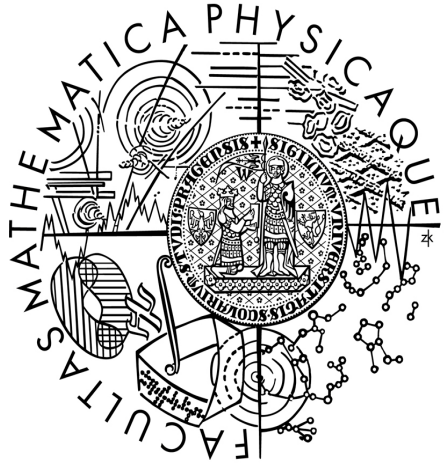
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Lenka Trochtová

Slévání rozdílných verzí textových souborů

Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Pavel Ježek

Studijní program: Informatika, programování

2006

Prohlašuji, že jsem svou bakalářskou práci napsala samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne Lenka Trochtová

Obsah

Název práce: Slévání rozdílných verzí textových souborů

Autor: Lenka Trochtová

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Pavel Ježek

e-mail vedoucího: pavel.jezek@mff.cuni.cz

Abstrakt: Cílem této práce bylo vytvořit program Merge sloužící k porovnávání a slévání dvou nebo tří verzí textových souborů a adresářů. Práce se zabývá programem Merge jak z uživatelského, tak i z programátorského hlediska. Součástí práce je uvedení do problematiky porovnávání a slévání souborů, porovnání programu Merge s některými existujícími nástroji, stručný popis základních funkcí programu, popis návrhu a implementace programu včetně použitých algoritmů textového a adresářového porovnání.

Klíčová slova: Merge, porovnání, slévání, text, adresář

Title: Merging of Different Text File Versions

Author: Lenka Trochtová

Department: Department of Software Engineering

Supervisor: Mgr. Pavel Ježek

Supervisor's e-mail address: pavel.jezek@mff.cuni.cz

Abstract: The goal of this thesis is to create a program (Merge) for comparing and merging two or three versions of text files and directories. The thesis describes the program from both user's and programmer's points of view. The text includes introduction into the background of file comparing and merging, comparison of the Merge program with another existing merging tools, a brief description of the main functions of the Merge program and also description of design and implementation of the program including the algorithms of text and directory comparison.

Keywords: Merge, compare, merge, text, directory

1. Úvod

1.1. Cíl práce

Cílem práce je vytvořit program Merge pro porovnávání a slévání několika verzí adresářového stromu. Hlavní částí programu je porovnání a slévaní jednotlivých souborů, které byly detekovány jako rozdílné v různých verzích. Program by měl podporovat zobrazení rozdílů v textových souborech a jejich zvýraznění pomocí barevného označení a grafických svorek. Při zobrazení rozdílů v textových souborech i adresářích by měl být kladen důraz především na přehlednost. Program by měl být navržen tak, aby byl v budoucnu dobře rozšiřitelný, zvláště o možnost porovnávat i jiné typy souborů.

2. Slévání souborů a existující nástroje

2.1. Úvod do problematiky slévání a jeho motivace

Tzv. slévání souborů je proces, kdy jsou postupnými úpravami odstraňovány změny mezi dvěma soubory. Používá se tam, kde z nějakého důvodu potřebujeme sjednotit dvě a více různých verzí téhož souboru. Totéž platí pro celé adresářové struktury, které obsahují odlišné verze k sobě logicky patřících souborů (příkladem může být třeba adresářová struktura souborů náležejících k jednomu projektu, na kterém se dlouhodobě pracuje a při průběžných zálohách vznikají jeho odlišné verze).

Ke slévání různých verzí souborů, či celých adresářů neodmyslitelně patří jejich porovnání. Abychom mohli dvě verze slévat, musíme samozřejmě napřed zjistit, čím se liší a co naopak mají společné. Slévání i samotné porovnávání by jistě mohl zvládnout sám uživatel s běžnými prostředky pro prohlížení a editaci adresářů a v nich obsažených souborů, bylo by to ale především u rozsáhlejších adresářů (případně souborů) neúnosně a především zbytečně namáhavé, nejobtížnější by pak bylo samotné nalezení všech změn.

Hlavním účelem programů, které mají tuto práci uživateli usnadnit, je tedy nalezení a pokud možno co nejpřehlednější zobrazení změn v porovnávaných verzích. Často dokonce samotné porovnání uživateli bohatě stačí – ne vždy je třeba slévat, někdy uživatel prostě jen potřebuje rychle zjistit, čím přesně se dvě verze liší. Není tedy divu, že existují nástroje, které se zabývají pouze porovnáváním – jako příklad může být uveden UNIXový příkaz diff.

Obohacení porovnávacích programů o slévací funkce, případně o další možnosti editace pak ještě více zlepší jejich použitelnost a efektivitu práce s nimi. Dalšího zlepšení lze dosáhnout přidáním možnosti porovnání nejen dvou verzí, ale i verzí tří, kde jedna z nich je považována za společného předka a zbylé dvě z ní vznikly na sobě nezávislým vývojem.

Protože nejčastějším typem porovnávaného souboru je textový soubor, je výhodné, aby porovnávací programy uměly nalézt nejen rozdílné řádky, ale v rámci sobě odpovídajících rozdílných úseků textů uměly na požádání najít i rozdíly na jemnější úrovni – třeba v jednotlivých znacích.

Programy s touto funkčností pak mají značné uplatnění v praxi.

Typickým příkladem praktického použití programů pro porovnávání a slévání různých verzí je použití při tvorbě rozsáhlejšího programového díla. V průběhu práce programátor zálohuje jednotlivé verze. Přestane-li náhle fungovat něco, co v dřívější verzi fungovalo, je snadné obě verze pomocí daného programu porovnat a nalézt nejen změněné soubory se zdrojovými texty, ale i přímo úseky v rámci jednotlivých textů, které se změnily. Pak už je na programátorovi, aby se rozhodl, které ze změn zachovat, a které vrátit zpět, protože způsobily chyby v programu.

V předchozím případě se uplatní především dvoucestná porovnání (vzájemné porovnání dvou textů nebo dvou adresářů). Možnost trojcestného porovnání (tj. porovnání dvou odlišných verzí se společným předkem) se nejlépe uplatní při týmové práci. Pracují-li například dva programátoři na stejném projektu, potřebují průběžně slévat výsledky své práce. Oba samostatně vytvoří úpravy původního společného kódu a následně potřebují své verze sjednotit. V příslušném programu na slévání mohou otevřít porovnání obou nových verzí adresářových struktur projektu s jejich společným předkem – verzí původní. Program graficky odliší soubory a adresáře do původní verze přidané, stejně jako graficky znázorní soubory a adresáře, na kterých některý z programátorů provedl změnu. K původní verzi lze slít nově přidané položky okamžitě na této úrovni porovnání, stejně tak i již existující položky změněné jen jedním z programátorů. Textové soubory vyskytující se změněně ve všech verzích je zajímavější nejdříve otevřít pro detailnější textové porovnání. To umožní použít v nové společné verzi změny obou programátorů, případně odhalit změny, které jsou v konfliktu, a rozhodnout se pro použití jedné z nich.

Program Merge vytvořený v rámci této práce byl vytvořen právě pro výše zmíněné použití. Umí porovnávat a slévat dvě, případně tři verze adresářů. Totéž umožňuje při práci s textovými soubory, kde kromě klasického porovnání textů jako posloupnosti řádek umožňuje navíc detailní nalezení rozdílů v jednotlivých znacích v rámci úseků textů detekovaných jako navzájem odlišných. Porovnání souborů na textové úrovní lze otevřít přímo z porovnání adresářů, kde se tyto soubory vyskytují. Nalezené změny při obou typech porovnání graficky

zvýrazňuje s důrazem na maximální přehlednost. Textové soubory lze navíc editovat běžným způsobem, včetně vyhledávání a náhrady vzorů v textu.

V obou výše uvedených motivačních příkladech lze Merge použít zmíněným způsobem. Při jeho vývoji se předpokládalo právě použití při porovnávání různých verzí projektů při vývoji softwaru, proto program také umožňuje zvýraznění syntaxe. Toto použití ale není podmínkou, program je univerzálním porovnávacím nástrojem. Merge lze obecně využít všude tam, kde se pracuje s více verzemi textů, případně adresářových struktur. Jediným omezením je, že na úrovni jednotlivých souborů zatím dokáže porovnávat jen čistě textové soubory (a zatím nepodporuje Unicode). Je zde ale možnost budoucího rozšíření, zejména o porovnávání dalších typů souborů, jako třeba binárních souborů, obrázků atd.

2.2. Existující nástroje a jejich porovnání s programem Merge

Úloha porovnávání různých verzí souborů (zejména textů) a adresářů je již velmi stará. Kupříkladu už ve starých verzích operačního systému UNIX byl příkaz diff. Programy pro porovnávání a slévání od té doby neztratily na účelnosti, naopak spíše neustále nalézají nové oblasti uplatnění. Není tedy divu, že od té doby vznikl nespočet nástrojů, které se tímto zabývají.

Narozdíl od příkazu diff, většina současných programů na porovnávání a případně slévání verzí jsou okenní aplikace s grafickým výstupem porovnání. Z těch, které pracují pod operačním systémem Windows stejně jako program Merge, lze uvést například komerční Merge od firmy Araxis, nekomerční WinMerge a nástroj WinDiff, který je součástí rozšiřujících balíčků s pomocnými nástroji pro systém Windows.

Následující text obsahuje srovnání těchto programů s programem Merge. V úvodu je tabulka s přehledem některých často používaných funkcí a jejich podpory jednotlivými programy, následuje podrobnější popis výhod a nevýhod těchto programů.

	Araxis Merge, profesionální verze	WinMerge	WinDiff	Merge
porovnání adresářů	ano	ano	ano	ano
porovnání textů	ano	ano	ano	ano
třícestné porovnání	ano	ne	ne	ano
zobrazení textů	vedle sebe – svorky, synchr. rolování	vedle sebe – vynechávání místa	v jednom textu - nepřehledné	vedle sebe – svorky, synchr. rolování
zobrazení adresářů	vedle sebe – vynechání místa	v jednom adresáři - nepřehledné	v jednom adresáři - nepřehledné	vedle sebe – vynechání místa
zvýraznění syntaxe	ne	ano	ne	ano
editace textů	ano	ano	ne	ano
slévání	ano	ano	ne	ano
nastavení porovnání – prázdné řádky	ano	ano	ano	ano
nastavení porovnání – mezery	ano	ano	ne	ano
nastavení porovnání – case-sensitive	ano	ano	ne	ano
nastavení porovnání – regulární výrazy	ano	ne	ne	ne
rozdíly v jednotlivých znacích	ano	první rozdíl, pak další, atd. na požádání - nepohodlné	ne	ano
cena (USD ($))	269.00	0	0	0

Tabulka 2.2.1 se srovnáním programů na porovnávání verzí.

Ze zmíněných programů má asi nejpestřejší paletu funkcí komerční Merge firmy Araxis (více o programu viz [4]). Jde zejména o jeho profesionální verzi – v té základní část funkcí chybí, především trojcestné porovnání. Jediné, co by se mu v této oblasti dalo vytknout, je absence možnosti zvýraznění syntaxe u porovnání textů. Vzhledem k tomu, že programy tohoto typu bývají často užívány programátory pro práci s různými verzemi zdrojových textů, je možnost zvýraznění syntaxe vítanou funkcí, která dále zvyšuje přehlednost grafického výstupu textového porovnání a tím zpříjemňuje práci s porovnávanými texty. Výraznou nevýhodou programu pak je jeho pořizovací cena. Například studentovi programování, který příležitostně použije podobný nástroj při práci na rozsáhlejším školním projektu, se taková investice nevyplatí. Pro firmu, ve které se podobný nástroj často používá, je to ale v každém případě dobrá volba.

Z nekomerčních programů je velmi zajímavý nástroj WinMerge (více o programu viz [5]). Umožňuje porovnávání i slévání textů i adresářů.

Za zmínku stojí především jeho část zabývající se porovnáváním textů. Texty lze pohodlně editovat, prohledávat a provádět náhrady. WinMerge podporuje zvýraznění syntaxe pro úctyhodné množství programovacích jazyků. Rozdíly jsou přehledně graficky znázorněny a sobě odpovídající společné řádky textů jsou zarovnány na stejnou výšku pomocní vkládání speciálních prázdných řádků (ty jsou od vlastních prázdných řádků textů barevně odlišeny). Domnívám se, že právě tento způsob řešení zarovnání společných řádků je silou, ale zároveň i slabostí tohoto nástroje. Pro porovnání dvou textů je naprosto ideální a přehlednou a tím i uživatelsky příjemnou variantou, nicméně, myslím, komplikuje rozšíření programu o porovnání textů tří. Použití v tomto případě by již nebylo tak přehledné a zároveň by bylo i složitější.

Ať už je příčina jakákoliv, WinMerge nepodporuje porovnání tří verzí ani pro texty ani pro adresáře. To značně omezuje jeho použití při týmové práci. Slévat dvě verze téže práce vzniklé ze společného předka nezávislým vývojem (viz motivační příklad spolupráce dvou programátorů) lze sice i za použití dvoucestného porovnání, ale je to varianta pro uživatele mnohem méně pohodlná. Především se zde ztrácí spojení s původní společnou verzí a tím je zamlžen původ jednotlivých změn. To samozřejmě znesnadňuje rozhodování, jak s danými změnami při slévání naložit, a zpomaluje se tak práce s programem.

Další nevýhodou programu je i jeho v porovnání s textovou částí poněkud slabší část zabývající se adresáři. Oba porovnávané adresáře jsou zobrazeny v jednom výpisu, kde je pouze malou ikonou a slovy označeno, zda se daná položka vyskytuje v obou adresářích (a je-li změněná), či pouze v jednom z nich (a v kterém). Krom toho vůbec není graficky znázorněna stromová struktura adresáře. Všechny položky adresáře i jeho podadresářů jsou vypsány na stejné úrovni v seznamu pod sebou, jejich umístění lze zjistit pouze z výpisu ve vedlejším sloupci. Položky podadresářů lze buď zobrazit všechny, nebo žádné, což je obzvláště nepohodlné při prohlížení větších adresářů. Porovnání adresářů tedy uživateli poskytne všechny základní informace o nalezených rozdílech, ale předkládá je v nepříliš atraktivní a značně nepřehledné a neintuitivní formě.

Poslední ze zmíněných existujících nástrojů, program WinDiff (více o programu viz [6]), pak ve srovnání dopadl asi nejhůře. Podporuje nejmenší množství zmíněných funkcí, vůbec neumožňuje slévání ani zobrazení detailnějších rozdílů při porovnání textů. Editace porovnávaných textů probíhá spuštěním externího programu (Poznámkový blok), což je pro uživatele méně přehledné a pohodlné, než by byla editace přímo v programu.

Velikým nedostatkem je také nepřehlednost zobrazení porovnání textů i adresářů. U adresářů je problém podobný jako v případě programu WinMerge. Zobrazení jejich porovnání spočívá ve výpisu dvou sloupců, v nichž jeden obsahuje cesty k daným položkám porovnání (adresářům a souborům) a druhý obsahuje slovní popis výsledku porovnání (vyskytuje se jen vlevo, jen vpravo atd.).

Zobrazení textů se pak nese v podobném duchu. Oba porovnávané texty jsou zobrazeny jako jeden, který je jakýmsi jejich sjednocením, pouze jsou barevně odlišeny řádky, které v některém z textů nejsou, nebo se liší. Zobrazení je tím pádem méně přehledné než zobrazení textů vedle sebe. Přehlednost tohoto zobrazení částečně zachraňuje postranní přídatný pohled se symbolickým znázorněním celého porovnání.

Program Merge vytvořený v rámci této práce nepřináší žádnou zásadní revoluční a zcela novou funkci, která by se nevyskytla v některém z již existujících programů. Přesto se domnívám, že má své právo na existenci. Program Merge dle mého názoru totiž představuje rozumný kompromis.

Je zadarmo a podporuje zvýraznění syntaxe jako WinMerge, umožňuje porovnávat nejen dvě, ale i tři verze adresářů a textů jako komerční program Merge firmy Araxis. Stejně jako tento nástroj zobrazuje změny u obou typů porovnání přehledným a elegantním způsobem. U porovnání textů to je barevné odlišení rozdílných řádek společně s grafickými svorkami včetně volitelné možnosti synchronizovaného rolování, u porovnání adresářů je to zobrazení stromové struktury adresářů vedle sebe se zarovnáním sobě odpovídajících položek na stejnou úroveň.

Merge dále obsahuje funkce pro slévání adresářů i textů. Pro práci s texty navíc obsahuje většinu běžných editačních funkcí, včetně práce se schránkou, funkcí undo a redo, vyhledávání a náhrad vzorů v textu.

Merge sice zatím rozhodně nedosahuje kvalit programu Merge od firmy Araxis, ale není v tomto ohledu příliš omezen do budoucna. Domnívám se, že většina funkcí, které má tento komerční program navíc, by se dala přidat bez zásadních změn stávajícího kódu. Jednalo by se tedy skutečně o rozšíření, nikoliv o předělání programu. Stejně tak by se dalo přiblížit pestrosti podporované syntaxe programu WinMerge. Podpora dalších jazyků by se dala přidat stejným způsobem, jako byly přidány jazyky již podporované. Jediným problémem je časová náročnost naprogramování příslušných funkcí pro větší množství jazyků.

3. Obecné řešení problému

3.1. Algoritmus porovnání

Program Merge obsahuje dva porovnávací algoritmy. První z nich je použit při porovnávání textů, druhý při porovnávání adresářů. Obě úlohy se totiž značně liší.

Texty jsou porovnávané jako posloupnosti řádek. Tyto posloupnosti jsou zcela obecné – nelze předpokládat jejich uspořádání (např. podle abecedy). Při jejich porovnání tedy jde o nalezení společné podposloupnosti, která je co nejdelší a zároveň minimalizuje počet rozdílů mezi oběma texty. Ten samý algoritmus se pak dá použít pro detailní porovnání sobě odpovídajících rozdílných úseků textů po jednotlivých znacích.

Naproti tomu adresáře nemají strukturu posloupnosti, jejich struktura je stromová. Navíc na jednotlivých hladinách adresářového stromu lze využít možnosti položky dané hladiny uspořádat (v tomto případě napřed podle typu – adresáře mají přednost - a pak podle jména).

Algoritmus porovnání textů

Jak již bylo řečeno, při porovnání textů hledáme vhodnou podposloupnost řádek. Podposloupnost by měla být co největší a zároveň chceme, aby výsledné rozdíly mezi oběma texty byly minimální.

Problém nalezení nejdelší společné podposloupnosti (LCS – Longest Common Subsequence – viz [7]) je známý a existuje algoritmus využívající dynamické programování, který tento problém řeší.

Úloha skutečně splňuje všechny předpoklady pro nasazení dynamického programování. Posloupnosti jsou obecné, nelze předpokládat jejich uspořádanost, tudíž nelze jednoduše zjistit, které prvky si budou ve výsledku vzájemně odpovídat – které tedy budou obsaženy v hledané podposloupnosti. Víme jen, že bude zachována vzájemná poloha těchto prvků. Pokud bychom chtěli vyzkoušet všechny možnosti, měla by úloha exponenciální složitost. Úloha má naštěstí dvě

vlastnosti, které umožňují nasazení dynamického programování – úloha je optimalizační (hledáme maximum) a části optimálního řešení jsou optimálními řešeními téže úlohy menší velikosti.

Zmíněný algoritmus má složitost O(nm), pro posloupnosti délek m a n.

Algoritmus používá dvě pomocná dvourozměrná pole, nazveme je například a a b. V poli a si budeme ukládat délku nejdelší společné podposloupnosti nalezené pro zpracované části vstupních posloupností. Prvek a[i][j] pole obsahuje délku nejdelší společné podposloupnosti prvních i členů první posloupnosti a prvních j členů druhé posloupnosti. V poli b si uložíme informace o směru, kterým se po vypočtení obou polí zpětně vydáme při hledání společné podposloupnosti.

Celý algoritmus má dvě části. V první části se současně vypočítá obsah obou polí, v druhé části se najde nejdelší společná podposloupnost.

První část algoritmu (posl1 a posl2 jsou vstupní posloupnosti):

```
L1 = length(posl1)
L2 = length(posl2)

for( i = 0; i <= L1; i++ )
      a[0][i] = 0

for( i = 0; i <= L2; i++ )
      a[i][0] = 0

for( i = 1; i <= L1; i++ )
      for( j = 1; j <= L2; i++ )
      {
            if( posl1[j] = posl2[i] )
            {
                  a[i][j] = a[i-1][j-1] + 1
                  b[i][j] = UPLEFT
            }
            else if(a[i-1][j] > a[i][j-1])
            {
                  a[i][j] = a[i-1][j];
                  b[i-1][j-1] = UP;
            }
            else
            {
                  a[i][j] = a[i][j-1];
                  b[i-1][j-1] = LEFT;
            }
      }
```

Po ukončení výpočtu první části obsahuje prvek a[L1][L2] délku hledané podposloupnosti a pole b obsahuje návod, kudy se vydat od konce při zpětném dohledání podposloupnosti.

Tím se pak zabývá druhá část algoritmu:

```
i = L1, j = L2
vysl = empty()    // výsledná podposloupnost
while( (i >= 0) && (j >= 0) )
      switch(b[i][j])
      {
      case UPLEFT:
            concatenate(posl1[i], vysl)   // zřetězení
            i--;
            j--;
            break;
      case UP:
            i--;
            break;
      case LEFT:
            j--;
            break;
      }
```

V proměnné vysl je nakonec uložena hledaná podposloupnost.

Původně byl pro porovnávání textů použit algoritmus v této podobě. Řádky společné podposloupnosti byly označeny jako společné pro oba texty a ostatní řádky byly považovány za změnu.

Ve většině případů byl výsledek dostačující. V některých případech, kde bylo více možných řešení, byla některá z nejednoznačností vyřešena tak, že vzniklo zbytečně mnoho změn. Typicky šlo o dvě vložení na místě, kde stačila jedna záměna. Problém spočíval v tom, že program zohledňoval pouze délku společné podposloupnosti, ale ne množství změn.

Bylo tedy třeba předělat úlohu maximalizace společných řádků na minimalizaci změn. Aby byla zachována ta vlastnost algoritmu, že výsledná podposloupnost je maximální, bylo třeba ohodnotit změny tak, že hodnota vložení (z pohledu protějšího textu smazání) je menší než hodnota změny (lze si představit dvě posloupnosti (a, b) a (b, a), ke změně jedné na druhou jsou potřeba dvě změny, případně jedno vložení a jedno smazání; v případě, že by algoritmus upřednostnil změny, byla by společná podposloupnost prázdná, i když by jinak mohla mít jeden prvek). Zároveň ale bylo třeba, aby dvě vložení měla dohromady větší hodnotu než jedna změna, jinak by se nevyřešil původní problém.

Upravený algoritmus je v následujícím výpisu.

První část upraveného algoritmu:

```
L1 = length(posl1); L2 = length(posl2);
CHANGE = 3; INSERT = 2;

for( i = 0; i <= L1; i++ ) a[0][i] = i*INSERT;
for( i = 0; i <= L2; i++ ) a[i][0] = i*INSERT;

for( i = 1; i <= L1; i++ )
      for( j = 1; j <= L2; i++ )
      {
            if( posl1[j] = posl2[i] )
            {
                  a[i][j] = a[i-1][j-1]
                  b[i][j] = UPLEFT
            }
            else if( ( a[i-1][j] <= a[i][j-1] ) &&
                        ( a[i-1][j] + INSERT < a[i-1][j-1] + CHANGE) )
            {
                  a[i][j] = a[i-1][j] + INSERT;
                  b[i-1][j-1] = UP;
            }
            else if( ( a[i][j-1] <= a[i-1][j] ) &&
                        ( a[i][j-1] + INSERT < a[i-1][j-1] + CHANGE) )
            {
                  a[i][j] = a[i][j-1] + INSERT;
                  b[i-1][j-1] = LEFT;
            }
            else
            {
                  a[i][j] = a[i-1][j-1] + CHANGE;
                  b[i-1][j-1] = UPLEFTCH;
            }
      }
```

A druhá část upraveného algoritmu:

```
i = L1, j = L2;
vysl = empty();   // výsledná podposloupnost
while( (i >= 0) && (j >= 0) )
      switch(b[i][j])
      {
      case UPLEFT:
            concatenate(posl1[i], vysl);
      case UPLEFTCH:
            i--;
            j--;
            break;
      case UP:
            i--;
            break;
      case LEFT:
            j--;
            break;
      }
```

Algoritmus porovnání adresářů

Algoritmus pro porovnání adresářů použitý v programu Merge využívá toho, že položky adresáře jsou předem uspořádané (k jejich uspořádání dochází při načtení adresáře). Algoritmus je rekurzivní stejně jako struktura adresáře. Adresáře i soubory jsou porovnávány podle obsahu.

Výsledkem algoritmu je označení každé z položek porovnávaných adresářů buď jako vložené, nebo jako přítomné v obou adresářích a změněné, nebo jako stejné.

Následuje symbolický zápis algoritmu. Značky < a > při porovnání položek adresáře jsou uspořádáním definovaným na těchto položkách (adresář < soubor, jinak porovnání podle jména).

```
function compare( dir1, dir2 )
{
      L1 = get_child_count(dir1), L2 = get_child_count(dir2);
      i = 0, j = 0;
      changed = false;

      while( i < L1 && j < L2 )
      {
            child1 = get_child(dir1, i); // i-tá položka adresáře dir1
            child2 = get_child(dir2, j); // j-tá položka adresáře dir2

            if( child1 < child2 )
            {
                  set_inserted(child1); // označí položku jako vloženou,
                                        // je rekurzivní pro adresáře
                  i++;
                  changed = true
            }
            else if( child1 > child2 )
            {
                  set_inserted(child2);
                  j++;
                  changed = true
            }
            else
            {
                  changed = changed || compare(child1, child2);
                  j++;
            }
      }


      // pokračování na další straně
```

```
	while( i < L1 )
	{
		child1 = get_child(dir1, i);
		set_inserted(child1);
		changed = true
	}

	while( j < L2 )
	{
		child2 = get_child(dir2, j);
		set_inserted(child2);
		changed = true
	}

	if( changed )
	{
		set_changed(dir1);
		set_changed(dir2);
	}
	return changed;
}


function compare( file1, file2 )
{
	file1.changed = file2.changed = false;
	file1.inserted = file2.inserted = false;

	if( file1.path == file2.path || file1.ctime == file2.ctime )
		return; // identické soubory, netřeba pokračovat v porovnání

	if( file1.lenght != file2.lenght ) // různá délka, soubory se liší
	{
		file1.changed = file2.changed = true;
		return;
	}

	while( !eof(file1) ) // porovnání stejně dlouhých souborů po bytech,
				// první nalezený odlišný byt ukončí
				// algoritmus s výsledkem „soubory jsou
				// různé“
	{
		byte1 = read(file1);
		byte2 = read(file2);
		if( byte1 != byte2 )
		{
			file1.changed = file2.changed = true;
			return;
		}
	}


}
```

3.2. Krátké odbočení k MFC

Program Merge je napsán v jazyce C++ za použití knihovny MFC a architektury dokument/pohled. Protože v následujícím textu používám některé základní pojmy, které se této knihovny a architektury týkají, pokusím se zde z nich krátce osvětlit alespoň to nejnutnější pro pochopení dalšího textu.

MFC (Microsoft Foundation Class Library – více viz [1] a [2]) je knihovna tříd, které tvoří jakousi tenkou objektovou obálku nad sadou základních funkcí API pro programování okenních aplikací pod operačním systémem Windows. Obsahuje hierarchii tříd pro základní objekty okenního systému (okna, tlačítka a další ovládací prvky, dialogy, menu atd.). Tyto třídy ve své implementaci volají funkce API pro manipulaci s objekty, které reprezentují. Většina metod těchto tříd jen volá příslušnou a typicky i stejnojmennou funkci API na objekt, který třída zastupuje.

Kromě toho, že MFC objektově zapouzdřuje související API funkce, obsahuje i některé složitější a pokročilejší funkce. Mezi ně bezesporu patří architektura dokument/pohled.

Základní myšlenkou této architektury je oddělení dat od způsobu jejich zobrazení. Proto také obsahuje dvě základní třídy – třídu dokumentu CDocument, která obsahuje reprezentaci dat a metody pro manipulaci s nimi, a třídu pohledu CView, která je potomkem třídy okna a stará se o zobrazení dat obsažených v dokumentu, ke kterému je připojena. Od těchto tříd dědí další specializovanější třídy dokumentů i pohledů.

V této architektuře platí, že jeden pohled zobrazuje data právě jednoho dokumentu, zatímco na jednom dokumentu může záviset více pohledů. Třída dokumentu obsahuje metody pro upozornění všech závislých pohledů. Třída pohledu kromě zobrazení dat dokumentu poskytuje část uživatelského rozhraní pro změnu těchto dat. Třída pohledu nikdy není samostatným oknem, musí být umístěna v tzv. rámcovém okně (to má rámeček, tlačítka na minimalizaci, zavření, apod., lze jej přemisťovat, měnit jeho velikost atd.), jehož klientskou část vyplňuje.

Aplikace může registrovat více typů pohledů a dokumentů. Děje se to prostřednictvím tzv. šablon dokumentů. Ty registrují konkrétní dokumenty

pro jejich použití společně s konkrétními pohledy a rámcovými okny. Šablona daného dokumentu jej umí vytvořit, otevřít i zavřít.

3.3. Porovnávání textů – obecně k návrhu řešení

První hrubý návrh řešení

Cílem návrhu bylo vytvořit přehledné porovnání textů, nástrojem byla knihovna MFC a architektura dokument/pohled.

Představa byla následující. Texty měly být umístěny v rámcovém okně daného porovnání vedle sebe a pohledy na sousední texty měly být proloženy speciálními pohledy se svorkami, které by propojovaly sobě odpovídající změněné části sousedních porovnávaných textů. Přičemž jedno bylo předem dáno použitou architekturou dokument/pohled – data budou v dokumentu, o jejich zobrazení se postarají pohledy.

Ačkoliv šlo o více samostatných textů a zdánlivě se tak nabízela možnost umístit každý do vlastního dokumentu (stejně jako datové struktury s jejich porovnáním), byla tato možnost zavržena. Důvodů bylo několik. V první řadě šlo o to, že knihovna MFC není na takové uspořádání připravena – jedno rámcové okno náleží vždy jednomu dokumentu. Není možné umístit do jednoho rámcového okna více pohledů, kde každý má vlastní dokument. Dalším důvodem byla vzájemná provázanost dat daných dokumentů a nemožnost samostatné existence jednoho bez ostatních. Logicky zkrátka všechna tato data patřila k sobě.

Bylo tedy rozhodnuto, že jedno porovnání bude odpovídat jednomu dokumentu. To znamenalo umístit všechny obsažené texty a dodatečné datové struktury s výsledkem jejich porovnání do jedné třídy dokumentu. Důsledkem tohoto rozhodnutí pak byla nutnost ručního naprogramování veškeré práce se soubory včetně příslušných ovládacích prvků – vytváření nových textů, otevírání a ukládání textů. Při tomto uspořádání nešlo použít předpřipravenou funkčnost vygenerovaného projektu, protože ta přepokládala rozvržení ve stylu jeden dokument – jeden soubor.

Ve výsledném návrhu tedy dokument obsahoval n datových struktur s texty a n - 1 datových struktur s porovnáním vždy dvou sousedních textů. Rámcové okno s pohledy na dokument pak obsahovalo n textových pohledů a n – 1 svorkových

pohledů. Rámcové okno při svém vytvoření požádalo dokument, aby vytvořil daný počet prázdných textů a dalších souvisejících datových struktur, a pak na tato data napojilo pohledy obou typů. Při potřebě jakékoliv manipulace s daty pak se daný pohled dotazoval dokumentu jen na ta data, která k pohledu patřila.

Výběr vhodné textové komponenty

Dalším krokem při návrhu bylo rozhodnutí, jak se vypořádat s vytvořením jednotlivých typů pohledů. Ty byly pro připomenutí dva – pohled na porovnávaný text a propojovací svorkový pohled.

V případě svorkového pohledu bylo rozhodnutí snadné, třída pohledu byla vytvořena jako přímý potomek základní třídy pohledu a neměla na starosti nic jiného než kreslení svorek synchronizovaně s vedlejšími textovými pohledy (musela tedy kvůli návaznosti svorek na texty znát pozici rolování obou textů).

Výběr vhodné třídy textového pohledu byl mnohem obtížnější.

První uvažovanou možností bylo vytvořit třídu jako potomka některé ze tříd standardních textových pohledů knihovny MFC. Takové třídy jsou dvě – CRichEditView a CEditView.

Třída CEditView neumožňuje formátovat text, celý text musí být zobrazen jedním fontem, jednou barvou písma atd. Třída ani nepočítá s tím, že by někdo mohl kreslit přes ni – kupříkladu v režimu R2_MASKPEN barevně (jakoby průhledným fixem) obtáhnout rozdílné řádky textu či do pohledu doplnit prodloužené svorky. Nebylo tedy možné tuto třídu použít z důvodu nemožnosti požadovaného grafického zobrazení nalezených rozdílů.

Naproti tomu třída CRichEditView formátování textu umožňuje. Tato třída ale umí spolupracovat pouze s dokumentem třídy CRichEditDoc. A tento dokument může obsahovat pouze jeden text. Tedy ani tato varianta řešení nebyla možná.

Další možností bylo vytvořit si třídu pohledu vlastní. Implementace opravdu kvalitní textové komponenty s veškerou obvyklou funkčností včetně vyhledávání

a náhrad v textu, mechanismu undo/redo, práce se schránkou a práce s textovými soubory by byla značně pracná.

Poslední možností bylo použít některou z volně šiřitelných open-source tříd nabízených na Internetu. Jako nejvhodnější z nich se jevila třída CCrystalEditView. Významnou výhodou této třídy bylo to, že byla napsána přímo v MFC, tedy do prostředí projektu by se dala začlenit bez větších problémů. Další výhodou byla její pružná architektura práce s textem. Zobrazovaný text nebyl umístěn přímo v nějakém speciálním dokumentu, ale v třídě CCrystalTextBuffer, ke které se pohled připojoval předefinováním virtuální metody LocateTextBuffer. Nebyl by tedy problém vytvořit v dokumentu porovnání více objektů tohoto typu a pohledy na ně napojit prostřednictvím vyžádání si příslušného textového bufferu od dokumentu v této metodě. Navíc třída umožňovala všechny obvyklé operace s textem a kromě nich ještě obsahovala podporu pro zvýraznění syntaxe.

Vzhledem k tomu, že tvorba vlastní textové komponenty nebyla předmětem této práce a třída CCrystalEditView nabízela tolik výhod, byla nakonec jako předek textového pohledu porovnání zvolena tato třída.

Začlenění Crystal Editu do projektu

Ačkoliv použití tříd CCrystalEditView a CCrystalTextBuffer nabízelo mnoho výhod, nebylo zcela bez problémů a také bylo třeba část vlastností těchto tříd upravit v jejich potomcích.

Samozřejmostí byla změna kreslících funkcí pohledu. Třída pohledu obsahovala virtuální metody pro nakreslení celé klientské části, pro nakreslení jedné řádky a pro nakreslení okraje (a pro zjištění jeho šířky). Cílem změn těchto metod bylo k jejich grafickému výstupu připojit dodatečné informace o porovnání, v případě okrajů pak šlo o změnu jejich účelu.

Nejjednodušší způsob, jak využít již existující kód pro kreslení, bylo původní metody v jejich předefinovaných verzích na začátku zavolat a pak kreslit přes jejich výstup. Takto je řešeno vykreslení prodloužených svorek se slévacími tlačítky v metodě pro nakreslení všech viditelných řádek, ale i barevné odlišení rozdílných řádek a případně jednotlivých znaků při detailním porovnání v rámci metody pro kreslení jedné řádky. V druhém případě bylo využito vlastností

kreslícího režimu R2_MASKPEN, kdy se barva štětce a barva podkladu smíchá aplikací logické operace and a výsledek vypadá, jako by daná oblast byla vybarvena průhledným barevným fixem.

V případě okrajů šlo o závažnější změnu a z původní metody nebylo zachováno téměř nic. Původním účelem okrajů bylo zobrazování grafických symbolů pro ladící zarážky, záložky a podobně. Autor CCrystal Editu nejspíše počítal s jeho použitím v nějakém nástroji pro překlad a ladění. V tomto projektu však nebylo těchto funkcí třeba. Okraj byl tedy využit na zobrazení čísel řádek a dále také slévacích tlačítek tam, kde bylo třeba. Kromě metody pro nakreslení okraje byla změněna i metoda určující šířku okraje tak, aby se přizpůsobila zobrazovaným informacím.

S použitím svorek jako způsobu zobrazení rozdílů a jejich propojení souvisejí další úpravy. Vzhledem k tomu, že svorky vzájemně propojují více pohledů, musely se sousední pohledy navzájem upozorňovat při jakémkoliv posunutí svého obsahu.

V případě vypnuté volby synchronizovaného rolování to bylo jednoduché. Textový pohled pouze požádal své svorky zobrazující sousedy, aby se překreslily, kdykoliv se vertikálně posunul jeho obsah. Svorkový pohled si od něj pak už jen zjistil nové posunutí textu a přizpůsobil mu svůj obsah.

V případě, že bylo synchronizované rolování zapnuté, byla situace složitější. Pohled, který byl rolován, byl pro své sousedy řídící a sousední pohledy se mu musely přizpůsobit. Pohled tedy pro své textové sousedy vypočítal nová posunutí textů a zavolal jejich metody pro synchronizaci s novou polohou. Synchronizované pohledy se pak na tuto polohu, pokud to bylo možné, přesunuly a stejným způsobem přinutily k synchronizaci i svého druhého souseda a upozornily svorkový pohled z již sesynchronizovaného porovnání na změnu. Synchronizace se tedy postupně rozšířila do celého řetězce propojených pohledů.

Výpočet nového posunutí sousedního textového pohledu závisel na prostřední řádce pohledu. Z výsledku porovnání se napřed zjistilo, která řádka této řádce odpovídá, následně byla tato řádka v sousedním pohledu zarovnána na stejnou y-ovou souřadnici jako prostřední řádka z řídícího pohledu. A jak se tedy zjistilo, která řádka dané řádce odpovídá? Pokud daná řádka byla řádkou společnou oběma textům, odpovídala jí ta samá řádka druhého textu. Pokud byla součástí rozdílu,

zjistila se napřed její relativní poloha v rámci rozdílného úseku textu a pak se jako její protějšek vzala řádka se stejnou relativní polohou v rámci téhož rozdílu v druhém textu porovnání.

Dalším významným krokem při začlenění Crystal Editu do projektu byla úprava mechanismu undo/redo. Crystal Edit obsahoval jeho vlastní implementaci. V textovém bufferu se ukládala undo historie, uživatelské příkazy na vrácení či znovu provedení akce pak zpracovávala samotná textová komponenta (třída pohledu) zavoláním příslušných metod svého bufferu pro práci s undo historií. Navíc některé akce změny textu byly slučovány do akce jedné – kupříkladu napsání více písmen neoddělených bílým znakem (tedy patřících k jednomu slovu) bylo považováno za jednu akci.

Toto chování bylo ale při použití v textovém porovnání zcela nevyhovující. Pokud by uživatel napsal několik písmen neoddělených bílým znakem v jednom textu, pak provedl změnu v sousedním textu, následně se vrátil do původního textu a tam pokračoval ve psaní rozepsaného slova a následně se rozhodl tyto ze svého pohledu tři akce postupně vrátit, choval by se program nesmyslně. Při prvním spuštění akce undo by se k překvapení uživatele smazalo celé slovo napsané v prvním z textů, aniž by se vrátila zpět změna provedená v sousedním textu mezi oběma vrácenými editacemi prvního textu. Tato změna by se nevrátila zpět ani při druhém (a ani při dalším) spuštění undo, protože undo by zpracovával aktivní textový pohled. To znamená, že editační akce by se při původním chování daly vracet ve správném vzájemném pořadí jen v rámci aktivního textu, ale nikoliv v rámci celého porovnání, což by ale bylo z pohledu uživatele logičtější.

Aby se nežádoucí chování odstranilo, bylo třeba provést dvě úpravy. V první řadě to bylo zrušení možnosti slučovat více akcí v undo historii textového bufferu do jedné. Tato původně hezká funkce Crystal Editu byla naneštěstí při použití v projektu nežádoucí.

Dalším krokem pak bylo vytvoření vlastní undo historie nad tou v Crystal Editu. Tato undo historie se stala součástí třídy dokumentu textového porovnání. Jediné, co se v ní zpočátku uchovávalo, byl index textu (pořadí textu v rámci porovnání), kterého se akce týkala. To znamená, že při vytvoření nového undo záznamu se použila původní funkčnost, pouze se navíc přidal záznam do undo historie v dokumentu. Při uživatelském vyvolání undo, případně redo akce se

aktivní pohled nejdříve přesvědčil, že daný undo záznam se týká tohoto pohledu (tj. index textu undo záznamu z dokumentu je stejný jako index jím zobrazovaného textu), a teprve pak jej zpracoval obvyklým způsobem. Pokud se daný záznam aktivního pohledu netýkal, předal tento pohled akci ke zpracování pohledu, kterého se záznam již týkal.

Později při implementaci slévání byla undo historie rozšířena o možnost slučování slévacích akcí. Pokud totiž uživatel provedl akci slití v podobě náhrady textu, považoval ji původní undo mechanismus za akce dvě – vymazání nahrazovaného textu a vložení textu, kterým se měl nahradit. Z pohledu uživatele to ale byla akce jedna – jedno slití. Rozšíření bylo provedeno tak, že se u daného undo záznamu v undo historii porovnání zapamatovalo, z kolika akcí se skládá. Tolikrát pak byla zavolána metoda textového bufferu na provedení undo/redo při žádosti uživatele o provedení příslušné undo/redo akce.

Implementace slévání

Součástí implementace slévání byly jednak samotné metody dokumentu textového porovnání pro slití dvou rozdílných úseků textů, jednak funkce pro procházení rozdílů a obsluhu slévacích tlačítek uvnitř pohledu.

Implementace metody pro slití daného intervalu rozdílných řádek do jemu odpovídajícího intervalu rozdílných řádek sousedního textu spočívalo v zavolání příslušných metod textových bufferů daných textů a úpravě záznamu v undo historii porovnání. Nejprve se z prvního textu získaly řádky, kterými se nahrazovalo, následně se v druhém textu smazaly řádky nahrazované a na jejich místo se pak vložily získané řádky. Obě akce – smazání a vložení – se pak v undo historii porovnání sloučily do jedné.

Kromě samotného slévání bylo třeba navrhnout způsob, jak by uživatel určil, který z nalezených rozdílů chce slít. Nakonec byly vytvořeny dva způsoby, jak vybrat daný rozdíl.

Prvním způsobem bylo užití slévacích tlačítek přímo v daném textovém pohledu. Ta byla nakreslena při kreslení svorek. Kromě toho, že pohled tlačítka nakreslil, zároveň si při jejich kreslení zapamatoval jejich polohu a index rozdílu a

porovnání, ke kterému se dané tlačítko vztahovalo. Při stisknutí tlačítka myši uživatelem nad daným pohledem pohled napřed zjistil, zda se uživatel netrefil do některého ze slévacích tlačítek (zjistit to nebylo příliš náročné, pohled si při každém kreslení zapamatoval pouze viditelná tlačítka a těch nemohlo být mnoho). V případě, že bylo zasaženo slévací tlačítko, předal pohled dokumentu žádost o provedení příslušné akce. Jinak bylo zpracování události předáno rodičovské třídě.

Jako druhý způsob pak bylo implementováno procházení rozdílů a funkce na slití aktuálního vybraného rozdílu. Řízení procházení i slévání bylo umístěno do třídy rámcového okna daného porovnání. To sice trochu zkomplikovalo návrh, ale umožnilo to uživateli otevřít si více samostatných procházení rozdílů ve více rámcových oknech zobrazujících totéž porovnání. Rámcové okno si tedy pamatovalo aktuální rozdíl a porovnání, ty umožňovalo měnit na základě uživatelských vstupů a výsledky pak předávalo v sobě obsaženým pohledům, aby mohly vybraný rozdíl graficky odlišit od rozdílů ostatních. Stejně tak rámcové okno předávalo uživatelské požadavky na slití příslušnému dokumentu.

Pro procházení a slévání rozdílů tímto způsobem byla vytvořena podnabídka hlavního menu, samostatný panel nástrojů a klávesové zkratky.

3.4. Porovnávání adresářů – obecně k návrhu řešení

První hrubý návrh řešení

Podobně jako u porovnání textů bylo cílem návrhu vytvořit přehledné porovnání adresářů, nástrojem byla knihovna MFC a architektura dokument/pohled.

Porovnávané adresáře měly být zobrazeny v okně s porovnáním vedle sebe. Zobrazení adresářů mělo mít podobu stromové struktury včetně možnosti adresáře rozbalovat a zabalovat. Sobě odpovídající položky porovnávaných adresářů měly být pomocí vynechávání místa zarovnány na stejnou y-ovou souřadnici a v případě, že byly adresáři, mělo se synchronizovat jejich rozbalování a zabalování. Tedy chování těchto adresářů bylo podobné, jako by se jednalo o adresář jeden, který je jejich sjednocením. Rozdílem by bylo to, že

v jednotlivých pohledech na adresáře byly zobrazeny jen ty položky, které se v daném adresáři vyskytovaly, a na místě těch, které v něm přítomny nebyly, by bylo vynecháno prázdné místo. Tato myšlenka (chování porovnávaných adresářů, jako by byly adresářem jedním) se pak stala základem implementace výše definovaného chování.

Podobně jako u porovnání textů bylo i zde rozhodnuto umístit veškerá data porovnání do dokumentu a jednotlivé adresáře zobrazovat v samostatných pohledech umístěných v rámcovém okně porovnání vedle sebe. Jednotlivé pohledy byly opět při vytvoření rámcového okna nastavením příslušné proměnné indexu připojeny ke svému adresáři, jehož obsah pak zobrazovaly a umožňovaly uživateli tento obsah měnit. Ještě předtím rámcové okno zavolalo metodu dokumentu pro vytvoření potřebného množství datových struktur reprezentujících tyto adresáře.

Výběr rodičovské třídy pohledu

Třída pohledu, kterou bylo třeba vytvořit pro výstup porovnání adresářů, měla umět kreslit stromové struktury adresáře, zvýraznit v nich nalezené změny a umožňovat vynechání místa mezi některými zobrazenými položkami (kvůli správnému zarovnání). Třída samozřejmě měla umožňovat rolování obsahu, aby šlo zobrazit i delší adresáře.

Třídu bylo možné vytvořit jako potomka třídy CView, nejobecnější třídy pohledu z knihovny MFC, a začít tak prakticky od nuly. Další možností bylo použít jako rodičovskou třídu CScrollView a ušetřit si alespoň práci s obsluhou rolování. Poslední možností pak bylo použít jako rodiče třídu CTreeView pro zobrazování stromových struktur.

Vzhledem k tomu, že třída CTreeView nedovoluje významněji měnit svůj grafický výstup a nebylo by tak možné dostatečně zvýrazňovat nalezené změny, natož pak vynechávat místo, byla tato možnost zavržena.

Tím nezbývalo nic jiného, než naprogramovat si vlastní, úloze na míru šitý prohlížeč stromových struktur porovnávaného adresáře. Rodičovskou třídou tohoto pohledu se z důvodu úspory práce s programováním vlastního mechanismu rolování stala třída CScrollView.

Použití třídy CScrollView jako rodičovské třídy pohledu

Výhodou použití této třídy je, že programátor se nemusí vůbec starat o obsluhu rolování. Sama třída posunuje v závislosti na rolování souřadnicový systém grafického kontextu pohledu. Kreslit se pak dá stejným způsobem, jako by k žádnému rolování nedocházelo.

Nicméně takový přístup je při kreslení rozsáhlejších grafických výstupů značně neefektivní. Třída CScrollView ale nabízí řešení – pomocí metody GetClipRect lze zjistit neplatnou část grafického výstupu a jednoduše překreslit jen ji. O zbytek se postará třída CScrollView.

Při použití této třídy v projektu se objevila zajímavá chyba, kterou zde nemohu nezmínit. Při testování již hotové třídy na rozsáhlejších adresářích nefungovalo rolování pomocí přetažení jezdce. Jezdec se po přetažení dolů vymrštil nahoru místo toho, aby zůstal na nově určeném místě a došlo k požadovanému rolování obsahu.

Chyba byla v samotné třídě CScrollView. Při použití mapovacího režimu MM_TEXT docházelo při přetažení jezdce u rozsáhlejších zobrazovaných dat k přetečení jedné z proměnných, ze které se vypočítávalo nové posunutí obsahu v pohledu. Ošetření této chyby se docílilo předefinováním virtuální metody OnScroll, kde bylo třeba při události přetažení jezdce nastavit správnou hodnotu proměnné nPos (správná hodnota se zjistila pomocí volání metody GetScrollInfo), která určovala novou polohu obsahu pohledu. (více k tomuto problému obsahuje článek [8])

Implementace porovnání a stromového zobrazení porovnaných adresářů

Pro reprezentaci porovnaného adresáře byla vytvořena stromová struktura, která obsahovala jména jednotlivých položek, cesty k nim a příznaky zda je daná položka v levém/pravém porovnání vložená, případně změněná. Dále obsahovala metody na načtení adresáře a porovnání adresáře s pravým sousedem (použit byl

rekurzivní algoritmus pro porovnání adresářů symbolicky zapsaný v kapitole o algoritmu porovnání).

Tato reprezentace dostatečně obecně řešila datovou reprezentaci porovnávaného adresáře. Těchto adresářů bylo možné umístit do dokumentu libovolný počet a pak je propojit porovnáním mezi sousedy.

Co ale tato reprezentace neřešila, bylo zobrazení daných dat. Jak již bylo řečeno, položky všech adresářů porovnání měly být navzájem správně zarovnány a rozbalování a zabalování sobě odpovídajících adresářů mělo probíhat synchronizovaně. Položky všech porovnávaných adresářů se tedy měly chovat, jako by byly součástí jednoho společného adresáře, zároveň však měly být zobrazeny v samostatných pohledech, které se týkaly vždy jen jednoho z porovnávaných adresářů.

Tento problém se vyřešil přidáním pomocné třídy, která reprezentovala sjednocení všech adresářů porovnání a sloužila k reprezentaci dat potřebných pro grafický výstup. Tato třída umožňovala pro sjednocený adresář vypočítat rozsahy umístění (v řádkách) jednotlivých obsažených položek (sjednocených podadresářů a souborů) a také obsahovala metody pro rozbalování a zabalování obsažených sjednocených adresářů. Každá položka obsahovala odkazy na odpovídající položky původních samostatných adresářů a metody pro přístup k jejich datům potřebným pro zobrazení adresářů (např. šlo o výsledky porovnání).

Toto řešení pomocí dvojí reprezentace adresářů sice částečně obsahuje redundantní informace, domnívám se ale, že je pro programátora přehlednější. Je zde totiž jasně oddělena reprezentace porovnání adresářů od pomocných dat grafického výstupu. Reprezentace porovnání adresářů je tím navíc znovupoužitelná pro jiné typy zobrazení výsledku.

Pomocná třída sjednocení adresářů sama neobsahuje konkrétní funkce pro kreslení obsahu, či pro obsluhu uživatelských akcí jako je kliknutí myši a podobně, ale obsahuje metody pro přijetí objektu, který rozumí jejímu obsahu a může přistupovat k jejím datům. Právě do objektů tohoto typu byla umístěna zmíněná funkčnost. Tím bylo do jisté míry dosaženo nezávislosti této pomocné třídy na např. použité knihovně pro práci s grafickým výstupem, okenním systémem, apod. Navíc to umožňuje snadno změnit způsob kreslení či obsluhu zmíněných událostí, aniž by bylo třeba měnit tuto třídu.

4. Implementace

4.1. Základní třídy projektu a jejich spolupráce

Aplikace je napsána v jazyce C++ za použití knihovny MFC.

Následující text obsahuje seznam nejdůležitějších tříd programu, popis jejich účelu v programu a vzájemné spolupráce. Více podrobností lze nalézt přímo ve zdrojových textech programu. Téměř každou funkci provází krátký komentář, který vysvětluje její účel a případně i použití.

- **třída aplikace - CMergeApp**

  reprezentuje samotnou MDI aplikaci, registruje základní typy dokumentů a provádí jejich správu. Uchovává šablony jednotlivých typů dokumentů, které umožňují vytváření a zavírání těchto dokumentů. Třída je potomkem třídy **CWinApp** knihovny MFC.

- **třída hlavního okna - CMainFrame**

  je třídou hlavního okna aplikace. Spravuje dětská okna s jednotlivými porovnáními. Umožňuje tedy otevírat v samostatných dětských oknech nová porovnání požadovaného typu (porovnání dvou nebo tří textových souborů nebo adresářů), stejně jako tato porovnání zavírat. Třída je potomkem třídy **CMDIFrameWnd** knihovny MFC.

- **třídy dětských rámcových oken:**

  pomocí třídy **CSplitterWnd** vedle sebe umisťují pohledy na jednotlivé texty nebo adresáře daného porovnání.

  V případě porovnání textů jsou pohledy instance tříd **CMergeTextView** (zobrazuje text) a **CLinesView** (zobrazuje svorky). Mezi sousedními pohledy textového typu je jeden pohled svorkového typu, který je propojuje.

  V případě porovnání adresářů jsou všechny pohledy typu **CMergeDirView**.

  Při svém vytvoření se postará o vytvoření příslušného počtu textů nebo adresářů v dokumentu a napojí na ně v sobě umístěné pohledy. Mimo to nastavuje pohledům parametry zobrazení a stará se o správné vzájemné zarovnání těchto oken a k nim patřícím prvkům uživatelského rozhraní (panelů nástrojů).

Třídy dětských oken jsou potomky třídy **CMDIChildWnd**.

- **CChild2TextsFrame** – dva texty
- **CChild3TextsFrame –** tři texty
- **CChild2DirsFrame –** dva adresáře
- **CChild3DirsFrame –** tři adresáře

- **třídy dokumentů porovnání:**

  datově reprezentují porovnání dvou a více textů či adresářů, kdy porovnávány jsou vždy dva sousední texty nebo adresáře. Jsou potomky třídy **CDocument** knihovny MFC.

  - **CMergeTextDoc -** obsahuje datovou reprezentaci textů a jejich porovnání a metody pro porovnávání a slévání textů a další operace s texty. Spolupracuje s třídami pohledů **CMergeTextView** a **CLinesView**, které slouží k zobrazení obsažených textů a výsledků jejich porovnání.

  - **CMergeDirDoc -** obsahuje datovou reprezentaci adresářů a jejich porovnání a metody pro porovnávání a slévání adresářů. Spolupracuje s třídou pohledu **CMergeDirView**, která slouží k zobrazení porovnávaného adresáře.

- **třídy pohledů:**

  slouží ke grafickému zobrazení výsledků porovnání.

  - **CMergeTextView -** slouží ke grafickému zobrazení jednoho z porovnaných textů ze třídy **CMergeTextDoc**. Při zobrazení nalezených rozdílů úzce spolupracuje s třídou **CLinesView**. Ta totiž zobrazuje grafické svorky propojující rozdíly ve dvou porovnávaných textech, svorky pak pokračují i v tomto pohledu. Třída je potomkem třídy **CCrystalEditView**, která je univerzální open-source editační textovou komponentou s podporou zobrazení syntaxe programovacích jazyků(více viz [3]).

- **CLinesView -** slouží ke grafickému zobrazení rozdílů dvou porovnávaných sousedních textů pomocí grafických svorek. Spolupracuje s třídou **CMergeTextView**, která tyto texty zobrazuje, dále s třídou dokumentu **CMergeTextDoc**, která obsahuje datovou reprezentaci zobrazovaného porovnání. Třída je potomkem třídy **CView** knihovny MFC.

- **CMergeTextView -** slouží ke grafickému zobrazení jednoho z porovnaných adresářů ze třídy **CMergeDirDoc**. Zobrazuje strom adresářové struktury obarvený na základě porovnání se sousedními adresáři. Třída je potomkem třídy **CScrollView** knihovny MFC.

- **cizí třídy (viz** [3]**):**

  - **CCrystalTextView -** zobrazuje text uložený ve třídě **CCrystalTextBuffer**, podporuje zvýraznění syntaxe. K zdroji dat (textu uloženém ve třídě **CCrystalTextBuffer**) přistupuje virtuální metodou LocateTextBuffer. Třída je potomkem třídy **CView** knihovny MFC.

  - **CCrystalEditView -** je potomkem třídy **CCrystalTextView** knihovny MFC, k níž přidává možnost editace zobrazeného textu.

  - **CCrystalTextBuffer -** obsahuje datovou reprezentaci textu zobrazeného předchozími dvěma třídami. Kromě seznamu řetězců jednotlivých řádek textu obsahuje metody pro editaci textu, práci se soubory a metody pro undo a redo akce.

4.2. Texty – objekty

Tato kapitola obsahuje podrobnější popis tříd použitých v textovém porovnání, zejména třídy CMergeTextDoc.

Třída dokumentu porovnání **CMergeTextDoc** obsahuje několik vnitřních tříd. Jsou to třídy CMergeTextBuffer (reprezentuje jeden porovnávaný text), CUndoHistory (vytváří undo historii celého porovnání nad undo historií implementovanou v textovém bufferu) a CTextComparison.

Třída **CMergeTextBuffer** je potomkem třídy CCrystalTextBuffer, která obsahuje text a umožňuje jej editovat. Ve třídě CMergeTextBuffer je pozměněna část metod rodičovské třídy a několik doplňujících pomocných metod bylo přidáno.

Změnou účelu metody SetModified bylo dosaženo možnosti detekovat všechny změny textu a upozorňovat na ně dokument, aby v reakci na tyto změny mohl aktualizovat datové struktury porovnání. Dosáhlo se toho možná trochu nepěkným trikem, kdy při každém nastavení proměnné určující, zda byl text modifikován, byla tato proměnná opět odnastavena, aby se metoda musela zavolat i při další změně. Pro zjištění, zda byl text modifikován pak byla přidána nová proměnná a nové metody (SetTextModified a GetTextModified), které plnily původní úlohu metod SetModified a GetModified. Důvod výběru tohoto řešení byl ten, že CCrystalTextBuffer původně nenabízel žádnou možnost, jak pohodlně reagovat na změny textu z jednoho místa. Samotná metoda SetModified byla volána vždy jen tehdy, pokud proměnná určující, že text byl změněn, byla nastavena na neplatnou hodnotu (tj. jen při první změně).

Další změněnou metodou byla metoda AddUndoRecord, k jejíž původní funkčnosti byla přidána spolupráce s undo historií porovnání v dokumentu. Nově tedy metoda přidávala záznamy do obou historií.

Třída **CUndoHistory** uchovává záznamy typu CUndoHistoryRecord. Třída **CUndoHistoryRecord** obsahuje pouze počet změn sdružených dohromady a index textu, kterého se dané změny týkají.

Třída CUndoHistory vytváří undo historii nad undo historiemi jednotlivých textových bufferů. Její metody CanUndo, Undo, CanRedo, Redo fungují tak, že pouze zavolají stejnojmenné metody textového bufferu, kterého se týká aktuální záznam historie.

Třída **CTextComparison** reprezentuje porovnání dvou textů. Obsahuje data s výsledkem tohoto porovnání, jako je seznam indexů společných řádek a jejich vzájemné mapování (hodí se v případě, že je třeba rychle zjistit, která řádka sousedního textu odpovídá dané řádce), seznam rozdílných intervalů řádek a seznam rozdílných intervalů znaků při detailním porovnání a další. Dále obsahuje metody pro výpočet a aktualizaci těchto dat na základě voleb porovnání nastavených uživatelem.

Třída **CMergeTextDoc** reprezentující porovnání n textů obsahuje n proměnných typu CMergeTextBuffer a n-1 typu CTextComparison. Dále jednu proměnnou s undo historií typu CUndoHistory a proměnné určující způsob porovnání (chování vzhledem k bílým znakům a prázdným řádkům, rozlišování velkých a malých písmen, volby synchronizovaného rolování a detailního porovnání). Dále poskytuje metody pro práci se soubory (otevírání, ukládání atd.), pro aktualizaci porovnání a metody pro slévání. Tyto metody většinou pouze jednoduchým způsobem využívají služeb zmíněných obsažených objektů.

Třída **CMergeTextView** slouží k zobrazení jednoho z porovnaných textů ze třídy CMergeTextDoc. Obsahuje data, která umožňují nastavit některé parametry zobrazení. Patří sem barvy, použité štětce, čáry apod., ale také ukazatel na instanci třídy CParser.

Třída **CParser** obsahuje virtuální metodu ParseLine, kterou volá třída CMergeTextView ve své vlastní stejnojmenné metodě. To, že třída pohledu takto předává tuto funkčnost objektu, na nějž má odkaz, umožňuje změnou typu tohoto objektu za běhu měnit algoritmus zvýraznění syntaxe a měnit tak na žádost uživatele programovací jazyk, jehož syntaxe je zvýrazňována. Kromě třídy CParser, jejíž metoda ParseLine nedělá nic (defaultní chování, odpovídá volbě vypnutého zvýraznění syntaxe), obsahuje třída další vnitřní třídy **CParserCPlusPlus** a **CParserJava**, které jsou potomky třídy CParser a poskytují tutéž funkčnost pro jazyky C++ a Java.

Třída CMergeTextView dále obsahuje metody pro kreslení porovnaného textu, pro obsluhu undo a redo akcí uživatele, obsluhu slévací tlačítek a pro synchronizaci rolování. Krom toho má tato třída na starost upozornění svých svorkových sousedů, kdykoliv se ve svislém směru přesune její obsah.

Základní metody kreslení jsou – OnDraw (celý grafický výstup), DrawSingleLine (jeden řádek), DrawMargin (okraj jednoho řádku), DrawLines (svorky) a DrawButtons (slévací tlačítka).

Metoda OnDraw nejprve volá stejnojmennou metodu rodičovské třídy, která pomocí volání virtuálních metod DrawSingleLine a DrawMargin pro všechny viditelné řádky nakreslí text a jeho okraj. Poté jsou volány metody DrawLines a DrawButtons, které do textu dokreslí svorky a slévací tlačítka. Metoda DrawSingleLine slouží k nakreslení jednoho řádku. Nejprve je opět zavolána metoda rodičovské třídy, která nakreslí text řádku se zvýrazněnou syntaxí, pak je do textu řádky dokresleno případné zvýraznění nalezených rozdílů. Metoda DrawMargin kreslí okraj řádku s jeho číslem, výstup této metody v rodičovské třídě měl jiný účel (zobrazení záložek a zarážek).

Obsluha slévacích tlačítek se děje v rámci metody OnLButtonDown, kde se zjistí, zda bylo při kliknutí myší zasaženo tlačítko, a pak je buď vyvolána akce patřící k tlačítku (v případě zásahu), nebo je zpracování předáno rodičovské třídě.

Obsluha undo a redo akcí a aktualizace souvisejícího uživatelského rozhraní se děje v rámci metod OnEditUndo, OnEditRedo, OnUpdateEditUndo a OnUpdateEditRedo. Všechny tyto metody nejprve zkontrolují, zda je daný pohled tím, který by měl tyto akce zpracovat (tj. zda pohledem zobrazovaný text je text, kterého se akce týkají), a podle výsledku buď akci zpracují, nebo ji předají ke zpracování pohledu, kterému akce patří. To se děje zavoláním téže metody na příslušný pohled.

Synchronizace rolování je naprogramována v rámci metod SynchronizeScrolling a ScrollLineToY. První z nich volá synchronizující pohled, v rámci ní je pak volána metoda ScrollLineToY na synchronizované sousedy tohoto pohledu. Metoda ScrollLineToY slouží k přesunutí obsahu pohledu tak, aby daná řádka byla umístěna na dané y-ové souřadnici. Metoda SynchronizeScrolling slouží k synchronizaci sousedů po přesunu obsahu pohledu. V rámci metody je zjištěno, která řádka sousedního pohledu odpovídá prostřední řádce rolovaného pohledu, a tato řádka je pak voláním metody ScrollLineToY tohoto souseda přesunuta na stejnou polohu jako prostřední řádka volajícího pohledu.

Kromě zmíněných tříd je součástí textového porovnání i sada globálních funkcí. Ty se nacházejí v souborech compare.h a compare.cpp a dokument porovnání a v něm obsažené třídy tyto funkce využívají.

Jsou to funkce obsahující implementaci porovnávacího algoritmu a předzpracování jeho vstupu na základě voleb porovnání. Samotný algoritmus (z kapitoly o algoritmu textového porovnání) je zapsán jako šablona, takže jej lze použít nejen na posloupnosti řádek (případně písmen) jako v textovém porovnání, ale na porovnání posloupností libovolného typu.

4.3. Adresáře – objekty

Tato kapitola obsahuje podrobnější popis tříd použitých v porovnání adresářů.

K reprezentaci jednotlivých adresářů slouží třídy CDirCmpItem, CDirItem a CFileItem.

Třída **CDirCmpItem** je abstraktním předkem zbylých dvou tříd. Reprezentuje obecnou položku porovnávaného adresáře a pomocí sady virtuálních metod definuje společnou funkčnost všech těchto položek. Mezi společné metody patří metody pro zjištění jména a cesty k položce, metody pro slévání a porovnání a metoda AcceptVisitor, která umožňuje procházet, číst a měnit stromovou strukturu dalším objektem, jehož třída je konkrétním potomkem abstraktní třídy **CDirCmpItemVisitor** (jediným účelem tohoto objektu a jeho potomků je vnější přidání nové funkčnosti k třídám CDirItem a CFileItem).

Třída **CFileItem** reprezentuje soubor a implementuje čistě virtuální metody abstraktní rodičovské třídy CDirCmpItem.

Třída **CDirItem** reprezentuje adresář. Kromě toho, že implementuje čistě virtuální metody abstraktní rodičovské třídy CDirCmpItem a část jejích virtuálních metod s naprogramovaným defaultním chováním předefinovává, definuje vlastní metody a data. K datům patří především pole ukazatelů na objekty typu CDirCmpItem, které reprezentuje položky daného adresáře (tedy obsažené soubory a podadresáře). Pro přístup k těmto položkám obsahuje metody GetChild a GetChildCount. Dále obsahuje metody pro načtení adresáře.

K reprezentaci pomocných slitých adresářů slouží podobná trojice tříd. Jsou to třídy CMergedDirCmpItem, CMergedDirItem a CMergedFileItem.

I zde je třída **CMergedDirCmpItem** společným abstraktním předkem zbývajících dvou tříd. Definuje společné metody pro vytvoření této pomocné datové struktury slitím s odpovídajícími objekty tříd CDirItem a CFileItem. Dále obsahuje metodu pro zjištění cesty a jména n-té slité položky a metody pro výpočet rozsahu dané položky v řádcích. Umožňuje přistupovat k původním adresářům/souborům, jejichž slitím daný objekt vznikl. Třída také obsahuje metodu AcceptVisitor pro přijetí objektu s dodatečnou funkčností (která se třídy týká, ale nebylo vhodné ji naprogramovat přímo do ní).

Třída **CMergedDirItem** reprezentuje slitý adresář. Obsahuje metody pro zabalení a rozbalení adresáře a mění metodu pro vypočítání rozsahu.

Třída **CMergedDirItem** reprezentuje slitý soubor.

Abstraktní třída **CMergedDirCmpItemVisitor** obsahuje čistě virtuální metody pro „návštěnu“ tříd CMergedDirItem a CMergedFileItem, tj. procházení adresářové struktury a provádění určitých akcí s jejími položkami. Prostřednictvím jejích potomků tak lze přidat novou funkčnost, aniž by byly změněny třídy CMergedDirItem a CMergedFileItem. V projektu jsou použiti dva potomci této třídy a to třídy CPainter a CClickedVisitor.

Třída **CPainter** umožňuje nakreslení vybrané části daného adresáře do zadaného kontextu zařízení. Zároveň při kreslení nastaví datové položky určující obdélník umístění popisku položky v pohledu, umístění zabalovacího a rozbalovacího tlačítka adresáře atd.

Třída **CClickedVisitor** slouží ke zpracování uživatelské akce kliknutí do pohledu s adresářem. Rozlišuje různé typy kliknutí – dvojité, jednoduché, pravým tlačítkem myši. Využívá informací nastavených při kreslení třídou CPainter.

Datovou část porovnání reprezentuje třída dokumentu **CMergeDirDoc**. Tato třída reprezentující porovnání n adresářů obsahuje n proměnných typu CDirItem a jednu pomocnou proměnnou typu CMergedDirItem, která obsahuje všech n adresářů slitých do jediného adresáře a usnadňuje kreslení grafického výstupu porovnání.

Dále třída obsahuje metody pro načítání adresářů, jejich slévání, aktualizaci porovnání a pro otevření porovnání vybraných textů. Většina metod, které se týkají porovnávaných adresářů, jejich slévání a aktualizace, využívá služeb tříd, které reprezentují tyto adresáře.

Prostředníkem mezi uživatelem a daty porovnání v dokumentu je třída pohledu **CMergeDirView**. Kreslí obsah adresáře, k němuž je připojená, a zpracovává akce uživatele – především kliknutí na klientskou část tohoto pohledu. Pro kreslení i zpracování kliknutí užívá služeb tříd CPainter a CClickedVisitor.

5. Program Merge z pohledu uživatele

5.1. Obecně

Program Merge je nástroj sloužící k porovnávání různých verzí adresářů a textových souborů. Zadané soubory či adresáře umožní otevřít v jednom okně vedle sebe a porovnat.

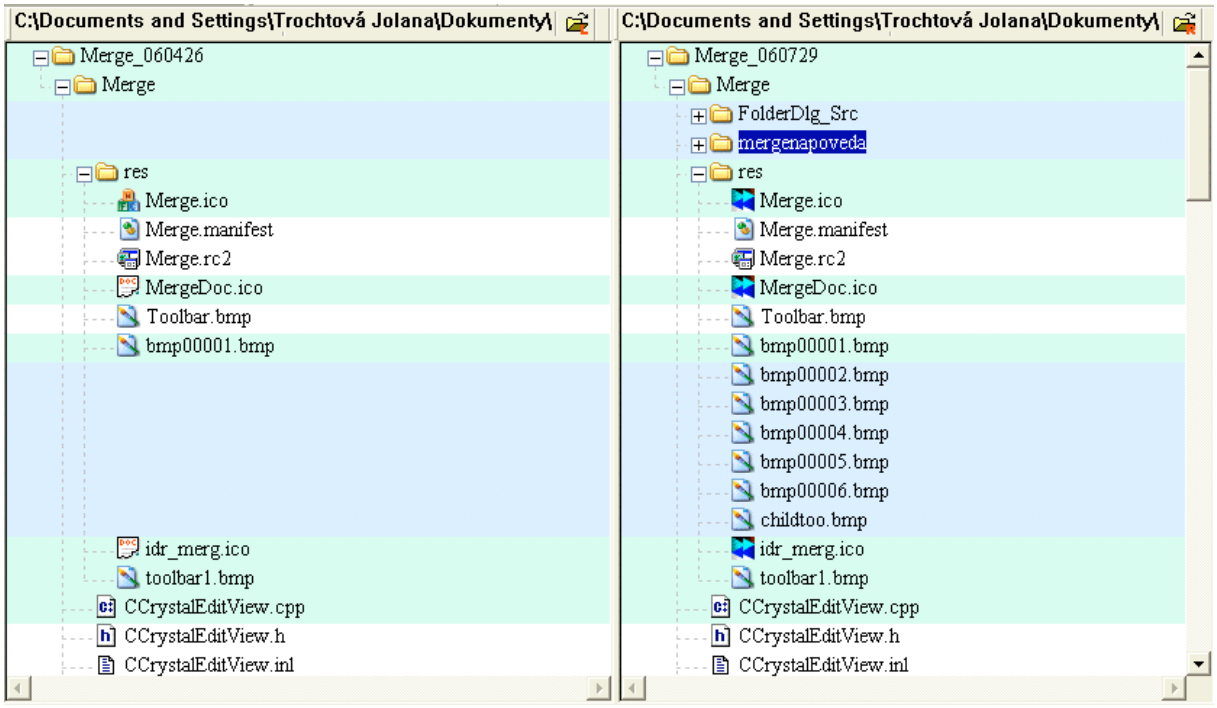
Nalezené rozdíly graficky zvýrazňuje a umožňuje snadné slévání různých verzí přenášením nalezených změn mezi porovnávanými soubory nebo adresáři. Porovnávat lze dva nebo tři texty, případně dva nebo tři adresáře. V případě porovnávání tří textů nebo adresářů je prostřední z nich považován za společného předka a krajní jsou s ním porovnávány.

Merge umožňuje mít zároveň otevřeno více porovnání a to i různých typů (porovnání adresářů nebo souborů, dvou nebo tří).

5.2. Adresáře

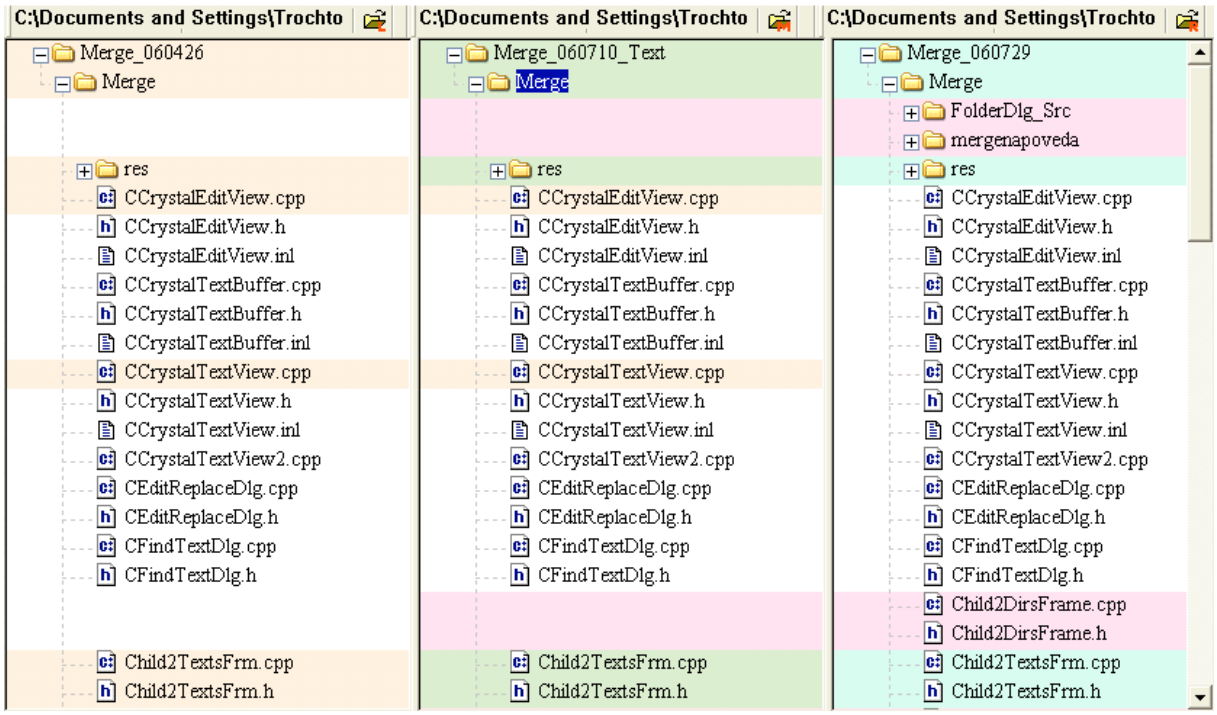
Jednotlivé soubory a adresáře detekované v různých verzích jako odlišné, případně vložené či naopak chybějící, jsou barevně odlišeny od souborů ostatních. Krom toho jsou adresářové složky i soubory v sousedních porovnávaných adresářích umístěny tak, aby sobě odpovídající verze souborů a adresářů byly vedle sebe.

Porovnávat lze adresáře dva



Obr. 5.2.1 – porovnání dvou adresářů. Modré položky adresáře jsou vložené, zelené změněné

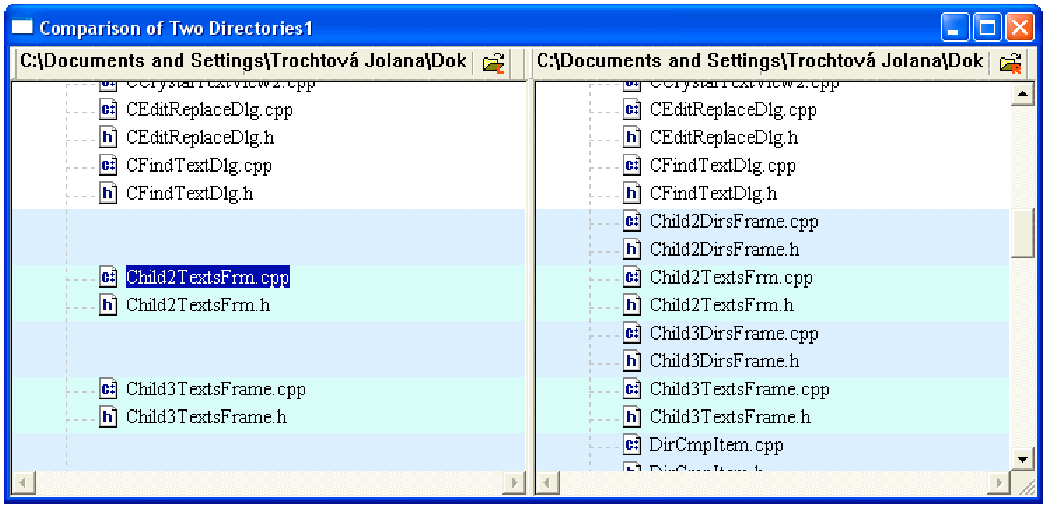
nebo tři.



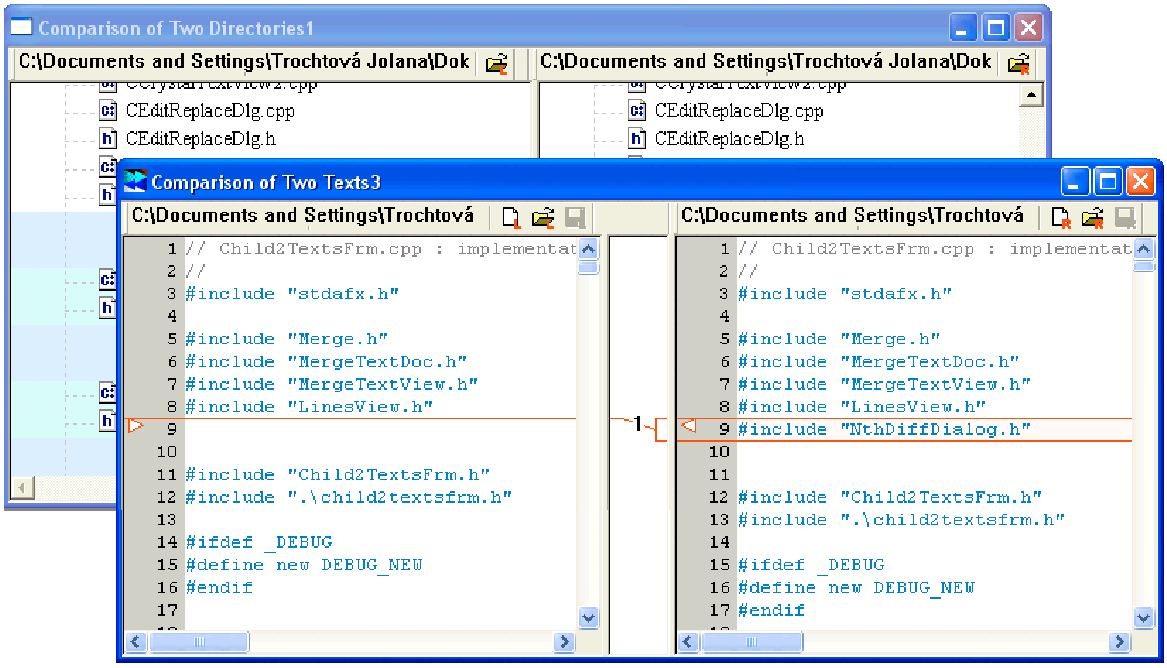
Obr. 5.2.2 – porovnání tří adresářů. V levém porovnání jsou oranžově zvýrazněné změněné položky, v pravém jsou zeleně zvýrazněné změněné položky a růžově vložené

Při porovnávání adresářů lze otevřít textové porovnání textů detekovaných jako rozdílných v daných verzích a prohlédnout si jejich vzájemné rozdíly a slévat

je na nižší úrovni. Stejně tak je ale možné adresářové struktury slévat kopírováním celých souborů a dokonce i celých adresářů.



Obr. 5.2.3. po dvojitém kliknuti na soubor, který je v pohledu vybrán, bude otevřeno jeho textové porovnání se sousedem

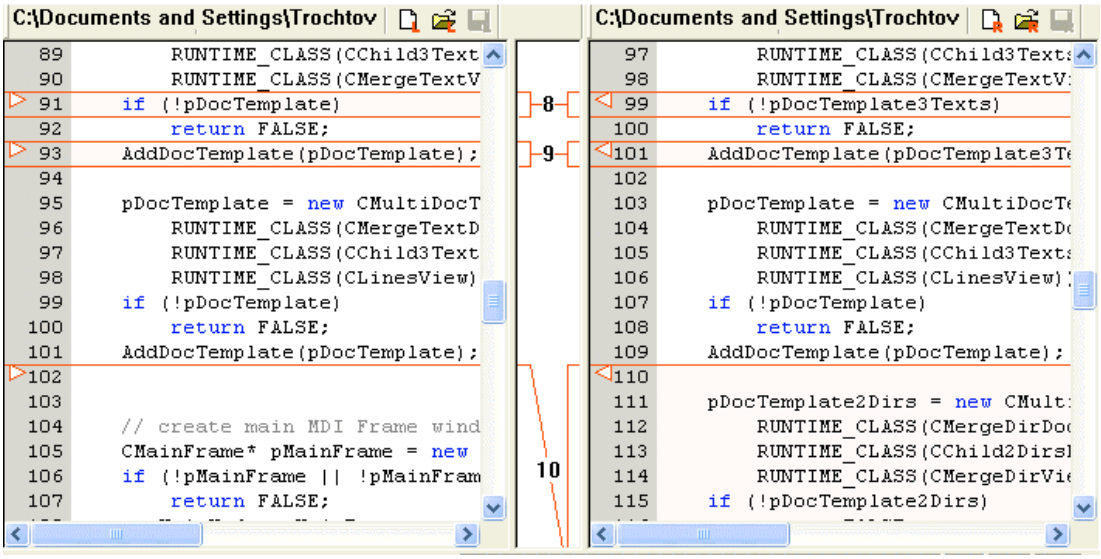


Obr. 5.2.4. stav po otevření textového porovnání vybraných souborů

5.3. Texty

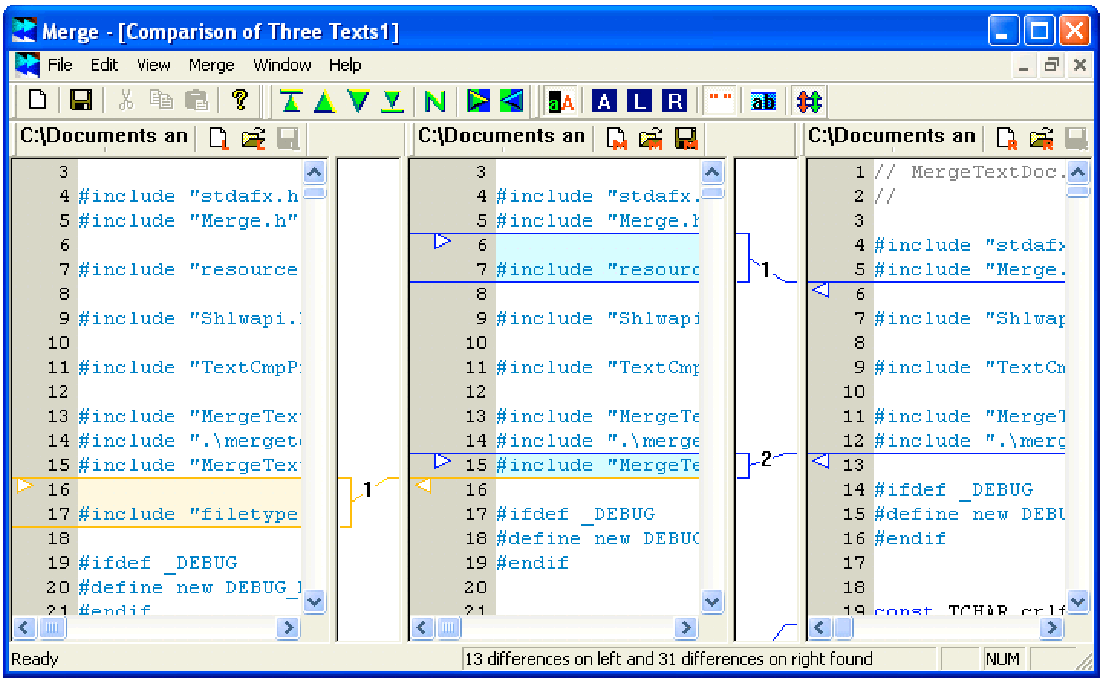
Rozdíly nalezené při porovnávání textů jsou barevně zvýrazněny, krom toho jsou sobě odpovídající rozdílné úseky porovnávaných sousedních textů propojeny pomocní grafických svorek.

Opět lze porovnávat dva



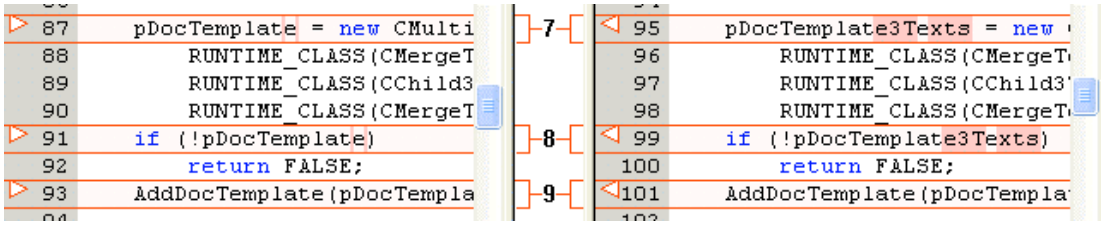
Obr. 5.3.1 porovnání dvou textů

nebo tři texty.



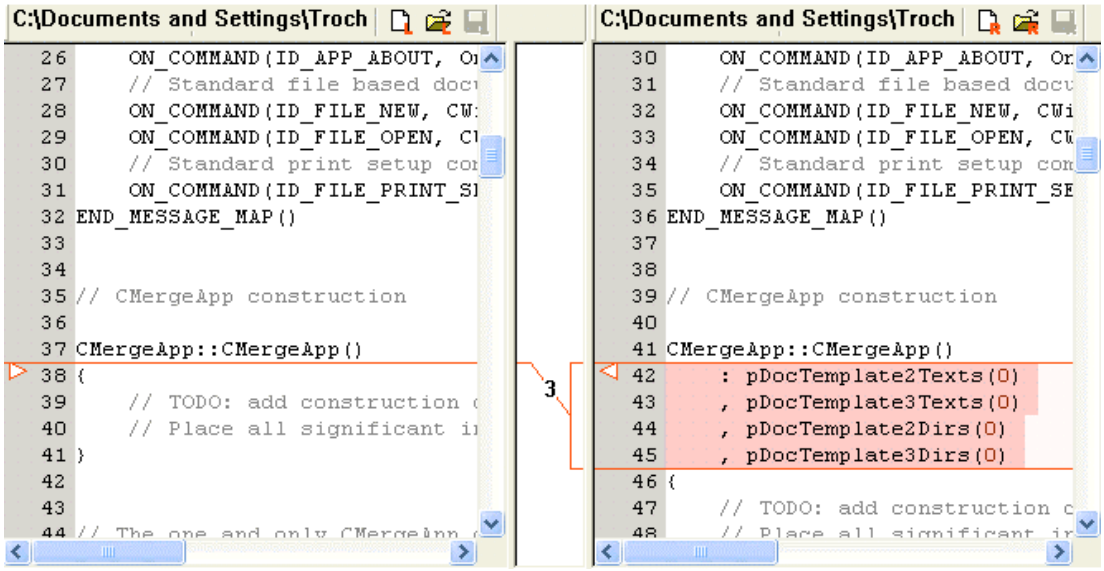
Obr. 5.3.2 porovnání tří textů

Kromě nalezení a zvýraznění rozdílných řádek lze na přání uživatele nalézt a zobrazit i rozdíly v jednotlivých znacích v rámci detailního porovnání dvou rozdílných sobě odpovídajících úseků řádků porovnávaných textů.

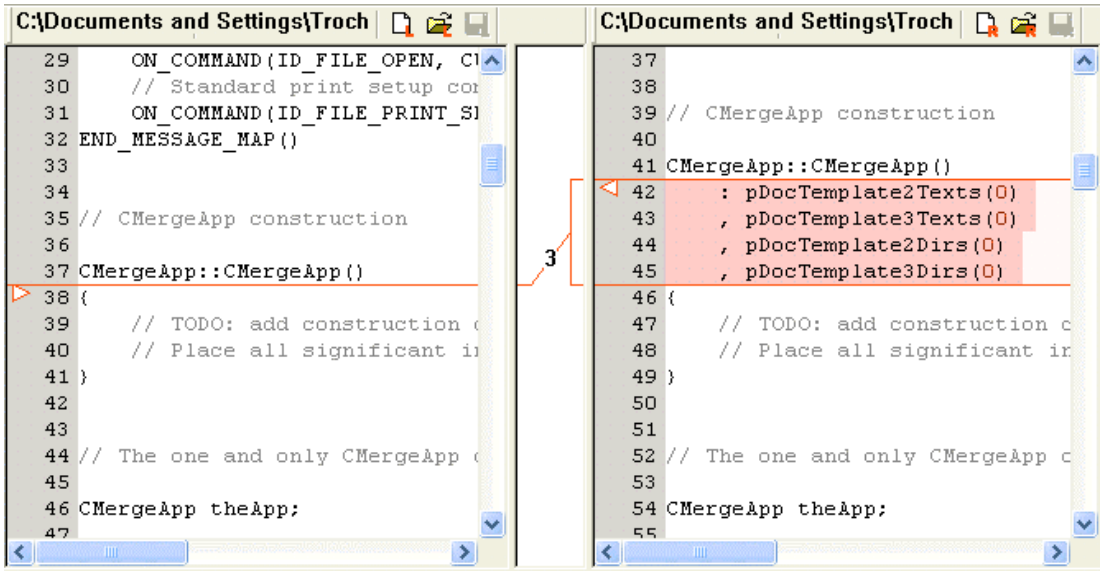


Obr. 5.3.3 detail – podrobné porovnání textů

Pro další zvýšení přehlednosti obsahuje program volitelnou vlastnost synchronizovaného rolování porovnávaných textů. Při této volbě pohyb s jedním textem ovlivňuje pohyb ostatních textů v porovnání tak, aby společné řádky textů byly pokud možno zarovnány na stejnou úroveň.

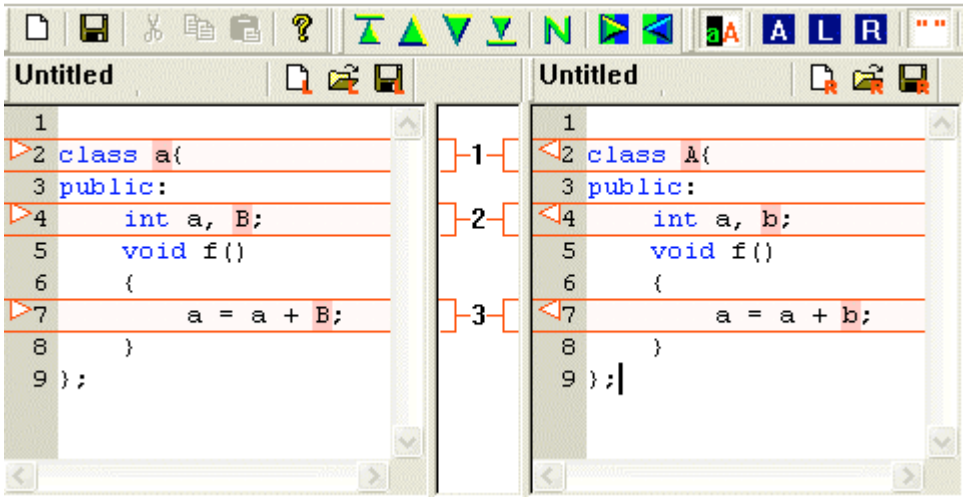


Obr. 5.3.4 synchronizované rolování – před posunutím levého textu dolů

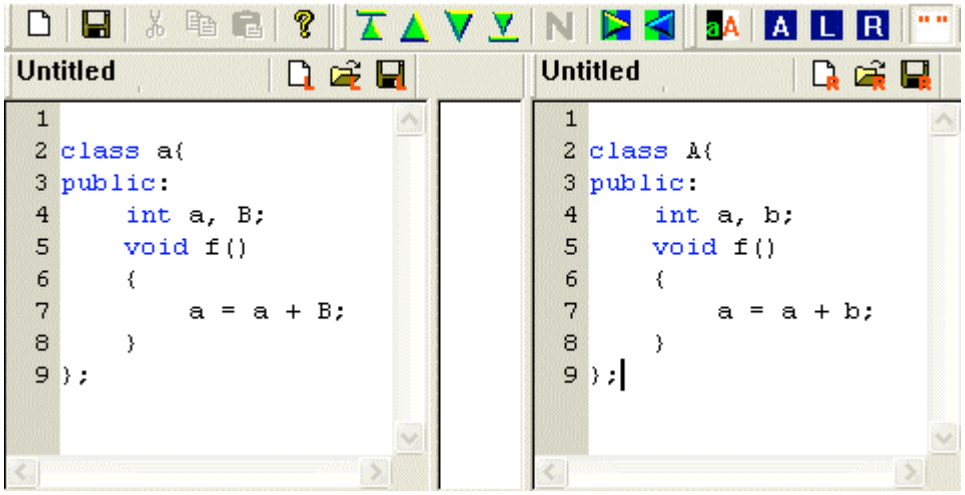


Obr. 5.3.5 synchronizované rolování – po posunutím levého textu dolů

Porovnávání textů lze ovlivnit množstvím voleb, které především ovlivňují chování porovnávacího algoritmu k bílým znakům, prázdným řádkům a rozdílům ve velikosti písmen. Je jen na uživateli, zda považuje tyto změny v různých verzích za stejně významné jako změny ostatní. V případě rozdílu mezi velkými a malými písmeny lze jejich rozlišování vypnout úplně.



Obr. 5.3.6 zapnutá volba case-sensitive porovnání



Obr. 5.3.6 vypnutá volba case-sensitive porovnání

Vzhledem k předpokládanému užití programu v praxi umožňuje program zobrazovat porovnávané texty se zvýrazněnou syntaxí některých programovacích jazyků (podporovány jsou jazyky C++ a Java).

6. Závěr

6.1. Zhodnocení programu vzhledem k cílům práce

Domnívám se, že program Merge splnil základní cíle svého vývoje. Stal se z něj program použitelný pro porovnávání a slévání adresářů i v nich obsažených textových souborů.

U obou typů porovnání předkládá výsledek své práce v přehledné a uživateli dobře srozumitelné formě. Porovnávané položky jsou zobrazeny vedle sebe, takže uživatel neztrácí při porovnání představu o podobě jednotlivých porovnávaných adresářů a souborů. Nalezené rozdíly jsou barevně odlišeny, u textového porovnání jsou navíc přidány svorky propojující odpovídající si rozdílné úseky a možnost synchronizovaného rolování, které výrazně zvyšuje pohodlí uživatele při prohlížení delších porovnávaných textů (u porovnání adresářů je synchronizace rolování obsahu sousedních adresářů dána automaticky charakterem zobrazení). Do textového porovnání bylo navíc oproti původním požadavkům přidáno volitelné zvýraznění syntaxe.

Především porovnání textů, na které měl být kladen největší důraz, poskytuje uživateli většinu funkcí a možností, které je schopen běžně využít. Jsou zde možnosti měnit chování porovnávacího algoritmu množstvím voleb, slévat texty, editovat je jako v jakémkoliv běžném textovém editoru včetně hledání a náhrad vzorů.

Porovnání adresářů je sice na funkce o něco chudší, ale rozhodně dává uživateli dobrou představu o vzájemném vztahu adresářů a je navrženo dostatečně obecně, takže není omezeno v budoucím rozšiřování.

Původní požadavek na budoucí rozšiřitelnost programu o nové typy porovnání je splněn díky použité architektuře. Rozšiřovat a vylepšovat lze samozřejmě i již existující typy porovnání. Více k tomuto tématu obsahuje následující kapitola.

6.2. Možná rozšíření do budoucna

Program Merge po svém jednoletém vývoji dospěl do stadia, kdy se již stal v praxi použitelným nástrojem. Přesto zůstává nespočet funkcí, které by se na něm daly dále vylepšovat.

Program není nijak omezen v možnosti budoucího přidání dalších typů porovnání. Nové typy porovnání lze přidat stejným způsobem, jako byly přidány stávající typy porovnání. Je zde tedy možnost přidání porovnání obrázků, binárních souborů či třeba porovnání některých souborů se stromovou strukturou – jako např. XML souborů, pokud by nestačilo jejich porovnání jako textů a bylo by místo toho požadováno porovnání více rozumějící jejich struktuře.

Ani stávající typy porovnání nejsou zcela definitivně hotové. Je zde prostor pro přidávání nových funkcí.

U obou typů porovnání lze zmínit třeba automatické slévání tří porovnávaných položek s detekcí kolidujících změn.

U textového porovnání by se daly dál rozšiřovat možnosti nastavení porovnání, např. o možnost nastavení významnosti/nevýznamnosti změn v řádkách odpovídajících zadanému regulárnímu výrazu. Užitečnou funkcí by jistě bylo označení slitých a editovaných řádek porovnávaných textů. Rozšířit by se daly možnosti zvýraznění syntaxe – vždyť zatím jsou podporovány pouze dva programovací jazyky. Velmi užitečná by byla i podpora dalších znakových sad, to by však vyžadovalo rozsáhlejší změny použité cizí textové komponenty. A jistě by se našlo velké množství dalších možných vylepšení.

Co se týče porovnání adresářů, dala by se rozšiřovat sada operací se soubory a adresáři. Užitečné by jistě bylo i rozšíření o undo historii a možnosti vracení provedených akcí zpět.

Téměř neustále by se dalo vylepšovat také uživatelské rozhraní.

Jistě by se našlo mnoho dalších námětů na vylepšení, to je ale spíše již otázka dlouhodobějšího vývoje a zapracování zkušeností s užíváním programu.

7. Seznam použité literatury

[1] Jeff Prosise (2000): Programování ve Windows pomocí MFC, Computer Press

[2] MSDN Library: http://msdn.microsoft.com

[3] The Code Project: http://www.codeproject.com

[4] Araxis Merge: http://www.araxis.com/merge/index.html

[5] Winmerge: http://winmerge.org

[6] Windiff: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tools/tools/windiff.asp

[7] Longest common subsequence problem – http://en.wikipedia.org/wiki/Longest-common_subsequence_problem

[8] PRB: CScrollView Scroll Range Limited to 32K: http://www.kbalertz.com/166473/CScrollView.Scroll.Range.Limited.aspx

8. Přílohy

Příloha A. Cizí části softwaru použité v projektu

Kromě standardních součástí knihovny MFC byly použity i některé části volně šiřitelného kódu nalezené na Internetu. Následuje seznam všech souborů v projektu s cizím zdrojovým kódem (původ viz [3]):

CCrystalEditView.cpp
CCrystalTextBuffer.cpp
CCrystalTextView.cpp
CCrystalTextView2.cpp
CEditReplaceDlg.cpp
CFindTextDlg.cpp
cplusplus.cpp
FolderDlg.cpp
CCrystalEditView.h
CCrystalTextBuffer.h
CCrystalTextView.h
CCrystalTextView2.h
cedefs.h
CEditReplaceDlg.h
CFindTextDlg.h
cplusplus.h
editcmd.h
editreg.h
FolderDlg.h