija nad oziroma z razredom `Predloga`. Ta dva razreda sta ključna za izvajanje katerih koli akcij nad predlogami.

Izluščiti vsebino predloge iz poizvedbe SQL je relativno preprosto, prav tako tudi nastavitev ostalih lastnosti razreda `Predloga`. Ko s tem opravimo, moramo rezultat prikazati uporabniku kot nov dokument XSLT. Preden dokument dejansko prikažemo, je na vrsti še rezervacija datoteke, če smo odprli predlogo z možnostjo urejanja. Pri rezervaciji datoteke moramo poskrbeti le za ustrezno povezavo s strežnikom TFS z ustrezno avtentikacijo. Rezervacijo izvedemo s pomočjo objekta tipa `Workspace`, ki predstavlja tako imenovani delovni prostor okolja strežnika TFS, v katerem se med drugim beležijo tudi vse rezervirane datoteke za določen projekt v strežniku TFS. Delovni prostor skrbi za povezavo med datotekami na TFS in našim lokalnim okoljem. Delovnih prostorov imamo lahko seveda več, na primer za vsak projekt imamo lahko svoj delovni prostor. Za vsak projekt velja, da je lahko vsebovan le v enem delovnem prostoru naenkrat, kar nam omogoča, da lahko na podlagi datoteke, ki jo obdelujemo, preprosto poiščemo ustrezno delovno okolje v strežniku TFS.

V primeru, da uporabnik odpre predloge le za pregled, se rezervacija datoteke ne izvede. Če se uporabnik kljub temu odloči, da bo predlogo urejal, je dodana funkcionalnost, ki prestreže vsako morebitno spremembo vsebine dokumenta in v tem primeru prikaže pojavno okno. Če se uporabnik na tem mestu odloči in izvede rezervacijo datoteke, se sprememba dokumenta dovoli, v nasprotnem primeru pa se prekliče in vsebina dokumenta ostane nespremenjena. To funkcionalnost dosežemo s preverjanjem pritisnjene tipke, ki povzroči spremembo vsebine predloge (na primer tipka s črko "a", tipka delete itd.), hkrati s preverjanjem, ali je predloga rezervirana ali ne. Če pritisnjena tipka povzroči spremembo, datoteka ni rezervirana, in da se uporabnik ne odloči za rezervacijo le-te, enostavno prekličemo pritisnjeno tipko in s tem onemogočimo spremembo vsebine dokumenta brez rezervacije.

Pri branju predlog iz repozitorja "LocPredloge" moramo obravnavati še prevode in način prikaza le-teh. Če smo za prikaz prevodov izbrali samo način "Kode", potem le prikažemo predlogo takšno kot

je. Če pa je izbran način "Kode & Text", pa je potrebna dodatna obdelava vsebine predloge. V predlogi so vsa besedila, primerna za prevajanje, označena s posebnim nizom, ki vsebuje šifro, s pomočjo katere poiščemo ustrezno besedilo v podatkovni bazi. Bolj nazorna predstavitev razlike teh dveh načinov je predstavljena na Sliki 13. Struktura niza je vedno enaka, spreminja se le številski del niza, ki predstavlja šifro besedila v podatkovni bazi. Če izberemo način "Kode & Text" potem se poleg šifre prikaže še dejansko besedilo, ki se skriva za šifro, v jeziku, ki smo ga nastavili v polju "Jezik za prikaz".

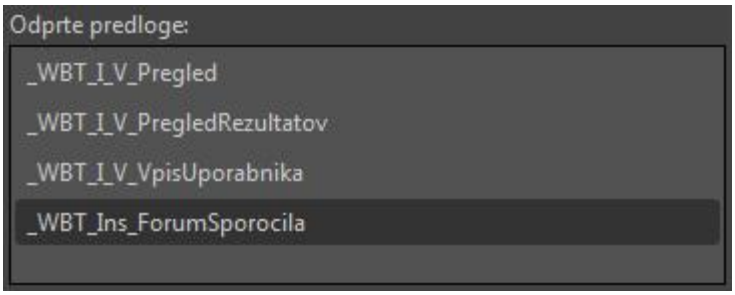


Slika 13: Rezultat prikaza prevedenega besedila v načinu "Kode" na vrhu in načinu "Kode & Text" spodaj.

Odpremo lahko seveda več različnih predlog zaporedoma, ne da bi prej zaprli prejšnjo predlogo. Kot že omenjeno, se za vsako odprto predlogo izdela nov primerek razreda `Predloga`, v katerem se hranijo vsi potrebni podatki in lastnosti, ki jih potrebujemo za delo s predlogami. Vse odprte predloge oziroma bolj natančno primerki objekta tipa `Predloga`, se dodajo v zbirko teh objektov. Zbirka se nato uporabi kot vir za povezovanje s podatki seznama odprtih predlog, kot je prikazano v naslednjem izrezku kode:

```
<ListView x:Name="lvPredlogeOdprte"
          ...
          ItemsSource="{Binding PredlogaAll}"
          SelectedItem="{Binding CurrentPredlogaBL.CurrentPredloga}"/>
```

Uporabo zbirke podatkov kot vir za podatkovno vezavo kontrol smo spoznali že v prejšnjem poglavju, medtem ko je koncept vezave lastnosti `SelectedItem` nov. S to povezavo dosežemo, da se ob spremembi aktivnega okna dokumentov znotraj Visual Studia v seznamu odprtih predlog ustrezno označi trenutno aktivna, kot je prikazano na Sliki 14. Vezava podatkov prav tako omogoča tudi avtomatsko osvežitev seznama v primeru, da odpremo novo ali zapremo že odprto predlogo.



Slika 14: Seznam vseh odprtih predlog. Označena predloga je trenutno aktivna.

Uporabnik ima tako boljši pregled nad tem, katera predloga je aktivna. To je pomembno zato, ker se nad aktivno predlogo izvajajo vse ostale akcije, kot so shranjevanje, izvajanje poizvedbe predloge nad podatkovno bazo in sprostitev datotek nazaj na TFS. Vse te akcije si bomo bolj podrobno ogledali v naslednjih poglavjih.

3.3.3. Shranjevanje in prevajanje predloge, izvedba poizvedbe nad bazo

Ko uporabnik konča z urejanjem predloge, je seveda naslednji korak shranjevanje. Shranjevanje predloge je v bistvu sestavljeno iz dveh delov – shranjevanje izvorne datoteke s končnico .sql in zagon poizvedb SQL, ki so vsebovane v tej datoteki, nad podatkovno bazo. Za to so na voljo trije gumbi, ki smo jih spoznali v prejšnjih poglavjih. Dva gumba sta namenjena eni oziroma drugi akciji, tretji gumb pa izvede obe akciji hkrati.

Ko datoteko shranjujemo, zapisana vsebina datoteke ni le vsebina XSLT, temveč celotna poizvedba SQL izvorne datoteke. Kot že omenjeno, se ob odpiranju vsake predloge kreira objekt tipa `Predloga`, kjer se hranijo in so dostopne vse lastnosti predloge, med katere spada tudi besedilo pred in za samo vsebino XSLT, ki jo urejamo v urejevalniku. To nam omogoča, da zelo preprosto sestavimo besedilo izvorne datoteke z novo vsebino dela XSLT. Izvorno besedilo je v bistvu kar poizvedba SQL, ki jo lahko direktno izvedemo nad podatkovno bazo. Več logike kot pri shranjevanju se skriva v pripravi vsebine predloge XSLT oziroma bolj natančno pri prevajanju besedila (seveda le v primeru, da gre za predlogo iz ustreznega repozitorija – "LocPredloga").

Preden je vsebina XSLT predloge pripravljena za shranjevanje, moramo izvesti še nekaj akcij, povezanih s prevajanjem besedila. Najprej moramo v predlogi poiskati besedila, ki so primerna za prevajanje. Ker je dokument XSLT v obliki XML, ga lahko kot takšnega tudi obravnavamo. To pomeni, da se lahko rekurzivno sprehodimo po vseh elementih in besedilo, ki je primerno za prevod, najdemo znotraj začetnega in končnega elementa. Obstajajo tudi primeri, ko je potrebno prevesti besedilo znotraj posameznih elementov, za kar pa žal ne obstaja preprosto generično pravilo. Za takšne primere obstaja dogovor, da okoli takšnega besedila postavimo poseben niz (#!#), kar nam olajša iskanje takšnega besedila.

Ko pridobimo vsa besedila, za katera potrebujemo prevod, jih posredujemo shranjeni proceduri v podatkovni bazi. Procedura najprej preveri, če posamezno posredovano besedilo že obstaja, in nam v tem primeru vrne šifro, ki je povezana s tem besedilom. V nasprotnem primeru se dodajo zapisi v ustrezne tabele, vrne pa se nova šifra, ki jo uporabimo v vsebini XSLT namesto besedila. Ker bi bilo iskanje in zamenjave le šifre znotraj predloge pri njeni dejanski uporabi precej težavno, če ne celo nemogoče, je dogovorjeno, da se okoli šifre doda še posebno zaporedje znakov: #S#P999#E#, kjer namesto števila 999 vstavimo šifro, ki jo vrne shranjena procedura. Vse to seveda velja le za besedila,

ki v predlogi še niso bila prevedena. Če je besedilo že povezano s šifro prevoda, potem te korake izpustimo, saj so že bili izvedeni.

Zamenjava besedila s šifro se izvede v vsakem primeru, tudi če nismo določili jezikov za prevod. V primeru, da smo vpisali enega ali več jezikov za prevod, pa se avtomatsko izvede tudi prevod besedila v vse te jezike. Za to se uporabi Googlov prevajalnik, katerega programski vmesnik je javno dostopen. Klic tega vmesnika je realiziran preko razširitvene metode (Extension Methods), ki so bile predstavljene v C# verziji 3.0. To so posebne statične metode, ki nam omogočajo dodajanje metod v obstoječe razrede, brez dedovanja iz že obstoječih. V naslednjem izrezku kode je predstavljena definicija takšne razširljive metode, ki doda metodo `Translate` vgrajenemu tipu `string`:

```
public static string Translate(this string str,
                               string sourceLanguage,
                               string targetLangugage)
{
    ...
    var req = (HttpWebRequest)WebRequest.Create(
    string.Format("http://ajax.googleapis.com/ajax/services/language/translate?v=1
.0&q={0}&langpair={1}%7C{2}",
        sts.Besedilo,
        sourceLanguage,
        targetLangugage));

    req.Proxy = Proxy;

    WebResponse response = req.GetResponse();
    var streamReader = new StreamReader(response.GetResponseStream());

    var serializer = new JsonSerializer();
    var translationResponse = (TranslationResponse)serializer.Deserialize(
                        new StringReader(streamReader.ReadToEnd()),
                        typeof(TranslationResponse));

    if (translationResponse.Data != null)
        sts.BesediloPrevod = translationResponse.Data.TranslatedText;
    else
        return "";
}
```

Razširitvene metode so definirane kot statične metode. Prvi parameter določa, nad katerim tipom metoda deluje, pred katerim pa mora biti modifikator `this`. Naša metoda poleg tega prejme še dva parametra, ki določata jezik izvornega besedila in jezik, v katerega želimo izvorno besedilo prevesti. V metodi lahko vidimo tudi klic programskega vmesnika Google za prevajanje in del obdelave rezultata odgovora. Sedaj ostane le še način uporabe te razširitvene metode, ki pa je prikazan v naslednjem izrezku kode:

```
tekst.Translate(CurrentPredloga.PredlogaIzvorniJezik.JezikKoda,
                jezikPrevod.JezikKoda);
```

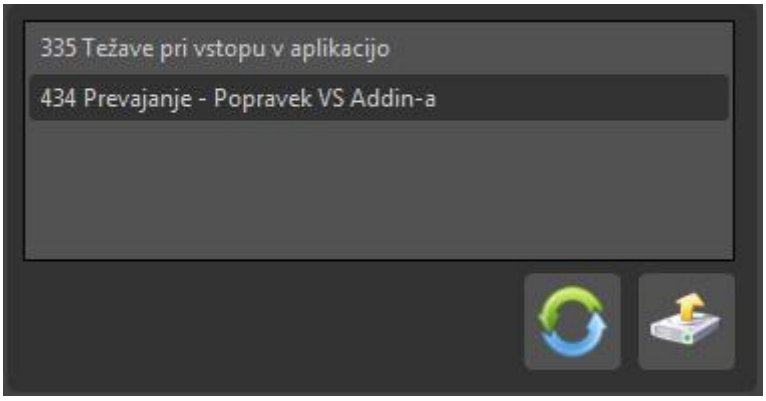
Spremenljivka `tekst` je tipa `string` in metodo `Translate` lahko kličemo kar neposredno z uporabo spremenljivke. Prav tako lahko vidimo, da metoda prejme samo dva parametra in ne tri, kot smo določili v definiciji metode. Prvi parameter v definiciji metode vsebuje modifikator `this`, kar pomeni,

da se za vrednost tega parametra vzame kar vrednost spremenljivke same, zato posredovanje le-te ni potrebno.

3.3.4. Sprostitev rezerviranih datotek v TFS in vezava sprememb na delovni nalog

Po uspešnem shranjevanju in posodobitvi podatkov v podatkovni bazi ostane še sprostitev rezerviranih datotek nazaj na TFS. Pri sprostitvi datotek moramo upoštevati pravilo, da se mora vsaka sprostitev, ki jo izvedemo, navezovati na en delovni nalog v strežniku TFS. To omogoča vodji projekta (in vsem ostalim, ki potrebujejo te podatke) lažji nadzor nad spremembami v projektu, saj ima tako vedno na razpolago pregled nad tem, kdo je izvedel spremembe, kdaj in zakaj oziroma v okviru katerega delovnega naloga je bila sprememba potrebna.

Glede na to, da ob sprostitvi datotek nazaj v strežnik TFS potrebujemo delovni nalog, na katerega se bodo zabeležile spremembe, moramo uporabniku prikazati seznam razpoložljivih delovnih nalogov. Ob kliku na gumb za sprostitev datotek se na uporabniškem vmesniku prikaže seznam ustreznih delovnih nalogov, kar je predstavljeno na Sliki 15. Razširjen uporabniški vmesnik vsebuje poleg seznama še dva gumba. S prvim seznam osvežimo, ker se ta ne posodobi avtomatično, ko je v strežnik TFS dodan nov delovni nalog. Drugi gumb pa izvede dejansko sprostitev datoteke nazaj v TFS in vezava le-te na izbran delovni nalog.



Slika 15: Seznam razpoložljivih delovnih nalogov, na katere se lahko veže sprostitev datotek.

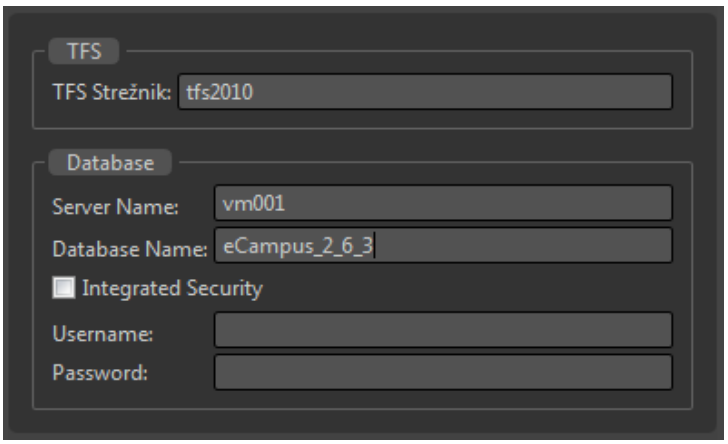
Za prikaz delovnih nalogov v seznamu je uporabljen že znani način povezovanja s podatki. Prav tako je relativno preprost način pridobitve delovnih nalogov. Dovolj je povezava do TFS in izvedba poizvedbe, ki ji določimo pogoja, da mora biti status delovnega naloga aktiven in da mora biti dodeljen trenutno prijavljenemu uporabniku. Kot že omenjeno, moramo najprej izbrati delovni nalog, da lahko izvedemo sprostitev datotek in vanj zabeležimo spremembe.

3.3.5. Nastavitve

Ker večina funkcionalnosti ne deluje pravilno oziroma sploh ne deluje, če razširitev nima definirane ustrezne povezave s strežnikom TFS in ustrezno podatkovno bazo, je dodana tudi možnost konfiguriranja teh lastnosti. Po kliku na gumb za urejanje nastavitev se prikažejo polja za pregled/vnos/spreminjanje nastavitev, kot je prikazano na Sliki 16.

Za strežnik TFS lahko določimo le ime, saj ostalo namreč ni potrebno. Strežnik se v našem primeru nahaja v lokalnem omrežju, zato je lokacija strežnika že vnaprej znana in bolj kompleksne določitve

poti niso potrebne. Prav tako ni potrebno določiti uporabnika za dostop do strežnika TFS, ker se za avtentikacijo uporabi kar uporabniški račun operacijskega sistema.



Slika 16: Polja za urejanje nastavitev.

Pri določanju strežnika SQL imamo odprtih več oziroma kar vse možnosti. Določimo lahko ime strežnika in podatkovne baze, nad katero se bodo izvajale poizvedbe. Poleg tega lahko določimo način avtentikacije. V primeru prijave z uporabo uporabniškega računa operacijskega sistema so polja za vnos uporabniškega imena in gesla onemogočena za vnos, saj teh podatkov pri takšnem načinu prijave ne potrebujemo. Če želimo izvesti prijavo z uporabniškim imenom in geslom, označimo primerna polja in vpišemo ustrezne podatke.

4. Sklepne ugotovitve

Glede na postavljene zahteve in pričakovanja, ki naj bi jih razširitev izpolnjevala, je bil projekt uspešno zaključen. Razvoj razširitve mi je osebno predstavljal kar precejšen izziv, saj sem se podal v nov teritorij, ki mi je bil popolnoma neznan. Prvič sem se srečal tako s koncepti in pristopi k razvoju razširitve kot tudi z novo tehnologijo, uporabljeno za gradnjo uporabniškega vmesnika. Presenečen sem bil tudi nad precej pomanjkljivo dokumentacijo razredov, metod, gradnikov, pristopov, ki se uporabljajo pri gradnji razširitev Visual Studia, kar ne velja za druge bolj standardne Microsoftove tehnologije. To je občasno pomenilo pristop s poskusi in napakami, ki je mučen, počasen, in kar je najpomembneje, rezultati niso optimalni. Prav tako bi si lahko precej olajšal gradnjo uporabniškega vmesnika z uporabo namenskega orodja Microsoft Expression Blend, ki omogoča bolj preprosto pozicioniranje, animacijo in določanje ostalih lastnosti grafičnih elementov v WPF.

Kot že rečeno, razširitev kljub številnim težavam in zapletom med razvojem ob zaključku deluje in vsebuje vse potrebne in zahtevane funkcionalnosti. Seveda ni ostalo le pri zahtevanih funkcionalnostih, saj so se med razvojem in testiranjem porodile nove ideje, ki bi uporabniku olajšale vsakodnevno uporabo. Pri testiranju se je, na primer, izkazala potreba po seznamu odprtih predlog, ki sem jo naknadno vključil v razširitev. Seznam v končni različici ni ostal le seznam, temveč je dobil še nekaj dodatnih funkcionalnosti. Ker se vse akcije izvajajo nad trenutno aktivnim oknom, v katerem urejamo predlogo, se v seznamu le-ta označi, kar zmanjša možnost napak in nejasnosti. Prav tako lahko iz seznama izberemo eno izmed odprtih predlog in s tem aktiviramo (prikažemo) okno za urejanje te predloge.

Razširitev so po končanem razvoju začeli in jo še vedno pri svojem delu uspešno uporabljajo razvijalci v našem podjetju. Kljub temu da je bilo dodanih precej dodatnih funkcionalnosti že med samim razvojem, se seveda vseeno pojavijo nove ideje in želje, ki bi še dodatno olajšale delo razvijalcev. V razširitev bi, na primer, lahko dodali še možnost dodajanja popolnoma nove predloge, prikaz zgodovine zadnjih 5 ali 10 odprtih predlog itd. Prav tako pa se mi je s spoznanjem razširitvenega modela za Visual Studio porodilo precej idej, ki bi mi lahko olajšale vsakdanje delo, zato jih bom v prihodnosti vsekakor poskusil uresničiti.

Literatura

[1] Wikipedia (2010) Integrated development environment. Dostopno na: http://en.wikipedia.org/wiki/Integrated_development_environment.

[2] Mike Snell, Lars Powers. Microsoft® Visual Studio® 2010 Unleashed. s.l. : SAMS, 2010.

[3] Nick Randolph, David Gardner, Chris Anderson, Michael Minutillo. Professional Visual Studio 2010. s.l. : Wiley Publishing, Inc., 2010.

[4] Robert Eisenberg, Christopher Bennage. Sams Teach Yourself WPF in 24 Hours. s.l. : Sams, 2009.

[5] Nathan, Adam. Windows Presentation Foundation Unleashed. s.l. : Sams, 2007.

[6] Novulo (2012) RAD (Rapid Application Development). Novulo. Dostopno na: http://www.novulo.com/Rad.aspx.

[7] Wikipedia (2010) Microsoft Visual Studio. Dostopno na: http://en.wikipedia.org/wiki/Microsoft_Visual_Studio.

[8] Nayyeri, Keyvan. Professional Visual Studio® 2008 Extensibility. s.l. : Wiley Publishing, Inc., 2008.