UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

VIZUÁLNY JAZYK NA KONVERZIU SÚBOROV

2011

Adam Poldauf

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFOMATIKY

VIZUÁLNY JAZYK NA KONVERZIU SÚBOROV

Bakalárska práca

Evidenčné číslo :	fc46c39e-2756-4db1-8048-2a3414591246
Študijný program :	Aplikovaná informatika
Študijný odbor:	9.2.9 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Mgr. Pavel Petrovič, PhD.

Bratislava 2011

Adam Poldauf

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Adam Poldauf
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský

Názov: Vizuálny jazyk na konverziu súborov

Cieľ: Navrhnúť a naimplementovať interpreter pre vizuálny jazyk na konverziu súborov. Príklad: textový súbor obsahuje zoznam študentov, údaje o každom študentovi zaberajú aj niekoľko riadkov... zo súboru potrebujeme vytiahnuť iba meno, priezvisko a rok narodenia a tieto údaje zlúčiť s údajom o známke študenta, ktorý sa nachádza v inom súbore, kde sú iba priezviská a známky. Konverzia sa vytvorí "nakreslením" dátových tokov medzi zdrojovými, spracujúcimi a cieľovými blokmi. Bloky môžu obsahovať aj skripty, regulárne výrazy a pod. ale vytvorenie takéhoto konverzného skriptu, ktorý je možné spúšťať aj v dávkach, by malo byť omnoho jednoduchšie ako vytvorenie skriptu pre sed, či awk.

Vedúci: Mgr. Pavel Petrovič, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 02.10.2010

Dátum schválenia: 25.10.2010

doc. RNDr. Mária Markošová, PhD.
garant študijného programu

.....
študent

.....
vedúci

Čestné prehlásenie

Čestne prehlasujem, že som túto bakalársku prácu ako aj priloženú aplikáciu vypracoval samostatne s použitím uvedenej literatúry za odbornej pomoci vedúceho bakalárskej práce.

Bratislava, 3.6.2011

.....

Poďakovanie

Rád by som sa poďakoval vedúcemu mojej bakalárskej práce, pánovi Mgr. Pavlovi Petrovičovi, PhD.za pomoc a cenné rady pri tvorbe tejto práce.

Abstrakt

POLDAUF, Adam: *Vizuálny jazyk na konverziu súborov* (Bakalárska práca), Univerzita Komenského v Bratislave. Fakulta Matematiky, Fyziky a Informatiky; Katedra aplikovanej informatiky. Vedúci práce: Mgr. Pavel Petrovič, PhD: FMFI UK, 2011, 43 strán.

Cieľom mojej bakalárskej práce je navrhnúť a implementovať nástroj na vytváranie grafických popisov konverzie súboru ktorý umožní vytvorené konverzie vykonávať. Práca poskytuje prehľad dostupných nástrojov na konverziu súboru, neanalyzuje však žiaden nástroj ktorý by daný problém riešil rovnakým, grafickým, spôsobom pretože podľa všetkých dostupných informácii taký nástroj zatiaľ neexistuje. Výsledná aplikácie umožňuje vytvorenie konverzie súboru v grafickom prostredí, jej vykonávanie, uloženie pre ďalšie použitie.

Kľúčové slová: vizuálny jazyk, spracovanie súboru, úprava dát.

Abstract

POLDAUF, Adam: *Visual language for file conversion* (Bachelor thesis), Comenius University in Bratislava. Faculty of Mathematics, Physics and Informatics: Department of Applied Informatics. Supervisor: Mgr. Pavel Petrovič, PhD: FMPH CU, 2011, 43 pages.

Topics of my bachelor thesis is to design and implement tool which is capable of creating visual scripts for file processing and also process files with created scripts. Thesis also gives an overview of available tools for file processing. To my best knowledge I am not aware of any solution of this kind which solves file processing in this way. Created application enables user to create visual script for file processing, execution of created script, and save it for next use.

Keywords: visual language, file processing, data conversion.

Obsah

Úvod

Cieľ bakalárskej práce

Navrhnúť a naimplementovať interpreter pre vizuálny jazyk na konverziu súborov. Príklad: textový súbor obsahuje zoznam študentov, údaje o každom študentovi zaberajú aj niekoľko riadkov... zo súboru potrebujeme vytiahnuť iba meno, priezvisko a rok narodenia a tieto údaje zlúčiť s údajom o známke študenta, ktorý sa nachádza v inom súbore, kde sú iba priezviská a známky. Konverzia sa vytvorí "nakreslením" dátových tokov medzi zdrojovými, spracujúcimi a cieľovými blokmi. Bloky môžu obsahovať aj skripty, regulárne výrazy a pod. ale vytvorenie takéhoto konverzného skriptu, ktorý je možné spúšťať aj v dávkach, by malo byť omnoho jednoduchšie ako vytvorenie skriptu pre sed, či awk.

Štruktúra bakalárskej práce

Práca je rozdelená podľa obsahu do viacerých kapitol. Úvodná kapitola oboznamuje s cieľom práce a popisuje logické členenie práce na kapitoly. Prvá kapitola je zameraná na analýzu už existujúcich nástrojov na konverziu a spracovanie súborov spolu s ich porovnaním.

V druhej kapitole sa nachádza špecifikácia zadávateľových požiadaviek na systém, spolu s návrhom vizuálneho jazyka. Obsahuje aj grafický návrh užívateľského rozhrania. Tretia kapitola obsahuje návrh riešenia.

V štvrtej kapitole sa nachádza zhodnotenie výsledného systému, popísané zmeny ktoré boli vyšpecifikované počas fázy implementácie a teda nie sú popísané v predchádzajúcich kapitolách. V kapitole sa nachádzajú aj príklady použitia systému spolu s návodom ako systém rozšíriť o novú funkcionalitu. Kapitola obsahuje ukážky z výslednej aplikácie.

V poslednej kapitole sa nachádza záverečné zhrnutie.

1. Analýza východísk

Podľa dostupných informácii neexistuje nástroj ktorý by umožňoval spracovanie súboru podľa grafického predpisu. Existuje však mnoho nástrojov ktorý riešia spracovanie súboru pričom predpis akým sa dáta v súbore upravujú sa určuje textovou formou. V nasledujúcej časti tieto nástroje predstavím a porovnám.

1.1. AWK

AWK je programovací jazyk, navrhnutý na analýzu alebo manipuláciu textových dát. Jeho syntax je podobná jazyku C. Prvá verzia AWK bola vytvorená v roku 1977. Názov vznikol z počiatočných písmen mien jeho tvorcov, ktorými boli Alfred V. **A**ho, Peter J. **W**einberger a Brian W. **K**ernighan. Dnes je súčasťou každého unixového operačného systému.

Jednou zo základných funkcií je prehľadávanie súboru na základe zadaných kritérií. Napríklad ak reťazec v súbore spĺňa kritéria, môžeme vykonať akcie, napríklad nahradiť ho iným.

AWK obsahuje v sebe built-in funkcie, taktiež umožňuje písanie vlastných funkcii, je v ňom možné definovať premenné, polia a definovať podmienky a používať cykly.

Ukážka syntaxe:

```
BEGIN {
  BEGIN_MSG  = "From"
  BEGIN_BDY  = "Precedence:"
  MAIN_KEY   = "Subject:"
  VALIDATION = "[MONTH REPORT]"

  HEAD = "NO"; BODY = "NO"; PRINT="NO"
  OUT_FILE = "Month_Reports"
  }

  {

  if ( $1 == BEGIN_MSG ) {
    HEAD = "YES"; BODY = "NO"; PRINT="NO"
    }

  if ( $1 == MAIN_KEY ) {
    if ( $2 == VALIDATION ) {
      PRINT = "YES"
      $1 = ""; $2 = ""
```

```
    print "\n\n"$0"\n" > OUT_FILE
    }
  }

if ( $1 == BEGIN_BDY ) {
  getline
  if ( $0 == "" ) {
    HEAD = "NO"; BODY = "YES"
  } else {
    HEAD = "NO"; BODY = "NO"; PRINT="NO"
    }
  }

if ( BODY == "YES" && PRINT == "YES" ) {
  print $0 >> OUT_FILE
  }
}
```

1.2 Sed (Stream editor)

Je stream editor vyvinutý z unixového editoru ed, vhodný na transformovanie sekvenčného prúdu textových dát. Sed prechádza vstupný súbor po riadkoch, podľa definovaných pravidiel riadok upraví a vypíše.

Funkcionalita umožňuje aj písanie komplexných textových transformácii, na rozdiel od AWK nepodporuje premenné a preto sa využíva najmä na pri riešení menších úloh. Najväčšia jeho výhoda oproti awk je rýchlosť spracovania dát.

Naučiť sa Sed je pomerne ťažké, vďaka jeho jednopísmenkovým príkazom.

Ukážka syntaxe:

```
#! /bin/sed -nf

:join
/<[^>]*$/ { N; s/[    *]\n[   *]/ /; b join; }
/<[   ]*\([aA]\|[iI][mM][gG]\)/!b
s/<[   ]*\([aA]\|[iI][mM][gG]|[aA][rR][eE][aA]\)[     ]\+/<a /g
s/<a\([^>]*\)[       ][hH][rR][eE][fF]=/<a\1 href=/g
s/<a\([^>]*\)[       ][aA][lL][tT]=/<a\1 alt=/g
s/href=\([^"   >]\+\)/href="\1"/g
s/alt=\([^"    >]\+\)/alt="\1"/g
s/\( alt="[^"]*"\)[^>]*\( href="[^"]*"\)/\2\1/g
s/<a[^>]* href="/<a href="/g
s/\(<a href="[^"]*\)"[^>]* alt="\([^"]*"\)/\1|\2/g
```

2. Funkcionalita systému

Časťou mojej práce bolo navrhnúť vizuálny jazyk na spracovanie súboru, všetky informácie uvedené v tejto kapitole obsahujú poznatky z tejto fázy mojej práce, pri ktorej som všetky moje výsledky konzultoval s mojím školiteľom. Výsledky hovoria o tom ako bude výsledný systém fungovať, popisujú jeho funkcionalitu, neobsahuje popis častí ktoré boli vymyslené až počas fázy implementácie.

Podkapitola 2.1. obsahuje stručnú špecifikáciu funkcionality akú by mal výsledný systém mať, ktorá vznikla na prvom stretnutí so školiteľom. Ostatné časti kapitoly odzrkadľujú výsledky návrhu vizuálneho jazyka.

2.1. Požiadavky na systém

Pri návrhu a implementácii systému sme sa zamerali na najmä nasledujúce ciele:

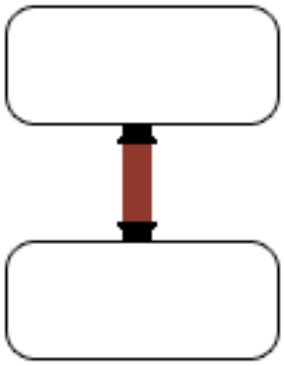
- Jednoduchý a graficky prehľadný, spracovanie by malo byť jednoduchšie ako pomocou sed alebo awk
- Popis konverzie by sa mal dať uložiť, následné spúšťanie z command-line
- Implementovaný tak, aby bol ľahko rozšíriteľný o novú funkcionalitu

Stručne opísaná funkcionalita, ktorú by mal systém zvládať:

- Textová konverzia:
    - Text – text
    - pridávať informácie
    - spájať viacero súborov
    - rozdeľovať súbor do viacerých
- Práca s METADATA rôznych formátov súborov
    - Zobraziť zoznam JPEG z priečinka spolu s ich vybranými METADATAMI
- Práca s XML, dolovanie údajov

2.2. Grafická konverzia

Konverzia je vytvorená graficky, na grafickej ploche systému za pomoci stavebných prvkov konverzie systému, ktoré predstavujú bloky a datovody. Blok predstavuje základnú funkčnú jednotku systému, počas vykonávania konverzie spracúva svoj vstup a výsledky posiela na svoje výstupy. Každý blok má určitý počet vstupov a určitý počet výstupov, ktoré sa prepájajú pomocou datovodov. Prepojením výstupu jedného bloku so vstupom iného vzniká medzi blokmi datavod, ktorý má za následok že výsledok práce prvého bloku sa objaví na vstupe druhého bloku. Takto poprepájané bloky pomocou datovodov vytvárajú funkčné konverzie.



Obr.1 schematický nákres stavebných prvkov konverzie

2.3 Bloky

Blok predstavuje základný stavebný prvok konverzie, má svoje vstupy a výstupy ktorých počet je možné meniť počas vytvárania konverzie, v rozsahu podľa špecifikácie uvedenej nižšie. Jedným prečítaním svojich vstupov, ich spracovaním a odoslaním výstupov vykonáva blok svoj krok.

Bloky sú rozdelené do nasledujúcich 3 kategórii:

- Source – obsahujúca bloky ktoré získavajú dáta
- Manipulation – obsahujúca bloky ktoré spracúvajú dáta
- Target – obsahujúca bloky ktoré zapisujú, poprípade zobrazujú dáta

2.3.1 Source Bloky

LineReader:

Číta vstupný súbor po riadkoch, každý riadok pošle na každý svoj výstup.

Počet vstupov: 0

Počet výstupov: 5

Parametre:

- vstupný súbor

DirectoryReader:

Prechádza obsah zložky a hľadá súbory definovaného typu. Na každý svoj výstup posiela absolútnu cestu k danému súboru.

Počet vstupov: 0

Počet výstupov: 5

Parametre:

- zložka(path)
- filetype

2.3.2 Manipulation Bloky

Spliter:

Delí reťazec zo vstupu na podreťazce na základe parametra. Pre každý výstup definujeme počet podreťazcov ktoré sa mu posielajú, formát zadávania je do textového poľa, zadávame jednotlivé počty pre výstupy zľava doprava oddelené bodkočiarkou. Ak zadáme menší počet čísel ako je počet výstupov, všetky ostatné podreťazce budú poslané do prvého

výstupu ktorému sme neudali počet. Ak má nastavaný príznak, po každom vykonaní kroku špeciálny znak konca riadku.

Príklad.:

Blok má 2 výstupy a má definovaný počet 1;4 . Pri takomto nastavení sa pošle 1 podreťazec do prvého výstupu a 4 do druhého, ostatné sa nevyužijú.

Blok má dva 2 výstupy a má definovaný počet 2; . Pri takomto nastavení sa pošle 2 podreťazece do prvého výstupu a všetky ostatné do druhého, ostatné sa nevyužijú.

Blok má dva 1 výstupy a má definovaný počet 2; . Pri takomto nastavení sa pošle 2 podreťazece do prvého, ostatné sa nevyužijú.

Počet vstupov: 1

Počet výstupov: 5

Parametre:

- oddeľovač (delimiter)
- počet podreťazcov pre každý výstup
- príznak, či má blok posielať znak konca riadku.

Glue:

Spája vstupy do jedného reťazca a ten posiela na výstup, ak je definovaný spojovník tak ho pridá vždy medzi dva vstupy. Ak ma nastavený príznak waitForLineEnd, drží na vstupe poslednú hodnotu pokým nepríde znak koniec riadku na všetky jeho vstupy.

Počet vstupov: 5

Počet výstupov: 1

Parametre:

- spojovník (glue)
- waitForLineEnd (bool)

Filter:

Filtruje vstup voči zadanému reťazcu. V prípade že blok nemá nastavený príznak regulárneho výrazu, ak vstupný reťazec obsahuje filter, tak je vstupný reťazec poslaný na každý výstup bloku. Ak má nastavený príznak regulárneho výrazu tak parameter filter považuje za regulárny výraz a vstupný reťazec posiela na každý svoj výstup ak vyhovuje danému regulárnemu výrazu. Blok akceptuje štandardnú syntax regulárnych výrazov používanú v jazykoch Java, Perl .

Počet vstupov: 1

Počet výstupov: 5

Parametre:

- filter (filter against)
- príznak regulárneho výrazu (is regexp)

Cutter:

Hľadá vo vstupnom reťazci pod reťazec definovaný parametrami. Ak daný podreťazec nájde pošle ho na výstup. Ak nájde viac výskytov v 1 vstupe, každý posiela ako samostatný výstup. Obsahuje príznak ktorý určuje či počiatočná a koncová sekvencia ponechá na výstupe alebo sa odstrihne

Počet vstupov: 1

Počet výstupov: 1

Parametre:

- start
- end
- príznak ponechania (Keep Sequences)

MetaMiner:

Na vstup mu prichádzajú súbory, on so súborov vytiahne definované metadata a pošle ich na výstupy.

Počet vstupov: 1

Počet výstupov: 5

Parametre:

- názov metatagu pre každý výstup

2.3.3 Target Bloky

LineWriter:

Zapisuje vstup ako riadok do výstupného súboru.

Počet vstupov: 1

Počet výstupov: 0

Parametre:

- výstupný súbor

2.4. Datovody

Datavod predstavuje funkčné prepojenie medzi blokmi, po ktorom sa posielajú dáta medzi blokmi.

Medzi blokmi sa v datovodoch posielajú dáta obalené v Packetoch, čo umožňuje jednotný prístup a jednotné narábanie s posielanými dátami.

Systém bude obsahovať 3 typy datovodov :

1. Neobmedzený – môže v sebe akumulovať neobmedzene veľa packetov, po prečítaní sa prečítaný packet vymaže
2. Obmedzený – môže obsahovať maximálne 1 packet, po prečítaní sa packet vymaže, ak sa blok snaží písať do obmedzeného datovodu v ktorom sa nachádza packet, tak sa pôvodný packet prepíše novým
3. Pamäť - môže obsahovať maximálne 1 packet, ktorý v ňom po prečítaní ostáva pokým nie je prepísaný iným packetom, teda môže byť prečítaný viac ako 1 krát

2.5. Vykonávanie

Mechanizmus vykonávania predstavuje jadro celého systému, popisuje v akom poradí budú jednotlivé bloky konverzie pracovať. Je možné, že rovnaká konverzia bude produkovať rôzne výsledky pri rovnakých vstupných dátach pri rozdielnom spôsobe vykonávania. V systéme budú nasledujúce dva mechanizmy vykonávania konverzie.

1. Synchrónne vykonávanie
2. Asynchrónne vykonávanie

2.5.1. Synchrónne vykonávanie

Celé spracovanie riadi globálny časovač. Všetky bloky súčasne vykonajú jeden krok(ak môžu, teda máju na vstupoch packety), ich činnosť je synchrónna.

Výsledok práce bloku s jedného kroku časovača sa objaví na vstupoch bloku ktorý je s ním prepojený pomocou datovodu až v nasledujúcom kroku časovača.

Ak v jednom kroku časovača nevykoná krok ani jeden blok konverzie vykonávanie sa končí.

Schematický zápis:

```
kým sa vykonal aspoň jeden krok bloku v minulom kole{
        pre všetky bloky konverzie C{
                vykonaj jeden krok C, výsledky ulož do cache
        }
        pre všetky bloky konverzie C{
                pošli výsledky ktoré sú v cache C, na príslušné datovody
        }
}
```

2.5.2. Asynchrónne vykonávanie

Celé spracovanie riadi front(FIFO) spracovania. Vykonávať svoj krok bude vždy iba blok, ktorý je na začiatku frontu.

Pred začatím vykonávania sa bloky ktoré získavajú dáta, sú typu “source“, zaradia do frontu. Následne sa spustí vykonávanie, vždy pracuje iba jeden blok, a po každom kroku sa upraví front podľa stavov na vstupoch blokov.

Vykonávanie sa končí keď je front prázdny.

Schematický zápis:

```
zaraď všetky “source“ bloky na front spracovania
kým nie je front prázdny{
        vyber blok B z frontu
```

```
        ak máju kapacitu výstupné datovody B{
                vykonaj jeden krok bloku B a posli výstupy ne jeho  výstupne datovody
        }
          pre všetky bloky C, na ktoré poslal výstup B{
                ak má C naplnené všetky vstupy, zaraď ho na front
        }
        ak sú naplnene všetky vstupy B, zaraď ho na front
}
```

2.6 Funkcionalita užívateľského rozhrania

Užívateľské rozhranie tvorí najdôležitejšiu časť systému, je potrebné aby bolo intuitívne, keďže sa v ňom bude dať vytvoriť grafická konverzia, poprípade upraviť.

Užívateľské rozhranie sa skladá z :

1. paleta blokov
2. grafická plocha
3. nastavenia bloku
4. textové okno

Bude poskytovať nasledovnú funkcionalitu:

- Vytvorenie novej konverzie:
    - pridanie nového bloku
    - odobratie bloku
    - pridať datovod
    - odobrať datovod
    - bloky bude možne v rámci plochy presúvať pomocou drag & drop
- Uloženie vytvorenej konverzie
- Otvorenie/ načítanie vytvorenej konverzie

2.7 Vykonávanie uloženej konverzie z príkazového riadku

Uloženú konverziu bude možné spúšťať aj z príkazového riadku a to nasledovne.

Pri zavolaní interpretera uloženej konverzie s parametrom ktorý predstavuje cestu k uloženej konverzii vypíše na štandardný výstup počet source a target blokov pre ktoré je potrebné zadať vstupné respektíve výstupné súbory.

Príklad volania:
$ VFLinterpreter konverzia.xml

Príklad výstupu na uvedené volanie :
$ Conversion from file konverzia.xml has 1 source(id:1) and 2 target(id:6,id:7) blocks.

Pri zavolaní interpretera uloženej konverzie s parametrom ktorý predstavuje cestu k uloženej konverzii, parametrom určujúcim spôsob vykonávania a všetkými vstupnými a výstupnými súbormi sa konverzia vykoná ako keby bola spustená s užívateľského rozhrania.

Parameter určujúci spôsob vykonávania:

- -a pre asynchrónne vykonávanie
- -s pre synchrónne vykonávanie

Spôsob zadávania vstupných a výstupných súborov:

-<id bloku> <cesta>

Príklad volania:
$ VFLinterpreter konverzia.xml -a -1 C:\input.txt -6 C:\output1.txt -7 C:\output2.txt

Príklad výstupu na uvedené volanie :
$ Conversion successfully executed. – v prípade bezproblémového vykonania
$ Error during execution of conversion. – v prípade výskytu chyby

3. Návrh riešenia

Návrh systému je dôležitá časť projektu. V návrhu budeme vychádzať z funkčných požiadaviek uvedených vyššie. Systém musí byť navrhnutý flexibilne, aby bol ľahko rozšíriteľný.

3.1. Použité technológie

Systém budem implementovať v jazyku Java v programovacom prostredí NetBeans. Jazyk Java som zvolil pretože je objektový a programátor sa nemusí starať o nízkoúrovňovú správu hardwaru ako je napríklad manažment pamäte, na viac poskytuje bohaté štandardné API s množstvom užitočných tried, ktoré mi uľahčia implementáciu.

3.2. Členenie návrhu

Návrh člením na nasledujúce logické celky:

- Užívateľské rozhranie
- Dátovú časť

Dátová časť nesie zodpovednosť za vykonávanie konverzie, teda prácu blokov, ukladanie a načítavanie konverzie i za spúšťanie uloženej konverzie z príkazového riadku pomocou interpretera.

Užívateľské rozhranie je zodpovedné za vytváranie konverzie. V zmysle jednoduchej rozšíriteľnosti systému, je nutné navrhnúť tak aby v ňom neboli nutné žiadne zmeny v prípade rozšírenia funkcionality systému o nové bloky. Je teda potrebné aby bol mechanizmus vykresľovania blokov jednotný.

Problém tohto logického členenia je nasledovný, každý blok má svoje špecifické vlastnosti ktoré je nutné nastavovať v užívateľskom rozhraní a úplné odseparovanie dátovej časti od užívateľského rozhrania nebolo možné.

Riešenie som vymyslel nasledovne:

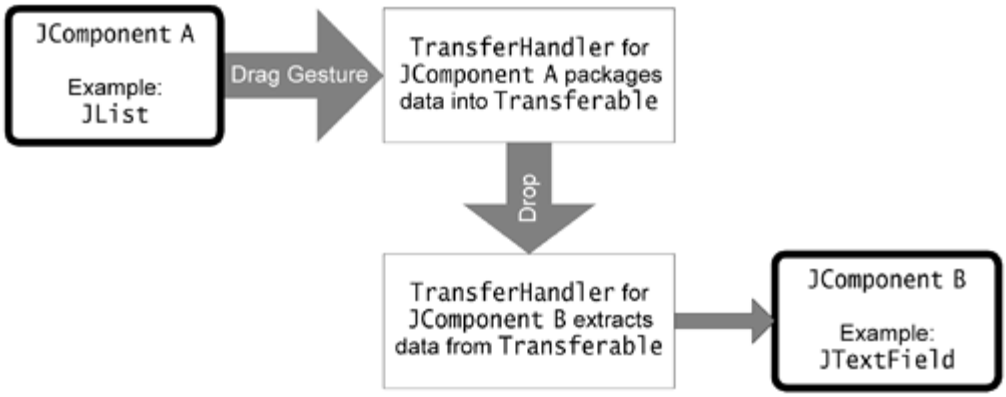
- každá trieda bloku z dátovej časti obsahuje inštanciu triedy JPanel, takzvaný propertiesPanel a taktiež je trieda zodpovedná za odchytávanie užívateľových interakcií s propertiesPanelom.
- každá trieda bloku z časti užívateľského rozhrania obsahuje inštanciu triedy bloku dátovej časti ktorú reprezentuje, tá jej poskytuje propertiesPanel keď ho potrebuje.

3.3. Užívateľské rozhranie

Všetky použité grafické prvky pochádzajú zo štandardnej knižnice jazyka JAVA, Swing.
Hlavnú časť predstavuje pridávanie nových blokov z palety blokov do grafickej plochy, ktoré bude pomocou Drag&Drop. Paleta blokov pozostáva z 3 paliet: Source, Manipulation, Target. Každé obsahuje inštancie triedy PalleteBlock podľa počtu blokov v danej kategórii. Grafická plocha pozostáva z plátna na ktoré je možno pridávať bloky a datovody.

3.3.1. Drag & Drop

Je akcia pri ktorej dochádza k premiestňovaniu vizuálnych objektov v užívateľskom rozhraní, teda o presun dát medzi jednotlivými komponentmi. Niektoré z komponentov knižnice Swing priamo podporujú túto funkcionalitu, pre tie ktoré nie je možnosť túto funkcionalitu doprogramovať, vytvorením vlastného objektu typu TransferHandler a jeho naviazaním na príslušné komponenty.

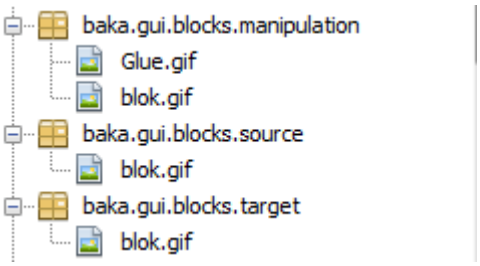


Obr.2 Drag & Drop mechanizmus v knižnici Swing

3.3.2. Triedy

PalleteBlock

Trieda zobrazujúca blok na palete. V premennej type nesie informáciu o tom aký blok reprezentuje, táto informácie je prenášaná pri pridávaní bloku na grafickú plochu. Táto informácia je využívaná i na kreslenie bloku na palete blokov. Najprv sa snaží načítať obrázok podľa mena bloku, ak taký nenájde použije univerzálny obrázok pre danú kategóriu s názvom "blok.gif".



Obr.3 Štruktúra ukladania obrázkov

GraphicBlock

Trieda zobrazujúca blok na grafickej ploche. Má na starosti kreslenie bloku, i s jeho vstupmi a výstupmi, ich počet získava z inštancie potomka triedy DataBlock ktorú obsahuje. Kreslenie bloku funguje rovnako ako v triede PalleteBlock.

GraphicCanvas

Trieda ktorá zobrazuje konverziu(bloky, datovody)

GraphicCanvasDropTargetListener

Trieda ktorá informuje objekt triedy GraphicCanvas o pozícii dropu nového bloku v rámci plochy.

GraphicCanvasMouseAdapter

Trieda ktorá ma na starosti akcie myši na objekte triedy GraphicCanvas.

- pri kliku na blok zobrazí v časti okna pre nastavenia bloku panel s nastaveniami pre kliknutý blok
- pri kliknutí na výstup bloku a následne vstup iného bloku vytvorí datovod ak tam nebol, ak bol tak ho odstráni.
- presúvanie blokov pomocou drag

BlockTransferHandler

Inštancia tejto triedy zabezpečuje presun dát pri drag&drop blokov z platy do grafickej plochy.

Datovod

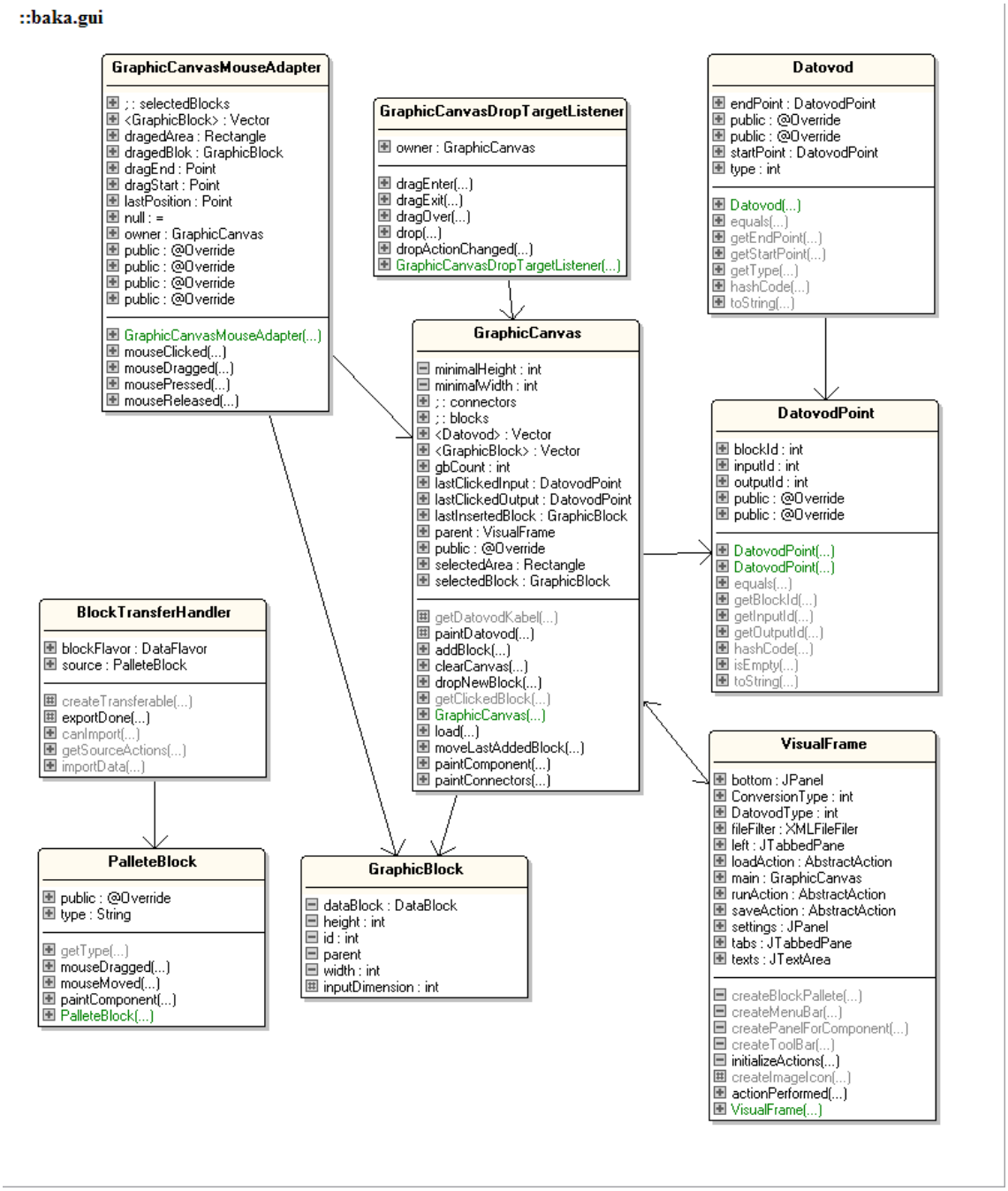
V grafickom rozhraní je datovod reprezentovaný dvojicou(začiatok, koniec) objektov typu DatovodPoint. Súradnice všetkých bodov pri kreslení datovodu sú vypočítavané pri každom kreslení.

DatovodPoint

Obsahuje id bloku a id vstupu alebo výstupu.

VisualFrame

Hlavná trieda grafického rozhrania, vytvára inštanciu komponentu JFrame, ktorá predstavuje okno systému a inicializuje všetky komponenty systému.



Obr.4 Triedny diagram užívateľského rozhrania

3.4. Dátová časť

Dátová časť ma na starosti vykonávanie vytvorenej grafickej konverzie, teda triedy tejto časti čítajú dáta uložené v súboroch, upravujú ich a následne zapisujú. Dôležitý je aj spôsob uloženia grafickej konverzie a jeho následné načítavanie.

3.4.1. Znaková sada

Všetky údaje v súboroch sú uložené v číselnej podobe. Znaková sada určuje spôsob akým sú znaky prevádzané na čísla, priradzuje každému znaku jedinečný identifikátor v rámci danej sady. Na príklad v znakovej sade ASCII, znaku A prislúcha 65, znaku B 66, atď. Existuje viac ako jedna znaková sada, medzi najpoužívanejšie patri ASCII, Unicode, Windows-1250.

Pri čítaní dát zo súboru je dôležité správne rozpoznať znakovú sadu konkrétneho súboru. Java obsahuje vo svojom štandardnom API mnoho tried na čítanie a zápis dát do súboru, ktoré riešia tento problém.

3.4.2. Ukladanie konverzie

Možnosť uložiť vytvorenú konverziu je veľmi dôležitá, dáva možnosť využívať rovnakú konverziu viac krát bez nutnosti jej opätovného vytvorenia, ktoré môže byť časovo náročné.

Konverziu som sa rozhodol ukladať vo formáte XML, ktorý poskytuje nasledujúce výhody:

- formát ktorý je čitateľný aj pre človeka
- editovateľný aj pomocou textového editora
- dobre prenosný na platforme nezávislý formát

3.4.3. Triedy

Conversion

Hlavná trieda konverzie, je v nej na implementované vykonávanie konverzie, ukladanie a načítavanie konverzie zo súboru.

DataBlock

Abstraktná trieda ktorá je nadtriedou všetkých blokov systému. Vykonáva úkony ktoré sú pre bloky spoločné, vytvorenie propertiesPanelu s prvkami na nastavovanie počtu vstupov a výstupov, ukladanie a načítavanie spoločných vlastností zo súboru. Obsahuje tri abstraktné metódy resetBlockState() ktorá sa spúšťa vždy na začiatku pred vykonávaním konverzie, processInput() ktorá vykonáva krok bloku, t.j. jedno prečítanie vstupov ich manipulácia, odoslanie výstupov a metóda finalizeBlockWord() ktorá sa spúšťa na konci vykonávania konverzie.

DataBus

Trieda reprezentujúca zbernicu, každý blok má vstupnú a výstupnú, ktorá slúži na čítanie vstupov a odosielanie výstupov bloku. Jej prvky sú objekty typu DataChannel. Poskytuje metódy na čítanie konkrétneho vstupu readSpecific(int id), zápis do konkrétneho výstupu writeSpecific(int id, String s), respektíve do všetkých výstupov writeToAll(String s). Taktiež poskytuje metódu hasInputs(), ktorá informuje o tom či každý DataChannel v zbernici obsahuje aspoň jeden Packet.

Trieda je taktiež zodpovedná za extrahovanie textovej informácie z Packetu pri čítaní a vytváranie Packetu pri zápise.

DataChannel

Trieda reprezentujúca prepojenie dvoch blokov po ktorom dochádza k presunu dát.

Packet

Trieda ktorá reprezentuje dáta posielané v DataChanneloch, ktorá umožňuje prípadnú internú zmenu ich uchovávanie , poskytuje metódy setTextContent(String s), getTextContent().

Config

Trieda obsahujúca parametre aplikácie medzi ktoré patria: veľkosť zobrazovaných blokov na grafickej ploche, zoznam blokov pre paletu blokov a mnohé iné, ktorú sú využívané na viacerých miestach v kóde aplikácie, pomocou tejto triedy môžeme parametre aplikácie pohodlne meniť.

Main

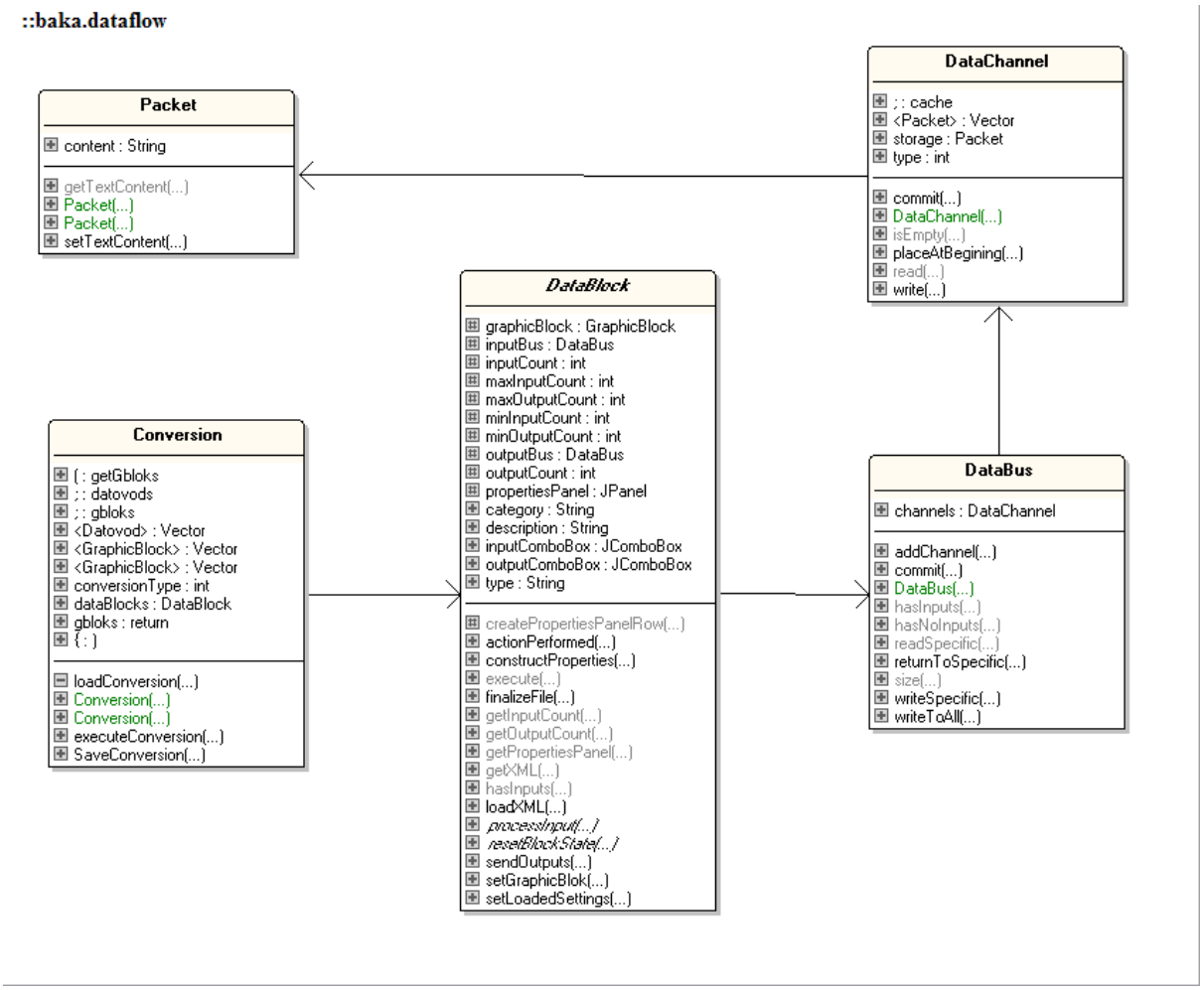
Trieda ktorá pri spustení vytvorí inštanciu triedy VisualFrame, teda vytvorí a zobrazí užívateľské rozhranie

VFLinterpreter

Trieda interpretera ktorý umožňuje spúšťanie uložených konverzií z príkazového riadku. Jej práca spočíva v triedení argumentov s ktorými bola trieda spustená a následnom vytvorení objektu triedy Conversion podľa príslušných parametrov.

Triedy blokov

Triedy blokov sú potomkami triedy DataBlock. implementujú metódu processInput() ktorá tvorí funkcionalitu bloku. Taktiež mnohé prekrývajú metódy na konštrukciu propertiesPanelu, ukladanie a načítavanie spoločných vlastností zo súboru. Vo všetkých metódach však využívajú funkcionalitu svojej nadtriedy explicitným volaním metódy nadtriedy.



Obr.5 Triedny diagram dátovej časti

4. Implementácia

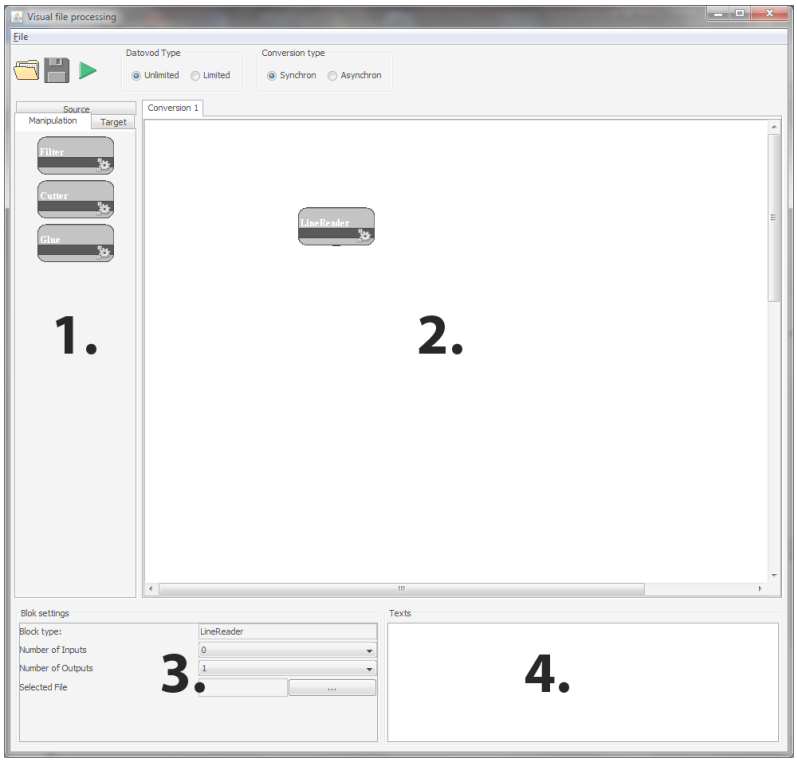
V tejto časti bakalárskej práce idem rozoberať ako sa mi podarilo splniť uvedenú špecifikáciu a implementovať návrh riešenia. Kapitola obsahuje obrázky aplikácie so stručnými popismi.

Systém sa mi podarilo naimplementovať tak že spĺňa takmer všetky funkčné požiadavky špecifikované v druhej kapitole a v súlade s návrhom riešenia z tretej kapitoly.

Za nedostatok považujem nedostatočné množstvo blokov, ktorých mohlo byť podstatnejšie viac, no vzhľadom na nedostatok času sa mi ich nepodarilo viac vymyslieť a naimplementovať. Táto kapitola však obsahuje aj návod ako systém možno pomerne jednoducho rozšíriť o novú funkcionalitu.

4.1. Užívateľské rozhranie

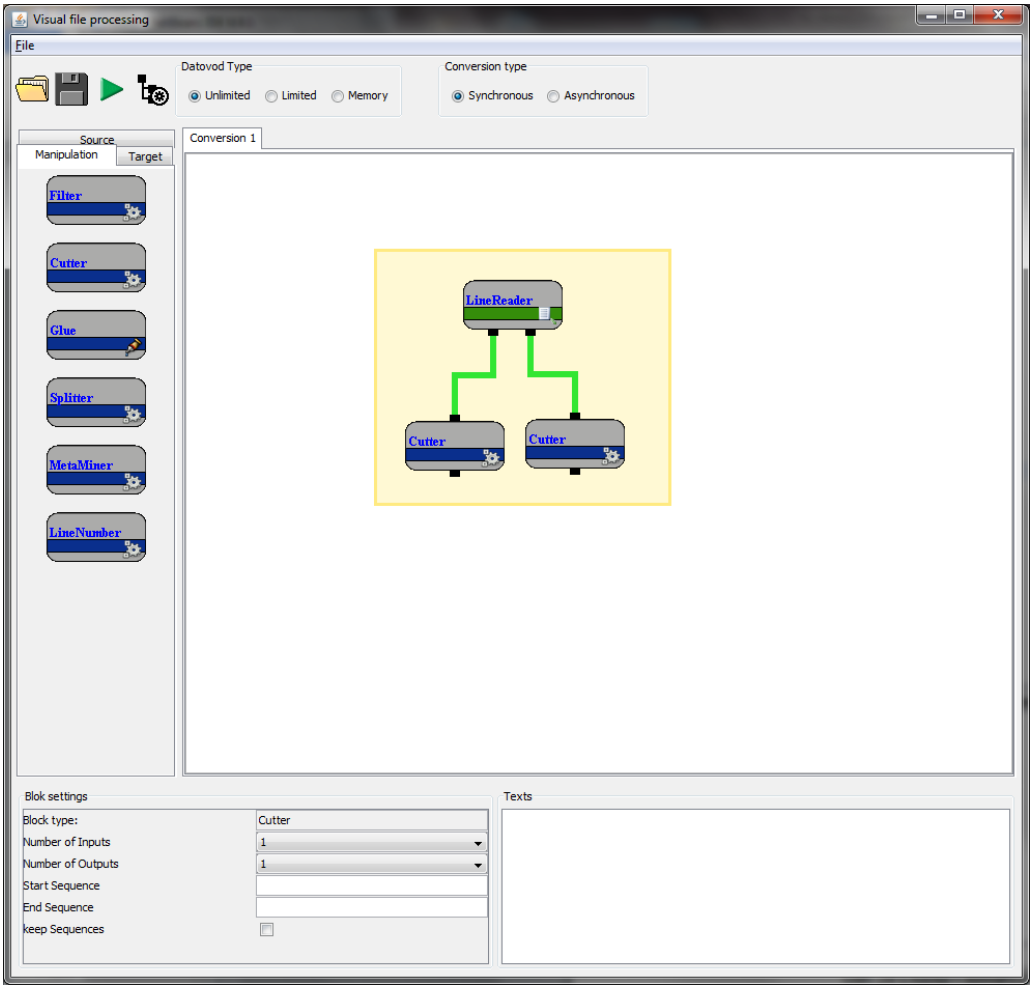
Užívateľské rozhranie sa skladá z palety blokov (1.), grafickej plochy (2.), oblasti pre nastavovanie vlastnosti bloku (3.), textového okna (4.) .



Obr. 7 Ukážka – užívateľské rozhranie

4.2. Presúvanie blokov

Na grafickej ploche je možné označiť a presúvať viac blokov naraz, táto funkcionalita bola pridaná počas implementácie, nie je uvedená v špecifikácii funkcionality.



Obr. 8 Ukážka – presúvanie blokov

4.3. Voľba typu datovodu a spôsobu vykonávania

V aplikácii je možné pri vytváraní konverzie použiť rôzne typy datovodov. V užívateľskom rozhraní je miesto kde je výber datovodu z 3

typov(Unlimited,Limited,Memory) aktuálny výber určuje typ nasledujúceho vytvoreného datovodu.

Rovnakým spôsobom je určovaný spôsob vykonávania konverzie, aktuálny výber určuje spôsob nasledujúceho vykonávania konverzie.

4.4. Príklady použitia

V tejto podkapitole sú uvedené príklady vzorové príklady konverzií ktoré je možné pomocou systému vytvoriť.

4.4.1 Prvý príklad - text

Máme vstupný súbor:

Autor kniha1 kniha2 kniha3 kniha4

Autor2 kniha1 kniha2 kniha3 kniha4

Autor3 kniha1 kniha2 kniha3 kniha5

....

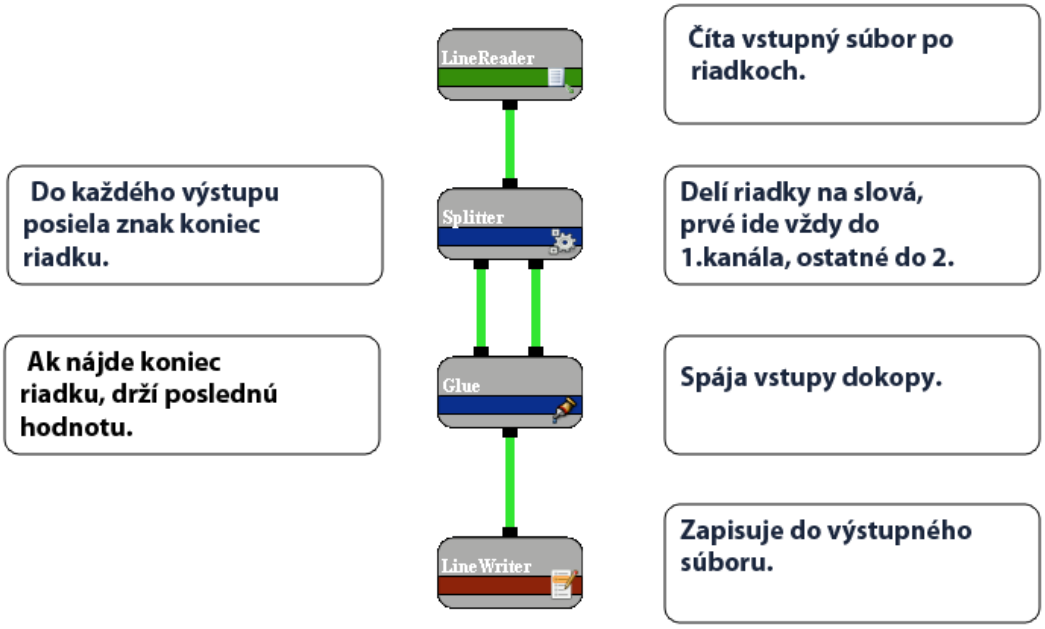
Náš cieľ je :

Autor kniha1

Autor kniha2

Autor kniha3

....



Obr.11 Príklad použitia č.1

4.4.2. Druhý príklad – text

Máme vstupný súbor:

Ide o html súbor v ktorom sú linky v tvare:

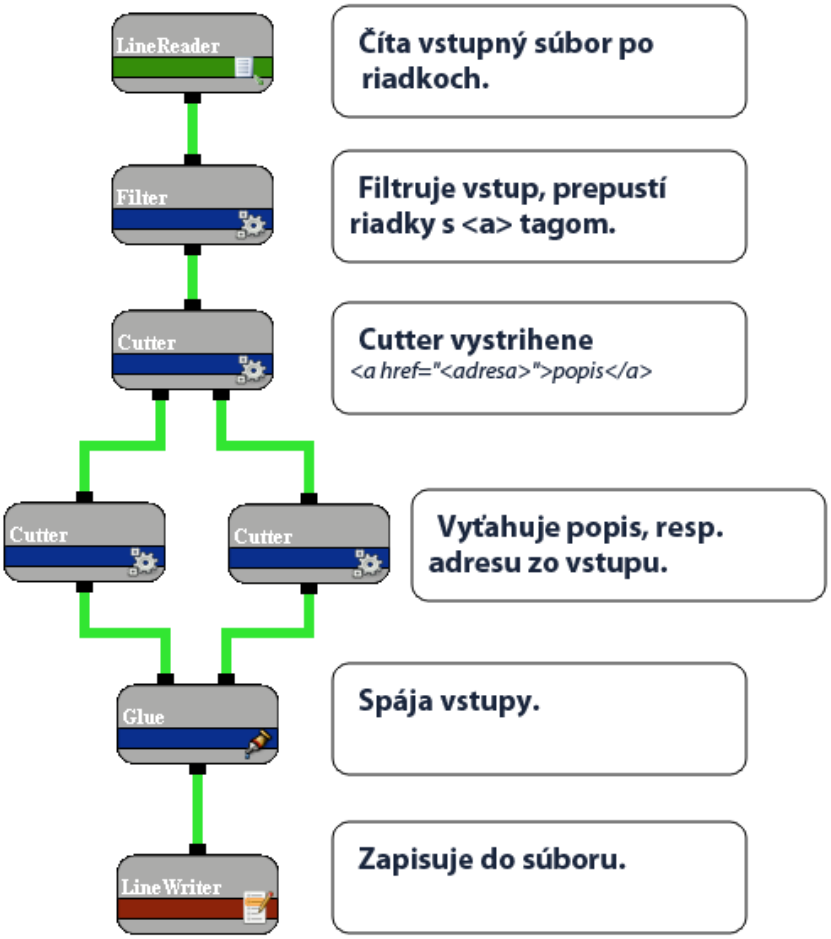
<a href="<adresa>">popis

Náš cieľ je:

Získať všetky linky a zapísať si ich vo formáte:

Popis <adresa>

Konverzia:



Obr.12 Príklad použitia č.2

4.4.3. Tretí príklad – META

Vstup:

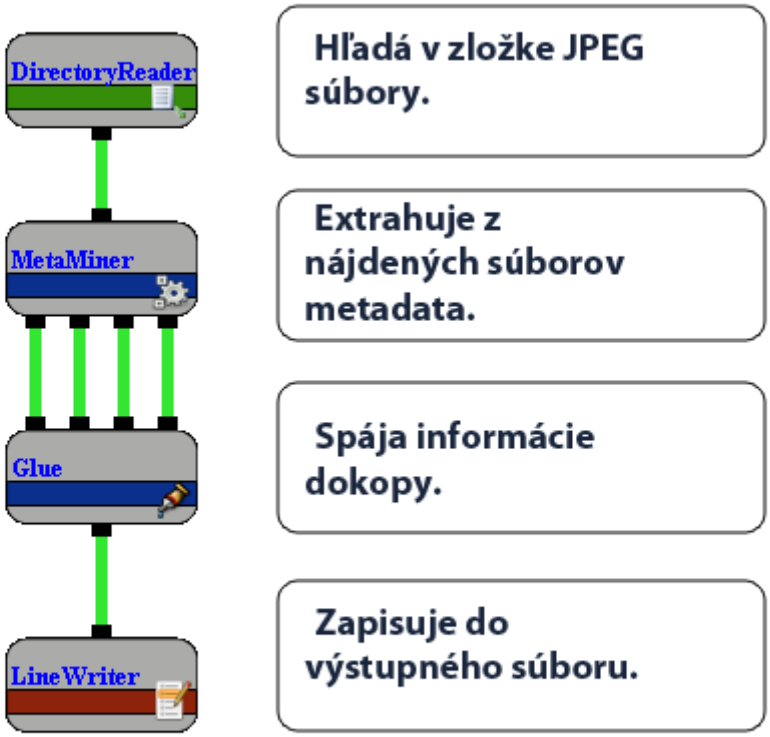
Zložka v ktorej sa nachádzajú JPEG súbory

Náš cieľ je:

Zoznam súborov v zložke

<meno> <dátum> <šírka> <výška>

Konverzia:



Obr.13 Príklad použitia č.3

4.5. Rozšíriteľnosť systému

V tejto časti idem ukázať ako rozšíriť funkcionalitu systému pomocou pripadania nového bloku.

Chceme aby systém zvládal číslovať riadky, teda dať poradové číslo každému riadku, potrebujeme blok ktorý by každému vstupu pridal poradové číslo v ktorom do bloku prišiel, nazvime ho LineNumber, bude logicky patriť medzi Manipulation bloky.

4.5.1. Postup implementácie

1. Vytvoriť triedu v balíčku baka.dataflow.blocks.manipulation nazvanú LineNumber, ktorá bude potomkom triedy DataBlock. Pre potreby zapamätania si poradového čísla vstupu bude mať trieda premennú LineNumber.

```
public class LineNumber extends DataBlock {

    private int LineNumber
}
```

2. Zaradiť triedu medzi Manipulation bloky do konfigurácie v triede Config, jej pripísaním do príslušného zoznamu.
3. Trieda LineNumber v konštruktore nastaví parametre bloku ako sú maximálny počet vstupov, výstupov, popis bloku.

```
public LineNumber() {
    this.description = "Add order number to each input.";
    this.category = "manipulation";
    this.type = "LineNumber";
    this.maxOutputCount = 1;
    this.outputCount = 1;
    this.maxInputCount = 1;
    this.inputCount = 1;
    constructProperties();
}
```

4. Aby trieda bola funkčná a samotná Java s ňou dokázala pracovať, musí trieda implementovať všetky abstraktné metódy svojej rodičovskej triedy.

Metóda resetBlockState() je vykonaná na každom bloku konverzie pred samotným vykonávaním konverzie.

```
@Override
public void resetBlockState() {
      this.LineNumber = 0;
}
```

Metóda finalizeBlockWork() je presný opak metódy resetBlockState() a je vykonaná na každom bloku po ukončení vykonávania konverzie. V prípade LineNumber ostáva prázdna.

```
@Override
public void finalizeBlockWork() {
}
```

Metóda processInput() je volaná počas vykonávania konverzie. Je to kľúčová metóda každého bloku, pretože vykonáva prácu bloku.

```
@Override
public void processInput() {
    String line = "";
    if ((line = inputBus.readSpecific(0)) != null) {
   this.LineNumber++;
        line = String.valueOf(this.LineNumber)+". "+line;
        outputBus.writeToAll(line);
    }
}
```

Po vykonaní týchto krokov sa blok objaví na palete blokov a bude poskytovať hore uvedenú funkcionalitu, teda pridá na začiatok každého vstupu poradové číslo.

4.5.2 Parametrizácia vytvoreného bloku

V prípade implementovania bloku ktorý potrebuje pre svoju prácu zadávanie parametrov od používateľa je potrebné pridať prvky na konfiguráciu bloku na jeho propertiesPanel, čo je obsahom nasledujúcej časti návodu v ktorej uvádzam ako upraviť nami vytvorený blok LineNumber, aby mal používateľ možnosť voľby či bude číslo umiestnené na začiatok alebo na koniec riadku.

1. Potrebujeme si reprezentovať informáciu o tom kde má byť číslo pridané a takisto túto informáciu vedieť meniť.

```
private int LineNumber;
private boolean atBegin; // slúži ako príznak toho kde má byť číslo
private JCheckBox positionCheckBox; // slúži na nastavovanie príznaku
```

2. Zmena tejto informácie je možná cez propertiesPanel ktorý je potrebné upraviť predefinovaním metódy, avšak je potrebné zavolať metódu nadtriedy pre vytvorenie ovládacích prvkov ktoré majú všetky bloky spoločné, t.j. počet vstupov a výstupov. Je potrebné inicializovať checkbox ktorý je následne pridaný spolu s popisom na propertiesPanel. Na vytvorenie riadku propertiesPanelu (popis, prvok) slúži metóda nadtriedy createPropertiesPanelRow(JComponent Caption, JComponent field).

```
@Override
public void constructProperties() {
    super.constructProperties(); // metóda nadtriedy
    this.positionCheckBox = new JCheckBox();
    this.positionCheckBox.setEnabled(true);
    this.positionCheckBox.setActionCommand("placePosition");
    this.positionCheckBox.addActionListener(this);
    JComponent row = this.createPropertiesPanelRow(new JLabel("Place at
beginning"), positionCheckBox);
    this.propertiesPanel.add(row);
 }
```

3. Obstarávanie odchytávania akcii ovládacích prvkov na propertiesPaneli sa stará trieda sama. V prípade viacerých ovládacích prvkov rozoznávame o ktorý prvok ide na základe actionCommandu ktorý sme nastavovali v predchádzajúcom kroku.

```
@Override
public void actionPerformed(ActionEvent e) {
    super.actionPerformed(e);
    if (e.getActionCommand().equals("placePosition")) {
       this.atBegin = this.positionCheckBox.isSelected();
    }
 }
```

4. Po takomto rozšírení bloku je nutné pre správne uloženie a načítanie konverzie ukladať si pri tomto bloku aj informáciu o tom kde má blok pridávať číslo. Na to je potrebné upraviť metódu pre uloženie getXML(Element e, Document doc), a metódy pre načítanie loadXML(Element e) ktorá vytiahne uložené informácii a setLoadedSettings() ktorá zabezpečí aby sa vytiahnuté načítané informácie správne zobrazili na propertiesPaneli.

```
public Element getXML(Element e, Document doc) {
    super.getXML(e, doc);

    Attr position = doc.createAttribute("position");
    position.setValue(String.valueOf(this.atBegin));
    e.setAttributeNode(position);
```

```
        return e;
    }

    @Override
    public void loadXML(Element e) {
        super.loadXML(e);
        String isReg = e.getAttribute("position");
        if (isReg.equals("true")) {
            this.atBegin = true;
        } else {
            this.atBegin = false;
        }
    }

    @Override
    public void setLoadedSettings() {
        super.setLoadedSettings();
        this.positionCheckBox.setSelected(this.atBegin);
    }
```

5. Na záver nesmieme zabudnúť upraviť samotnú činnosť bloku v metóde processInput().

```
    @Override
    public void processInput() {
        String line = "";
        if ((line = inputBus.readSpecific(0)) != null) {
            this.LineNumber++;
            if (atBegin) {
                line = String.valueOf(this.LineNumber) + ". " + line;
            } else {
                line = line + " " + String.valueOf(this.LineNumber) + ".";
            }
            outputBus.writeToAll(line);
        }
    }
```

Po takomto upravení blok LineNumber dosiahol požadovanú funkcionalitu.

V tejto časti návodu boli možnosti vytvárania komplikovanejších blokov, za pomoci umiestňovania ich ovládacích prvkov na propertiesPanel.

5. Záver

Cieľom mojej bakalárskej práce bolo navrhnúť a implementovať nástroj na vytváranie grafických popisov konverzie súboru ktorý umožní vytvorené konverzie vykonávať. Jej cieľová skupina je široká, keďže spracovanie dát patrí medzi neoddeliteľnú súčasť mnohých procesov v informatike.

Vývoj systému prebehol podľa očakávaní s drobnými problémami, no všetky problémy boli odstránené vyhľadaniu pomoci na internete alebo vďaka konzultácii so školiteľom.

S výsledkami mojej práce som vcelku spokojný, pretože spĺňa všetky požiadavky stanovené na začiatku, avšak počas vývoja sa objavili najmä zo strany môjho školiteľa nápady na vylepšenie . Niektoré s týchto nápadov sa podarili implementovať no niektoré kvôli nedostatku času nie.

Do budúcnosti by som chcel rozšíriť aplikáciu o funkcionalitu ktorá by umožnila vytvárať tzv. makro bloky, teda vytvorenú konverziu uložiť ako makro a neskôr ju mať pri vytváraní k dispozícii, čo by podstatne uľahčilo vytváranie rozsiahlejších konverzii.

Použitá literatúra

ELLIOTTE, R. H. 2006. *Java™ I/O*. 2. vyd. Sebastopol : O'Reilly, 2006. ISBN 0-596-52750-0.

HEROLD, H. 2004. *Awk & sed* : Příručka pro dávkové zpracování textu. 1. vyd. Brno : Computer Press, 2004. ISBN 80-251-0309-9.

WALRATH, K. a kol. 2004. *The JFC Swing Tutorial :* A Guide to Constructing GUIs. 2. vyd. Palo Alto : Addison Wesley, 2004. ISBN 0-201-91467-0.

HEROUT, P. 2007 *Java - grafické uživatelské prostředí a čeština.* 1. vyd. České Budějovice : Kopp, 2007. ISBN 80-7232-150-1.

Java SE Tutorials. [online]. Oracle, 2011. [cit.16.5.2011]. Dostupné na internete: <http://download.oracle.com/javase/tutorial/.

MCLAUGHLIN, B. 2001. *Java & XML*. 2. vyd. Sebastopol : O'Reilly, 2001. ISBN 0-596-00197-5.

Príloha č.1

CD s kompletnými zdrojovými kódmi aplikácie a aplikáciou samotnou. Súčasťou CD je aj súbor „Readme.txt“ popisujúci obsah CD.