Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Aplikácie konečných transformácií

(Bakalárska práca)

Miroslav Hoták

Vedúci: Prof. RNDr. Branislav Rovan, PhD.

Bratislava, 2009

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Poďakovanie

Rád by som sa poďakoval vedúcemu svojej bakalárskej práce profesorovi Branislavovi Rovanovi za cenné rady a inšpirácie pri jej tvorbe.

Abstrakt

V tejto práci sa popisuje návrh a realizácia prostredia umožňujúceho jednotným spôsobom pomocou grafického uživateľského rozhrania transformovať údaje medzi rôznymi formátmi s využitím externých transformačných programov s konzolovým uživateľským rozhraním.

Kľúčové slová: *formát, transformácia, transformačný predpis, GUI*

Obsah

Kapitola 1

Úvod

Informačné technológie človeku umožňujú v súčasnosti uchovávať v digitálnej forme údaje rozličnej povahy (text, audio, video, grafika, ...). Ucelená množina digitalizovanej informácie je v počítači väčšinou uložená vo forme súboru. Formát súboru určuje, akým spôsobom sa informácia kóduje do binárnych hodnôt pri ukladaní do súboru, a na druhej strane by mal poskytovať návod ako spätne reprezentovať uloženú informáciu.

1.1 Prečo vzniká potreba konverzie formátov

Informácie jedného druhu sú prevažne zapísateľné viacerými spôsobmi v rôznych formátoch súborov. Rozličné formáty sa odlišujú vo svojich vlastnostiach, preto často vzniká potreba konvertovať informáciu z jedného formátu na iný. Uvediem vlastnosti súborov, ktoré vnímam ako kľúčové pri vzniku nutnosti konverzie medzi formátmi:

1. Veľkosť súboru - Zmena formátu súboru z dôvodu ušetrenia diskového priestoru, často sprevádzaná stratou informácie (kvality súboru).

2. Rozšírenosť formátu - Konverzia formátu z dôvodu umožnenia narábania

s obsahom súboru (prezeranie, editácia, ...) v uživateľom preferovanej aplikácii.

3. Interoperabilita medzi počítačovými systémami - Konverzia dát vynútená používaním štandardov na výmenu dát a využívanie funkcii externých systémov (napríklad Štandardy pre informačné systémy verejnej správy)

4. Rozsah špecifikácie formátu - Niekedy je potreba konverzie do formátu, ktorý ma potenciál niesť prídavnú informáciu oproti konvertovanému formátu.

1.2 Existujúce transformačné riešenia

Momentálne dostupné riešenia na transformáciu formátov by som rozdelil na dva hlavné typy:

1. Proprietárne riešenia s grafickým uživateľským rozhraním poskytujúce širokú sadu transformácii, väčšinou rozdelené na viacero aplikácii, každá zabezpečuje transformácie medzi formátmi rovnakého typu údajov (napríklad riešenie ABC Amber)

2. otvorené jednoúčelové konverzné programy, väčšinou využívajúce konzolové uživateľské rozhranie

Bolo by vhodné mať možnosť využívať tieto otvorené konverzné programy jednotne pomocou grafického uživateľského rozhrania, pričom využitie by nebolo striktne definované pre uzavretú množinu formátov, ale všeobecné.

1.3 Cieľ bakalárskej práce

Cieľom tejto práce je vytvoriť prostredie, v ktorom by sa jednotným spôsobom, jednoducho a pohodlne dali využiť rôzne existujúce konverzné programy.

Kapitola 2

Rozbor problému

2.1 Globálny graf transformácii

Problém existencie rôznych formátov a transformácii medzi nimi možno abstrahovať do orientovaného grafu.

Vrcholy grafu reprezentujú jednotlivé existujúce formáty a hrany predstavujú transformácie medzi formátmi. Orientovanosť hrán odlišuje východzí formát od výstupného formátu. Každá hrana je nositeľom návodu, ako sa má daná transformácia realizovať.

V ideálnych podmienkach je tento graf úplný, čiže dokážeme realizovať transformáciu medzi ľubovoľnou dvojicou formátov. V reálnych podmienkach, však väčšinou táto podmienka neplatí z rozličných dôvodov ako napríklad:

- transformácia nie je implementovaná alebo nám známa
- formáty sú nositeľom dát rozličných charakterov, konverzia medzi nimi by nedávala zmysel

Graf všetkých známych formátov a transformácii, nad ktorými má vyvíjané prostredie pracovať nazveme globálny graf transformácii.

2.2 Obmedzenia na graf transformácii

Zo špecifickosti domény popísanej grafom vyplývaju v praxi nasledovné charakteristiky grafu:

- zakázané slučky, transformácia musí prebiehať medzi dvojicou rôznych formátov
- povolené násobné hrany, možnosť existencie viacerých konverzných programov medzi dvoma formátmi

2.3 Transformovateľnosť

Tým, že globálny graf transformácii poskytuje nadhľad nad všetkými dostupnými transformáciami, vzniká možnosť neobmedziť sa len na využitie samostatných konverzných programov, ale transformáciu medzi dvojicou formátov možno realizovať aj sériou čiastkových transformácii, pri ktorej vznikajú medziprodukty potrebné ku nasledujúcim transformáciam. Po skončení série transformácii tieto môžu byť odstránené. Je vhodné, aby sa pre používateľa javila konverzia pomocou série transformácii rovnako monolitická ako transformácia s použitím jediného konverzného programu.

Transformovateľnosťou nazveme reláciu medzi dvojicou formátov, ktorá predstavuje možnosť transformovať jeden formát na druhý. V globálnom grafe transformácii je táto relácia ekvivalentná s existenciou cesty medzi dvojicou vrcholov. Základnou vlastnosťou tejto relácie je tranzitívnosť. Ak je formát A transformovateľný na formát B a formát B transformovateľný na formát C, potom je aj formát A transformovateľný na formát C.

Kapitola 3

Návrh back-endovej vrstvy prostredia

V tejto časti ukážem ako sa časti globálneho grafu transformácii namapujú na typy objektov v implementácii navrhovaného prostredia.

3.1 Typy objektov

Globálny graf transformácii sa v praxi väčšinou rozdelí do niekoľkých komponentov súvislosti. Možnosť transformácii je vždy obmedzená len na formáty nachádzajuce sa v spoločnom komponente.

Z tohto hľadiska zadefinujeme **okruh formátov** ako základnú podštruktúru globálneho grafu transformácii. Predstavuje ucelenú množinu formátov, ktoré spája rovnaká povaha uložených dát a medzi ktorými existujú reálne alebo predpokladáme neskôr doplnené potencionálne konverzné programy. Okruh formátov teda zgrupuje v implementácii jeden alebo viacero komponentov grafu.

Typ **Formát** reprezentuje skutočný typ formátu súboru. Je charakterizovaný svojou príponou a príslušnosťou k okruhu formátov.

Typ **Transformácia** predstavuje konverziu z jedného formátu do druhého. Transformácia je viazaná na konkrétny okruh formátov a dvojicu formátov z tohto okruhu medzi ktorými prebieha. Hlavným atribútom transformácie je *transformačný predpis*. Je to konzolový príkaz, ktorým sa realizuje konverzia.

Typ **Transformačná reťaz** reprezentuje cestu v globálnom grafe transformácii. Jej atribútom je dĺžka cesty a implementovaná je ako spájaný zoznam transformácii. Z grafu prirodzene vyplývajú nasledovné vlastnosti:

- Všetky transformácie v zozname prislúchajú k tomu istému okruhu formátov.

- Pre transformačné reťaze dĺžky aspoň 2 platí nasledujúci vzťah:
  $\forall i \in 1, 2, .., len(TR) - 1 : To_i = From_{i+1}$, kde $len(TR)$ označuje dĺžku transformačnej reťaze, To_i výstupný formát i-tej transformácie v reťazi a $From_i$ vstupný formát i-tej transformácie. Tento vzťah reprezentuje to, že transformačná reťaz predstavuje postupnú sériu transformácii formátov, ktorá sa vonkajšiemu pozorovateľovi javí ako konverzia z formátu $From_1$ na formát $To_{len(TR)}$.

- Označme $M = \{From_i \mid 1 \leq i \leq len(TR)\} \cup \{To_{len(TR)}\}$ množinu formátov, na ktorých prebieha séria konverzii v transformáčnej reťazi. Platí vzťah: $\forall i, j \ (1 \leq i, j \leq |M|) \wedge (i \neq j) : M_i \neq M_j$
  Je to vyjadrenie faktu, že transformačné reťaze nereprezentujú také cesty v globálnom grafe, ktoré obsahujú kružnicu, a teda v nich dochádza ku zbytočnej postupnosti transformácii, kde dospejeme ku formátu, v ktorom sme sa už nachádzali.

- Ku každej jednoduchej transformácii definovanej v globálnom grafe transformácii priradíme transformačnú reťaz dĺžky 1, ktorá obsahuje túto jednoduchú transformáciu.

3.2 Výpočet transformačných reťazí

Globálny graf transformácii, nad ktorým má vyvíjané prostredie pracovať, môže byť na vstupe popísaný do rôznych stupňoch detailnosti:

1. Graf je popísaný kompletným statickým zoznamom formátov a korektných transformačných reťazí. Môže to byť realizované napríklad vo forme tabuľky s uloženými trojicami typu (Vstupný formát, Výstupný formát, Transformačná reťaz). Nevýhodou tohto prístupu je, že tabuľku treba v prípade zmien v grafe vypočítavať nanovo.

2. Graf je na vstupe popísaný minimalisticky, len svojimi okruhmi formátov, do nich patriacich formátov a jednoduchými hranami. Program vypočítava uzáver grafu obsahujúci existujúce transformačné reťaze až po jeho načítaní. Toto sa dá realizovať dvoma spôsobmi:

   (a) Vypočítať čiastočný uzáver s pevným začiatočným vrcholom, t.j. vstupným formátom až po jeho zvolení užívateľom. Ten ostáva platný až do momentu zmeny vstupného formátu alebo opätovného načítania grafu.

   (b) Vypočítať kompletný uzáver grafu priamo po jeho načítaní, ktorý ostáva rezidentne v pamäti a používa sa pri transformáciach až do opätovného načítania grafu. Uzáver celého grafu možno vypočítať postupne ako uzáver nad každým okruhom formátov, keďže transformácie sú definované v rámci okruhu.

Pri implementácii som použil prístup 2b, pri ktorom je vstupný globálny graf transformácii popísaný len svojimi vrcholmi a hranami a výpočet ciest nastáva po jeho načítaní programom.

Tento prístup možno realizovať nasledovným algoritmom popísaným v pseudokóde:

```
1  /*
2  okruhy[] - pole okruhov nacitanych z grafu
3  okruh.formaty[] - pole formatov v danom okruhu
4  retaze[f1][f2] - pole transformacnych retazi z formatu f1 do f2,
5        operator += zvacsi pole o pridavanu retaz
6  disjunktne(tr1, tr2) - vracia true, ak retaze tr1 a tr2
7        neobsahuju spolocny format, inak false
8  spoj(tr1, tr2) - vracia retaz vznikunutu spojenim tr1 a tr2
9  */
10 for (okruh in okruhy[]) {
11    formaty[] = okruh.formaty[];
12    for (middle in formaty[]) {
13       for (from in formaty[] ) {
14          for (to in formaty[]) {
15             if (from != middle && middle != to) {
16                continue;
17             }
18             fromRetaze = retaze[from][middle];
19             toRetaze = retaze[middle][to];
20             for (retazZaciatok in fromRetaze) {
21                for (retazKoniec in toRetaze) {
22                   if (disjunktne(retazZaciatok, retazKoniec)) {
23                      retazSpojena = spoj(retazZaciatok, retazKoniec);
24                      retaze[from][to] += retazSpojena;
25                   }
26                }
27             }
28          }
29       }
30    }
31 }
```

Tento algoritmus je modifikáciou Floyd–Warshallovho algoritmu s tým rozdielom, že nehľadá iba najkratšie cesty v grafe, ale všetky existujúce cesty, na ktorých sa neopakujú vrcholy. V závislosti od počtu hrán v grafe môže byť čas výpočtu až exponenciálny. To však nie je prekážkou, ak zoberieme do úvahy fakt, že počet formátov je rádovo malý. Ukážem, že algoritmus je korektný.
Najprv ukážem, že každá existujúca akceptovaná cesta sa dostane do výsledku. Urobím to indukciou vzhľadom na dĺžku cesty.

1. Cesty dĺžky 1 sú vytvorené priamo pri načítavaní grafu. Z nich sa potom vypočítavajú ďalšie cesty.

2. Zadefinujme si pojem *vnútorný vrchol cesty*. Je to každý vrchol cesty, ktorý nie je na prvý ani posledný vrchol cesty. Cesta dĺžky n má teda *n-1* vnútorných vrcholov. Jednotlive vrcholy grafu možno ohodnotiť podľa poradia, v akom sa načítavajú do konfigurácie, teda v akom poradí sú v pseudokóde v poli *formaty[]*. Nová transformačná reťaz(cesta) vzniká v algoritme vždy spojením dvoch ciest, ktoré majú spoločný posledný a prvý vrchol. Nazvime tento vrchol *spojový vrchol*. V algoritme sa postupne prechádza všetkými formátmi podľa ohodnotenia, pričom vždy sa aktuálny prechádzaný vrchol používa ako spojový vrchol na vytvorenie nových ciest.

   Majme korektnú cestu dĺžky n s vrcholmi $V_0, V_1, \ldots, V_n$, o ktorej chceme dokázať, že algoritmus ju nájde a zaradí do výsledku. Podľa indukčného predpokladu všetky cesty kratšie ako n dokáže algoritmus nájsť. Zoberme si vnútorný vrchol tejto cesty s najväčším ohodnotením V_{max}. Ak má byť prešetrovaná cesta nájdená, musela by vzniknúť spojením ciest $V_0, \ldots, V_{max}$ a $V_{max}, \ldots, V_n$ a s vrcholom V_{max} ako spojovým vrcholom. Vieme z indukčného predpokladu, že cesty $V_0, \ldots, V_{max}$ a $V_{max}, \ldots, V_n$ budú nájdené algoritmom po jeho zbehnutí, lebo majú dĺžku menšiu

ako n. No tieto cesty musia byť nájdené algoritmom už v čase, keď je aktuálný spojový vrchol (premenná middle) V_{max}, pretože všetky vnútorné vrcholy týchto ciest (podľa ktorých tieto cesty vznikali) majú ohodnotenie menšie ako V_{max}. V čase keď je aktuálny spájaný vrchol V_{max}, teda už existujú cesty $V_0, \ldots, V_{max}$ a $V_{max}, \ldots, V_n$ a teda prešetrovaná cesta naozaj vznikne.

Indukciou vzhľadom na dĺžku cesty tiež ukážem, že každá korektná cesta sa do výsledku nemôže dostať viac ako raz.

1. Cesty dĺžky 1 načítané z grafu sú do výsledku zaradené len raz.

2. Majme cestu dĺžky n. Predpokladajme, že všetky kratšie cesty ako n sú vo výsledku najviac raz. Označme V_s vrchol, ktorý môže byť použitý ako spojový vrchol. Ukážeme, že jediným kandidátom je V_{max}, teda vnútorný vrchol prešetrovanej cesty s najväčším ohodnotením. V prípade, že by to bol iný vrchol, tak jedna z ciest, z ktorých vznikla prešetrovaná cesta ($V_0, \ldots, V_s$ a $V_s, \ldots, V_n$), by musela obsahovať V_{max} ako svoj vnútorný vrchol. To je však spor s možnou existenciou tejto cesty, lebo v čase keď je aktuálny spojový vrchol vrchol s menším ohodnotením ako V_{max}, tak cesty s vnútorným vrcholom V_{max} ešte nemohli existovať. Keďže jediný spojový vrchol V_{max} prichádza do úvahy pri prešetrovanej ceste a obe cesty, z ktorých táto cesta vznikla sú vo výsledku z indukčného predpokladu najviac raz, tak aj prešetrovaná cesta tam bude najviac raz.

Týmto spôsobom vzniká úplný uzáver načítaného grafu transformácii. Vďaka nemu má užívateľ možnosť spúšťať konverziu medzi všetkými dvojicami formátov patriacich do relácie transformovateľnosti, pričom má na výber zo všetkých existujúcich ciest neobsahujúcich kružnicu.

3.3 Transformačný predpis

Na spúšťanie externých konverzných aplikácii budeme používať rozhranie príkazového riadku. Výhodou je jednoduchosť volania príkazov bez nutnosti existencie špecializovaného API. Návod na to, aký príkaz zavolať na spustenie konverzie by mal byť ukrytý v *transformačnom predpise*, ktorý je atribútom transformácie. Model transformačného predpisu som vytváral na základe analýzy charakteru konzolových progamov. Spúšťaný príkaz je v konečnom dôsledku textová hodnota. No hodnotu príkazu na spúšťanie programov možno konfigurovať na základe rozličných vstupných volieb a parametrov. Pri konverzných programoch sú to minimálne dva nevyhnutné parametre, a to vstupný a výstupný súbor (adresár).

Syntax UNIXových príkazov

Všeobecný formát unixového príkazu vyzerá nasledovne:

command [flags] [arguments]

command - špecifikuje názov príkazu, statická časť
flags - príznaky, modifikujú funkčnosť základného príkazu, väčšinou začínajú znakom '-', ako napríklad '-f'
arguments - najčastejšie sú to názvy súborov alebo adresárov, na ktorých sa príkaz vykonáva

Uzly transformačného predpisu

Na základe syntaxe príkazov som sa rozhodol navrhnúť transformačný predpis ako zoznam uzlov rôznych typov. Výstupom každého uzla je text, a definitívny príkaz na konverziu dostaneme zreťazením výstupov jednotlivých uzlov predpisu (podľa poradia v predpise). Predpis pozostávajúci z uzlov

nemusí byť striktne jednoúrovňový, ale predpis niektorých typov uzlov môže pozostávať zo zoznamu poduzlov. Výstupom takéhoto typu uzla je potom zreťazenie výstupov jeho poduzlov.
Navrhované typy poduzlov:

1. Text - statický text, výstup tohto uzla je samotný text
2. Vstupný súbor - reprezentuje súbor, ktorý užívateľ zvolí na konverziu, výstupom je text rovný ceste ku vstupnému súboru v adresárovom strome
3. Výstupný súbor - reprezentuje požadovaný výstup konverzie, môže nadobúdať dve podoby, podľa toho, čo požaduje konverzný program, prvá podoba je celá cesta ku výstupnému súboru, druhou podobou je cesta po rodičovský adresár výstupného súboru, výstupnou hodnotou uzla do predpisu je prislúchajúca textová hodnota cesty
4. Input Box - vstupné pole, umožňuje ľubovoľný textový vstup do predpisu od užívateľa, výstupom do predpisu je vložený text, má atribút 'predvolená hodnota', t.j. textová hodnota, ktorá sa použije do výstupu v prípade, že užívateľ explicitne nezadá svoj vstup
5. Check Box - zaškrtávacie pole, obsahuje ako hodnotu zoznam poduzlov (poduzly rovnakého typu ako práve definujeme) a booleovskú premennú(zaškrtnutá/odškrtnutá možnosť), výstupom tohto uzla je výstup zoznamu jeho poduzlov v prípade, že je booleovská premenná pravdivá, v opačnom prípade prázdny textový reťazec, tento uzol predstavuje analógiu ku dvojhodnotovému prepínaču áno/nie zo syntaxe konzolových príkazov
6. Select Box - výberové pole, zovšeobecnenie uzla Check Box, ako hodnotu obsahuje zoznam možných volieb, každá voľba obsahuje ako svoju

hodnotu zoznam poduzlov(každý je jedného z 5 práve definovaných typov), výstupom voľby je zreťazenie výstupov jej poduzlov, Select Box ďalej obsahuje referenciu na jednu zo svojich volieb, ktorá je práve vybratá, ako výstup Select Boxu sa použije výstup práve vybratej voľby, analógia ku viacvoľbovým prepínačom

S navrhovanou sadou uzlov by sa mala dat pokryť variabilita všetkých konzolových príkazov. Uzly môžeme podľa ich výstupov rozdeliť na *terminálne* (Text, Vstupný súbor, Výstupný súbor a Input Box) a *neterminálne* (Check Box, Select Box). *Terminálny uzol* poskytuje priamo výstup bez ovplyvnenia výstupu inými uzlami (je to list v lese stromov vytvorených z uzlov predpisu), zatiaľ čo výstup *neterminálneho uzla* vzniká z výstupov jeho poduzlov (nikdy to nie je listový uzol). Ďalšim kritériom rozdelenia uzlov je *modifikovateľnosť* výstupného predpisu užívateľom pomocou uzla. Uzol Text je v tomtom zmysle statický a užívateľ nemá možnosť ovplyvniť jeho výstup do predpisu. Uzlami Vstupný súbor a Výstupný súbor užívateľ ovplyvňuje predpis nepriamo, keďže tieto uzly reflektujú užívateľom zvolený vstupný súbor, resp. výstupný adresár. Výstup uzlov Input Box, Check Box a Select Box do hodnoty predpisu môže užívateľ ovplyvniť priamo. Pri uzle Input Box priamo zadaním vstupného textu, pri Check Boxe zaškrtnutím (odškrtnutím) voľby a pri Select Boxe výberom jednej z volieb. Tieto typy uzlov, ktoré môže užívateľ nastavovať pred spustením konverzie, sa dobre mapujú na štandardné formulárové komponenty z knižníc na vývoj GUI (Graphical User Interface).

3.4 Globálny graf transformácii v súbore

Popísal som všetky typy objektov, ktoré sa nachádzajú v globálnom grafe transformácii a aplikácia ich využíva ku realizácii konverzii. Aplikácia načítava tento graf zo súboru pri svojom štarte. Otázkou je voľba štruktúry

súboru popisujúceho graf. Za formát súboru som si zvolil XML z nasledujúcich dôvodov:

- je to štandardný formát
- existujú bežné parsery XML súborov, odpadá nutnosť písať pri implementácii vlastný parser
- je to veľmi variabilný formát, schopnosť definovať si ľubovoľné elementy usporiadané v ľubovoľnej stromovej štruktúre
- v prípade budúceho rozšírenia grafu o nové prvky ľahké zaistenie spätnej kompatibility
- prijateľná čitateľnosť konfiguračného súboru pre človeka, v prípade dobre naformátovanej štruktúry XML súboru

Súbor popisujúci globálny graf transformácii má nasledovnú štruktúru zapísanú v jazyku XML Schema:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="vstup"/>
<xs:element name="vystup"/>

<xs:element name="inputbox">
  <xs:complexType>
    <xs:attribute name="meno" type="xs:string" use="required"/>
    <xs:attribute name="default" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="checkbox">
  <xs:complexType mixed="true">
    <xs:all>
      <xs:element ref="vstup" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="vystup" minOccurs="0" maxOccurs="unbounded"/>
```

```
      <xs:element ref="inputbox" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="selectbox" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="checkbox" minOccurs="0" maxOccurs="unbounded"/>
    </xs:all>
    <xs:attribute name="meno" type="xs:string" use="required"/>
    <xs:attribute name="selected" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="yes"/>
          <xs:enumeration value="no"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>

<xs:element name="option">
  <xs:complexType mixed="true">
    <xs:all>
      <xs:element ref="vstup" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="vystup" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="inputbox" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="selectbox" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="checkbox" minOccurs="0" maxOccurs="unbounded"/>
    </xs:all>
    <xs:attribute name="meno" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="selectbox">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="option" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="default" type="xs:string"/>
    <xs:attribute name="meno" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="konverzia">
  <xs:complexType mixed="true">
    <xs:all>
      <xs:element ref="vstup" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="vystup" minOccurs="0" maxOccurs="unbounded"/>
```

```
      <xs:element ref="inputbox" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="selectbox" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="checkbox" minOccurs="0" maxOccurs="unbounded"/>
    </xs:all>
    <xs:attribute name="do" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="format">
  <xs:complexType>
    <xs:all>
      <xs:element ref="konverzia" minOccurs="0" maxOccurs="unbounded"/>
    </xs:all>
    <xs:attribute name="meno" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="formaty">
  <xs:complexType>
    <xs:all>
      <xs:element ref="format" minOccurs="0" maxOccurs="unbounded"/>
    </xs:all>
  </xs:complexType>
</xs:element>

<xs:element name="okruh">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="formaty"/>
    </xs:sequence>
    <xs:attribute name="meno" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="konverznyProgram">
  <xs:complexType>
    <xs:all>
      <xs:element ref="okruh" minOccurs="0" maxOccurs="unbounded"/>
    </xs:all>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Popísaná back-endová vrstva aplikácie mapuje do objektov štruktúru načítaného globálneho grafu transformácii, vytvára nad grafom nové typy objektov odvodené zo základných typov, poskytuje možnosť parametrizovania transformačných príkazov, vracia výstupy parametrizovaných príkazov, umožňuje modifikáciu načítaného grafu, výstup grafu do súboru so zvolenou XML štruktúrou, z akej bol graf načítaný a poskytuje rôzne funkcie, ktoré využíva prezentačná vrstva aplikácie pri interakcii s používateľom.

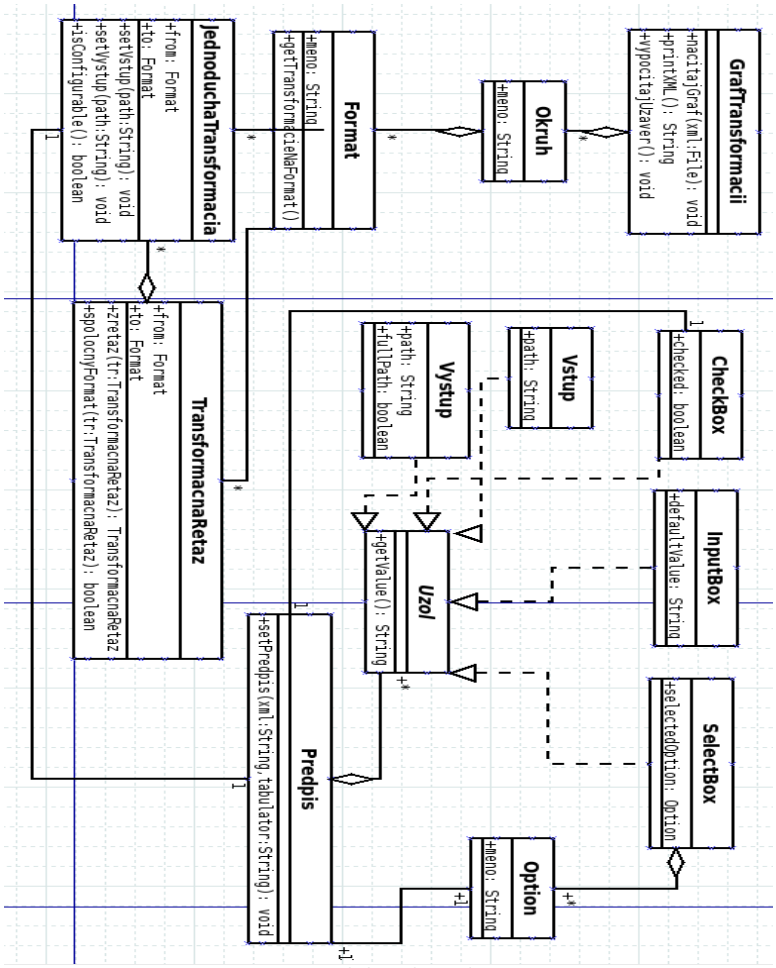


Diagram tried backendovej vrstvy

Kapitola 4

Návrh prezentačnej vrstvy prostredia

Prezentačná vrstva slúži na komunikáciu s používateľom, pričom je podporovaná funkciami backend vrstvy.

4.1 Použité technológie

Na implementáciu oboch vrstiev som použil jazyk Java. To zabezpečuje aplikácii možnosť použitia na rozličných platformách. Na tvorbu grafického užívateľského rozhrania som použil knižnicu Java Swing.

4.1.1 Tvorba GUI

Popíšem základné princípy GUI programovania, ktoré som použil. Objekty pomocou, ktorých sa vytvára GUI možno rozdeliť do 2 hlavných typov.

1. **Komponenty** - nízkoúrovňové objekty, ktoré majú vizuálnu reprezentáciu a ktoré umožňujú interakciu s používateľom, príklady takýchto

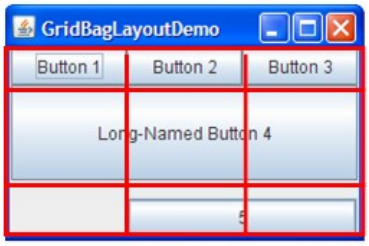
typov objektov sú tlačidlá, vstupné polia, textové polia, zaškrtávacie políčka, skrolovacie plátna atď.

2. **Kontajnery** - vysoko-úrovňové objekty, ktoré slúžia ako plátno na zobrazovanie iných kontajnerov alebo komponentov, patria sem koreňové objekty postavené najvyššie v hierarchii grafických objektov ako JFrame (štandardné okno), JDialog (dialógové okno) alebo odľahčenejšie kontajnerové objekty ako napríklad JPanel

Každé okno aplikácie s grafickým prostredím je vytvorené ako strom grafických objektov s koreňom typu vysoko-úrovňového kontajnera a listami sú výlučne komponenty. Každý kontajner obsahuje zoznam svojich synov a je k nemu priradený objekt implementujúci rozhranie *LayoutManager* (správca rozvrhnutia). Parametre tohto objektu nastavené pri vkladaní grafického objektu do kontajnera určujú pozíciu a veľkosť zobrazenia vkladaného objektu v kontajneri. Existuje veľa typov správcov rozvrhnutia, ja som použil *GridBagLayout*.

GridBagLayout

GridBagLayout je definovaný priamo v balíku javax.swing . Je to jeden z najflexibilnejších a najkomplexnejších správcov rozvrhnutia.



Pridávané komponenty ukladá do mriežky stĺpcov a riadkov, pričom je možnosť pridať aj komponent presahujúci cez viac riadkov alebo stĺpcov. Výška všetkých riadkov nemusí byť nevyhnutne rovnaká, a podobne je to so šírkou stĺpcov.

Spôsob, akým sa špecifikuje veľkost a pozícia vkladaného objektu do kontajneru s rozložením GridBagLayout určuje inštancia typu GridBagConstraints (ohraničenie) vkladaná do metódy *Container.add(component, constraint)*. Popíšem jeho niektorého stavové premenné.

- gridx, gridy- špecifikujú stĺpec a riadok, kam sa má vložiť komponent, ich číslovanie je od ľavého horného rohu a začína nulou
- gridwidth, gridheight - počet stĺpcov a riadkov, cez ktoré komponent presahuje
- fill - určuje roztiahnutie komponentu, ak jeho preferovaná veľkosť je menšia ako veľkosť pridelenej bunky
- insets - špecifikácia vonkajšieho odsadenia medzi komponentom a hranicami jeho bunky
- anchor - určuje orientáciu komponentu v bunke, ak je jeho veľkosť menšia ako veľkosť bunky

Vytvorená inštancia typu GridBagConstraints je znovupoužiteľná pri vkladaní rôznych komponentov do rozličných kontajnerov.

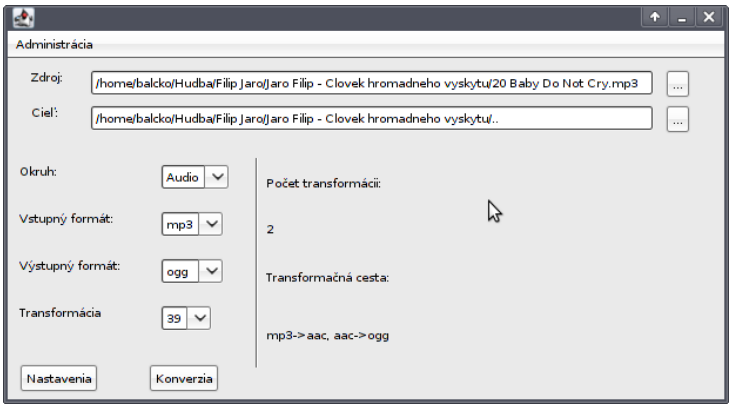
Tvorba grafického uživateľského rozhrania pozostáva rámcovo z nasledujúcich činností: definícia a inicializácia objektov, nastavenie ich vlastností, rozmiestnenie komponentov do kontajnerov, priradenie obsluhy udalostí s definovanými obslužnými metódami ku komponentom a vykreslenie objektov.

4.2 Rozhranie pre ovládanie konverzii

Toto používateľské rozhranie by malo umožňovat jednoducho,intuitívne a jednotne realizovať načítané konverzie z globálneho grafu.

4.2.1 Výber konverzie

Výber transformácie uživateľom poskytuje trieda MainFrame odvodená od triedy JFrame. Vstupnými parametrami zadávanými od používateľa sú vstupná cesta, výstupná cesta, okruh, vstupný formát, výstupný formát a vybratá transformácia medzi nimi. Výber vstupu a výstupu je realizovaný cez triedu JFileChooser, ktorá poskytuje dialógové okno na jednoduché hľadanie objektov v adresárovom strome. Po výbere vstupného súboru sa aplikácia podľa jeho prípony snaží zistiť požadovaný vstupný formát. Ponechal som možnosť explicitného zvolenia typu vstupného súboru používateľom, pretože rovnaký typ formátu môže mať rôzne prípony a nemusela by nastať zhoda medzi príponou súboru a skutočným požadovaným formátom z grafu transformácii (napríklad htm-html, jpg-jpeg, ...). Po nastavení vstupného formátu aplikácia poskytne na výber len tú množinu výstupných formátov, do ktorých existuje konverzia. Po zvolení transformačnej reťaze, ktorá ma konverziu prevádzať vidí v informačnom paneli umiestnenom vpravo dole užívateľ detaily zvolenej transformácie.



Obrázok 4.1: Okno výberu konverzie

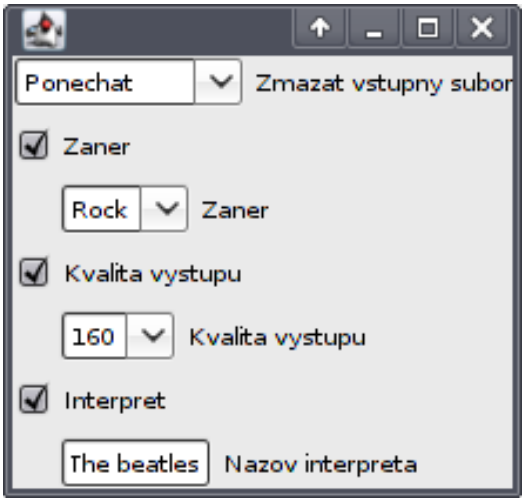
Trieda MainFrame je hlavnou triedou aplikácie. Vo svojej metóde main() vytvorí svoju inštanciu, ktorá vykresľuje okno výberu konverzie na obrázku 4.2.1. Jeho zavretím aplikácia končí. Jej dôležitou stavovou premennou je objekt typu Program, ktorý obsahuje načítaný graf transformácii a informácie o aktuálnych uživateľských vstupoch. Na základe týchto informácii dochádza ku riadeniu programu a vykresľovaniu okna. Životný cyklus tohto objektu trvá počas celého behu aplikácie. Pri zvolení takej transformačnej reťaze, ktorá obsahuje obsahuje aspoň jednu transformáciu s modifikovateľnými uzlami v predpise, je tlačidlo *Nastavenia* povolené, a pomocou neho sa otvára okno umožňujúce zmenu nastavení použitých transformácii.

4.2.2 Nastavenia konverzie

Nastavenie transformačného príkazu možno vykonávať zmenou vlastností modifikovateľných uzlov v jeho predpise. Tie meníme pomocou grafických komponentov, ktoré zodpovedajú týmto uzlom. Uzlu InputBox priradíme grafický komponent typu JTextField a zápisom hodnoty do tohto vstupného poľa vkladáme paramater do konverzného príkazu. Uzlu CheckBox priradíme komponent JCheckBox a jeho zaškrtnutím/odškrtnutím povolíme hodnote CheckBoxu dostať sa do príkazu. Uzlu SelectBox priradíme komponent JComboBox s možnosťami rovnými názvom volieb v SelectBoxe a medzi týmito možnosťami si môžme vyberať. O zobrazovanie okna obsahujúceho tieto grafické komponenty zodpovedajúce modifikovateľným predpisom sa stará inštancia triedy SettingsFrame, ktorá je odovodená od triedy JFrame.

Rozvrhnutie komponentov v nastaveniach konverzie

Cieľom v okne SettingsFrame je rozmiestniť komponenty prirodzene ako za sebou nasledujú v predpise, pričom komponenty vnorené do neterminálnych uzlov zachovávajú aj svoje vnorenie pomocou formátovania (Obrázok 4.3.2).



Obrázok 4.2: Okno nastavení konverzie

To som dosiahol stromovou hierarchiou objektov typu JPanel kopírujucou strom modifikovateľných uzlov predpisu. Do hlavného panelu postupne pridávam pod seba grafické komponenty zodpovedajúce uzlom predpisu. Ak sa vyskytne uzol typu CheckBox alebo SelectBox vytvorím k nemu nový objekt typu JPanel, vykreslím naň príslušné poduzly rovnakým spôsobom, a vykreslený panel umiestnim pod grafický komponent zodpovedajúci rodičovskému uzlu. Umiestnim ho však pomocou inštancie GridBagConstraints s nastaveným ľavým vonkajším odsadením a s veľkým horizontálnym presahom buniek, aby som nepokazil horizontálne rozvrhnutie uzlov vyššie v hierarchii. Takto je možné rekurzívne vykresliť celý strom modifikovateľných uzlov. Popíšem ešte ako sa prejavujú udalosti na grafických komponentoch na vykresľovanie. Pri zaškrtnutí CheckBoxu sa panel, na ktorom sú vykreslené jeho poduzly stane viditeľným, pri odškrtnutí sa zneviditeľní. Pri výbere možnosti zo SelectBoxu sa k nemu prislúchajúci panel poduzlov prekreslí poduzlami predpisu vybratej voľby. Pri prekreslovaní nedochádza ku kompletnému prekreslovaniu všetkých panelov v strome, ale každý panel si pamätá referenciu na

svojho rodiča a postupne sa tak prekreslí cesta od zmeneného panelu až po koreňový panel.

4.2.3 Spúšťanie konverzie

Po zadaní všetkých vstupných údajov používateľom a vybratí si transformačnej reťaze je možne spustiť konverziu. Finálne podoby konverzných príkazov dostaneme dosadením zadaných vstupných a výstupných ciest do uzlov typu Vstup a Výstup a použitím metódy getValue() objektu Predpis. Externé konverzné programy sa spúšťajú v samostatnom procese, ktorý je spúšťaný v oddelenom vlákne s nižšou prioritou, aby nedochádzalo ku blokovaniu grafického uživateľského prostredia. Na spúšťanie príkazu sa používa metóda objektu Runtime *exec(String[] cmdarray)*. Jej argumenty dostávame rozdelením konverzného príkazu do poľa reťazcov s medzerou ako oddeľovačom. Treba však dávat pozor na to aby vstupná a výstupná cesta nebola rozdelená ani v prípade, že obsahuje medzery. Metóda môže hádzať výnimku typu IOException, ktorá sa zachytáva a uživateľovi sa hádže chybová hláška s popisom vzniknutej chyby. Spúšťanie konverzie prebieha v samostatnom okne implementovanom triedou KonverziaFrame rozširujúcou JFrame. Toto okno obsahuje trojicu textových polí informujúcich uživateľa o stave konverzie. V prvom okne sa postupne vypisujú príkazy, ktoré spúšťajú transformácie. Úspešne ukončené transformácie sú označené. V druhom okne je výpis štandardného výstupu z procesu, v ktorom beží konverzný program. V treťom okne je štandardný chybový výstup z procesu. V prípade série transformácii vznikajú medziformáty potrebné pre nasledujúcu transformáciu. Po skončení transformačného procesu sú tieto zmazané.

4.3 Administračný mód

Dá sa predpokladať, že sa vyvinie potreba zmeny globálneho grafu transformácii z rozličných dôvodov(definícia nových formátov, transformácii, editácia existujúcich transformácii, ...). Preto je vhodné umožniť intuitívnu grafickú formu editácie globálneho grafu bez nutnosti ručne editovať XML súbor popisujúci tento graf. Toto prostredie by uľahčovalo aj prvotnú definícu grafu. Zmena grafu ovplyvňuje funkčnosť celej aplikácie, preto je vhodné aby existovala možnosť oddeleného administračného módu vyžadujúceho autentifikáciu používateľa pri vstupe.

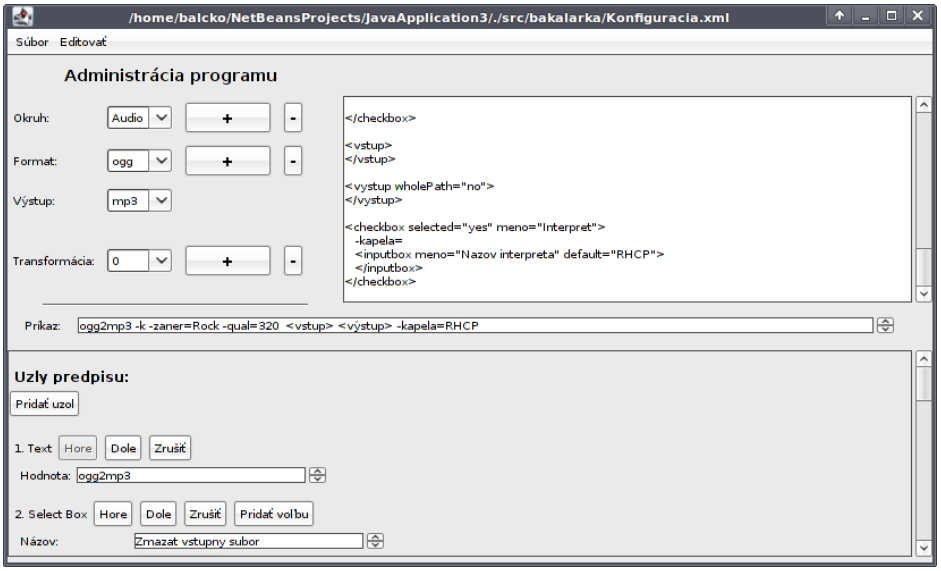
4.3.1 Bezpečnosť administračného módu

Administračné rozhranie som sa rozhodol oddeliť autentifikáciou heslom od používateľskej časti aplikácie. Pri prvotnom spustení aplikácie je administračné rozhranie prístupné bez hesla, s možnosťou definovať heslo, poprípade ho neskôr zmeniť. Zvolené heslo sa ukladá do konfiguračného súboru aplikácie ako zhašovaná hodnota. Na hašovanie sa používa trieda *BCrypt.java* implementujúca šifrovací algoritmus Blowfish uverejnený Bruceom Schneierom v roku 1993. Na hašovanie sa používa jej metóda *hashpw(String password, String salt)*, pričom bezpečnostný reťazec salt sa generuje metódou *genSalt()*. Pri autentifikácii sa zadané heslo kontroluje pomocou metódy *checkpw(String plaintext, String hashed)*. Je dôležité, aby práva na zápis do konfiguračného súboru mali len oprávnené osoby, aby nebolo možné obísť autentifikáciu prepísaním uloženej zhašovanej hodnoty.

4.3.2 Funkcie administračného módu

Administračný mód umožňuje pridávanie a odoberanie všetkých typov objektov definovaných v grafe transformácii, t.j. okruhov formátov, formátov

a transformácii. Pri pridávaní a odoberaní okruhov a formátov je používateľ vyzvaný pomocou dialógového okna zadať názov pridávaného objektu. V prípade, že objekt s daným menom už existuje, uživateľ obdrží chybové hlásenie. Ďalej administračný mód umožňuje upravovať existujúce transformácie.



Obrázok 4.3: Administračné okno

Administračné okno implementuje trieda AdministraciaFrame odvodená od JFrame. Ľavá horná časť okna uživateľovi umožňuje pridávať, odoberať objekty grafu a pohybovať sa po nich. Úpravu transformačného predpisu možno realizovať dvoma spôsobmi. Buď pomocou grafických komponentov vykresľujúcich les uzlov predpisu a umožňujúcich meniť ho, alebo editáciou samotného uzla z XML súboru popisujúceho graf, ktorý popisuje transformačný predpis, čo môže byť niekedy efektívnejšie. Tieto prístupy možno kombinovať, pretože každá platná úprava xml sa prejaví prekreslením lesa grafických komponentov zodpovedajúcemu zmenenému transformačnému

predpisu a každá udalosť v grafických komponentoch meniaca predpis sa prejaví prepísaním xml popisu. V prípade chybnej zmeny xml nenastáva zmena transformačného predpisu a popis chyby sa zobrazí v chybovom riadku na spodku okna. Pre lepšiu orientáciu pri editácii transformačného predpisu sa predvolený výstup upravovaného predpisu zobrazuje v stavovom riadku v strede okna. Atribút cesta v uzloch typu Vstup a Výstup je naplnený reťazcom <*vstup*> , respektíve <*výstup*>. Najdôležitejšou stavovou premennou okna AdministraciaFrame je objekt typu Editor, ktorý obsahuje načítaný graf a udržuje informácie o práve zvolených objektoch grafu v administračnom režime. Pri vytváraní rozvrhnutia grafických komponentov mapujúcich les uzlov predpisu som použil podobný rekurzívny princíp s vnorenými panelmi ako u okna nastavení konverzie. Rozdiel pri prekresľovaní tohto lesa nastáva pri editácii predpisu pomocou úpravy xml. Vtedy sa nedá jednoznačne povedať, ku zmene ktorého uzla dochádza, pretože pri jednej udalosti ich môže byť pozmenených niekoľko (mazanie vyznačeného úseku, vkladanie textu z clipboardu, ...), preto dochádza pri každej zmene ku prekresleniu celého lesa. V prípade, že v administračnom móde nastali zmeny grafu transformácii, po jeho opustení si objekt triedy Program načíta aktuálnu verziu grafu a vytvorí nad ním uzáver podobne ako pri štarte aplikácie.

4.3.3 Problém interpretácie medzier

Keďže aplikácia má umožniť editáciu transformačného predpisu úpravou xml, je vhodné, aby tento xml mal prirodzené formátovanie dobre čitateľné pre človeka. Pri takomto formátovaní platí, že každá ďalšia vnorená úroveň elementov je od rodičovského elementu odsadená zľava o jednu úroveň viac ako je odsadený jej rodič. Úroveň odsadenia predstavujú neviditeľné znaky, väčšinou je to tabulátor. Vzniká však problém s interpretáciou hodnoty tex-

tových uzlov, pretože musíme vedieť rozlíšiť znaky použité na odsadenie od znakov, ktoré sú hodnotou textového uzlu transformačného predpisu. Pre správnu interpretáciu textových uzlov a reálnych medzier v nich treba text získaný parserom ešte dodatočne upraviť nasledovne:

1. Ak má nájdený textový uzol rodičovský uzol, tak sa do výsledku dostanú všetky riadky okrem prvého a posledného upravené tak, že sa odreže z ich začiatkov prislúchajúce odsadenie k danému textovému uzlu. Nakoniec sa ešte odreže posledný znak, ktorým je znak ukončenia predposledného riadku.

2. Ak nemá nájdený textový uzol rodičovský uzol, vezme sa ako výsledok vyparsovaný text

   (a) orezaný o prvý riadok, ak má tento textový uzol súrodenecký uzol nad sebou

   (b) orezaný o posledný riadok a znak ukončenia predposledného riadku, ak má tento textový uzol súrodenecký uzol pod sebou

Až takto upravená hodnota sa môže použiť ako obsah textového uzla predpisu.

4.3.4 História zmien

Pri úprave grafu transformácii môže dochádzať ku rôznym preklepom alebo nechceným akciám, preto by bolo vhodné mať možnosť vrátiť späť vykonané zmeny. Rozhodol som sa preto ukladať históriu zmien v administračnom móde a poskytovať operácie *Späť* a *Dopredu* známe zo štandardných editorov. História zmien je implementovaná pomocou triedy AdministraciaHistoria. Tá si udržiava zoznam historických záznamov typu HistoriaZaznam a poradové číslo záznamu, ktorý je totožný s aktuálnym stavom editora.

Trieda HistoriaZaznam je abstraktná trieda obsahujúca stavové premenné popisujúce stav aplikácie v istom časovom bode a dvojicu neimplementovaných metód undo() a redo(). Implementáciu tejto abstraktnej triedy realizujú triedy reprezentujúce jednotlivé možné typy zmien transformačného predpisu. Trieda AdministraciaHistoria poskytuje tri základné metódy:

- pridajZaznam(HistoriaZaznam zaznam) - odstráni zo zoznamu záznamy s poradovým číslom vyšším ako je aktuálne poradové číslo, pridá na koniec záznamu argument zaznam a nastaví aktuálne poradové číslo na koniec zoznamu

- undo() - vráti späť poslednú zmenu v editore a aplikáciu nastaví do stavu pred ňou, realizuje sa to zavolaním metódy undo na aktuálnom zázname a zmenšením poradového čísla o jedno

- redo() - vykoná opäť poslednú vrátenú zmenu, realizované volaním metódy redo na aktuálny záznam v histórii a zväčšením poradového čísla o jedno

Abstraktnú triedu HistoriaZaznam realizuje spolu sedem rôznych tried reprezentujúcich možné vykonané zmeny počas editácie. Sú to nasledovné zmeny: Pridanie alebo vymazanie okruhu, formátu alebo transformácie a zmena transformačného predpisu.

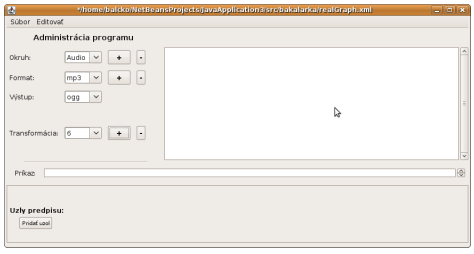
Kapitola 5

Príklady použitia

V tejto kapitole ukážem reálne príklady použitia aplikácie.

5.1 Mapovanie konverzného programu na transformačný predpis

Ukážem návod pre administrátora, ako namapovať konverzný program podľa jeho manuálových stránok do predpisu. Použijem program *mp32ogg* transformujúci hudobné súbory formátu mp3 na formát ogg. V prípade, že ešte nemám definovaný okruh na transformáciu audio súborov a v ňom zadefinované formáty mp3 a ogg, tak ich vytvorím. V ďalšom kroku pridám novú transformáciu medzi tymito formátmi zatiaľ prázdnu.



Manuálové stránky príkladu vyzerajú takto:

```
MP32OGG(1)                                                    MP32OGG(1)

NAME
       mp32ogg - Convert MP3 to Ogg Vorbis

SYNOPSIS
       mp32ogg [options] files directories...

DESCRIPTION
       This manual page documents briefly the mp32ogg command. This manual
       page was written for the Debian distribution because the original pro
       gram does not have a manual page.

       mp32ogg is a program that converts MP3 files and directories to Ogg
       Vorbis.

OPTIONS
       These programs follow the usual GNU command line syntax, with long
       options starting with two dashes ( - ). A summary of options is
       included below.

       -h, --help
              Show summary of options.

       --quality=[-1..10]
              Set Ogg/Vorbis quality level. By default, set to the bitrate of
              original .mp3.

       --delete
              Delete files after converting

       --rename=format
              Instead of simply replacing the .mp3 with .ogg for the output
              file, produce output filenames in this format, replacing %a, %t
              and %l with artist, title, and album name for the track

       --lowercase
              Force lowercase filenames when using --rename

       --verbose
              Verbose output

       --preserve-timestamp
```

```
        Preserve file timestamp

    --from-charset=CHARSET
        Reencode IDv3 tags from CHARSET to UTF8.

SEE ALSO
    oggenc(1),

AUTHOR
    This manual page was written by Julien Danjou <acid@debian.org>, for
    the Debian GNU/Linux system (but may be used by others).

                                August 28, 2002                MP32OGG(1)
```

Ukážem ako možno vytvoriť z toho transformačný predpis. Samozrejme názvy jednotlivých uzlov, a rozhodnutie, ktoré prepínače použiť v predvolenom nastavení programu závisia čisto od preferencií administrátora. Vypíšem za sebou aké uzly budú nasledovať a ich hodnoty. Pri textových uzloch budem dávať hodnoty do úvodzoviek, aby boli viditeľné medzery.

1. Text - ‘mp32ogg ‘
2. CheckBox - meno ‘Kvalita výstupu‘
   1 . Text - ‘–quality=‘
   2 . SelectBox - meno ‘Kvalita výstupu‘, vložíme sem 12 volieb, každá bude obsahovať textový uzol s číslom od ‘-1‘ po ‘10‘, defaultnú voľbu nastavíme na 10
   3 . Text - ‘ ‘
3. CheckBox - meno ‘Zmazať súbor po konverzii‘, defaultne odškrtnuté
   1 . Text - ‘–delete ‘
4. CheckBox - meno ‘Formátovaný názvo výstupu‘, defaultne odškrtnuté
   1 . Text - ‘–rename=‘

2 . InputBox - meno ‘použite %a ako meno autora, %t titul, a %l názov albumu‘, defaultná hodnota ‘%a - %t‘

3 . Text - ‘ ‘

4 . CheckBox - meno ‘Malé písmo‘, defaultne odškrtnuté

1 . Text - ‘–lowercase ‘

5. CheckBox - meno ‘Rozsiahly výstup‘, defaultne odškrtnuté

1 . Text - ‘–verbose ‘

6. CheckBox - meno ‘Zachovať časovú pečiatku‘, defaultne odškrtnuté

1 . Text - ‘–preserve-timestamp ‘

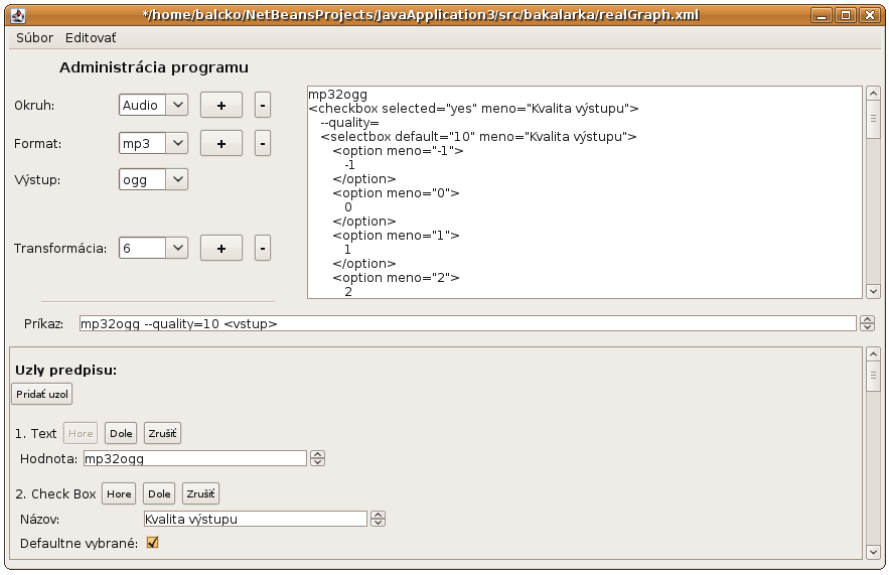
7. CheckBox - meno ‘Zmena kódovania na UTF8‘, defaultne odškrtnuté

1 . Text - ‘–from-charset=‘

2 . InputBox - meno ‘Pôvodné kódovanie‘, defaultná hodnota “

3 . Text - ‘ ‘

8. Vstup

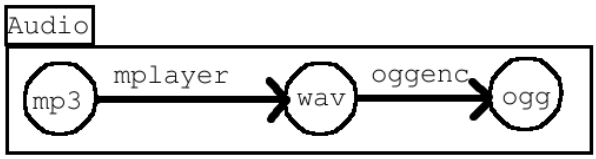


5.2 Príklad konverzie

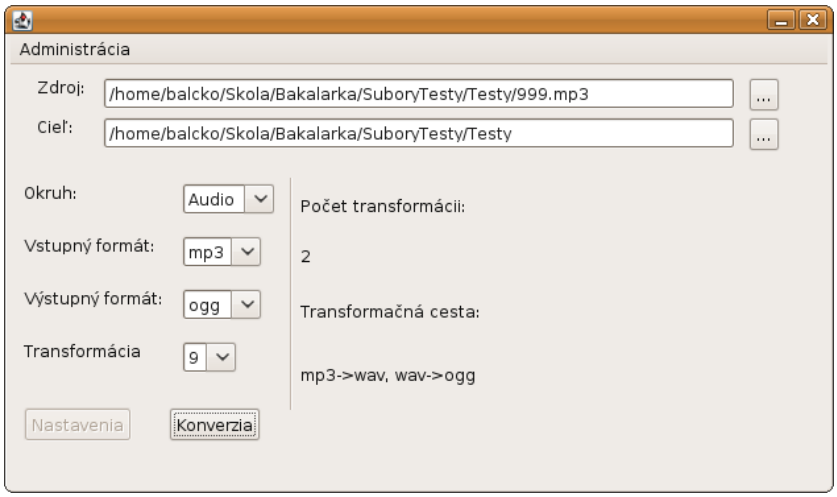
Definujem si vzorové transformácie a ukážem výsledok reálnej konverzie nad audio súbormi. Vytvorím si okruh Audio a doňho trojicu formátov mp3, ogg a wav. Definujem si medzi nimi nasledovné transformácie:

1. mp3 -> wav pomocou programu *mplayer*
2. wav -> ogg pomocou programu *oggenc*

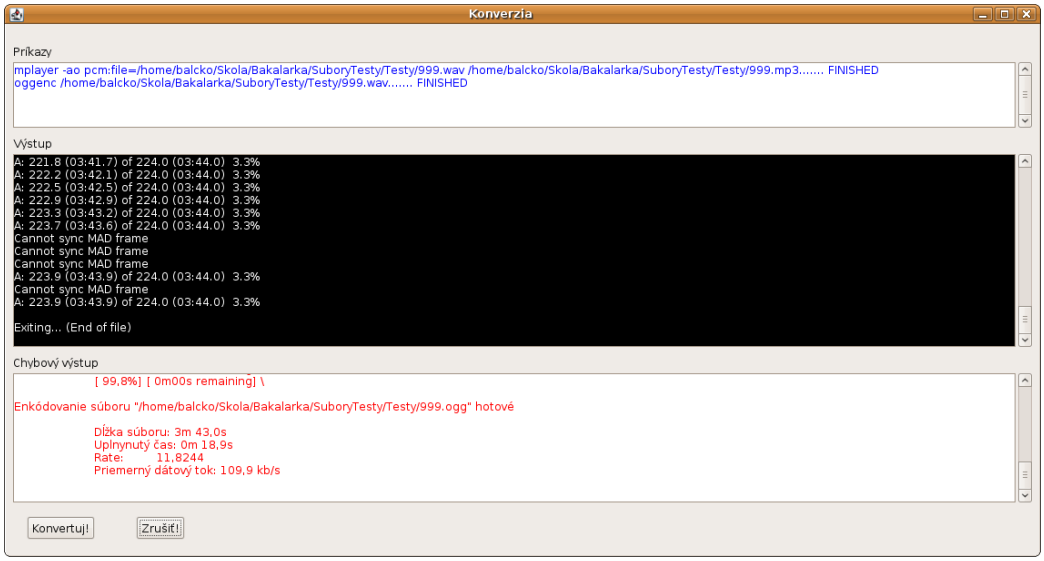
Graf transformácií bude vyzerať nasledovne:



Predvediem dvojprechodovú konverziu medzi z formátu mp3 na ogg.



Výsledok konverzie:



Kapitola 6

Záver

Touto prácou sa podarilo zrealizovať prostredie, v ktorom sa dá jednotným a intuitívnym spôsobom pristupovať ku využitiu rozličných konverzných programov s konzolovým uživateľským rozhraním. Prostredie umožňuje nastavovanie a spúšťanie konverzii bez nutnej znalosti detailov manuálových stránok programov.

Celá funkcionalita prostredia závisí od rozsahu spektra programov pripojených ku prostrediu a správneho namapovania príkazov a ich volieb na objekty prostredia, čo sú úlohy pre administrátora systému.

Literatúra

[ABC] Abc amber software.
`http://www.processtext.com/index.html`.

[GrB] How to use gridbaglayout.
`http://java.sun.com/docs/books/tutorial/uiswing/layout/gridbag.html`.

[Pir99] Vartan Piroumian. *Java Gui Development*. Sams; 1st edition (August 1999), 1999.

[PN96] Michael Morrison Patrick Naughton. *The Java Handbook*. Osborne/McGraw-Hill, 1996.

[Sis] Štandardy pre informačné systémy verejnej správy.
`http://standardy.informatika.sk/`.

[UML] The unified modeling language.
`http://www.uml.org/`.