UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Vasilij Femec

UPORABNIŠKI VMESNIK ZA OSEBNO RAČUNOVODSTVO

Diplomska naloga
na visokošolskem strokovnem študiju

Mentor:
prof. dr. Dušan Kodek

Ljubljana, 2008

Povzetek

V tem diplomskem delu je opisan postopek nastanka uporabniškega vmesnika za aplikacijo Bilanca, ki je namenjena pregledu osebnih finančnih tokov. Gre za preprosto aplikacijo, ki iz podanih prihodkov in odhodkov izračuna trenutno stanje. Delo se prične s poglobljeno analizo najboljših možnih načinov, ki naj uporabniku čim bolj olajšajo uporabo. Temu sledi implementacija v okolju Delphi. Izkaže se, da tudi tako na pogled preprosta aplikacija, kot je Bilanca, zahteva od snovalca veliko kodiranja, če želi, da bo uporabniški vmesnik brezhiben.

Avtor ugotavlja, da bi uporabniki največkrat segli po orodju Microsoft Excel, če bi sami želeli rešiti problem, ki ga rešuje Bilanca. Ko analizira uporabniško izkušnjo v Excelu, na podlagi ugotovitev iz literature izpostavi nekaj možnih izboljšav, ki so nato implementirane v praktičnem delu. Nastala aplikacija naj bi tako uporabniku nudila večje udobje, saj je, za razliko od Excela, prilagojena točno določenemu problemu: prikazovanju osebnih finančnih tokov.

Diplomsko delo vsebuje pregled literature s področja uporabniških vmesnikov, določanje tipičnega uporabnika Bilance, izdelavo prototipa, analizo uporabniške izkušnje z Excelom, iskanje možnosti za izboljšave in potek izdelave aplikacije v okolju Delphi vključno s težavami, na katere je avtor naletel. Na koncu je opisano tudi testiranje aplikacije z naključnimi uporabniki, ki pokaže, da je bilo delo večinoma uspešno, podane pa so tudi nove ideje za izboljšave.

Ključne besede

uporabniški vmesnik, Delphi, Excel

Abstract

This diploma work describes a method for user interface development for an application Bilanca that is intended for a review of personal financial flows. It is a simple application that subtracts outcome from income and shows the current financial state. The work begins with a detailed analysis of the best possible user interface options that give the most comfortable user experience. This is followed by the implementation in a Delphi environment. The results show that even a simple application like Bilanca takes a lot of coding to produce a flawless user interface.

The author points out that users would normally use Microsoft Excel when trying to solve a problem like the one solved by Bilanca. As he analyses the Excel's user interface and results from the literature, some possible improvements are found and used in the practical implementation. The resulting application is focused on solving only the specific user problem, namely the management of personal financial flows, and gives a better user experience than Excel.

The diploma work contains a research of literature on user interfaces, definition of a typical user of Bilanca, prototype development, user experience analysis with Excel, new ideas for improving the user interface, and finally the development of Bilanca in a Delphi environment together with a description of difficulties that were found. It concludes with a description of the process of testing Bilanca with random users which shows that the work has been successful and suggests some new ideas for improvement.

Key words

user interface, Delphi, Excel

1 Uvod

1.1 Namen diplomskega dela

Cilja tega diplomskega dela sta bila

- raziskovanje področja uporabniških vmesnikov in
- uporabna aplikacija, ki ne bo služila le kot praktični del diplomskega dela, ampak jo bodo ljudje dejansko uporabljali tudi po mojem zagovoru.

Uporabniški vmesniki me zanimajo že od nekdaj. Že več let vsakodnevno uporabljam aplikacije za montažo zvoka in videa, kot sta npr. Adobe Premiere in Steinberg Cubase. Pri tovrstni programski opremi, ki uporabniku nudi veliko različnih možnosti, še posebej pride do izraza dobra organiziranost uporabniškega vmesnika. Pri montaži videa je značilno, da se uporabnikova dejanja ponavljajo, od uporabniškega vmesnika pa je odvisno, kako mu bo tovrstno delo olajšal. Šele ob primerjavi z drugimi programskimi orodji sem začel zares ceniti uporabniške vmesnike aplikacij, ki jih največkrat uporabljam. Začel sem postajati pozoren na razlike (največkrat malenkosti), ki so nek kos programske opreme naredile popularnejši od ostalih. Obe prej omenjeni aplikaciji sta najbolj razširjeni na svojem področju v svetu računalnikov PC.

Skozi dolgotrajnejšo uporabo programske opreme se sčasoma pojavijo ideje, kako določene lastnosti uporabniškega vmesnika izboljšati, dopolniti, poenostaviti itd. Ko sem na takšne ideje naletel, so se sočasno porodila tudi druga vprašanja, navajam samo tri med mnogimi:

- Kako sploh zasnovati tako kompleksen uporabniški vmesnik, ki rešuje toliko različnih problemov in kljub vsemu deluje zelo učinkovito?
- Koliko testov je potrebnih, da aplikacija končno ustreza določenemu spektru uporabnikov?
- Ali so podjetja pripravljena prisluhniti idejam in predlogom svojih uporabnikov?

Področje uporabniških vmesnikov je široko in se mnogokrat križa tudi z drugimi vedami, kot sta na primer psihologija in oblikovanje. Pri svojem delu sem se osredotočil na tiste dele področja, ki so me v zastavljenem času lahko pripeljali do izdelane končne aplikacije. Ker je bil čas največja ovira, sem mnogokrat moral prekiniti raziskovanje in se raje osredotočiti na izdelavo praktičnega dela. To pomeni, da bi z več razpoložljivega časa morda izdelal še boljši uporabniški vmesnik.

Da bi izdelano aplikacijo ljudje dejansko uporabljali, mora ta reševati nek njihov problem. Po možnosti mora ta problem reševati bolje od ostalih aplikacij, ki so na voljo. [1] Odločil sem se za reševanje problema nadzora nad lastnim finančnim stanjem. Kot študent sem se velikokrat znašel v položaju, ko mi je zmanjkalo denarja. Ko sem si začel svoje finančne tokove vestno zapisovati, sem se takšnim položajem lažje izognil. Tovrstne težave verjetno pestijo še mnoge študente in zato mislim, da bi nastalo aplikacijo z veseljem uporabljal še kdo poleg mene. Tako je nastala aplikacija z imenom Bilanca, ki uporabniku nudi celovit pregled nad finančnimi tokovi. Njen uporabniški vmesnik je zgrajen na osnovi pridobljenih znanj v sklopu tega diplomskega dela, kar pomeni, da

- je preprost za uporabo,
- vsebuje bližnjice za učinkovitejšo manipulacijo in
- je konsistenten z uporabniškimi vmesniki uveljavljnih aplikacij.

1.2 Opis tipičnega uporabnika

Tipičen uporabnik, na katerega cilja aplikacija Bilanca, je študent z nešteto listki z zapisi, komu je koliko dolžan in koliko je kdo dolžan njemu. Pričakuje nakazilo za študentsko delo, ki pa, kot ponavadi, spet zamuja. Da bi poračunal, ali si lahko privošči izlet med vikendom (kakšno je njegovo dejansko stanje), mora iz žepov in predalov izbrskati vse listke z dolgovi in jih vnesti v kalkulator. Pri tem obstajata nevarnosti, da ni našel vseh listkov in da je v kalkulator vnesel napačen znesek. Še največja tegoba pri vsem tem pa je časovna potrata.

Aplikacija Bilanca takšnemu uporabniku nudi shranjevanje vseh podatkov o dolgovih na enem mestu. Prav tako lahko v njej shrani podatke o nakazilih, ki jih pričakuje. Aplikacija seveda tudi sama sproti prikazuje pravilno stanje, vse skupaj pa za uporabnika pomeni nič več iskanja listkov in nič več računanja stanja. Poleg tega celovit pregled nad osebnimi finančnimi tokovi pozitivno vpliva na poravnavo dolgov. Dolgovi, ki jih shranjujemo v mislih in na listkih, ne izgledajo tako resno, kot takrat, ko so zbrani na enem, jasnem in preglednem mestu. Dolgove v mislih lahko namenoma ali nenamenoma spregledamo, da bi si olajšali vest. Bilanca je do takšnega ravnanja manj popustljiva. Poleg tega tipičen uporabnik uporablja najbolj razširjen operacijski sistem pri nas, to je Microsoft Windows.

1.3 Problemska domena

Če želi uporabnik sam ustvariti tabelo s prihodki in odhodki, s samodejnim izračunom stanja, bo praviloma segel po orodju Microsoft Excel. To pomeni, da mora aplikacija Bilanca uporabniku nuditi vsaj tako dober uporabniški vmesnik, kot ga ima Excel, imeti pa mora tudi določene izboljšave, ki ga bodo prepričale v trajnejšo uporabo.

Cilj je bil torej izdelati uporabniški vmesnik, ki bo boljši od Excelovega. Boljši uporabniški vmesnik pomeni [4]:

- manj klikov,
- večja prilagojenost problemu,
- učinkovitejša manipulacija vmesnika,
- manj naporna predstavitev podatkov.

Izdelati boljši vmesnik od tistega, ki ga izdela priznano podjetje, je velik izziv. Obstaja velika verjetnost, da je to podjetje izvedlo mnogo raziskav na področju uporabniških vmesnikov in vsakršno oddaljevanje od njihovih rešitev je tvegano početje. Tudi sicer je pravilo za gradnjo dobrih uporabniških vmesnikov, naj se snovalci zgledujejo po uveljavljeni programski opremi iz naslednjih dveh razlogov [4]:

- Za uveljavljeno programsko opremo ponavadi stojijo podjetja z dovolj sredstvi, da si lahko privoščijo raziskave.
- Uveljavljene programske opreme so uporabniki že navajeni, zato bodo aplikacije s podobnimi uporabniškimi vmesniki lažje uporabljali.

Na podlagi tega pravila sem se odločil, da bom izdelal uporabniški vmesnik, ki bo podpiral vse akcije, ki jih je vajen uporabnik Excela. Poleg tega pa bom dodal še nekaj dodatnih možnosti, ki naj bi izboljšale uporabnikovo izkušnjo. Med testiranjem prototipa, izdelanega v Excelu, so se pokazale tudi hibe Excelovega uporabniškega vmesnika in s tem možnosti za izboljšave. Naj poudarim, da je Excel multidisciplinarno orodje, ki zna rešiti veliko več problemov, kot samo odštevati odhodke od prihodkov. Ko govorim o hibah Excela, govorim le z vidika problemske domene Bilance. Če bi bil Excel namenjen zgolj prikazu osebnih finančnih tokov, bi imel verjetno tudi temu prilagojen uporabniški vmesnik. In moj cilj je bil ugotoviti, katere bi te prilagoditve bile in kako jih implementirati. Pomankljivosti Excelovega vmesnika, izboljšave in primerjave z vmesnikom Bilance so opisane v razdelku *2.2 Primerjava uporabniških vmesnikov.*

1.4 Uporabljeno znanje, pridobljeno na FRI

Znanje s fakultete, ki sem ga neposredno lahko uporabil pri izdelavi praktičnega dela diplomskega dela, sem pridobil pri naslednjih predmetih:

- **Osnove algoritmov in podatkovnih struktur 1 in 2**
  Uvod v objektno programiranje in sortirni algoritmi.
- **Razvoj aplikacij**
  Spoznavanje vseh faz nastanka aplikacije, od zajema zahtev, do uvedbe v uporabo.

- **Uporabniški vmesniki**
  Spoznavanje osnovnih gradnikov uporabniških vmesnikov, odnosov med njimi in njihova implementacija.

1.5 Znanje, pridobljeno na spletu

Med študenti FRI se v pogovornem jeziku čedalje pogosteje pojavlja fraza "Google programming" (programiranje s pomočjo brskalnika Google). Za vsak programski jezik, ki sem ga uporabljal tekom izobraževanja na Fakulteti za računalništvo in informatiko, sem vso potrebno pomoč našel na spletu. Tudi vse probleme, ki so se pojavili med programiranjem aplikacije, ki je del diplomskega dela, sem rešil s pomočjo virov na spletu. Splet postaja čedalje bolj dovršen vir znanja o programiranju. Na njem sem našel vse, od začetniških tečajev, do rešitev zahtevnejših problemov, vključno s komentarji in izboljšavami programerjev z vseh koncev sveta. Na spletu sem prav tako našel tudi mnogo nasvetov in smernic za oblikovanje dobrih uporabniških vmesnikov. Pri programiranju mi je bila v največjo pomoč stran na naslovu *http://delphi.about.com.*

1.6 Znanje, pridobljeno v literaturi

S knjigami sem pridobil nekaj dodatnega znanja na področju izgradnje uporabniških vmesnikov. Naj naštejem nekaj bistvenih ugotovitev, ki so skozi izdelavo diplomskega dela služila kot izhodišča [3, 4]:

- Uporabniški vmesnik je dobro oblikovan takrat, ko se program obnaša točno tako, kot pričakuje uporabnik.

- Pri načrtovanju uporabniških vmesnikov se moramo zatekati k najpreprostejšim rešitvam - le te naredijo vmesnik res dober.

- Velikokrat nam ravno mnenje naključnih uporabnikov razkrije pravo pot do idealnega vmesnika.

- Šest je zadostno število uporabnikov, ki testirajo uporabniški vmesnik. Že s šestimi testnimi uporabniki dobimo dokončno sliko o tem, kaj vmesniku manjka in kaj je zadovoljivo. Pri večjem številu preizkuševalcev se rezultati začnejo očitno ponavljati.

- Kadar koli uporabniku ponujamo možnosti, od njega zahtevamo odločitev.

- Uporaba dobro oblikovanih izdelkov je jasna že takrat, ko jih samo gledamo. To ne velja samo za uporabniške vmesnike, ampak v industriji nasploh. Uporaba video kamere je pri dobri oblikovni zasnovi jasna že na pogled.

- Konsistenca razbremenjuje uporabnikov spomin. Ljudje se bodo hitreje navadili na aplikacijo, ki ima podoben uporabniški vmesnik, kot ostale aplikacije.

- Oblikovalska kreativnost prej škoduje uporabnosti vmesnika, kot pa ji koristi.

- Uporabniki ob uporabi naše aplikacije nimajo pri sebi priročnika. Tudi, če bi ga imeli, ga ne bi uporabljali.

- Ne le, da uporabniki ne berejo priročnikov, ne berejo skoraj ničesar. Branje je za večino uporabnikov napor. Zato jih ne poskušajmo poučevati o načinu uporabe našega programa, ampak jih raje vodimo z drugimi prijemi, kot so npr. metafore, ikone, vizualni ključi, ...

- Misliti moramo tudi na uporabnike, ki ne znajo natančno upravljati z miško - teh je več, kot si mislimo.

- Ljudje si veliko lažje zapomnimo stvari, če nas nanje nekaj spominja.

- Če program od nas želi ime datoteke, ga bomo raje poiskali na seznamu, kot pa v lastnem spominu in ga potem vpisali.

- Ne zahtevajmo od uporabnika, da si zapomni stvari, ki si jih lahko zapomni računalnik.

Spoznal sem tudi načrtovanje na podlagi aktivnosti. Gre za načrtovanje, ki razvijalca sili, da ne razmišlja na način, ki mu je najbolj domač (kaj vse mora aplikacija podpirati, da bo uporabnik lahko uspešno opravil svoje delo), ampak da se postavi v vlogo uporabnika in si odgovarja na vprašanja, katere so aktivnosti, ki jih bo uporabnik intuitivno želel izvesti, da po najhitrejši poti pride do rezultata. To vrsto načrtovanja je zasnoval Microsoftov razvijalec Mike Conte v timu za razvoj orodja Microsoft Excel. Rezultat takšnega načrtovanja je manj zapleten vmesnik, ki ni preplavljen z možnostmi, ampak je bolj preprost in uporabniku bolj prijazen. Gre za učinkovit način, kako razvijalcem delno preprečiti, da bi precenili sposobnosti uporabnika - s to težavo se podjetja pogosto srečujejo.

Naslednja metoda je izumljanje namišljenih uporabnikov. Gre za metodo, pri kateri si razvijalec izmisli neko osebo, ji določi poklic, izobrazbo, znanje v računalništvu, operacijski sistem, ki ga uporablja (če sploh ve, kaj je to) itd. Šele ko je ta oseba natančno definirana, jo v mislih postavimo pred našo aplikacijo in si skušamo zamisliti, kako bi se odzvala na uporabniški vmesnik. Skušamo si izmisliti več navideznih oseb, ki so si med seboj čim bolj različne - od skoraj računalniško nepismenih, do izkušenih uporabnikov. Če bodo vse osebe znale uporabljati naš program, smo zasnovali dober uporabniški vmesnik.

Šest korakov, ki vodijo do dobrega uporabniškega vmesnika:

1. Ustvarimo nekaj namišljenih uporabnikov.

2. Ugotovimo, katere so glavne aktivnosti aplikacije (iz uporabnikovega stališča).

3. Ugotovimo, kako uporabnik pričakuje, da bo prišel uspešno do cilja v našem programu.

4. Skiciramo vmesnik.

5. Skico znova in znova poenostavljamo, dokler ne ustreza vsem namišljenim uporabnikom.

6. Opazujemo prave uporabnike, kako uporabljajo naš program in popravimo dele vmesnika, kjer je prišlo do zapletov.

Ob tem je treba predstaviti tudi delno nasprotujoč vidik [1], ki zagovarja stališče, da se na opazovanje uporabnikov pri delu ni mogoče močno zanašati. Kadar uporabniki vedo, da so opazovani, njihovo delo ne poteka tako, kot ponavadi. Trudijo se, da bi naredili čim manj napak, ravno pristne napake pa so tisto, kar bi pri njih želeli videti. Zato se mora razvijalec zavedati, da kljub vsemu sam nosi največjo odgovornost za dober uporabniški vmesnik.

2 Snovanje uporabniškega vmesnika

2.1 Prototip in testiranje

Prototip sem izdelal z orodjem Microsoft Excel. Excel je, po pogovoru z naključnimi uporabniki, tudi sicer največkrat izbrano orodje, kadar želijo uporabniki sami rešiti problem, ki ga rešuje Bilanca.

Prototip je najprej vseboval tabelo s štirimi stolpci:

- dolgovi (kredit),
- mesečni odhodki (kredit),
- terjatve (debet),
- prihodki (debet).

Poleg tega je vseboval še posebna polja, ki so služila shranjevanju podatkov o bančnih računih: transakcijski, varčevalni, ...

	Odhodki skupaj	**820**							Na dan lahko zaprviš:
	Prihodki skupaj	**2668**		Stanje	1848				5,063013699
MESEČNI	ODHODKI		DOLGOVI		PRIHODKI		TERJATVE		
150	Stanovanje		0	Blagajna za video	200	Aleš - videospot			
21	Vzajemna		220	Nejc - za kamero					
30	Mobitel		500	Mama					
40	Bencin		10	Andrej - boni	200	Osebni račun	13	Jan	
30	Registracija				15	Denarnica			
10	Avto dodatno		5	Nejc - pizza					
10	Frizer		85	Janez - vikend	1650	Projekt za Simobil	50	Sestra	
30	Oblačila								
25	Badminton		.400	Diploma	.100	Sklad za dolgove			
30	Počitnice		.200	Bančni limit					
80	Zabava				0	Varčevalni sklad			
					0	Mesečni skladi			
			.150	Stanovanje					
			.21	Vzajemna	500	Nona			
					40	Kritje stroškov za spot			
			.200	Varčevalni sklad					
			0	Mesečni skladi					

Slika 2.1: Začetni prototip v Excelu.

Več mesečna uporaba prototipa je pokazala, da so štirje stolpci odveč. Prav tako posebna polja za bančne račune. Od vseh elementov sem ohranil samo dva stolpca: odhodke in prihodke. Odvečne elemente sem odstranil predvsem zato, da bi imel uporabniški vmesnik manj elementov. S tem sem računal na večjo uporabnost in privlačnejši videz uporabniškega vmesnika. Elemente sem odstranil kljub temu, da sem se nanje navadil in so popestrili videz aplikacije. Uporabniški vmesnik s samo dvema stolpcema, namesto štirih, je bolj v skladu z zastavljenimi izhodišči predvsem z vidika prvega vtisa na uporabnika - tabele so same po sebi neprivlačne in uporabniku dajejo vtis, da od njega pričakujejo veliko dela.

	Odhodki skupaj	**810**	
	Prihodki skupaj	**1200**	
	Stanje	**390**	
ODHODKI		**PRIHODKI**	
0	Blagajna za video	200	Aleš - videospot
220	Nejc - za kamero		
500	Mama		
		650	Osebni račun
5	Nejc - pizza	0	Denarnica
85	Janez - vikend		
		350	MIDI melodije
.400	Diploma		
.200	Bančni limit	0	Varčevalni sklad
		0	Mesečni skladi
.150	Stanovanje		
.21	Vzajemna		

Slika 2.2: Končni prototip v Excelu.

Na začetku je bil namen aplikacije, poleg pregleda nad trenutnim stanjem, uporabniku nuditi tudi pregled nad njegovo celotno porabo. Od tod tudi ideja o stolpcu za mesečne odhodke.

Mesečni odhodki	
Prehrana	150
Zabava	80
Šport	90
Internet	40
Telefon	30
...	

Tabela 2.1: Mesečni odhodki so se skozi testiranje izkazali za nepomembne.

Uporabnik bi si v tem stolpcu lahko beležil vse možne porabe sredstev in si tako ustvaril natančno sliko o tem, za katere stvari in storitve porabi svoj denar. Vendar se je s tem pojavila težava, kako te stroške prikazati v stanju. Če so se obračunali sproti, so obremenili stanje vnaprej (še preden je do njih dejansko prišlo), kar je slabo vplivalo na

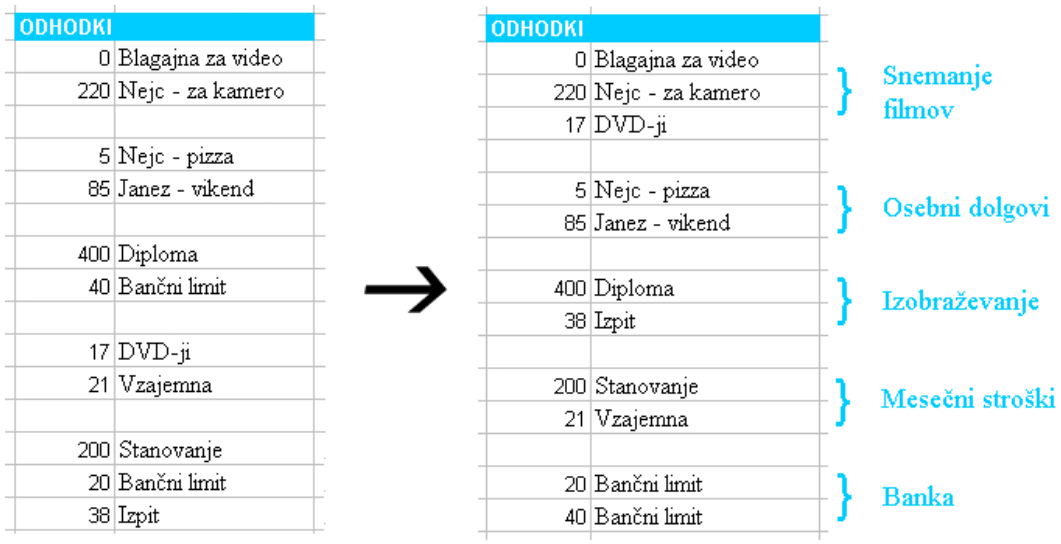
uporabnikovo moralo. Stanje je bilo včasih celo negativno že takoj, ko je uporabnik vnesel svojo plačo. Poleg tega sem opazil, da sem v ta stolpec sčasoma vpisal vse vrste svojih mesečnih odhodkov, njihov obračun onemogočil, potem pa sem nanje pozabil. Stolpec mesečnih odhodkov se je tako izkazal za nebistvenega.

2.2 Primerjava uporabniških vmesnikov

Med uporabo prototipa so se pokazale naslednje potrebe po izboljšavah:

2.2.1 Premikanje polj v Excelu

Vnosi v stolpcih odhodkov in prihodkov se pogosto spreminjajo. Ko je stolpec prazen, ni težav z grupiranjem vnosov - sorodne odhodke oz. prihodke pač vpišemo zaporedno. Ko pa je v stolpcu veliko razdrobljenih vnosov, se sčasoma pojavi potreba po sortiranju. Edini način, kako v Excelu urediti vnose v novem vrstnem redu, je z uporabo akcij izreži in prilepi.



Slika 2.3: Razdrobljeni vnosi in grupirani vnosi.

Takšno urejanje je zamudno in nepriročno. Veliko priročnejša bi bila uporaba akcije vleci in spusti, ki pa je Excel ne podpira.

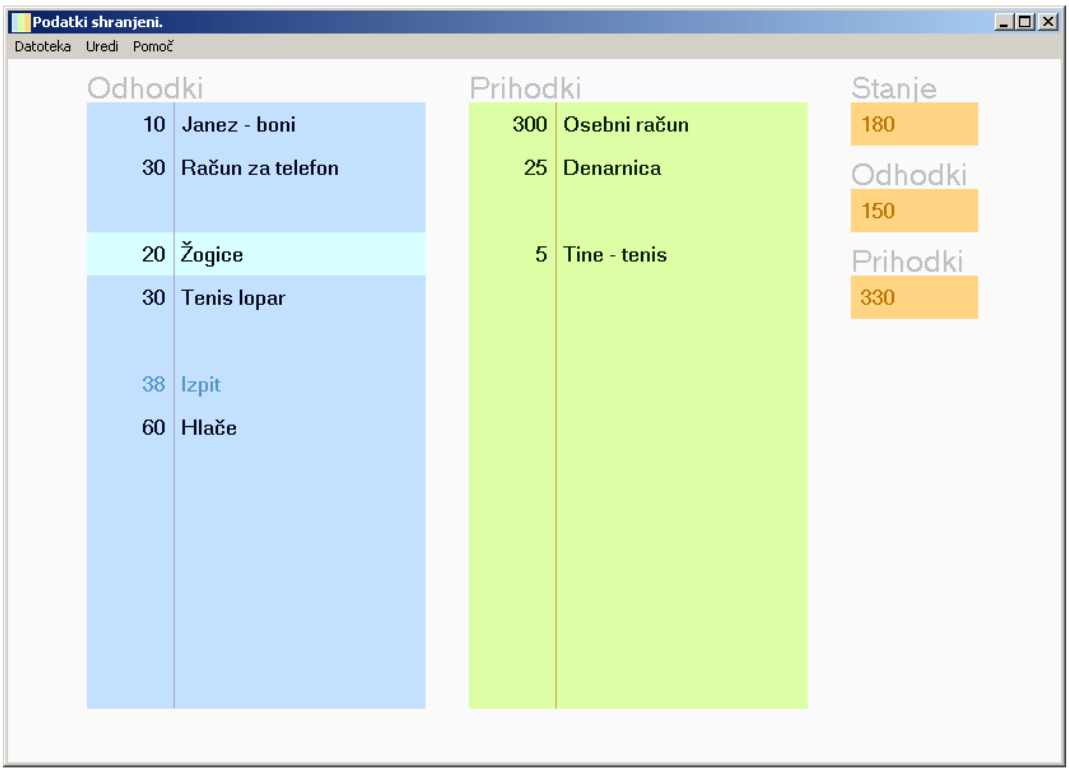
2.2.2 Upoštevanje polj v Excelu

Stanje se izračuna tako, da se vse vrednosti v stolpcu odhodkov odštejejo od vseh vrednosti v stolpcu prihodkov. Včasih se na primer pojavi kakšen strošek, ki si ga želimo zabeležiti, a ne želimo, da bi vplival na stanje. Primer takšnega stroška je dolg prijatelju, ki ga nočemo pozabiti, poravnati pa ga nameravamo šele čez nekaj mesecev. Excel zahteva od uporabnika kar nekaj truda, če želi iz izračuna stanja izvzeti eno samo polje iz stolpca stroškov.

10	dolg Andreju
.85	dolg Katarini

Slika 2.4: Polje dolg Katarini ni vračunano v stanje.

Kot je razvidno s slike, sem se na koncu zatekel k naslednji rešitvi: znesek, ki ga nisem želel upoštevati, sem opremil z neštevilskim znakom. Tako je Excel smatral, da ne gre za številčno vrednost in jo je zato izpustil iz končnega izračuna stanja. Seveda pa to ni elegantna rešitev, ki jo je bilo prav tako potrebno izboljšati.

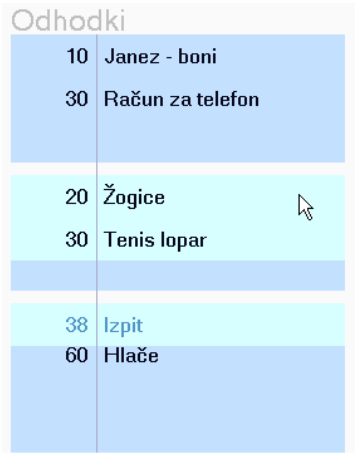


Slika 2.5: Končna podoba aplikacije Bilanca.

Aplikacija Bilanca navedene pomanjkljivosti odpravlja na naslednje načine:

2.2.3 Premikanje polj v Bilanci

Urejanje zaporedja polj je možno z vlečenjem in spuščanjem. S pritiskom tipke shift lahko uporabnik izbere več polj hkrati in jih simultano premika.



Slika 2.6: Premikanje polj z vlečenjem.

2.2.4 Upoštevanje polj v Bilanci

Uporabnik lahko določi, ali se vrednost upošteva v stanju ali ne, na dva načina:

- z enim samim klikom (srednji miškin gumb) ali
- z izbiro v dvižnem meniju (z desnim klikom na polje).

Slika 2.7: Polje Izpit se ne upošteva v izračunu stanja.

2.3 Ostale lastnosti uporabniškega vmesnika Bilance

Bilanca nudi še naslednje možnosti, ki olajšajo uporabnikovo izkušnjo:

2.3.1 Povečava pogleda

Spletni brskalniki so postali lep zgled za razbremenjevanje uporabnikovih oči. Kadar uporabnik z zaslona natančno prebira besedilo oz. podatke, večji pogled močno izboljša uporabniško izkušnjo. Po vzoru brskalnika Mozilla Firefox, tudi Bilanca omogoča povečavo in pomanjšanje pogleda s pomočjo tipk *CTRL* + in *CTRL* -.

2.3.2 Bližnjice

Bilanca ima na voljo nekaj bližnjic, ki bolj izkušenim uporabnikom omogočajo še hitrejšo in učinkovitejšo uporabo aplikacije. Vse pomembne operacije nad polji (premikanje med polji, vpisovanje vrednosti, brisanje, določanje, ali naj se polje upošteva v končnem izračunu) je možno izvajati s tipkovnico.

Prav tako aplikacija pozna tudi nekaj bližnjic z miško. V razdelku *2.2.4 Upoštevanje polj v Bilanci* sem že opisal klik s srednjim gumbom, poleg tega pa tudi dvojni klik na formo v bližini stolpca doda novo polje ustreznemu stolpcu. To je hitrejša alternativa izbiri v dvižnem meniju, ki ga prikličemo z desnim klikom na formo.

2.3.3 Odpuščanje napak

"Hitro lahko tipkam zato, ker vem, da imam na voljo tudi tipko za brisanje."
- Paul Butcheit, Google [2]

Če hočemo uporabniku omogočiti hitro uporabo aplikacije, mu mora uporabniški vmesnik odpuščati napake vseh vrst. Aplikacija Bilanca uporabniku omogoča razveljavitev izvedenih akcij nad polji: pisanje, brisanje, odstranjevanje in dodajanje polj, ... Izvedba razveljavitve zadnjega uporabnikovega dejanja je opisana v razdelku *3.8 Razveljavitev zadnjega dejanja.*

2.4 Prva uporaba aplikacije

Prvi vtis aplikacije na uporabnika je zelo pomemben, saj je od njega odvisno, ali se bo uporabnik sploh še kdaj vrnil. Prazna tabela, za katero mora uporabnik sam ugotoviti, čemu služi, zagotovo ne vodi v najudobnejšo uporabniško izkušnjo. Zato se aplikacija Bilanca ob prvi uporabi samodejno napolni z nekaj podatki, ki uporabniku prikažejo osnovno poanto aplikacije že pred njegovim prvim klikom. [1] Ob prvi uporabi je že nekaj vnešenih odhodkov in prihodkov, ki imajo dva namena:

- uporabniku namesto praznih polj prikazujejo, kako uporabniški vmesnik izgleda v uporabi in
- uporabniku sugerirajo, kakšne vrste podatkov lahko vpisuje v polja.

Tako predizpolnjena polja skušajo uporabnika spodbuditi, da pomisli na lastne prihodke oz. odhodke in jih vpiše v polja. Poleg tega je eno predizpolnjeno polje izključeno iz končnega izračuna stanja, kar pomeni, da je obarvano drugače, kot ostala polja. S tem je uporabnik spodbujen k nadaljnjemu raziskovanju - z desnim klikom na polje se dvigne dvižni meni, ki mu nudi odgovor na vprašanje, zakaj je polje na videz drugačno od ostalih.

Kot verjetno mnoge, je tudi mene najprej spreletela misel, da aplikaciji dodam navodila za uporabo, ali vsaj seznam vseh bližnjic. Vendar bi takšen dodatek pomenil, da uporabniški vmesnik ni dovolj dober, da bi uporabnik uspel sam intuitivno ugotoviti, kako se ga uporablja. Zato sem se odločil, da ne bo priloženih nobenih navodil. Namesto pisanja navodil sem se raje lotil ponovnega razmisleka, kako vmesnik oblikovati tako, da jih ne bo potreboval.

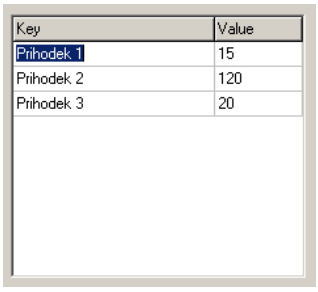
3 Implementacija

3.1 Izbira programskega jezika

Da bi si zagotovil kar največjo svobodo pri ustvarjanju, sem seveda moral poseči po nizkonivojskem jeziku. Če sem želel narediti boljši uporabniški vmesnik od Excelovega, sem ga moral premagati v detajlih. V izboru sta bili okolji Visual C++ in Delphi. Odločil sem se za slednje, ker sem že od nekdaj programiral v Pascalu in sem s tem upal na manj začetnih zapletov. Delphi podpira izgradnjo lastnih komponent, kar se mi je na začetku zdela prava smer za začetek raziskovanja, a sem kasneje elemente vmesnika sestavil kar iz že obstoječih komponent.

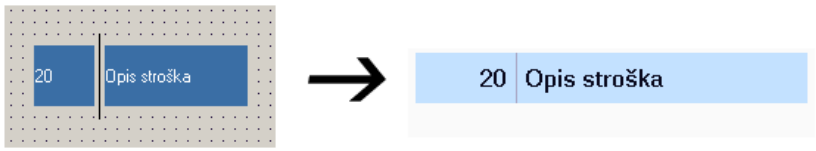
3.2 Iskanje primernih komponent za izvedbo uporabniškega vmesnika

Delphi pozna tabelo s ključi in vrednostmi, a je dokaj okorna za uporabo. Vrstnega reda polj se ne da spreminjati z vlečenjem in spuščanjem (enako kot v Excelu), kar bi pomenilo, da bi to možnost moral dodati sam. Četudi bi uspešno dodal možnost vlečenja in spuščanja, bi še vedno pogrešal nekatere druge lastnosti, na primer tiste, ki vplivajo na vizualno podobo tabele. Ob tem pa se že pojavi vprašanje, ali ni bolje ustvariti svoje komponente, ki se bo v celoti obnašala v skladu z zastavljenimi izhodišči. Tabelo s ključi in vrednostmi sem tako opustil.



Slika 3.1: Delphijeva tabela s ključi in vrednostmi.

Delphi omogoča pisanje lastnih komponent. To možnost sem ovrgel zaradi spoznanja, da lahko vsem izhodiščnim zahtevam ugodim s sestavljanjem že obstoječih komponent. Tako sem eno polje v stolpcu preprosto sestavil iz dveh label - ena nosi vrednost, druga pa opis odhodka oz. prihodka.

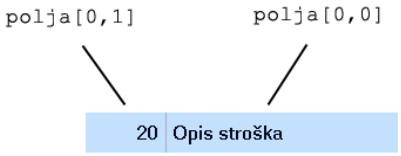


Slika 3.2: Dve spojeni labeli, ki z nekaj popravki atributov tvorita polje. Čez njiju poteka črta ločnica.

S tem se je iskanje ustreznih komponent zaključilo. Med izgradnjo uporabniškega vmesnika sem preizkusil tudi gumbe, ki pa so me soočili z nenavadno težavo. S klikom na gumb se osredotočenje (angl. focus) prenese z glavne forme na gumb. Ukaz, naj se osredotočenje prenese nazaj na formo (`form1.setfocus`) v mojih poskusih ni deloval. Tako gumbov sploh ne bi mogel uporabiti, tudi, če bi jih želel. Če forma izgubi osredotočenje, namreč ne zazna več dogodkov na tipkovnici, kar pa močno ohromi uporabo. Ta problem zagotovo je rešljiv, vendar rešitve zanj v okviru tega dela žal nisem uspel najti.

3.3 Dinamično kreiranje komponent

Število polj v stolpcu ni končno, možno jih je brisati in kreirati, ko je program v teku.



Slika 3.3: Predstavitev polja s podatkovno strukturo tabela (angl. array).

Dinamično kreiranje labele brez starša (nil) s klicem konstruktorja razreda TLabel:

```
polja[0, 0] := TLabel.create(nil);
```

Dinamično kreiranje prinaša tudi nevarnost, da po nepotrebnem porabimo preveč pomnilnika. Zato je potrebno vsako instanco labele uničiti, ko je ne potrebujemo več. To storimo s klicem metode free:

```
polja[0, 0].free;
```

Kljub veliki previdnosti še vedno lahko pride do prevelike porabe pomnilnika, zato je priporočljivo uporabiti blok try ... finally.

```
try
  begin
     polja[0, 0] := TLabel.create(nil);

     ...

  end
```

```
finally
     polja[0, 0].free;
end;
```

Bloka try ... finally nisem uporabil zaradi nepriročnosti. Pri uničevanju instanc se zanašam na števec, ki kaže trenutno število obstoječih label. Ob vsakem kreiranju nove labele se števec ustrezno poveča, ob vsakem uničenju pa pomanjša. Zaradi majhnega števila polj v stolpcih prihodkov in odhodkov (maksimalno 200) je s tem ob morebitnih napakah v kodi tudi dokaj majhna nevarnost za prekoračitev zmožnosti pomnilnika.

3.4 Premikanje polj z akcijo vleci in spusti

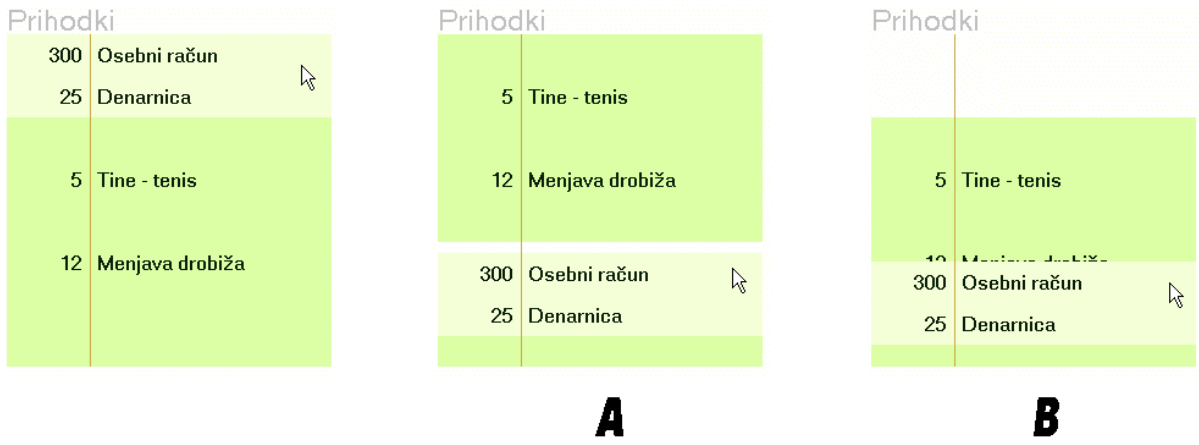
Vsaka komponenta zazna naslednje miškine dogodke:

- OnMouseClick,
- OnMouseDown,
- OnMouseUp,
- OnMouseMove,
- OnMouseDblClick.

Z uporabo dogodkov OnMouseDown, OnMouseUp in OnMouseMove sem realiziral vlečenje in spuščanje. Ob dogodku OnMouseDown se nastavi zastavica `LabelDrag := True`. Pod tem pogojem vsa označena polja sledijo miškinemu kazalcu (samo vertikalna smer), vse dokler ne pride do dogodka OnMouseUp.

3.5 Spreminjanje zaporedja polj v stolpcu

Ko uporabnik vleče označena polja čez ostala polja, se njihovo zaporedje sproti prilagaja. Tako ima uporabnik ves čas jasno predstavo o tem, kakšno bo končno zaporedje, ko označena polja spusti. Alternativa, ki bi bila lažje izvedljiva, je ta, da vsa neoznačena polja mirujejo, vse dokler uporabnik ne spusti označenih polj. Vendar je takšno urejanje uporabnika puščalo v dvomih, zato sem se odločil za izvedbo zahtevnejše različice, kar pa je prineslo kar nekaj težav.



Slika 3.4: Sortiranje polj med vlečenjem.
A - Sprotno prilagajanje neoznačenih polj.
B - Neoznačena polja med vlečenjem mirujejo.

Algoritem, ki skrbi za pravilno zaporedje polj, je podoben normalnemu vstavljanju. Zaradi majhnega števila polj časovna zahtevnost tega algoritma ne povzroča nobenih zakasnitev, ki bi poslabšale uporabniško izkušnjo. Vsakič, ko uporabnik med vlečenjem premakne miško,

- se premaknejo označena polja sorazmerno z miškinim kazalcem,
- glede na velikost premika se določi njihovo novo mesto v zaporedju in
- sortirni algoritem vsa ostala polja prilagodi novemu stanju.

Kljub temu, da sem za ta segment izvorne kode porabil največ časa, včasih še vedno pride do manjših napak pri sortiranju. Pri tako nizkonivojskem jeziku, kot je Delphi, je treba paziti na vse najmanjše podrobnosti. Segment kode, ki sem ga razvijal več dni, sem kasneje zavrgel in napisal novega, ker starega nisem mogel prilagotiti primeru, ko uporabnik med vlečenjem sunkovito premakne miško. Nov sortirni algoritem se s to težavo spopada dosti bolj uspešno, a žal še vedno ne brez napak. Do napak pride ob kombinaciji hitrih gibov z miško, medtem ko pogled samodejno drsi po formi (angl. auto scrolling). Sicer pa uporabnik, ki ne preizkuša meja zmogljivosti uporabniškega vmesnika, načeloma ne bo opazil nobenih hib.

3.6 Vnos opisa in vrednosti

Labela žal ne dovoli, da bi uporabnik vanjo vpisal vrednost ali besedilo. Komponenta, ki to dovoljuje, je vpisno polje (angl. edit box). Ko želi uporabnik Bilance vpisati vrednost, labelo prekrije vpisno polje, in uporabniku ponudi vnos. Porodi se vprašanje, zakaj sploh uporabljati labele, če bi lahko namesto njih uporabili vpisno polje. Odgovor leži v atributih obeh komponent. Ti se namreč med seboj rahlo razlikujejo. Na primer, vpisno polje nima možnosti desne poravnave besedila, medtem ko labela to možnost podpira. Ta možnost je natanko tisto, kar sem potreboval pri oblikovanju polj - vrednost je namreč poravnana desno, ob črti ločnici.

Slika 3.5: Labelo, ki prikazuje vrednost polja, prekrije vpisno polje, ki ne podpira desne poravnave besedila.

Ko uporabnik vpisuje vrednost v polje (vpisno polje prekrije labelo) se poravnava vnosa spremeni iz desne v levo. Ideje, kako bi to težavo zlahka odpravil, v procesu nastajanja kode nisem dobil. Ker ta oblikovna nekonsistenca ni preveč moteča (nanjo sem se hitro navadil), sem se odločil, da ji ne bom več posvečal pozornosti.

Aplikacija ob vsakem vnosu vrednosti preveri, ali je bila dejansko vnešena vrednost - torej samo številke. Če uporabnik vpiše neštevilski znak, obvelja prejšnja vrednost. Pri uporabi vpisnega polja sem naletel še na eno prepreko. Uporabnik Excela je navajen prehajati z enega polja tabele na drugega s tipko *TAB*. Kljub temu, da vpisno polje podpira dogodka OnKeyDown in OnKeyPress, nisem uspel zaznati pritiska tipke *TAB*. Poskusil sem na enak način, kot sem to uspešno storil v kontekstu forme. Dodatno začudenje je povzročilo to, da sem lahko brez težav zaznal pritiske na vse ostale tipke, vključno s črkami in številkami.

3.7 Shranjevanje podatkov

Delphi ponuja veliko možnosti za manipulacijo z datotekami. Najprej sem podatke nameraval shranjevati v ASCII datoteki, kasneje sem se odločil kar za ini datoteko, saj Delphi prinaša knjižnico IniFiles, ki delo s tovrstnimi datotekami močno olajša. Aplikacija Bilanca uporablja dve podatkovni datoteki:

- ini datoteko s podatki o nastavitvah glavnega okna (forme) in
- ini datoteko s podatki o stolpcih odhodkov in prihodkov.

Ini datoteke so uporabne predvsem zaradi uporabe kategorij in ključev. Z enim stavkom pridemo do podatka, ki se lahko nahaja kjer koli v datoteki. Na primer, stavek

```
Cena := IniFile.ReadInteger('Cenik', 'Sesalnik', -1);
```

vrne vrednost cene sesalnika v kategoriji cenik. Če ključ ali kategorija ne obstajata, je vrnjena vrednost, določena s tretjim argumentom metode, v našem primeru -1.

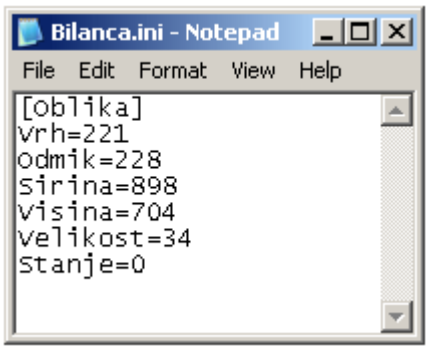
Del procedure PodatkiShrani, ki shranjuje podatke o odhodkih in prihodkih, prikazuje uporabo knjižnice IniFiles:

```
procedure PodatkiShrani;
var x: integer;
    msg: string;
begin

IniFile := TIniFile.Create(ChangeFileExt
  (Application.ExeName,'.pod'));
```

```
IniFile.WriteInteger('Podatki','P1', Polja1Stevec);
IniFile.WriteInteger('Podatki','P2', Polja2Stevec);

  for x := 0 to _MAXPOLJ do begin
    if x <= Polja1Stevec then begin

      msg := cut(polja[x, 0].caption);
      IniFile.WriteString
        ('odhodki', 'opis' + inttostr(x), msg);

      msg := cut(polja[x, 1].caption);
      IniFile.WriteString
        ('odhodki', 'vrednost' + inttostr(x), msg);

      if polja[x, 0].font.color = Polja1DisableColor then
        IniFile.WriteInteger
          ('odhodki', 'oz' + inttostr(x), 1)
      else
        IniFile.WriteInteger
          ('odhodki', 'oz' + inttostr(x), -1);

    end else begin

      IniFile.WriteString
        ('odhodki', 'opis' + inttostr(x), '');

      IniFile.WriteString
        ('odhodki', 'vrednost' + inttostr(x), '');

      IniFile.WriteInteger
        ('odhodki', 'oz' + inttostr(x), -1)
    end;
  end;  //end for

...

end;
```

Opomba: procedura *Cut* odreže presledke na začetku in na koncu niza.

Iz kode je razvidno, da sta pisanju v datoteko namenjeni metodi WriteString in WriteInteger. Podobno je pri branju, na voljo sta ReadString in ReadInteger.



Slika 3.6: Ini datoteka s podatki o lastnostih glavnega okna.

Obe datoteki se shranita v mapo, kjer se nahaja izvršna datoteka aplikacije. Shranjevanje v mapo okolja Windows (ponavadi c:\Windows) lahko pripelje do zapletov, če želimo ponesreči uporabiti ime datoteke, ki ga uporablja že neka druga aplikacija.

3.8 Razveljavitev zadnjega dejanja

En večjih izzivov pri gradnji Bilance je bila možnost razveljavitve zadnjega uporabnikovega dejanja. Ideja je sicer preprosta: vsakič, ko uporabnik naredi spremembo, se shrani trenutno stanje vseh polj. To stanje se doda v tabelo vseh ostalih stanj iz preteklosti. Spodnji podatkovni strukturi sta bili uporabljeni za shranjevanje stanj:

```
type TPolje = record
  opis: string;
  vrednost: string;
  id: integer;
  oznacen: boolean;
  enabled: boolean;
end;

type TStanje = record
  stevec1: integer;
  stevec2: integer;
  stroski: array[0.._MAXPOLJ] of TPolje;
  prihodki: array[0.._MAXPOLJ] of TPolje;
end;
```

Težave niso nastale pri shranjevanju stanj, ampak pri zaznavanju sprememb. Izkazalo se je, da je zaznavanje sprememb lahko zapleten proces. Lovljenje dogodkov na tipkovnici in miški ni dovolj. Vzemimo za primer polje brez opisa, ki ima vrednost 10. Uporabnik začne urejati vrednost, vpiše 20, potem pa si premisli in spet vpiše 10, nato pa potrdi vnos. Kljub temu, da je prišlo do dogodkov na tipkovnici, ki so v resnici večkrat spremenili vrednost polja, rezultata ne smemo obravnavati kot spremembo. To dejstvo je prineslo nove spremenljivke, ki so ob domnevnih spremembah služile preverjanju novega stanja s prejšnjim. Ker so to v veliki večini globalne spremenljivke, so hkrati tudi poslabšale pregled nad kodo in povečale nevarnost za nastanek napak, ki jih prevajalnik ne javi in jih je zato težko odkrivati. Bilanca je enostavna aplikacija z majhnim številom različnih možnih dogodkov, a mi je zaznavanje sprememb povzročilo nepričakovano veliko težav. Ne predstavljam si prav dobro, kako upravljajo s spremembami in razveljavljanjem dejanj pri zapletenih aplikacijah z veliko možnostmi.

3.9 Samodejno drsenje pogleda

Kadar je polj na formi več, kot jih je možno prikazati naenkrat, se samodejno prikažejo drsniki, s katerimi lahko pogled prestavimo na željena polja. Drsenje pogleda po formi lahko v Bilanci upravljamo tudi z miškinim kolescem. Kadar želi uporabnik z vlečenjem premakniti polje izven meja pogleda, pričakuje, da se bo pogled temu prilagodil. V preteklosti sem že opazil, da se aplikacije v takšnem primeru odzivajo zelo različno, kar

me je presenečalo. Zdaj vem, zakaj je temu tako - drsenje pogleda je v celoti prepuščeno programerju. Delphi ne ponuja nobene samodejne podpore pri tovrstnih dogodkih.

Samodejno drsenje pogleda sem realiziral s pomočjo štoparice (angl. timer). Ko se polje med vlečenjem približa robu forme, se omogoči delovanje štoparice, ki vsakih 70 stotink sekunde premakne pogled navzdol ali navzgor. Ko uporabnik odmakne polje od roba forme, se delovanje štoparice onemogoči in drsenje se ustavi. Pri uveljavljenih aplikacijah je ponavadi hitrost drsenja usklajena s tem, kako daleč od roba je objekt, ki ga vlečemo. Bolj, ko objekt prekorači mejo, hitreje pogled drsi po formi. Bilanca uporablja dve različni hitrosti drsenja. Ob tem sem spoznal, da koordinate na formi niso statične. Če pogled premaknemo dol po formi in postavimo objekt na lokacijo (0, 0), bo objekt v celoti viden. To pomeni, da se z vsakim premikom pogleda spremenijo tudi koordinate vseh objektov na formi, kar me je presenetilo, saj to pomeni tudi, da imajo lahko objekti na formi negativne koordinate.

3.10 Odpravljanje vizualnih pomankljivosti

Z naraščanjem števila objektov na formi se pri njihovem izrisu pojavi težava z osveževanjem. V primeru Bilance je to pomenilo migotanje (angl. flicker) vseh objektov, kadar je uporabnik izvajal vlečenje. Več, ko je bilo polj v stolpcih, močnejše in bolj moteče je bilo migotanje. Tovrstne težave se odpravljajo z uporabo navideznega zaslona. Formo z vsemi objekti najprej v celoti izrišemo na navidezni zaslon, potem pa ga z eno potezo preslikamo v pomnilnik grafične kartice. Tako se učinek migotanja močno zmanjša. To tehniko sem najprej nameraval izpeljati sam v celoti, potem pa sem odkril zelo preprosto alternativo. Delphi namreč nudi samodejno upravljanje z navideznim zaslonom pri izrisu forme, to možnost pa aktiviramo z nastavitvijo vrednosti globalne spremenljivke `DoubleBuffered := True`.

3.11 Testiranje z uporabniki

Aplikacijo sem izdelal z namenom, da bi jo uporabljal sam in tudi moji prijatelji. Zato sem kar njih povabil na preizkušanje. Naletel sem na različne odzive - od mlačnih vprašanj, zakaj bi takšno aplikacijo sploh potrebovali, do prijetnih sporočil, da bodo aplikacijo dejansko uporabljali. Mislim, da bi dobil bolj realno sliko, če bi aplikacijo ponudil naključnim uporabnikom in ne prijateljem, ker bi tako dobil bolj iskrene odgovore, brez strahu, kakšna bo moja reakcija. Še boljšo sliko pa bi dobil, če bi aplikacijo dejansko ponudil tistim, ki ustrezajo opisu tipičnega uporabnika - tistim, ki bi radi izboljšali pregled nad svojimi financami. Od preizkuševalcev sem dobil kar nekaj namigov in predlogov, kako dopolniti aplikacijo:

- uporaba kombinacije tipk *CTRL-A* za označevanje vseh polj naenkrat,
- uporaba tipke *TAB* med vpisovanjem vrednosti oz. opisa v polje (ta problem sem že opisal v razdelku *3.6. Vnos opisa in vrednosti*),
- izbira imena datoteke, kamor se shranjujejo podatki o poljih,

- težava pri shranjevanju podatkov v operacijskem sistemu Windows Vista, ...

Zadnja točka se nanaša na shranjevanje podatkov v direktorij c:\. Windows Vista tega ne dovoli, zato uporabnik, ki Bilanco poganja iz tega direktorija, ne more shraniti nastalih sprememb. Težave nisem odpravil, ker ni bistvena za okvire tega diplomskega dela. Aplikacije se ponavadi naložijo z inštalacijskim programom, ki v tem primeru preprosto ne bi dovolil inštalacije v c:\ in težava bi bila tako odpravljena. Prav tako nisem upošteval predloga, opisanega v predzadnji točki. Med uporabo Excelovega prototipa se mi namreč niti enkrat ni pripetilo, da bi si želel shraniti trenutno stanje v drugo datoteko in ustvariti novo. Pregled osebnih financ na enem mestu pomeni tudi, da so vsi podatki shranjeni na enem mestu. Ponujanje možnosti, da si uporabnik ustvari več datotek, se mi zdi popolnoma odveč. Če bi nekdo zares potreboval to možnost pa bi jo z majhno pomočjo iznajdljivosti lahko tudi dosegel s kopiranjem izvršne datoteke v neko drugo mapo.

Prvo točko sem upošteval in dodal novo bližnjico. Prav tako sem v glavni meni dodal tudi možnost, da uporabnik poveča ali pomanjša pogled. Opazil sem namreč, da nihče od preizkuševalcev ni niti pomislil, da uporabniški vmesnik to omogoča (z uporabo tipkovnice).

Na koncu sem pred Bilanco posedel tudi svoja starša, ki računalnik zelo malo uporabljata. Izkazalo se je, da so nekatere bližnjice, ki so bile dodane kot razširitev uporabnosti uporabniškega vmesnika, v bistvu prva možnost, ki jo neizkušeni uporabnik poskusi. To se je zgodilo pri urejanju opisov polj - moja mama ni začela urejati opisa z dvojnim klikom na polje, ampak je na polje samo kliknila in začela pisati. Kadar je označeno samo eno polje, Bilanca namreč omogoča tudi to, da se urejanje polja začne samodejno, ko uporabnik začne uporabljati tipkovnico. To je vsekakor pozitivna ugotovitev, ki je v skladu z navedbami v razdelku *1.6 Znanje, pridobljeno v literaturi.*

4 Sklepne ugotovitve

4.1 Doseženi cilji, nove ideje in ugotovitve

Aplikacijo Bilanca ocenjujem kot uporabno in med prijatelji, ki so jo testirali, jo dva že redno uporabljata. Prepričan sem, da ciljnemu uporabniku nudi olajšanje in nov pogled na osebne finance. S tega stališča sem s svojim delom zadovoljn. Seveda pa obstaja še veliko manjših in malo večjih stvari, ki bi jim lahko posvetil še nekaj pozornosti:

- izboljšati bi se dalo sortirni algoritem za urejanje vrstnega reda polj med vlečenjem,
- izboljšati bi se dalo samodejno drsenje pogleda med vlečenjem,
- uporabniku bi lahko ponudil tudi nastavitev barv vmesnika, saj bi si tako izgled aplikacije še približal in jo zato raje uporabljal,
- razmisliti bi veljalo tudi o sistemu, ki bi uporabniku nudil pregled nad vsem zasluženim denarjem v določenem obdobju,
- kodo bi lahko naredil bolj prijazno kasnejšemu urejanju, ...

Kljub temu, da gre za na pogled zelo enostavno aplikacijo, je izvorna koda Bilance dolga več kot 2.300 vrstic. To se mi zdi veliko, glede na to, da aplikacija v bistvu zgolj odšteva odhodke od prihodkov in jih hrani v datoteki. Koda je tako dolga zato, ker je njena večina namenjena uporabniškemu vmesniku in ne funkcionalnosti aplikacije. Če bi Delphi nudil večjo podporo samodejnemu upravljanju z gradniki uporabniškega vmesnika, bi bila lahko koda opazno krajša. Delphi nudi razvijalcem veliko svobodo pri ustvarjanju, kar izkušenim programerjem zagotovo pride prav, manj izkušene pa ta svoboda lahko ohromi. Tudi mene je na začetku. Če bi še enkrat želel ustvariti Bilanco od začetka, bi verjetno dobro premislil o kakšnih drugih okoljih, ki od programerja ne zahtevajo dela na tako nizkem nivoju arhitekture uporabniških vmesnikov. Kljub vsemu sem z odločitvijo za Delphi zelo zadovoljn, saj sem si v času razvoja nabral veliko izkušenj, ki mi bodo prej ali slej prišle prav.

Slike

Tabele

Priloga in viri

Priloga

Temu diplomskemu delu je priložen CD z izvorno kodo aplikacije Bilanca.

Viri

[1] 37signals, *Getting real*, 37signals, 2006

[2] P. Butcheit, *http://paulbuchheit.blogspot.com*

[3] D. A. Norman, *The design of everyday things*, Basic Books, 2002

[4] J. Spolsky, *User interface design for programmers*, Apress, 2001

Izjava o samostojnosti dela

Izjavljam, da sem diplomsko delo izdelal samostojno pod vodstvom mentorja prof. dr. Dušana Kodeka. Izkazano pomoč drugih sodelavcev sem v celoti navedel v zahvali.

Ljubljana, 30. september 2008

Vasilij Femec

Kazalo

Zahvala

Za pozorno in vzpodbudno mentorstvo se najlepše zahvaljujem prof. dr. Dušanu Kodeku. Zahvaljujem se tudi vsem prijateljem, ki so prijazno priskočili na pomoč pri testiranju nastale aplikacije. Na koncu se želim zahvaliti svojim staršem in spremljevalki Katarini, ki so mi skozi študij trdno stali ob strani.