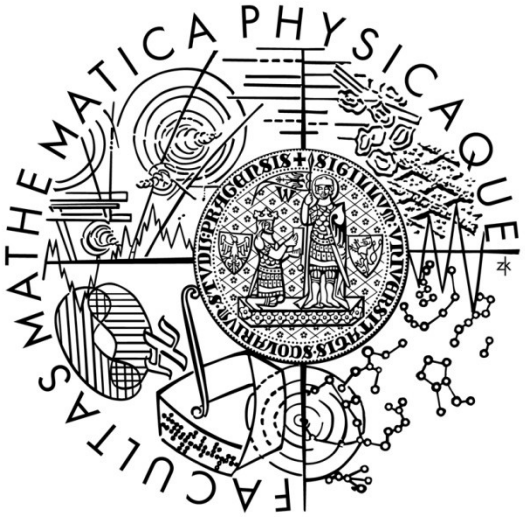
UNIVERZITA KARLOVA V PRAZE
MATEMATICKO-FYZIKÁLNÍ FAKULTA

BAKALÁŘSKÁ PRÁCE



Michal Novák

Vizualizace PML souborů

Katedra softwarového inženýrství

VEDOUCÍ BAKALÁŘSKÉ PRÁCE:
Mgr. Václav Novák
Ústav formální a aplikované lingvistiky

Studijní program: informatika, obor programování
2008

Poďakovanie

Ďakujem môjmu vedúcemu Mgr. Václavovi Novákovi za jeho podporu počas vývoja projektu, jeho rýchlu reakciu na všetky moje otázky a za cenné rady pri tvorbe programu i textu tejto práce.

Prehlasujem, že som svoju bakalársku prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičiavaním práce a jej zverejňovaním

V Prahe dňa 24.5.2008 Michal Novák

Obsah

Názov práce: Vizualizace PML souborů
Autor: Michal Novák
Katedra (ústav): Katedra softwarového inženýrství
Vedúci bakalárskej práce: Mgr. Václav Novák
E-mail vedúceho: novak@ufal.mff.cuni.cz

Abstrakt:

Predmetom tejto práce je aplikácia PMLViewer. Táto aplikácia je určená na vizualizáciu súborov vo formáte PML. Umožňuje vykresliť závislostné alebo zložkové stromy popísané v tomto formáte, pričom poskytuje užívateľovi možnosť modifikácie vizuálnych parametrov. Vizuálne zmeny je možné uložiť do formátu Visual PML, ktorý je takisto špecifikovaný v tejto práci. Aplikácia navyše umožňuje zobrazené stromy exportovať do rôznych vektorových a rastrových formátov.

Kľúčové slová: PML, vizualizácia, Pražský závislostný korpus, závislostný strom, zložkový strom

Title: PML Visualization
Author: Michal Novák
Department: Department of Software Engineering
Supervisor: Mgr. Václav Novák
Supervisor's email address: novak@ufal.mff.cuni.cz

Abstract:

The subject of this work is the application PMLViewer. This application is designed for visualization of files in PML format. It is able to draw dependency and constituency trees described in this format and in addition it offers a possibility to modify visual parameters. These visual changes can be stored in Visual PML format, which is also specified in this work. Furthermore, this application permits the export of displayed trees into various vector and raster image formats.

Keywords: PML, visualization, Prague dependency treebank, dependency tree, constituency tree

Úvod

Jazyk Prague Markup Language (PML) síce nie je zatiaľ príliš rozšírený, napriek tomu si vďaka jeho nesporným výhodám pri popise lingvistických dát získava stále väčšiu popularitu. Dáta, ktoré jazyk vo väčšine prípadov popisuje, sú závislostné alebo zložkové stromy. Tie je možné rôznorodo graficky znázorniť. Avšak tento jazyk sám osebe neumožňuje zachytiť vizuálnu stránku lingvistických dát. Na vizualizáciu sa zatiaľ používa aplikácia Tred[1], ktorá umožňuje aj editáciu PML. Tá na modifikáciu vzhľadu zobrazovaných dát (stromov) používa tzv. „stylesheety".

Hlavným cieľom tejto práce bolo vytvorenie aplikácie PMLViewer, ktorá k vizualizácii PML dokumentov pristupuje odlišným spôsobom ako Tred. Pri vytváraní tejto aplikácie som kládol dôraz najmä na jednoduchosť a priamočiarosť ovládania. PMLViewer umožňuje načítať PML dokument a následne upravovať jeho vzhľad. Výstupom môže byť dokument vo formáte Visual PML (VPML), ktorý v sebe na rozdiel od obyčajného PML formátu nesie aj vizuálnu informáciu. Ďalšou možnosťou výstupu je export do niekoľkých najbežnejšie používaných vektorových a rastrových obrázkových formátov.

Prvá kapitola je teoretickým úvodom do problematiky. Oboznamuje čitateľa s donedávna jedinou implementáciou jazyka PML – Pražským závislostným korpusom[2] (PDT[3]). Mojou prioritou pri vytváraní aplikácie PMLViewer bola schopnosť korektne zobraziť dáta práve z tohto korpusu a viaceré funkcie programu majú význam iba pre dáta z PDT. V tejto, teoretickej, časti bude stručne popísaná aj špecifikácia samotného formátu PML a jeho implementovaná podmnožina v aplikácii PMLViewer. Rovnako bude špecifikovaný formát VPML ako možnosť vizuálneho popisu PML dát.

Druhá kapitola je užívateľská dokumentácia k aplikácii PMLViewer, ktorá slúži ako návod na použitie aplikácie.

Tretia, najobsiahlejšia, kapitola ukazuje aplikáciu PMLViewer z pohľadu analytika a programátora. Bližšie vysvetľuje spôsob implementácie rôznej funkcionality. Pritom poukazuje na rôzne problémy alebo viaceré možnosti riešenia, z ktorými som sa pri implementácii stretol.

Posledná kapitola obsahuje porovnanie aplikácií PMLViewer a Tred, pričom sa poukazuje na výhody a nevýhody každej z nich.

Táto práca obsahuje niekoľko obrázkov a diagramov, ktoré sú prevzaté buď priamo z aplikácie PMLViewer, alebo boli vytvorené pomocou UML dizajnéra vývojového prostredia NetBeans 6.0.

---

[1] http://ufal.mff.cuni.cz/pdt2.0/doc/tools/tred/index.html

[2] http://ufal.mff.cuni.cz/pdt2.0/

[3] PDT – Prague Dependency Treebank

1 Teória

1.1 Pražský závislostný korpus (PDT)

PDT vznikol na **Ústave formálnej a aplikovanej lingvistiky**[4] a momentálne sa nachádza vo verzii 2.0. Tento korpus bol vyvíjaný s ohľadom na dve požiadavky:

- Prakticky na skutočných textoch ukázať správnosť a použiteľnosť Funkčne-generatívneho popisu, a tým nadviazať na pražskú lingvistickú tradíciu, konkrétne na prácu Pražského lingvistického krúžku.
- Vytvoriť dostatočne veľkú množinu, ktorú môžu špecializované nástroje využiť prostredníctvom metód strojového učenia na automatickú analýzu a generovanie jazykových dát.

Texty v PDT pochádzajú z týchto periodík: Lidové noviny, MF Dnes, Českomoravský profit, Vesmír [1]. Vďaka ich väčšej žánrovej rozmanitosti tento korpus reprezentuje reálny jazyk viac ako napr. **Penn Treebank**[5], ktorého primárnym zdrojom bol ekonomický denník The Wall Street Journal [2].

Dáta obsahujú okolo 2 milióny slov a sú anotované na 4 vrstvách. Tieto vrstvy sú medzi sebou previazané , avšak pre niektoré vety chýba korešpondujúci popis na zložitejších rovinách (a-rovine, t-rovine). Platí však pravidlo, že veta, ktorá je anotovaná na nejakej hladine, je anotovaná aj na všetkých nižších. Roviny v PDT, zotriedené od najnižšej, sú tieto:

1. Slovná, tzv. **w-rovina**. Na tejto vrstve sú jednotlivé zdrojové vety rozdelené na tokeny (slová, interpunkčné znamienka). W-rovina pokrýva celú databázu textov v PDT, čiže okolo 2 milióny slov.
2. Morfologická, tzv. **m-rovina**. Ku každému tokenu z w-roviny je na tejto hladine pridané morfologické informácie, predovšetkým morfologické zaradenie do kategórií a základný tvar slova. Takisto sú opravené prípadne pravopisné chyby, ktoré sa vyskytli na w-rovine. Tak sa môže stať, že k niektorému prvku m-roviny nie je priradený žiadny prvok w-roviny (napr. chýbajúca čiarka). Tento jav však nastáva iba príležitostne, preto m-rovina obsahuje približne 2 mil. slovných jednotiek.
3. Analytická, tzv. **a-rovina**. Táto vrstva zachytáva povrchovú syntaktickú štruktúru. Jednotky z morfologickej hladiny sú na a-rovine zapojené do závislostného stromu. Uzly tohto stromu sú usporiadané podľa vetného poriadku a spolu s hranami doplnené o ďalšie informácie. Na analytickej hladine je anotovaných už menej, 1,5 mil. slov.
4. Tektogramatická, tzv. **t-rovina**. Poslednou je vrstva hĺbkovej syntaxe. Príslušná veta je tu reprezentovaná rovnako ako v a-rovine závislostným stromom. Medzi uzlami tohto stromu a slovnými jednotkami z vety však už nie je bijektívne zobrazenie. Niektoré uzly sú vynechané

[4] http://ufal.mff.cuni.cz/

[5] http://www.cis.upenn.edu/~treebank/

(neplnovýznamové slová) a naopak iné uzly sú pridané (napr. zamlčaný vetný subjekt). Každému uzlu aj hrane je priradené množstvo atribútov (napr. odkaz do valenčného slovníka na valenčný rámec slovesa). V tektogramatickej rovine je zahrnutý aj popis aktuálneho členenia a koreferencií. T-rovina je zo všetkých rovín najzložitejšia, a preto je na nej anotovaných ešte menej dát ako na a-rovine, konkrétne 0,8 mil. slov.

Dáta z PDT vo verzii 1.0 boli popísané formátom CSTS[6], ktorý bol založený na SGML[7] (pôvodne bol vytvorený pre účely Českého národního korpusu[8]). Používanie SGML však prináša mnohé nevýhody. Hlavný problém je jeho obsiahlosť a neexistencia univerzálneho parsovacieho nástroja.

Z tohto dôvodu je vo verzii 2.0 primárnym formátom PDT na XML založený **Prague Markup Language (PML)**, ktorý je popísaný ďalej.

Súčasťou PDT je aj niekoľko nástrojov, ktoré zjednodušujú editáciu a názorne vizualizujú (Tred), sprostredkovávajú vyhľadávanie podľa rôznych kritérií (NetGraph[9]) atď. PMLViewer vzniká ako ďalší nástroj umožňujúci vizualizáciu, a to nielen dát z PDT ale všeobecne popisov stromov pomocou PML.

Pri popise PDT 2.0 som čerpal z [1].

1.1 Jazyk PML a práca aplikácie PMLViewer s ním

Pri popise jazyka PML som čerpal zo dvoch verzií špecifikácií [3] a [4], pričom technická správa [3] obsahuje navyše úvodnú motivačnú časť. Špecifikácia jazyka PML je v nej však iba vo verzii 1.0, zatiaľ čo zdroj [4] je špecifikácia PML verzie 1.1.

1.1.1 Motivácia pre vznik jazyka PML

Pri vytváraní druhej verzie PDT bolo nutné z vyššie spomenutých dôvodov zmeniť pôvodný formát uloženia dát. Nový formát mal spĺňať nasledujúce kritéria:

- možnosť anotácie vo viacerých autonómnych rovinách,
- možnosť referencií v rámci dokumentu, medzi rovinami, ale aj do externých dokumentov,
- schopnosť zachytiť lineárne aj štruktúrované dáta s možnosťou rekurzívneho zanorenia,
- schopnosť uložiť alternatívy, ktoré sa z dôvodu vágnej povahy jazyka často objavujú,
- zrozumiteľnosť a čitateľnosť pre človeka,
- rozšíriteľnosť,
- byť založený na XML, ktoré je jednoduché a existuje množstvo nástrojov na prácu s ním.

[6] https://wiki.ufal.ms.mff.cuni.cz/format-csts

[7] http://www.w3.org/MarkUp/SGML/

[8] http://ucnk.ff.cuni.cz/

[9] http://quest.ms.mff.cuni.cz/netgraph/

Tieto podmienky nedokázal uspokojiť žiaden z vtedy existujúcich formátov na popis lingvistických dát. Z toho dôvodu sa rozhodlo o deklarovaní nového formátu – Prague Markup Language (PML) [3].

1.1.2 Štruktúra jazyka PML

Jazyk PML na rozdiel od XHTML[10], DocBook[11] atď. nie je slovníkom s heslami, ktorých význam je už špecificky určený. Skôr sa jedná o systém na popis takéhoto slovníka [4]. Z toho dôvodu sa vytváranie PML dokumentu skladá z dvoch častí. Najprv je nutné vytvoriť tzv. **PML schému**[12], ktorá používa konštrukcie definované v norme jazyka PML. Táto slúži ako schéma (podobne ako XML Schema alebo Relax NG pre obyčajné XML dokumenty) pre iný súbor, tzv. **PML inštanciu**, ktorá už v sebe nesie konkrétne informácie.

Jedna schéma môže byť samozrejme použitá na viacero inštancií zároveň. Tak je to aj pri popise PDT 2.0. Štruktúra dát každej roviny (napr. analytickej) je vytvorená pomocou PML schémy (`adata_schema.xml`) a všetky dáta príslušnej vrstvy (napr. `sample0.a.gz`) vyhovujú tejto schéme.

Koreňovým elementom PML schémy je `pml_schema`. Jeho podelementy môžu byť referencie na iné súbory, označenie verzie, prípadne vysvetľujúci popis schémy. Najdôležitejším potomkom je však element `root`, ktorý v sebe zahrňuje už samotnú schému, podľa ktorej sa vytvárajú PML inštancie.

Každá PML inštancia takisto obsahuje časť, ktorej štruktúra je vždy rovnaká, je to tzv. hlavička PML inštancie a je uvedená elementom `head`. Obsahuje v sebe referencie na poddokumenty a odkaz na PML schému, ktorej daná inštancia prislúcha.

PML inštancia pozostáva z konštrukcií, ktorých predloha je určená v PML schéme prostredníctvom typov. Tieto typy je možné kategorizovať nasledovným spôsobom:

- abstraktné
  - atomické
  - komplexné
- konkrétne
  - anonymné
  - pomenované

Abstraktné atomické a komplexné dátové typy sú definované v špecifikácii PML. Tieto typy sú len akési kostry, úplný zmysel dostávajú až vtedy, keď sa dodefinujú v PML schéme, stanú sa z nich konkrétne typy. Stručne spomeniem ich názvy a akému účelu slúžia.

Atomické typy by sa dali zoradiť podľa šírky obmedzenia na ich hodnotu. Najviac obmedzeným je **typ `constant`**, ktorého jediná možná hodnota musí byť priamo definovaná v PML schéme. O čosi viac možností pre svoj obsah má

[10] http://www.w3.org/TR/xhtml1/

[11] http://www.docbook.org/

[12] V publikáciách [4] a [3] sa označuje aj ako PML aplikácia. Ja sa však v tejto práci budem držať termínu PML schéma, ktorý podľa mňa lepšie vystihuje charakter takéhoto súboru.

typ `choice`, ktorý v rámci svojej definície v schéme určí množinu hodnôt, ktoré môže nadobúdať. Najuniverzálnejším atomickým typom je však **`cdata`**. Môže nadobúdať akúkoľvek hodnotu, ktorá je obmedzená iba prítomnosťou atribútu `format`. Ten prípadne môže obmedziť hodnoty na nezáporné celé čísla, unikátne identifikátory, na odkazy smerujúce na takéto identifikátory alebo iným spôsobom (určeným v špecifikácii PML). Ako jediný z abstraktných typov už nemusí byť v PML schéme dodefinovaný.

Komplexné typy sú svojou podstatou kontajnery a majú zmysel tiež len vtedy, ak je definované, akým spôsobom má vyzerať ich obsah. Konštrukt v PML inštancii, ktorý zodpovedá takémuto typu, je možné naplniť hodnotami atomických, ako aj komplexných typov. Jednotlivé komplexné typy sa od seba odlišujú svojimi vlastnosťami, ktoré sa následne odzrkadľujú pri ich použití v PML inštancii.

Typ `structure` („**štruktúra**“) reprezentuje množinu členských atribútov, čiže dvojíc názov – hodnota, ktorá je veľmi podobná typu `struct` z jazyka C. Obsah „štruktúry“ je definovaný XML elementmi s názvom `member`. XML atribúty tohto elementu určujú mimo iného aj meno členského atribútu. Obsah členského atribútu môže byť takmer akýkoľvek iný typ (obmedzenia stanovuje špecifikácia PML), ale tento typ musí byť stanovený.

Ďalším typom je **`list`** („**zoznam**“) a plní úlohu homogénneho zoznamu. Typ jeho položky musí byť v PML schéme definovaný obsahom XML elementu `list` alebo referenciou na pomenovaný typ pomocou XML atribútu `type`. V PML inštancii sa položky „zoznamu“ označujú elementom `LM`. V PML schéme je možné určiť aj to, či má byť konkrétny „zoznam“ usporiadaný (pomocou XML atribútu `sorted`).

Podobnú funkciu ako „zoznam“ plní **typ `sequence`** („**sekvencia**“) s tým rozdielom, že jeho položky nemusia byť rovnakého typu. Typy obsahov, ktoré pri danej „sekvencii“ pripadajú do úvahy, je nutné zadefinovať pomocou podelementov s názvom `element`. V PML inštancii sa potom aplikácie týchto typov objavujú pod názvami definovanými v XML atribúte `name` XML elementu `element`. Súčasťou obsahu „sekvencie“ môže byť aj samotný text neuzavretý do žiadneho ďalšieho elementu. Poradie a počet opakovaní položiek sekvencie je určený XML atribútom `content_pattern` XML elementu `sequence`.

Formou podobný „zoznamu“, ale funkciou od neho odlišný, je **typ `alt`** („**alternatíva**“). Ide o typ, ktorého inštancie zvyčajne nadobúdajú jednu hodnotu, ale v niektorých prípadoch je povolených niekoľko alternatívnych hodnôt [4]. Podobne ako pri type `list` sú položky „alternatívy“ v PML inštancii označované XML elementom so špeciálnym názvom, v tomto prípade `AM`.

Posledným komplexným abstraktným typom je **`container`** („**kontajner**“). Jeho účel je podobný ako pri „štruktúre“ avšak oproti typu `structure` má „kontajner“ viacero obmedzení. Všetky jeho položky, okrem jednej, musia byť atomického typu (nie však typu `constant`). Tieto položky sú v PML schéme definované XML elementom `attribute` a v PML inštancii sa potom

reflektujú ako XML atribúty. Tá jediná položka, ktorá nemusí byť atomická, určuje priamy obsah „kontajnera“.

Doteraz spomenuté komplexné typy sú iba prostriedok na vytvorenie konkrétneho typu. Aplikovať je ich možno buď priamo v koreňovej vetve PML schémy ako anonymné typy alebo zadefinovaním pomenovaného typu pomocou elementu `type`. Takýto typ je následne možné použiť na všetkých miestach, kde sa vyžaduje nejaký typ, a to pomocou XML atribútu type zadaním jeho názvu. Miesta, kde je možné použiť pomenovaný typ, definuje špecifikácia PML (napr. pri členoch abstraktného typu `structure` je to XML atribút elementu `member`, pri „zozname“ je to XML atribút elementu `list` atď.).

S verziou 1.1 do PML prišli tzv. **modulárne schémy**. Tie umožňujú umiestniť pomenované typy do iného súboru, ale najmä dávajú možnosť takéto typy modifikovať. Ak potrebujeme v našej schéme nejaký konkrétny typ a podobný typ je už použitý v inej schéme, stačí nám do našej PML schémy tento typ vložiť pomocou elementu `import` a následne ho modifikovať prostredníctvom elementu `derive`. Importovanie pomenovaného typu je navyše možné podmieniť číslom revízie PML schémy, z ktorej sa importuje.

1.1.3 Role

Tak, ako bol jazyk PML popísaný doteraz, by bol popri DTD, XML Schema, Relax NG a iných iba ďalším variantom popisu XML dokumentov. To čo ho robí odlišným, sú tzv. **role**. Role sú ortogonálne k typom, čiže každý typ môže byť označený akoukoľvek rolou [4] (v skutočnosti to nie je pre všetky prípady pravda a existujú obmedzenia). Význam rolí je taký, že dodávajú PML schéme nejakú sémantickú informáciu. Hodnoty rolí jednotlivých typov tak môžu pomôcť automatickým nástrojom k „pochopeniu“ a spracovaniu PML dokumentu. Inak tomu nie je ani pri aplikácii PMLViewer.

Špecifikácia PML definuje nasledujúce role:

- `#TREES` – typ, ktorý táto rola označuje, predstavuje zoznam závislostných alebo zložkových stromov
- `#NODE` – označuje typ, ktorý predstavuje jeden uzol stromu
- `#CHILDNODES` – označuje typ, ktorý reprezentuje zoznam potomkov uzlu
- `#ID` – označuje jednoznačný identifikátor konštruktu
- `#KNIT` – typ označený touto rolou má byť nahradený konštruktom, na ktorý odkazuje jeho hodnota
- `#ORDER` – typ s touto rolou určuje poradové číslo uzlu
- `#HIDE` – štruktúra s touto rolou by mala byť skrytá

1.1.4 Vzhľad PML dokumentov podporovaných aplikáciou PMLViewer

Aplikácia PMLViewer je schopná otvoriť a spracovať inštancie PML, ktoré popisujú závislostné alebo zložkové stromy. Tieto súbory musia splňovať ešte požiadavku, že stromový vzťah rodičovský uzol – syn je v PML zaznamenaný prostredníctvom vnorených dátových štruktúr tak, ako v nasledujúcej ukážke:

```
<node>      <!-- parent -->
  ...
  <children>
    <LM>   <!--child 1-->
      <node>
        ...
      </node>
    </LM>
    <LM>   <!--child 2-->
      ...
    </LM>
  </children>
</node>
```

Na takýto dokument je možné korektne aplikovať PML role (`#TREES`, `#NODE`, `#CHILDNODES`). Ak v dokumente tieto role nemožno rozumne použiť, PMLViewer ho nedokáže načítať. Príklad takéhoto súboru je taký, ktorého stromové vrcholy sú popísané spoločne ako množina a ich vzájomné vzťahy sú popísané zas ako množina hrán s referenciou na začiatočný a koncový vrchol. Taký súbor by mohol vyzerať napríklad takto:

```
<nodes>
  <LM><node id="1">...</node></LM>
  <LM><node id="2">...</node></LM>
  <LM><node id="3">...</node></LM>
  ...
</nodes>
<edges>
  <LM><edge start="2" end="1">...</edge></LM>
  <LM><edge start="2" end="3">...</edge></LM>
</edges>
```

Ďalším podporovaným formátom je **Visual PML (VPML)**, ktorý som vytvoril špeciálne pre túto aplikáciu a jedná sa o PML dokument obohatený o vizuálne informácie (detailnejšie bude popísaný v časti 1.2).

Súbory PML ako aj VPML môžu byť skomprimované vo formáte gzip.[13]

1.1.5 Implementovaná podmnožina jazyka PML v aplikácii PMLViewer

Aplikáciu PMLViewer som vytváral s predpokladom, že načítanie dokumentu by malo prebehnúť čo najrýchlejšie. Z toho dôvodu, ale aj preto, lebo to nebolo cieľom tejto aplikácie, som zavrhol striktnú validáciu PML inštancie voči PML schéme a schémy voči norme. Rozhodol som sa ignorovať niektoré obmedzenia, a to tie, ktorých porušenie nespôsobí závažný problém pri spracovaní PML dokumentu. Konkrétne sú to tieto:

- XML atribút `format` pri type `cdata`
- XML atribút `content_pattern` typu `sequence`

Takisto som sa rozhodol zatiaľ nepodporovať modulárne schémy. Ako oficiálny nástroj k PML existuje skript `pml_simplify`[14], ktorý prevedie modulárnu schému na jednoduchú a tá už môže byť spracovaná aplikáciou PMLViewer.

[13] http://www.gzip.org/

[14] http://ufal.mff.cuni.cz/pdt2.0/tools/pml/pml_simplify

Zatiaľ nepodporované sú rovnako aj interné schémy, čiže umiestnenie PML schémy priamo v súbore PML inštancie. To súvisí so spôsobom spracovania PML inštancie v aplikácii PMLViewer (popísaný viac v kapitole 3.1.5.2), ktorý však bude do nasledujúcej verzie prehodnotený.

Rola #HIDE nie je v žiadnej súčasnej aplikácii jazyka PML použitá. Z toho dôvodu som sa rozhodol neimplementovať prácu s touto rolou, pri spracovaní PML dokumentu je ignorovaná.

1.2 Formát Visual PML (VPML)

VPML je formát vytvorený pre potreby aplikácie PMLViewer. Jeho úlohou bolo uchovávať vizuálne informácie o stromoch tak, aby bolo možné opätovne načítať vizuálne parametrizovaný strom.

Súbor typu VPML vznikne tak, že sa k PML inštancii pridajú vizuálne informácie. Tieto informácie sú vo forme elementov a atribútov patriacich do nasledujúceho menného priestoru (namespace):

```
http://ufal.mff.cuni.cz/pdt/pml/visual/
```

V dokumentoch generovaných aplikáciou PMLViewer je tomuto namespacu priradený prefix `vpml`. Po odstránení všetkého, čo náleží do tohto namespacu, zostane pôvodná PML inštancia.

Ďalej v texte budem značkou VPML označovať samotné vizuálne rozšírenie. Je však zrejmé, že formát VPML môže existovať iba v súčinnosti s prislúchajúcou PML inštanciou.

Štruktúra VPML je takáto:

- samostatná sekcia definície globálneho vizuálneho štýlu
- sekcia lokálnych nastavení (zakomponovaná vnútri popisu PML inštancie)

Obe sekcie zahŕňajú nastavenia pre uzly, hrany a atribúty uzlov. Špeciálne pre hrany a uzly je možnosť nastaviť spôsob ich zvýraznenia. Určia sa parametre, ktoré majú byť odlišné od normálneho stavu a ich hodnoty. Zvýraznený objekt potom vznikne prekrytím normálneho nastavenia tým zvýrazňovacím, pričom neprekryté parametre majú hodnoty ako v normálnom stave.

1.2.1 Zápis hodnôt špeciálnych typov

Na všetkých miestach, kde sa vyžaduje farba, je nutné ju zapisovať v tvare #RRGGBB, kde RR je hexadecimálne vyjadrenie podielu červenej zložky (podobne GG zelenej a BB modrej).

Štýl fontu musí nadobúdať tieto hodnoty: `0` – normálne, `1` –tučné, `2` – kurzíva, `3` – tučná kurzíva.

Štýl čiary sa musí zapisovať jednou z týchto hodnôt: `solid` – spojitá, `dashed` – čiarkovaná, `dash-dotted` – bodkočiarkovaná, `dotted` – bodkovaná.

Atribút typu boolean nadobúda hodnoty `true` alebo `false`.

Nakoniec v sekcii lokálnych nastavení je stav objektu určený pomocou hodnôt: `0` – globálne nastavenie, `1` – lokálne nastavenie.

1.2.2 Sekcia globálnych nastavení

Pri popise štruktúry sekcie **globálnych vizuálnych nastavení** budem používať DTD[15] výrazy na popis obsahu elementov. V globálnej sekcii sa môžu vyskytovať tieto elementy:

`global`

Tento element ohraničuje globálnu sekciu.

Obsah: `(attrs, edgeNormal, edgeHighlighted, nodeNormal, nodeHighlighted)`

`attrs`

Obsahuje globálne nastavenia atribútov uzlov. Na prvej hladine sú jednotlivé typy uzlov. Táto hierarchická štruktúra musí odpovedať štruktúre konkrétneho typu uzlu v PML schéme (pri vynechaní rekurzívnych atribútov).

Obsah: `(attr+)`

`attr`

Ak má prázdny obsah, svojimi XML atribútmi určuje globálne nastavenie nejakého atribútu uzlov. Ak obsah nie je prázdny, zaobaľuje v sebe viacero atribútov (presne podľa PML schémy).

Atribúty (XML):

`name`	názov atribútu uzlu
`visible`	viditeľnosť atribútu
`font`	názov fontu
`style`	štýl fontu
`size`	veľkosť písma v bodoch
`color`	farba písma

Obsah: `(attr*)`

`edgeNormal`

Globálne nastavenie vizuálnych atribútov hrany s normálnym zobrazením (nie je zvýraznená).

Atribúty (XML):

`edgeStyle`	štýl čiary, ktorou sa vykresľuje hrana
`edgeWidth`	šírka čiary, ktorou sa vykresľuje hrana
`edgeColor`	farba hrany
`edgeVisible`	true, ak má byť hrana viditeľná, inak false
`edgeUndercolor`	farba podfarbenia hrany (zobrazí sa okolo hrany polopriehľadne)

`edgeHighlighted`

[15] http://www.w3.org/TR/REC-xml/#dt-doctype

Nastavenie zvýrazňovania. Tento element obsahuje rovnaké XML atribúty ako `edgeNormal`. Niektoré z nich však môžu chýbať. Tie, ktoré sú prítomné, určujú, ktoré parametre zobrazenia sa budú pri zvýraznení meniť.

`nodeNormal`

Globálne nastavenie vizuálnych atribútov uzlu s normálnym zobrazením (nie je zvýraznený).

Atribúty (XML):

`nodeKind`	druh tvaru uzlu
`nodeSizeW`	šírka uzlu
`nodeSizeH`	výška uzlu
`nodeColor`	farba uzlu
`nodeVisible`	viditeľnosť uzlu
`nodeBorderColor`	farba okraju uzlu
`nodeBorderStyle`	štýl okrajovej čiary uzlu
`nodeBorderWidth`	hrúbka okrajovej čiary uzlu
`nodeUndercolor`	farba podfarbenia uzlu (zobrazí sa okolo uzlu polopriehľadne)

`nodeHighlighted`

Nastavenie zvýrazňovania uzlu. Platí to isté ako pre hranu, akurát sa v tomto elemente vyskytujú XML atribúty z elementu `nodeNormal`.

1.2.3 Sekcia lokálnych nastavení

Lokálne nastavenia vizualizácie určujú, či sa zobrazenie daného atribútu, uzlu alebo hrany riadi individuálne alebo globálne. Pri individuálnom charaktere konkretizuje parametre tohto zobrazenia.

U atribútov sa môžu takéto údaje nachádzať iba pri tých, ktoré sú atomického typu, čiže `cdata`, `choice` alebo `constant`. Realizujú sa pridaním XML atribútov z namespacu VPML do elementu daného typu v PML inštancii. Problémy vyvstávajú pri atribútoch, ktoré sú v PML inštancii realizované ako XML atribúty a pri type `constant`, ktorý nie je v PML inštancii realizovaný vôbec.

Prvý problém sa vyskytuje pri členoch `structure` typu, ktoré sú reprezentované ako XML atribúty alebo pri `attribute` zložkách typu `container`. Riešenie spočíva vo vložení prázdneho podelementu do elementu obsahujúceho daný atribút. Tento podelement má rovnaké meno ako atribút, ale je zaradený do namespacu VPML, aby sa zamedzilo konfliktom mena atribútu a podelementu a taktiež za účelom jednoduchého ignorovania v prípade potreby spracovania len čistého PML dokumentu.

Riešenie problému s typom `constant` je v niektorých prípadoch nejednoznačné. Ak je tento typ členom typu `structure`, do elementu – štruktúry sa podobne ako pri predchádzajúcom probléme vloží prázdny podelement s menom daného konštantného člena a namespacom VPML. Typ `list` (príp. `alt`) musí mať veľkosť svojho obsahu definovanú v inštancii pomocou elementov `LM` (príp. `AM`). Vizuálne dáta sa vložia do nich. Čo sa týka

typu container, jeho atribúty nemôžu byť konštantné a vizualizačné informácie sa za každých okolností pridávajú do elementu, ktorý reprezentuje typ container. Nejednoznačnosti nastávajú pri type sequence. Špecifikácia PML neupravuje, či konštantné členy sekvencie majú byť realizované aj v PML inštancii pomocou prázdných elementov s menami týchto členov. Ak by to však tak nebolo, počet a pozíciu členov typu constant v rámci sekvencie by musel upravovať XML atribút content_pattern. Počet výskytov nejakého atribútu v ňom ale môže byť definovaný intervalovo pomocou zástupných znakov *, + alebo ?. V takom prípade už počet konštánt s daným menom v sekvencií nie je možné určiť.

Lokálne vizuálne nastavenie uzlu je vždy v tom elemente, ktorý je charakterizovaný rolou #NODE.

Hranová vizualizácia je určená rovnako v elemente s rolou #NODE. Ten však musí byť potomkom elementu s rolou #CHILDNODES. Takže popis výzoru hrany z otca do syna je umiestnený v elemente syna.

Parametre lokálnej vizualizácie sú rovnaké ako parametre globálnej pre tie druhy objektov, ktoré neslúžia na zvýrazňovanie (attr, nodeNormal, edgeNormal). Ich prítomnosť však závisí od hodnoty parametru určujúceho štýl použitý na daný objekt (lokálny, globálny). Uzly a hrany obsahujú ešte k tomu parameter, či sú alebo nie sú zvýraznené. Pre uzol môže byť definovaná aj jeho pozícia.

Element uzlu

Vizuálne nastavenia pre daný uzol.

Atribúty (XML):

nodeState	druh štýlu použitý pre konkrétny uzol
nodeHighlighted	ak je true, uzol je zvýraznený
nodeX	x-ová súradnica pozície uzlu
nodeY	y-ová súradnica pozície uzlu
všetky parametre z nodeNormal	ak sú lokálne nastavenia aktívne

Element synovského uzlu

Vizuálne nastavenia pre hranu vedúcu do tohto uzlu z otcovského.

Atribúty (XML):

edgeState	druh štýlu použitý pre konkrétnu hranu
edgeHighlighted	ak je true, hrana je zvýraznená
všetky parametre z edgeNormal	ak sú lokálne nastavenia aktívne

Element atomického atribútu

Vizuálne nastavenia atribút.

Atribúty (XML):

attrState	druh štýlu použitý pre konkrétny atribút
všetky parametre z attr	ak sú lokálne nastavenia aktívne

2 Užívateľská dokumentácia

2.1 Inštalácia a adresárová štruktúra aplikácie

Aplikácia PMLViewer nevyžaduje žiadnu špeciálnu inštaláciu. Stačí iba skopírovať celý adresár PMLViewer z CD na požadované miesto.

Obsah adresárov adresárovej štruktúry aplikácie PMLViewer je nasledujúci:

`PMLViewer/dist` – samotná aplikácia a knižnice

`PMLViewer/doc` – dokumentácia, táto bakalárska práca

`PMLViewer/sample` – ukážkové dokumenty

`PMLViewer/schema` – potrebné PML schémy

`PMLViewer/src` – zdrojové súbory

PMLViewer je naprogramovaný v jazyku Java[16] a z toho dôvodu je platformovo nezávislý. Na spustenie aplikácie je však potrebné mať nainštalovaný Java Standard Edition Runtime Enviroment 6.[17]

2.2 Spustenie

PMLViewer je možné spustiť v dvoch režimoch:

1. režim grafického užívateľského rozhrania (GUI)
2. dávkový režim

Spustiť aplikáciu v GUI režime je jednoduché. Stačí dvojklik na súbor `PMLViewer.jar` alebo z príkazového riadku:

```
java -jar PMLViewer.jar
```

Takýmto spôsobom sa aplikácia spustí, ale neotvorí sa žiaden dokument. Na to, aby sa tak stalo hneď po štarte aplikácie, je potrebné pri spustení z príkazového riadku zadať cestu k otváranému dokumentu ako parameter:

```
java -jar PMLViewer.jar [cesta k dokumentu]
```

Dávkový režim umožňuje exportovať strom z PML dokumentu bez toho, aby sa muselo vykresliť grafické užívateľské rozhranie. Toto sa uskutoční pomocou prepínača -e:

```
java -jar PMLViewer.jar -e [cesta k dokumentu] [poradie
stromu v dokumente] [výstupný súbor]
```

Formát výstupného súboru, čiže výsledku exportu sa určí pomocou jeho koncovky. Formáty, do ktorých je možné exportovať, budú popísané v kapitole 2.6. Poradie stromu v dokumente sa počíta od 1.

Všetky možnosti spustenia aplikácie PMLViewer je možné vidieť po zadaní príkazu:

```
java -jar PMLViewer.jar -h
```

alebo

[16] http://java.sun.com/

[17] http://java.sun.com/javase/downloads/index.jsp

```
java -jar PMLViewer.jar --help
```

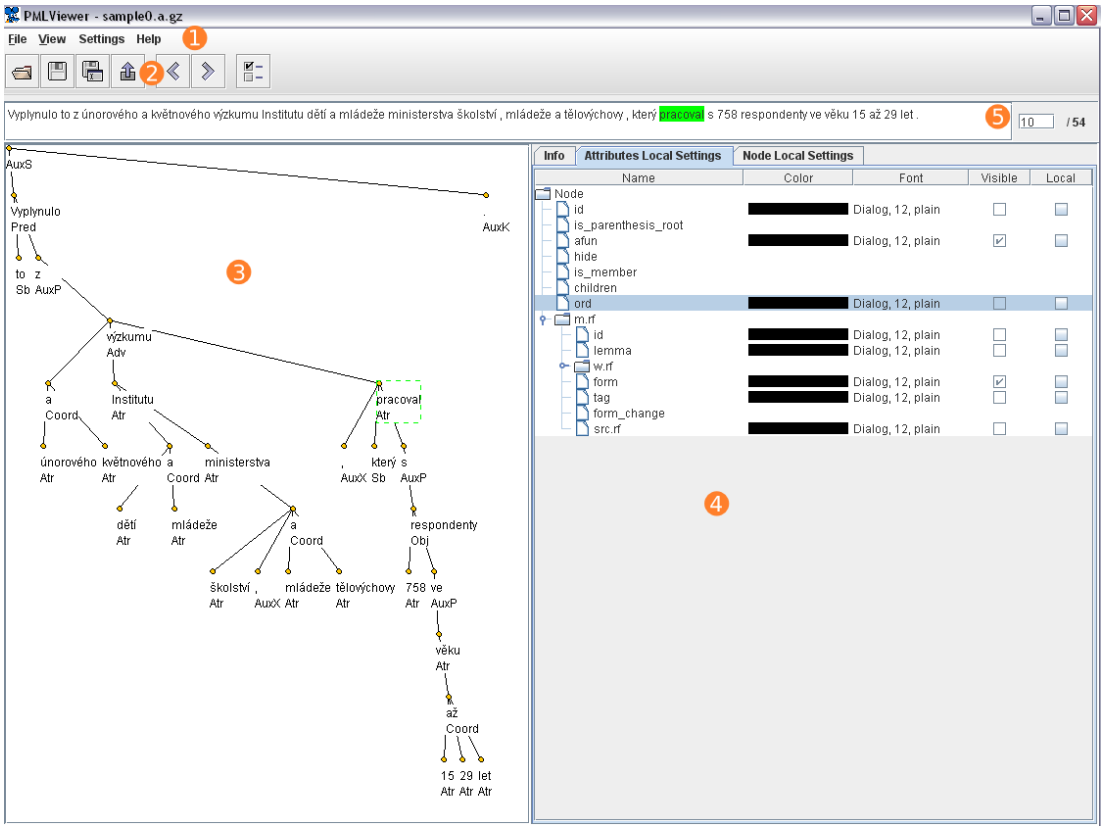
2.3 Hlavné okno aplikácie

V prípade spustenia aplikácie PMLViewer v režime GUI sa objaví hlavné okno (viď Obrázok 2.1). V jeho najvrchnejšej časti sa nachádza **hlavné menu**, odkiaľ je prístup ku veľkej časti funkcionality. Pod ním je umiestený **panel nástrojov** (toolbar), ktorý umožňuje zrýchlený prístup ku niektorým funkciám.

V zostávajúcej a zároveň najväčšej časti hlavného okna sa nachádza **panel otvoreného dokumentu**. Ak nie je žiaden dokument otvorený, je tento priestor prázdny. Inak je rozdelený do troch častí:

- **graf zobrazujúci strom**
- **kontextové menu**
- **panel zobrazenia vety**

Pomer veľkostí týchto častí je možné meniť ťahom za hraničnú priečku medzi nimi. Jednotlivé súčasti hlavného okna a najmä panela dokumentu budú popísané v ďalšom texte.



Obrázok 2.1 Hlavné okno aplikácie PMLViewer (1 – hlavné menu, 2 – panel nástrojov, 3 – graf zobrazujúci strom, 4 – kontextové menu, 5 – panel zobrazenia vety)

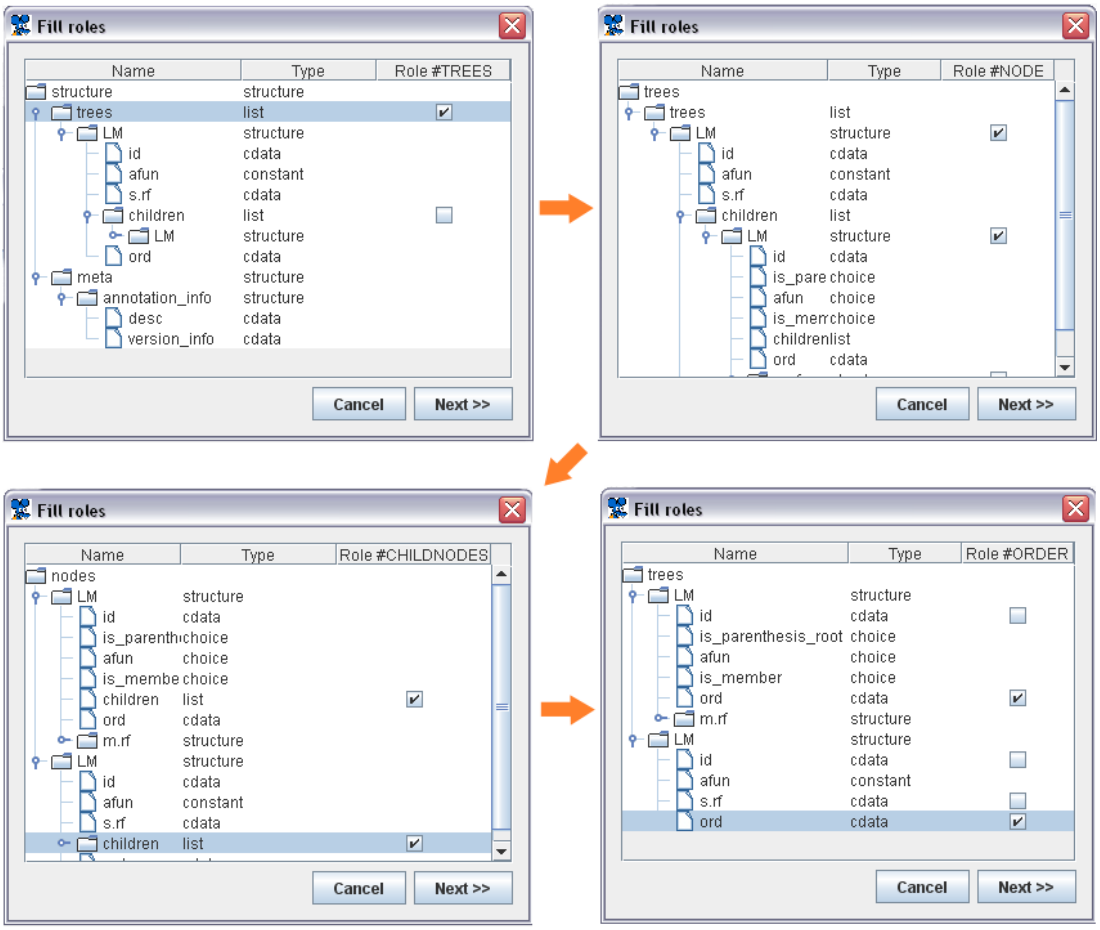
2.4 Otvorenie a spracovanie dokumentu

Ako som už spomenul, PML dokument je možné otvoriť priamo pri spustení aplikácie. Ďalšou možnosťou je použiť hlavné menu, konkrétne pomocou položky **File | Open...** Rovnakú akciu spôsobí použitie klávesovej skratky CTRL + O alebo kliknutie na príslušnú ikonu v paneli nástrojov. Výsledkom je, že sa zobrazí štandardné dialógové okno na otvorenie súboru. Po tom, čo

užívateľ zvolí súbor a potvrdí svoj výber, sa začne načítavať a spracovávať dokument.

Celé spracovanie PML dokumentu prebieha vo vlákne rozdielnom od toho, v ktorom prebieha odbavovanie prichádzajúcich udalostí. To znamená, že aplikácia je počas časovo náročného spracovania dokumentu schopná reagovať na požiadavky užívateľa (na obmedzenú množinu) prekreslením svojho užívateľského rozhrania, čiže nepôsobí dojmom, že tzv. „zamrzla“.

Počas spracovávania dokumentu sa zobrazí dialógové okno informujúce užívateľa o priebehu načítania a zároveň umožňujúce zrušiť celú akciu.



Obrázok 2.2 Sekvencia dialógových okien na doplnenie rolí

Ak nenastane žiadna chyba, ktorá môže byť spôsobená rôznymi okolnosťami, výsledkom je zobrazenie panela dokumentu adekvátneho spracovanej PML inštancii. Môže sa však stať, že v PML schéme prislúchajúcej k spracovávanému dokumentu nie sú štruktúry označené rolami, ktoré pomáhajú rozpoznať sémantiku dokumentu. V takom prípade je užívateľ počas načítavania o tomto informovaný a požiadaný o nápravu. Zobrazia sa postupne dialógové okná na doplnenie role `#TREES`, `#NODE`, `#CHILDNODES` a `#ORDER` (viď). Požiadavka na doplnenie role však nastane len vtedy, keď sa v príslušnej PML schéme nenachádza ani raz. Užívateľ môže doplnenie niektorej z rolí odmietnuť. Potom môže síce parsing pokračovať ďalej, ale iba dovtedy, dokiaľ nebude prítomnosť nejakej štruktúry s danou rolou

vyžadovaný. Tak je napríklad možné odmietnuť vyplnenie role `#ORDER`, čo nespôsobí chybu, no zobrazované stromy nebude možno usporiadať ako závislostné stromy.

Jedna inštancia aplikácie PMLViewer dovoľuje mať otvorený iba jeden dokument zároveň. V prípade požiadavky na súčasné otvorenie viacerých dokumentov je nutné spustiť ďalšie inštancie aplikácie.

2.5 Práca s dokumentom

2.5.1 Navigácia v dokumente

Po načítaní dokumentu sa zobrazí panel, ktorý obsahuje prvú vetu v dokumente a jej strom. Počet viet v dokumente je možné vidieť v pravej časti panelu zobrazenia vety. Na tom istom mieste sa dá priamo prejsť na akúkoľvek inú vetu zadaním jej poradového čísla do textového poľa. Postupný prechod o jednu vetu ďalej alebo naspäť umožňujú položky v hlavnom menu **View | Next tree** (respektíve **View | Previous tree**) alebo ikony v paneli nástrojov ako aj klávesové skratky ALT + $\rightarrow$ (ALT + $\leftarrow$).

2.5.2 Zobrazenie vety

Panel zobrazenia vety slúži najmä na vypísanie celej zobrazovanej vety. Toto je však k dispozícii iba pre aktuálne známe aplikácie formátu PML, t.j. pre dokumenty z a-vrstvy a t-vrstvy Pražského závislostného korpusu. Pri iných PML dokumentoch je tento panel neaktívny a prezentuje užívateľovi jedine informáciu o tom, že aktuálny dokument nie je touto funkcionalitou podporovaný. Neaktívny je aj pre súbory m-vrstvy, napriek tomu, že je ich možné aplikáciou PMLViewer načítať a ich štruktúra je známa. Problém s nimi je v tom, že nemajú definované poradie uzlov, čiže vypisovaná postupnosť slov by nemusela byť v správnom vetnom poriadku. Pre podporované formáty sa veta vypíše a pri označení nejakého uzlu v grafe sa zvýraznia slová, ktoré sa k danému uzlu vzťahujú. Výber hrany alebo viacerých objektov nespôsobí žiadne zvýraznenie v paneli zobrazenia vety. Takisto to samozrejme platí aj vtedy, keď nie je označený žiaden objekt.

2.5.3 Zobrazenie grafu, zmena jeho tvaru

Spodná časť hlavného okna je priečne rozdelená na dve. V tej ľavej sa nachádza graf, ktorý reprezentuje zobrazovaný strom. Výzor tohto grafu závisí od toho, či práve otvorený dokument je vo formáte PML alebo jeho VPML rozšírením. Vzhľad stromu po načítaní VPML dokumentu je presne taký, aký bol v čase jeho uloženia. Jediný rozdiel môže nastať pri usporiadaní zoznamu viditeľných atribútov, pretože informácia o tomto poradí sa do dokumentu VPML neukladá. Obyčajný PML dokument v sebe na rozdiel od jeho vizuálneho rozšírenia neobsahuje prezentačné informácie, preto sa aplikácia PMLViewer pri zobrazovaní jeho stromov implicitne správa nasledovným spôsobom:

- tvar grafu je určený prítomnosťou atribútu uzlu s rolou `#ORDER`. V prípade jej prítomnosti u každého uzlu sa graf zoradí ako závislostný strom, inak sa zoradí ako zložkový strom

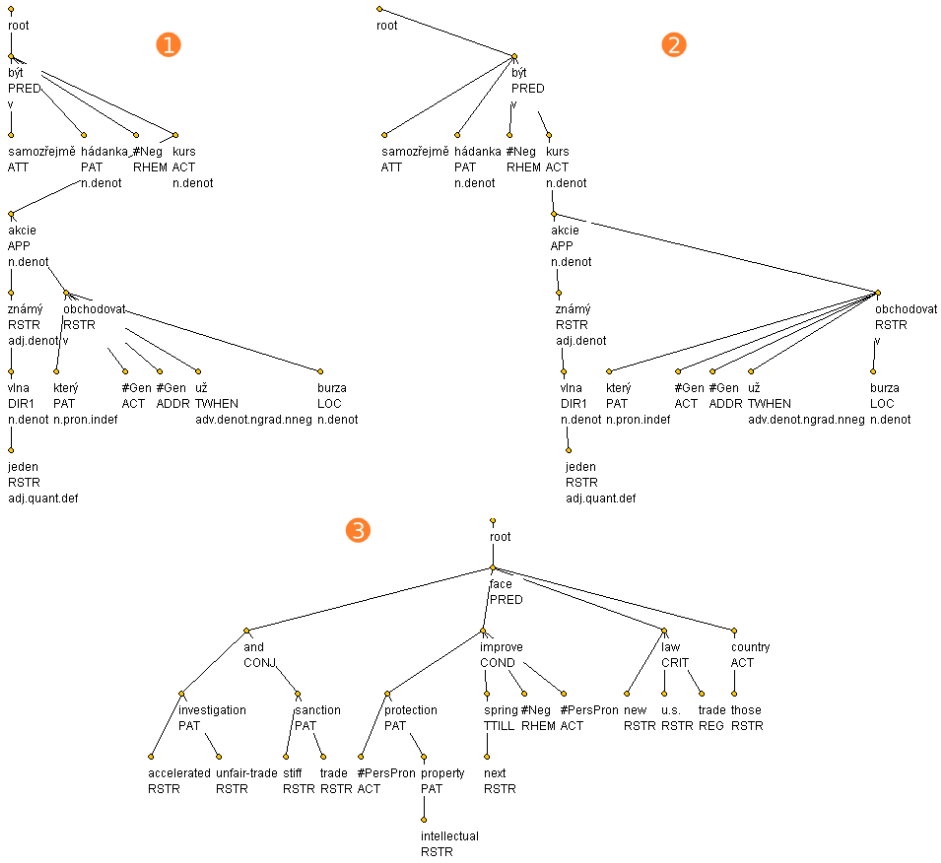
- parametre uzlu sú nasledovné: viditeľný, tvar kruhový, veľkosť 5x5px, oranžovej farby, okraj čierny spojitý šírky 1px, bez podfarbenia
- parametre hrany: viditeľná, čierna, spojitá šírky 1px, bez podfarbenia
- atribúty sú zobrazené čiernou farbou a fontom Dialog veľkosti 12pt a normálnym štýlom. Či je atribút viditeľný, závisí od typu dokumentu. Tu sú názvy viditeľných atribútov pre jednotlivé typy dokumentov:
  - dokument m-vrstvy: pre koreňový uzol žiadny atribút, pre ostatné uzly atribúty `form` a `tag`
  - dokument a-vrstvy: pre koreňový uzol atribút `afun`, pre ostatné uzly sú to atribúty `afun` a `m.rf/form`
  - dokument t-vrstvy: pre koreňový uzol atribút `nodetype`, pre ostatné uzly sú to atribúty `t_lemma`, `gram/sempos` a `functor`
  - pre ostatné dokumenty sa zobrazujú tie atribúty, ktoré majú rolu `#ID` (nemusia byť žiadne)

Tvar grafu sa dá meniť rôznymi spôsobmi. Najvariabilnejší z nich je pomocou myši. Každý uzol je možné označiť a premiestniť pretiahnutím myšou. Výber môže obsahovať aj niekoľko uzlov a hrán, s ktorými je potom možné myšou dohromady posúvať. Výber objektov sa vytvorí obdĺžnikovým vytýčením pomocou ťahu myšou so stlačeným ľavým tlačidlom alebo prostredníctvom kliknutia na objekty pri súčasne stlačenej klávese CTRL alebo SHIFT.

Ďalšou možnosťou zmeny tvaru grafu je programovo pomocou funkcií v hlavnom menu. Položky **View | Expand | Horizontally** (respektíve **Vertically**) a **View | Shrink | Horizontally** (respektíve **Vertically**) umožňujú zväčšiť, prípadne zmenšiť, horizontálne alebo vertikálne vzdialenosti medzi uzlami. Implicitne sa pri zväčšení táto vzdialenosť zdvojnásobuje, ale tento koeficient je možné zmeniť v nastaveniach (viac v časti 2.8).

Posledným spôsobom je automaticky meniť tvar celého grafu pomocou radenia uzlov. Ako som už spomenul, triedenia grafu sa aplikujú aj pri prvotnom zobrazení. V aplikácii PMLViewer rozlišujem 3 druhy triedenia (viď):

1. zoradenie **„bez poradia“** – uzly v rovnakej hĺbke sú vykreslené na rovnakej hladine, horizontálne sú na každej hladine zvlášť zoradené postupne od ľavého okraja za sebou doprava.
2. zoradenie **„ako závislostný strom“** – vertikálne umiestenie je zhodné s predchádzajúcim prípadom, horizontálne sú uzly usporiadané od ľavého okraja za sebou doprava. Rozdiel oproti predchádzajúcemu prípadu je v tom, že táto postupnosť je globálna pre všetky uzly. Na takéto zotriedenie je potrebná prítomnosť atribútu označeného rolou `#ORDER` pre každý uzol, je zdrojom pre vytvorenie postupnosti uzlov.
3. zoradenie **„ako zložkový strom“** – vertikálne usporiadanie je zhodné s predchádzajúcimi prípadmi. Pri horizontálnom umiestnení sa dodržuje pravidlo, že predok leží presne v strede medzi svojím prvým a posledným potomkom. Listy sú zoradené od ľavého okraja postupne za sebou doprava (môžu byť na rôznych hladinách).



Obrázok 2.3 Vzhľad stromov pri rôznych druhoch automatického triedenia uzlov (1 – metóda „bez poradia“, 2 – metóda „ako závislostný strom“, 3 – metóda „ako zložkový strom“)

Ak chceme zamedziť zmenám pozície uzlov, poslúži nám na to zaškrtávacia položka **View | Fixed nodes** v hlavnom menu.

2.5.4 Kontextové menu a jeho vplyv na zobrazenie grafu

Kontextové menu sa nachádza v pravej časti panela dokumentu. Skladá sa z niekoľkých záložiek, ktorých počet a obsah závisí od označeného objektu v grafe:

- označený nie je žiaden objekt, panel má tri záložky:
  - **globálne vizuálne nastavenie atribútov**
  - **globálne vizuálne nastavenie uzlu**
  - **globálne vizuálne nastavenie hrany**
- označený je uzol, panel má tri záložky:
  - **informácia o hodnotách atribútov**
  - **lokálne vizuálne nastavenie atribútov**
  - **lokálne vizuálne nastavenie uzlu**
- označená je hrana, panel má jednu záložku:
  - **lokálne vizuálne nastavenie hrany**

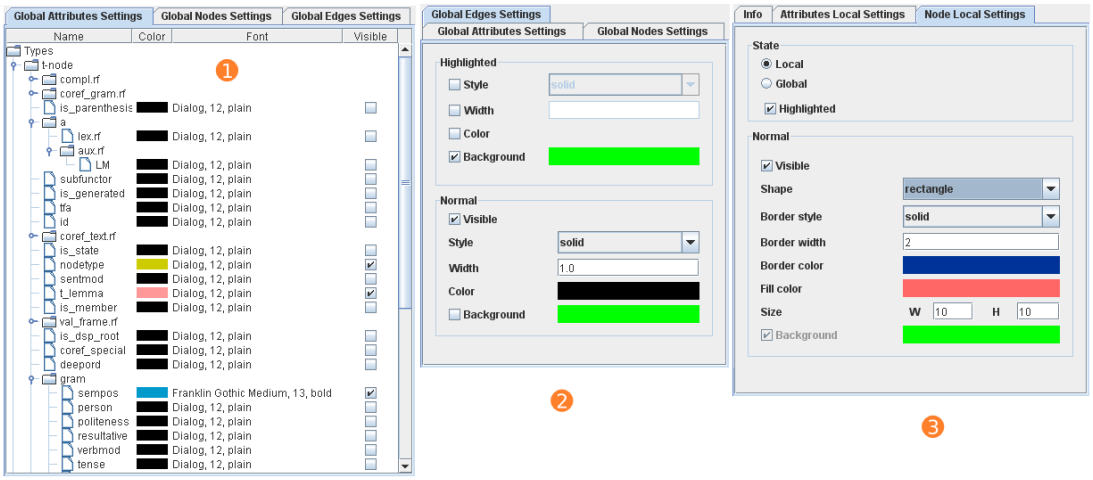
Aby bolo možné nastaviť všetko potrebné pre atribúty a zároveň zachovať hierarchický charakter atribútovej štruktúry, prebieha zmena parametrov atribútov pomocou hierarchickej tabuľky. Na vrchu hierarchie tejto tabuľky sú rôzne typy uzlov a až potom ich atribútové štruktúry, pretože v jednej PML

schéme môže byť viacero konštruktov označených rolou `#NODE` a tieto môžu mať rôznu atribútovú štruktúru. Pre globálne nastavenia je v tabuľke možné pre každý atribút zvoliť jeho viditeľnosť, farbu a font, u lokálneho ešte indikátor toho, či sa má daný atribút správať podľa globálneho alebo lokálneho nastavenia. V lokálnom vizuálnom nastavení atribútov je zamedzené meniť parametre, pokiaľ nie je tento indikátor nastavený na lokálne nastavenie. V takom prípade sa na mieste nemodifikovateľných parametrov zobrazujú informácie z globálneho nastavenia. Pokus o zmenu farby vyvolá zobrazenie dialógového okna na výber farby (podobne pre font). Poradie viditeľných atribútov v grafe závisí od času pridania, najnovšie zviditeľnené atribúty sa pridávajú na spodok zoznamu.

Hierarchická tabuľka atribútov a ich hodnôt má iba informatívny charakter, zmenu nevizuálnych hodnôt aplikácia PMLViewer neumožňuje.

Panel globálnych vizuálnych nastavení hrán alebo uzlov sa skladá z dvoch častí. Tá spodná umožňuje meniť parametre normálneho zobrazenia objektov (nezvýrazneného). Vrchná časť naopak určuje, zmenou ktorých parametrov sa bude uskutočňovať zvýraznenie a akú hodnotu tie parametre budú nadobúdať. Pre uzol sú k dispozícii tieto vizuálne parametre: viditeľnosť, veľkosť, tvar, farba a podfarbenie uzlu, hrúbka, štýl a farba jeho okraja. Pre hranu sú to nasledovné parametre: viditeľnosť, hrúbka, štýl, farba a podfarbenie hrany.

Možnosti lokálnych vizuálnych nastavení hrany alebo uzlu sú takmer rovnaké ako tých globálnych. Rozdiel je, podobne ako pre atribúty, v možnosti určiť, či sa označený objekt bude správať podľa lokálneho alebo globálneho nastavenia. Ak je aktívne globálne nastavenie, užívateľovi je zamedzené meniť lokálne nastavenie. Ďalšou možnosťou na tomto paneli je zvýraznenie objektu. Po tejto akcii sa aktuálne nastavenie parametrov (buď globálne alebo lokálne) prekryje maskou zvýraznenia, čiže parametre vybrané v globálnom nastavení zvýraznenia prepíšu tie pôvodné, ostatné ostanú nezmenené. Tak ako nie je možné meniť parametre v lokálnom nastavení, ak je aktívne globálne, nie je možné meniť v lokálnom nastavení zvýrazneného objektu ani tie parametre, ktoré určujú zvýraznenie.



Obrázok 2.4 Niektoré záložky kontextového menu (1 – globálne nastavenie atribútov, 2 – globálne nastavenie hrany, 3 – lokálne nastavenie uzlu)

Všetky zmeny v kontextovom menu sa okamžite odzrkadľujú v grafe. Ak užívateľ skryje hranu, táto v grafe nezmizne, ale zobrazí sa ako sivá čiarkovaná. Rovnako ak sa skryje uzol aj všetky jeho atribúty, v grafe sa zobrazí malý sivý čiarkovaný štvorec. Funguje to takto za tým účelom, aby užívateľ mal možnosť takýto skrytý objekt označiť a prípadne mu parametre zmeniť. Pri exporte však už takéto skryté objekty vo výsledku miznú úplne.

2.6 Export grafu

Veľkou výhodou aplikácie PMLViewer je možnosť exportu grafu zobrazujúceho lingvistický strom. Aplikácia ponúka export do nasledujúcich vektorových formátov:

- Scalable Vector Graphics (SVG)[18]
- Portable Document Format (PDF)[19]
- Encapsulated PostScript (EPS)[20]

Takisto umožňuje export do nasledovných rastrových formátov:

- JPEG[21]
- Portable Network Graphics (PNG)[22]

Prístup k tejto funkcionalite je cez položku **File | Export** hlavného menu ako aj prostredníctvom ikony v paneli nástrojov. V hlavnom menu sa nachádza zoznam položiek exportu do formátov, ktoré rozdeľuje čiara na vrchné vektorové a spodné rastrové. Kliknutím na niektorú z položiek sa zobrazí dialógové okno exportu, kde je prednastavený formát, do ktorého užívateľ chcel exportovať. Na tomto mieste je však ešte možné formát zmeniť. Užívateľ si vyberie, kam chce výsledok exportu uložiť. Ak meno súboru zadá aj s koncovkou prislúchajúcou danému formátu, súbor sa uloží pod týmto menom. Inak sa k názvu automaticky správna koncovka pripojí. Po kliknutí na tlačidlo export sa buď zaháji export, o ktorého priebehu informuje dialógové okno (v prípade rastrových formátov), alebo sa zobrazí ešte jedno okno, kde užívateľ môže upraviť parametre exportu (u vektorových formátov). Pre každý formát toto okno vyzerá inak, pretože každý má rozdielne parametre. Viac o týchto parametroch je popísané v špecifikáciách jednotlivých formátov. Ak nemá užívateľ záujem nič meniť a chce zachovať implicitné nastavenia, stačí už iba potvrdiť tlačidlom OK.

2.7 Uloženie dokumentu

Po vizuálnych zmenách dokumentu PML môže užívateľ tieto zmeny uchovať. Aplikácia PMLViewer umožňuje uložiť vizuálne upravený dokument v špeciálnom formáte VPML, ktorý je rozšírením PML (viac o formáte VPML v kapitole 1.2). Na rozdiel od exportu sa uchová kompletný dokument a je možné ho kedykoľvek opätovne načítať a pracovať s ním.

---

[18] http://www.w3.org/Graphics/SVG/

[19] http://www.adobe.com/devnet/pdf/pdf_reference.html

[20] http://partners.adobe.com/public/developer/en/ps/5002.EPSF_Spec.pdf

[21] http://www.jpeg.org/jpeg/index.html

[22] http://www.libpng.org/pub/png/

Možnosť uloženia sa nachádza v hlavnom menu pod položkami **File | Save** a **File | Save as...** Rozdiel medzi týmito dvoma voľbami je badateľný iba pri VPML dokumentoch. Vtedy prvý variant ukladá do toho istého súboru, zatiaľ čo druhý ponúkne užívateľovi výber umiestnenia a spôsobu uloženia. Naopak, pri obyčajných PML dokumentoch musí byť užívateľ za každých okolností informovaný a požiadaný o to, aby zadal potrebné údaje na uloženie súboru. Ak však takýto PML dokument niekde uloží, aktuálne zobrazovaným dokumentom v aplikácii sa stane ten uložený, ktorý obsahuje vizuálne údaje. Dôsledkom toho položka File | Save zmení svoju funkciu a od toho momentu sa pomocou nej ukladá vždy do rovnakého súboru (až dokým užívateľ neotvorí nový PML dokument).

Ako som už spomenul, pred zahájením ukladania sa môže užívateľovi zobraziť okno na výber cesty k súboru, do ktorého sa bude ukladať, a spôsobu uloženia. Jedným spôsobom je vytvorenie VPML dokumentu, tým druhým je ešte jeho následná komprimácia do formátu gzip. Výsledkom teda bude súbor s koncovkou `.vpml` alebo `.gz`. Uloženie prebehne po potvrdení kliknutím na tlačidlo Save.

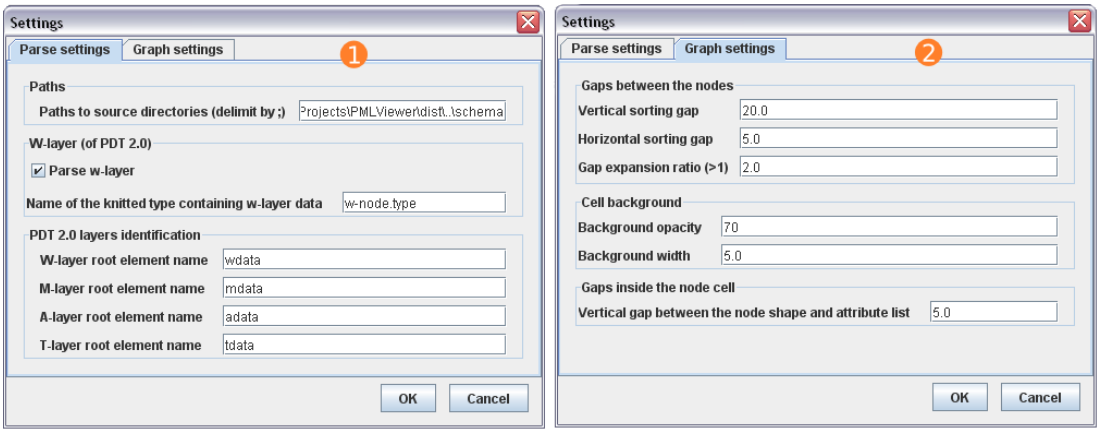
2.8 Nastavenia aplikácie

Za položkou **Settings** v hlavnom menu sa skrývajú nastavenia aplikácie. Tieto nesúvisia konkrétne iba s nejakým dokumentom, sú pre všetky spracovávané súbory rovnaké. Skladajú sa z nastavení parsingu PML dokumentu a z grafickým nastavení. Oba druhy majú v hlavnom menu svoju vlastnú položku, avšak po kliknutí na akúkoľvek z nich sa otvorí dialógové okno spoločné pre všetky nastavenia (viď Obrázok 2.5). V ňom sú jednotlivé druhy nastavení vyčlenené záložkami, takže nie je problém sa rýchlo dostať k inému typu nastavení. Toto okno sa zobrazí aj po kliknutí na príslušnú ikonu v paneli nástrojov.

Nastavenia spracovania dokumentu sa týkajú hlavne rozpoznávania dokumentov z PDT 2.0. V tejto časti je možné pre každú vrstvu Pražského závislostného korpusu zmeniť názov identifikátoru, ktorý by sa mal zhodovať s názvom koreňového elementu dokumentov tej danej vrstvy. Rozpoznanie dokumentov z PDT 2.0 je dôležité pri implicitnom zobrazení viditeľných atribútov a pri korektnom naplnení panela vety. Ďalšou zaujímavou funkciou pre tento jazykový korpus je možnosť zamedziť spracovaniu w-vrstvy. Ak je táto možnosť aktivovaná, počas spracovania PML inštancie z PDT sa príslušný poddokument w-vrstvy vôbec nepoužije a zabráni sa akémukoľvek pokusu o vkladanie dát z tejto vrstvy pomocou role `#KNIT`, užívateľovi sa zobrazia iba názvy referencií. Takéto chovanie trochu urýchli spracovanie PML dokumentu. Dôležitou súčasťou je možnosť nastavenia ciest k adresárom so súbormi, ktoré sú počas spracovania vyžadované (PML schémy, poddokumenty). Jednotlivé cesty sa oddeľujú znakom „;“. V dokumentoch z PDT 2.0 sú cesty k požadovaným súborom zadané iba názvom súboru, čiže tie musia ležať v rovnakom adresári ako nadriadený dokument. Ak sú napríklad schémy umiestnené na inom mieste v adresárovej štruktúre, je nutné do požadovaného poľa v dialógovom okne nastavení aplikácie túto cestu pridať. V opačnom

prípade spracovanie zlyhá. Pri úvodnom spustení aplikácie je pridaná cesta k adresáru `schema` v adresárovej štruktúre aplikácie PMLViewer, ktorý obsahuje všetky potrebné PML schémy. Zmeny nastavení parsingu sa prejavia pri ďalšom otváranom dokumente.

Záložka grafických nastavení pozostáva z možnosti upraviť parametre rozmiestňovania uzlov pri triedení, čiže veľkosť vertikálnej a horizontálnej medzery medzi susednými uzlami. Ďalej je možné zmeniť koeficient, pomocou ktorého sa vypočítavajú nové vzdialenosti medzi uzlami pri automatickom rozširovaní alebo zužovaní (táto hodnota nesmie byť menšia ako 1). Tieto zmeny sa prejavia pri nasledujúcom použití spomínaných funkcií. Rozmiestnenia sa týka aj parameter zvislej vzdialenosti medzi symbolom uzlu a jeho viditeľnými atribútmi. Ten je rovnako možné v grafických nastaveniach zmeniť. Nakoniec záložka grafických nastavení ponúka zmenu parametrov týkajúcich sa podfarbenia objektov. Tu je možné nastaviť mieru nepriehľadnosti a šírku podfarbenia. Zmeny troch posledne spomínaných parametrov sa v zobrazovanom grafe prejavia okamžite po zavretí dialógového okna nastavení tlačidlom OK.



Obrázok 2.5 Panely nastavení aplikácie (1 – nastavenia spracovania dokumentu, 2 – grafické nastavenia)

3 Implementácia, varianty a problémy

Program PMLViewer bol napísaný v jazyku Java. Zdrojový kód som vytváral vo vývojovom prostredí Netbeans[23] vo verziách 5.0, 5.5 a nakoniec 6.0. Tento vývojový nástroj som si vybral hlavne preto, lebo jeho súčasťou je dizajnér grafického užívateľského rozhrania pre knižnicu Swing[24]. Ten mi urýchlil návrh grafických prvkov.

3.1 Načítanie PML dokumentu

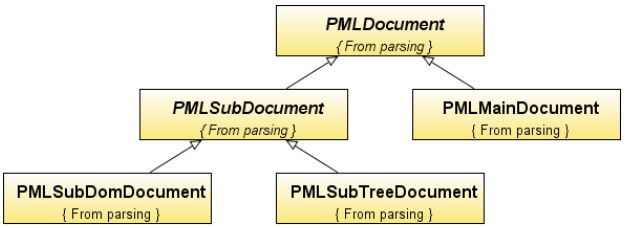
Toto je podstatná a, povedal by som, najdôležitejšia časť aplikácie PMLViewer. Zároveň je aj časovo najnáročnejšia. Ak by sa operácia načítania dokumentu spúšťala priamo v tzv. event-dispatching vlákne (ED vlákne), počas jej vykonávania by sa nemohli spracovávať udalosti, na ktoré sa reaguje zväčša prekreslením časti užívateľského prostredia. Vzhľadom ku dĺžke doby spracovania dokumentu by táto strata reakcií bola zjavná. Z toho dôvodu som sa rozhodol operáciu načítania (ako uvidíme ďalej, tak aj uloženia a exportu) presunúť do iného vlákna, a tým zamedziť zablokovaniu odozvy aplikácie. Na tento účel mi poslúžila štandartná trieda Javy 6 SE `SwingWorker`. Tá sama vytvorí iné vlákno, v ňom spustí dlhotrvajúcu akciu a nakoniec o jej skončení notifikuje zavolaním metódy, ako aj vyvolaním udalosti už v ED vlákne. Počas priebehu dlhotrvajúcej akcie môže takisto prostredníctvom udalostí v ED vlákne oboznamovať o svojom stave a pomocou nejakého grafického komponentu o tomto informovať aj užívateľa.

V aplikácii PMLViewer slúži na uskutočnenie popísanej funkcionality trieda `OpenProvider` (potomok triedy `SwingWorker`). Pred jej spustením sa zobrazí okno, ktoré počas celého priebehu užívateľa informuje o stave načítania. To je v ED vlákne notifikované práve udalosťami vyvolanými z vlákna dlhotrvajúcej akcie. Po ukončení načítania sa zobrazí výsledok v podobe panela dokumentu, eventuálne sa zobrazí chybová správa.

Celý proces načítania, parsingu a vytvorenia špeciálnej dátovej reprezentácie sa spustí vytvorením inštancie (zavolaním konštruktora s parametrom typu `File`) triedy `PMLMainDocument`. Táto trieda reprezentuje práve hlavný spracovávaný dokument. Je podedená od triedy `PMLDocument`, ktorá je abstraktným predkom rovnako abstraktnej triedy `PMLSubDocument`. Tá reprezentuje dokument referencovaný z hlavného PML súboru a rozširujú ju ďalšie dve triedy: `PMLSubDomDocument` a `PMLSubTreeDocument`. Bližší popis týchto tried a rozdielu medzi nimi je popísaný v časti 3.1.2.

[23] http://www.netbeans.org/

[24] http://java.sun.com/docs/books/tutorial/uiswing/index.html



Obrázok 3.1 Diagram dedičnosti tried reprezentujúcich dokumenty

Spracovanie hlavného dokumentu prebieha v piatich fázach, ktorých poradie prevedenia sa nemôže zmeniť, pretože vstup neskoršej fázy závisí od výstupu tej skoršej. Sú to:

1. parsing do JDOM[25] stromu
   a. PML inštancie
   b. PML schémy
2. spracovanie referencovaných PML dokumentov (ďalej len „**poddokumentov**“)
3. spracovanie PML schémy, ktorej daný dokument odpovedá
4. načítanie alebo vytvorenie globálnych vizuálnych informácií
5. samotné spracovanie PML inštancie, vytvorenie špecifickej dátovej reprezentácie dokumentu

V nasledujúcom texte každú z týchto častí samostatne popíšem.

3.1.1 XML parsing do JDOM stromov

Ako som už spomínal, formát PML je založený na XML. Preto je možné na jeho parsovanie použiť štandardné nástroje. Existujú tri rozdielne dátové modely XML: udalosťami riadený model (SAX[26]), stromová reprezentácia (DOM[27]) a objektová stromová reprezentácia (JDOM) [5, s. 59]. Napriek tomu, že vo väčšine prípadov PMLViewer pristupuje k XML sekvenčne, vyžaduje sa pri parsovaní PML z dôvodu interných referencií aj náhodný prístup (napr. definovanie a používanie vlastných typov). Preto som sa rozhodol pre reprezentáciu stromom.

Pre Javu existujú viaceré knižnice umožňujúce vytvorenie stromového modelu z XML. Ja som sa rozhodol do tohto projektu zvoliť už spomenutú objektovú knižnicu JDOM, s ktorou je intuitívna a jednoduchá manipulácia. Umožňuje tak postupný prechod štruktúrou dokumentu, ako aj dotazy v jazyku XPath 1.0.[28]

XML parsing v aplikácii PMLViewer spočíva najprv v spracovaní PML inštancie (čiže otváraného súboru) a následne prislúchajúcej PML schémy. Ak je nejaký zo súborov skomprimovaný do formátu gzip, je pomocou štandardných nástrojov pre prácu s gzip archívmi v Jave rozbalený a ďalej sa už postupuje rovnakým spôsobom.

[25] http://www.jdom.org/

[26] http://www.saxproject.org/

[27] http://www.w3.org/DOM/

[28] http://www.w3.org/TR/xpath

Korektný XML parsing je nutnou podmienkou pre ďalšie fázy spracovania dokumentu.

3.1.2 Spracovanie poddokumentov

Za poddokumenty sa považujú iba tie, ktoré sú určené v PML schéme rodičovského dokumentu pomocou elementu `reference`. Názvy týchto referencií sú konfrontované s obsahom elementu `references` v PML inštancii, aby ich prienik bol nakoniec množinou, z ktorej sa pre každú referenciu vytvorí potomok objektu `PMLSubDocument`, a tým sa spracuje príslušný poddokument.

Ak pri spracovaní nejakého poddokumentu nastane chyba, neznamená to ešte okamžité ukončenie celého procesu načítavania. Chybový alebo na danom mieste neexistujúci dokument nemusí byť použitý pri vytváraní dátových štruktúr. Môže slúžiť napr. na vydolovanie informácií potrebných na zobrazenie celej vety, ktorá je pomocou PML anotovaná (tak je to u poddokumentov a-vrstvy a m-vrstvy pri načítavaní dokumentu t-vrstvy). Preto sa takto spôsobené chyby ošetrujú až na mieste použitia.

Abstraktnú triedu `PMLSubDocument` rozširujú triedy `PMLSubDomDocument` a `PMLSubTreeDocument`. Hľadisko, podľa ktorého sa rozhoduje, aký z týchto dvoch objektov pre danú referenciu na poddokument vytvoriť, je obsah atribútu `readas` elementu `reference` v PML schéme – `trees` alebo `dom`. Na určenie rozdielu medzi týmito dvoma typmi som sa rozhodol použiť pravidlo: ak sa má poddokument čítať ako stromy (hodnota `trees`), spracuj aj jeho poddokumenty; naopak, ak sa má čítať ako DOM, nespracuj jeho poddokumenty. Je pravda, že interpretácia tohto atribútu sa mierne odlišuje od špecifikácie PML, ale je to zapríčinené iným spôsobom spracovania hlavného dokumentu voči poddokumentu. Poddokument sa z dôvodu rýchlosti nikdy nespracováva tak komplexne ako hlavný dokument, použije sa iba vtedy, keď je to potrebné (pri spájaní pomocou role `#KNIT`). Na príklade schémy a-vrstvy PDT 2.0 je možné ukázať, že všetko, čo je nutné k spracovaniu hlavného dokumentu je definované v jeho schéme (alebo importované pomocou modulárnych schém). Z toho dôvodu by táto interpretácia urýchľujúca spracovanie nemala spôsobovať problémy.

Načítanie poddokumentu je podobne ako u hlavného dokumentu rozdelené do niekoľkých fáz:

1. parsing do JDOM stromu
    a. PML inštancie
    b. PML schémy
2. rekurzívne spracovanie referencovaných PML dokumentov (v závislosti na vyššie spomenutom type)
3. samotné spracovanie PML inštancie, vytvorenie slovníka odkazov do dokumentu

Prvé dve fázy prebiehajú rovnako ako pri hlavnom dokumente.

Úlohou poslednej časti načítania poddokumentu je vytvoriť slovník (mapu) obsahujúcu odkazy na dátové štruktúry označené jednoznačnými identifikátormi. Táto mapa je indexovaná práve týmito identifikátormi.

Pomocou slovníka je možné do hlavného dokumentu pripájať náležiace štruktúry z iných vrstiev na tých miestach, kde sa objaví rola `#KNIT`. Pre položku typu `cdata`, ktorá tvorí obsah typu `list` alebo `alt`, sa v súlade so špecifikáciou PML rola `#ID` nedeteguje. To isté platí pre priamy obsah typu `container`, kde neexistuje spôsob ako ho tou rolou označiť (samozrejme sa to netýka obsahu atribútov typu `container`).

3.1.3 Spracovanie schémy

Ešte pred zahájením spracovania samotnej PML inštancie je potrebné prejsť schému, zistiť, či je v nej všetko potrebné definované a ako sekundárny produkt vytvoriť štruktúru, ktorá sa neskôr doplní o vizuálne údaje a použije ako globálne vizuálne nastavenie atribútov.

Na toto som vytvoril triedu `PMLSchemaParser`, ktorá po zavolaní metódy `parse` vráti štruktúru pripravenú na ďalšiu fázu. Táto metóda pri svojom vykonávaní prechádza JDOM stromom schémy, pričom kontroluje prítomnosť štyroch rolí (`#TREES`, `#NODE`, `#CHILDNODES`, `#ORDER`). Pre každú rolu stačí, aby bola minimálne na jednom mieste použitá. Ak nie je, užívateľ je informovaný dialógovým oknom na doplnenie informácií (viď. časť 2.4). Neprítomnosť rolí v schéme sa v súčinnosti s užívateľom rieši v nasledujúcom poradí:

1. ak nie je žiadna štruktúra označená rolou `#TREES`, na užívateľský výber sa zobrazí strom celej schémy
2. ak nie je nič označené `#NODE` rolou, užívateľ rolu vyberá v zozname stromov
3. v prípade, že sa v schéme nenachádza rola `#CHILDNODES`, na výber sa zobrazí zoznam rôznych druhov uzlov pomenovaných atribútom `name` (viď nižšie)
4. pri chýbajúcej `#ORDER` role sa zobrazí v dialógovom okne takisto zoznam druhov uzlov

Na to, aby sa mohol zobraziť model schémy pre užívateľský výber, je potrebné pri prechádzaní stromom vytvárať špeciálnu štruktúru – mapu objektov typu `PMLSchemaAttribute`, ktorý je podedený od typu `PMLAttribute` (jednoduchá trieda reprezentujúca dvojicu meno – hodnota). Ako hodnota (členská premenná `value`) môže byť akýkoľvek objekt, takže aj opäť mapa. Takýmto spôsobom sa vytvorí hierarchický strom celej schémy.

Trieda `PMLSchemaAttribute` je obohatená o rôzne parametre potrebné v tejto časti. Dôležitá je napríklad položka `recursiveItems`, ktorá označí miesto v strome atribútov, kde sa štruktúra schémy zacykluje. V tomto prípade sa hodnota nastaví na `null` a pri zobrazovaní stromu atribútov sa použijú iba hodnoty `value`. Naopak pri spracovávaní PML inštancie, kedy sa paralelne s JDOM stromom inštancie prechádza aj táto štruktúra (viď. časť 3.1.5), je potrebné nasledovať rekurzívny charakter dokumentu obmedzený inštanciou – použije sa položka `recursiveItems`.

Užívateľ má možnosť zakázať spracovanie w-vrstvy pri otváraní dokumentov z PDT. Z toho dôvodu sa pri prechode schémou sleduje prítomnosť role `#KNIT`

a skutočnosť, či odkazuje na w-vrstvu. V prípade splnenia daných podmienok sa za obsah odpovedajúcej štruktúry zoberie priamy podelement a nie ten, na ktorý sa odkazuje pomocou atribútu `type` (pripájané dáta).

Výsledná mapa (jej vrchná hladina) je indexovaná názvami štruktúr reprezentujúcimi uzol (s rolou `#NODE`). Tieto názvy v prípade typu `structure` zodpovedajú nepovinnému atribútu `name`. Ak nie je vytvorený alebo uzol reprezentuje typ `sequence` respektíve `container`, tento atribút sa do schémy za účelom zachovania konzistencie pridá.

3.1.4 Spracovanie globálnych vizuálnych nastavení

V tejto fáze sa vytvoria objekty interne reprezentujúce **globálne vizuálne nastavenia** a naplnia sa dátami. Tie sa buď načítajú z JDOM stromu spracovávaného dokumentu (VPML), alebo vzniknú dosadením implicitných údajov (čisté PML), ktoré sa však uložia ako globálna sekcia VPML do JDOM stromu PML inštancie. Vytvorí sa tým objekt `GlobalVisualSettings` zaobaľujúci a sprístupňujúci dáta zo sekcie globálnych nastavení VPML.

Nastavenia uzlov zastupujú inštancie tried `NodeVisual` (resp. jej neabstraktní potomkovia `NodeNormalVisual`, `NodeHighlightedVisual`), hrán `EdgeVisual` (resp. `EdgeNormalVisual`, `EdgeHighlightedVisual`). Stromová štruktúra atribútov pre rôzne druhy uzlov, ktorá je výsledkom spracovania schémy je v tejto fáze doplnená v koncových uzloch o globálne nastavenia atribútov pomocou inštancií triedy `AttrVisual` a zabalená do objektu typu `GlobalAttributesTree`. S týmito triedami sa stretneme aj v nasledujúcej časti.

3.1.5 Spracovanie PML inštancie

Teraz je už všetko pripravené na kompletný prechod cez JDOM strom PML inštancie a vytvorenie vlastnej reprezentácie dokumentu, ktorá sa stane obsahom objektu triedy `PMLMainDocument`.

3.1.5.1 Štruktúra dátovej reprezentácie dokumentu

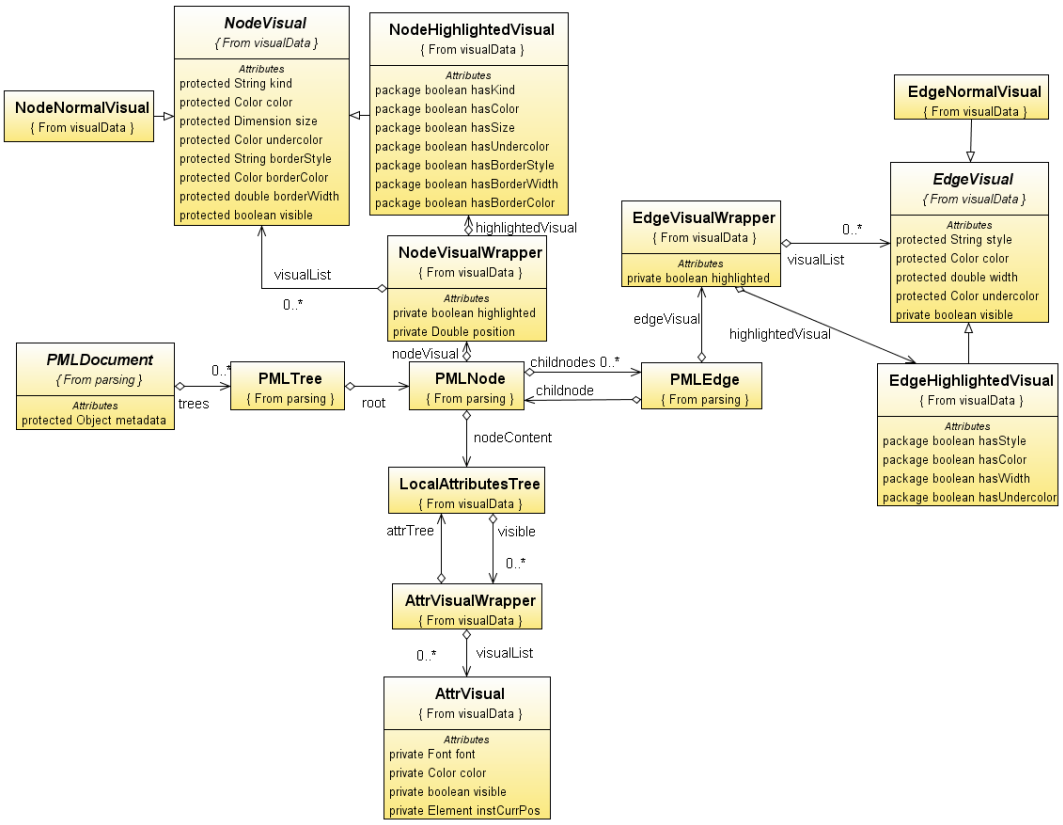
Táto reprezentácia sa skladá z dvoch častí:

- metadáta
- zoznam stromov

Do metadát patrí všetko čo je obsahom elementu `root` a nie je obsahom elementu s rolou `#TREES`. Tie sa síce spracovávajú, ale v samotnej aplikácii PMLViewer sa nijak nevyužívajú. Rozhodol som sa to spraviť takto preto, aby bola tieto metadáta k dispozícii a v prípade potreby nebol veľký problém doprogramovať zobrazovanie niektorých z nich. Navyše spracovanie metadát má marginálny dopad na celkový čas parsingu, keďže v priemere zaberajú veľmi malú časť celého dokumentu.

Zoznam stromov tvoria už relevantné dáta. Každý strom je zaobalený do objektu triedy `PMLTree`. Jeho obsahom sú navzájom poprepájané **uzly** (objekty triedy `PMLNode`) a **hrany** (`PMLEdge`).

Uzly v sebe obsahujú už pre daný uzol konkrétnu **lokálnu atribútovú štruktúru**, ktorá je reprezentovaná objektom typu `LocalAttributesTree`. Jej koncové atribúty môžu byť buď obyčajné (typu `PMLAttribute`), alebo s pridanou vizuálnou hodnotou. Obyčajnými sa stanú vtedy, ak k nim nie je dostupné globálne vizuálne nastavenie. V opačnom prípade sa vytvorí objekt triedy `AttrVisualWrapper`, ktorá je podedená od `PMLAttribute`. Tento objekt v sebe obsahuje presne tie informácie, ktorými sa odlišuje vo VPML lokálna sekcia atribútov od tej globálnej, čiže aktuálne nastavený štýl (globálny, lokálny). Okrem toho v sebe zahŕňa ešte referencie na objekty typu `AttrVisual`. Jeden je globálny, vytvorený počas spracovávania globálnych vizuálnych dát, a druhý vzniká práve v tejto časti, ale až vtedy, keď sa aktuálnym štýlom stane lokálne nastavenie (to je buď načítané z VPML dokumentu alebo vyplnené implicitne). Práve tieto objekty spolu s objektom `AttrVisualWrapper` sa starajú o zápis do VPML súboru po každej zmene nejakého vizuálneho parametru.



Obrázok 3.2 Schéma dátovej reprezentácie dokumentu (obyčajné šípky predstavujú dedičnosť, šípky s kosoštvorcom členskú premennú)

Podobne ako obsah terminálnych atribútov tvorí `AttrVisualWrapper` objekt, tak je súčasťou uzlu objekt `NodeVisualWrapper`, ktorý plní analogickú funkciu pre uzly. Na rozdiel od atribútu obsahuje tento objekt aj informáciu, či je uzol zvýraznený. Z toho dôvodu z neho vedie referencia na popis zvýraznenia uzlu v globálnych informáciách. Tento popis je typu `NodeHighlightedVisual`, ktorý je síce potomkom `NodeVisual`, ale plní trochu inú funkciu.

V mojom pôvodnom návrhu bola vizualizácia zvýraznených objektov rovnaká ako normálnych, len so zmenenými hodnotami parametrov. Tým sa stalo zvýraznenie len iným štýlom, takže pri zvýraznení nejakého uzlu sa zmenili všetky parametre vizualizácie. To som však postupom času prehodnotil a dodal som zvýrazneniu jeho skutočný význam. Nemenia sa všetky parametre, iba niektoré. Ktoré to budú, je možné si zvoliť tak, ako aj ich hodnotu. Na samotné zvýraznenie uzlu slúži metóda `highlight` objektu `NodeHighlightedVisual`, ktorá pretransformuje pôvodné vizuálne dáta na zvýraznené.

Dôležitou súčasťou obsahu uzlu je zoznam referencií na hrany, ktoré spájajú daný uzol s jeho deťmi, ako aj mapa rolí a atribútov, ktoré sú nimi označené. Je to potrebné napríklad pri zobrazovaní stromu, kedy sa vzájomná poloha uzlov určuje podľa ich role `#ORDER`.

Pre objekt typu `PMLEdge` ako reprezentanta hrany platia podobné skutočnosti ako pre uzol. Obsahuje objekt `EdgeVisualWrapper`, v ktorom sa o zvýrazňovanie stará inštancia triedy `EdgeHighlightedVisual`. Okrem toho v sebe drží referenciu na uzol – potomka.

Celá štruktúra je prehľadne zobrazená na obrázku Obrázok 3.23.2.

3.1.5.2 Priebeh spracovania

O spracovanie a vytvorenie štruktúry, o ktorej pojednávala predchádzajúca časť, sa stará niekoľko metód. Každá z nich zodpovedá za spracovanie konkrétneho PML typu (špeciálna pre typ `structure`, `list`, `sequence` atď. ako aj pre ich obsahy napr. `member`, `item`, `element` atď.) a sú medzi sebou previazané nepriamou rekurziou. Metódy, ktoré fungujú ako terminálne v tejto možnej rekurzii, sú práve tie, čo sa starajú o spracovanie atomických PML typov (`cdata`, `choice`, `constant`).

Väčšina „neterminálnych“ metód potrebuje pre svoje správne fungovanie týchto 5 parametrov:

- aktuálne spracovávaný element z JDOM stromu PML inštancie
- aktuálne spracovávaný element z JDOM stromu PML schémy
- aktuálne spracovávaný atribút z atribútovej štruktúry globálnych vizuálnych dát (nemusí byť žiaden)
- aktuálne spracovávaný uzol (nemusí byť žiaden)
- rola nadradeného elementu (nemusí byť žiadna)

Prítomnosť pozície v JDOM strome PML schémy a pozície v atribútovej štruktúre globálnych vizuálnych dát, ktorá vznikla zas iba z parsingu PML schémy, sa môže zdať redundantná. Avšak v objektoch typu `PMLSchemaAttribute`, z ktorých je atribútová štruktúra postavená, nie sú obsiahnuté všetky potrebné informácie. No nebol by veľký problém túto triedu rozšíriť, aby dokázala zachytiť aj zostávajúce parametre, ktoré je teraz potrebné vo fáze spracovania PML inštancie získavať z PML schémy. V takom prípade by bolo možné z metód starajúcich sa o parsing jednotlivých PML typov odstrániť odkaz na pozíciu v JDOM strome PML schémy a na tento účel by postačila vytvorená globálna atribútová štruktúra. Takéto riešenie by

neprinieslo úsporu v dĺžke doby spracovávania dokumentu, ale by skrátilo a možno aj sprehľadnilo kód. Ja som sa napriek tomu rozhodol implementovať tieto metódy súčasným riešením, a to z dôvodu podobnosti a jednotnosti týchto metód s metódami v `PMLSubDocument` a sčasti aj `PMLSchemaParser`. Tam sú na účely spracovania (špecifikované v predchádzajúcich častiach) použité rovnomenné metódy. Ich podobnosť (a tým aj rýchlejšie pochopenie) som chcel zväčšiť použitím referencií priamo do JDOM stromu PML schémy. Tento postup je však hodný prehodnotenia do ďalšej verzie aplikácie PMLViewer.

Pôvodne som v nasledujúcom texte zamýšľal každú z týchto metód stručne popísať, pretože parsing PML inštancie je fundamentálnou časťou spracovania celého PML dokumentu. Bolo by to však zbytočne zdĺhavé, a preto ďalej upozorním len na najdôležitejšie a netriviálne záležitosti. Záujemca sa môže bližšie o fungovaní týchto metód dozvedieť z priloženej po anglicky písanej Javadoc[29] dokumentácie.[30]

3.1.5.3 Spájanie (knitting)

Doteraz som ešte nespomenul, akým spôsobom je riešená jedna z dôležitých konštrukcií jazyka PML. Je ňou možnosť napojiť na nejaké miesto v dokumente časť z iného dokumentu tak, aby to vyzeralo kompaktne, ako jednotný celok, inými slovami knitting.

Druhá fáza spracovania dokumentu vytvorí vnútornú reprezentáciu referencovaných dokumentov. Zatiaľ však tieto dokumenty neboli potrebné. Knitting je konštrukt, ktorý ich využíva. Okrem knittingu je možné ukazovať do týchto poddokumentov aj obyčajnými referenciami. V aplikácii PMLViewer sa však pri výskyte nejakého takéhoto odkazu nevykonáva nijaká špecifická akcia. Výnimku tvoria jedine odkazy do a-vrstvy v t-vrstve PDT 2.0, ktoré sa využívajú pri zobrazovaní kompletnej vety vykresľovaného stromu.

Knitting sa uskutočňuje všade tam, kde sa vyskytne rola `#KNIT`. Nie vždy to však musí platiť. Užívateľ má možnosť v nastaveniach zamedziť spracovaniu w-vrstvy PDT 2.0 pri načítaní dokumentu, ktorým je referencovaná. Vtedy neprebehne ani spracovanie poddokumentu, a samozrejme ani knitting, keďže nie je k dispozícii zdroj, ktorý by sa mal pripájať. K takejto referencii sa potom pristupuje ako k obyčajnému odkazu, čiže sa interpretuje ako atomická hodnota.

Rola `#KNIT` sa môže vyskytovať v týchto troch PML typoch (častiach typov): položka `structure` – `member`, položka `sequence` – `element` a `list`. Čo sa týka implementácie knittingu, prvé dva typy sa trochu odlišujú od posledného, ale v princípe je to rovnaké.

Referencia sa musí najprv rozdeliť na dve časti. Tie od seba oddeľuje symbol `#`. Prvú časť tvorí odkaz na poddokument, v ktorom sa vkladaný obsah nachádza. Druhá časť je identifikátor, ktorým je tento obsah označený. Vyčlenenie

[29] http://java.sun.com/j2se/javadoc/

[30] Je k dispozícii adresári `doc/javadoc` adresárovej štruktúry aplikácie PMLViewer

štruktúr, ktoré sú označené nejakým identifikátorom (rola `#ID`), prebehlo už vo fáze parsingu poddokumentov, takže sa požadovaný obsah rýchlo dohľadá. Nakoniec sa celý skopíruje a vloží do PML inštancie hlavného dokumentu namiesto pôvodnej referencie. PML schéma sa meniť nemusí, pretože je v nej popis vkladaného obsahu už prítomný v podobe užívateľského typu. Preto môže spracovanie PML inštancie pokračovať ďalej štandardným spôsobom, akoby tam bol vkladaný obsah už od samého začiatku.

Knitting typu `list` sa odlišuje tým, že sa tu môže vyskytovať zoznam referencií, kde každá z nich musí byť nahradená. Malú odchýlku v spôsobe spracovania zapríčiňuje skutočnosť, že tento zoznam môže byť jedno- alebo viacprvkový. Ak je jednoprvkový, nenachádza sa v ňom podelement `LM`. Preto sa k tomuto knittingu pristupuje ako ku položkám typov `structure` alebo `sequence`. V prípade, že je viacprvkový, musia byť do jednotlivých podelementov `LM` prekopírované zodpovedajúce obsahy z referencovaného PML dokumentu.

Naskytá sa otázka, či musia byť obsahy z poddokumentov kopírované. Nestačilo by sa k týmto referenciám postaviť naozaj ako k referenciám, čiže pri výskyte rovnakej referencie na dvoch odlišných miestach v hlavnom dokumente by tieto ukazovali na ten istý obsah, čím by odpadlo kopírovanie? Odpoveď je jednoduchá, nie! Problém by vznikol pri vytvorení VPML súboru (a rovnako už v samotnej vnútornej reprezentácii), kedy by takýmto, síce obsahovo rovnakým, no pozične odlišným štruktúram museli byť priradené tie isté vizuálne dáta. A to by užívateľ vo väčšine prípadoch nechcel. Koniec koncov v samotnej špecifikácii PML je stanovené, aby sa referencovaný obsah na miesto s rolou `#KNIT` kopíroval [4].

3.1.5.4 Spracovanie typu container

Typ `container` bol do špecifikácie PML pridaný až vo verzii 1.1 a je oproti ostatným zloženým typom v niečom trochu iný. U všetkých ostatných typov sú ich položky špeciálne definované a pomenované (`member`, `element`, `item`) a až pod týmito zložkami sú uložené ďalšie typy. U typu `container` takisto existuje takto pomenovaná položka (`attribute`), no okrem toho obsahuje nejaký jeden z typov priamo, bez medzikroku.

Z tohto dôvodu je nutné pri vytvorení internej reprezentácie ako mapy vyriešiť, pod akým indexom bude tento hlavný obsah v nej uložený. U položiek `attribute` problém nie je, sú uložené pod svojími menami (XML atribút `name`). Pre hlavný obsah som bol nútený vymyslieť taký názov indexu, aby v žiadnom prípade nekolidoval s ostatnými.

Toto je možné zaručiť tým, že sa vyberie nejaké pomenovanie (deterministicky alebo náhodne) a pokiaľ koliduje, vyberie sa iné. Takýmto spôsobom sa určite zvolí názov, ktorý nie je v rozpore s ostatnými.

Ja som sa však vydal inou cestou. V špecifikácii XML 1.0 je napísané, že znak „:“ by sa nemal používať v definícii mena, iba na oddelenie menného priestoru. Napriek tomu by mali XML procesory vedieť spracovať aj takéto mená [6, kapitola 2.3]. Ako sa však dočítame na domovskej stránke projektu JDOM v sekcii FAQ, autori tejto knižnice majú iný názor a vyhradzujú

dvojbodke v názve elementu alebo atribútu iba funkciu oddeľovača namespacu od skutočného názvu konštruktu [7]. Z toho vyplýva fakt, že samotný názov atribútu nemôže obsahovať dvojbodku. Takže mi vo vnútornej reprezentácii stačí pomenovať index do mapy, pod ktorým bude uložený hlavný obsah typu container, názvom obsahujúcim znak „:“. Ja som zvolil pomenovanie `pml:container_content`. Ak by niekto rovnakým spôsobom pomenoval atribút v kontajneri, spracovanie dokumentu zlyhá už pri parsingu PML inštancie do JDOM stromu. Takéto chovanie knižnice JDOM je na jednej strane zo strany jej autorov nedodržanie normy XML 1.0, na strane druhej si však neviem predstaviť, že by niekto dával takéto zavádzajúce pomenovania pre elementy alebo atribúty.

3.1.6 Ošetrenie chýb

Pri spracovaní PML dokumentu môže nastať niekoľko chýb. Môžu to byť chyby spojené so vstupno-výstupnými operáciami, XML parsingom do JDOM stromu alebo chyby vznikajúce porušením pravidiel PML. Práve na posledný prípad slúži trieda výnimky `PMLParsingException`. Súčasťou tejto výnimky je informácia o ceste k súboru, v ktorom chyba nastala a chybová správa. Môže sa však stať, že táto správa je nedostačujúca. Chýba hlavne informácia o čísle riadku v súbore, kde nastala chyba. Knižnica JDOM neumožňuje priamo zistiť z pamäťovej reprezentácie napr. nejakého elementu číslo riadku v súbore, kde sa tento element nachádza. Vyžadovalo by si to rozšírenie JDOM knižnice o túto funkcionalitu. Takéto rozšírenie je vhodné na zváženie do budúcej verzie aplikácie.

3.2 Zobrazenie stromov, zmena ich parametrov a užívateľské rozhranie

Po načítaní dokumentu a vytvorení jeho vnútornej reprezentácie je všetko pripravené na to, aby sa vykonali akcie, ktoré dokončia prípravu a nakoniec zobrazia výsledok – dokument, ktorého vzhľad a vzhľad jeho častí (stromov) bude možné modifikovať.

V tejto časti sa budem venovať najmä vykresleniu stromu, bočného kontextového panela vizuálnych nastavení, ako aj panela zobrazenia vety.

3.2.1 Vykreslenie stromu

Na samotnú vizualizáciu som sa rozhodol použiť knižnicu JGraph[31], ktorá je vynikajúcim prostriedkom na vykresľovanie grafov. Je postavená na Swing komponentoch a dodržiava MVC (Model-View-Controller) architektúru.[32] Napriek tomu na to, aby som docielil zobrazenie, aké som požadoval, bolo nutné doprogramovať viacero rozšírení. Medzi inými bolo potrebné špecificky

[31] http://www.jgraph.com/

[32] Architektúra MVC je založená na oddelení troch súčastí komponentu [8]:

1. model, ktorý reprezentuje dátovú zložku
2. pohľad, ktorý určuje, akým spôsobom sa budú dáta z modelu zobrazovať
3. radič, ktorý zabezpečuje užívateľskú interakciu

zobrazovať uzly stromu, reagovať na akcie, doplniť rôzne druhy usporiadania uzlov v stromoch a iné.

Teraz popíšem, aké všetky súčasti z knižnice JGraph sú potrebné na správne zobrazenie stromu, a ktoré som v aplikácii rozšíril alebo preťažil. Všetky tieto súčasti bude v ďalšom texte súhrnne označovať pojmom **„graf“**.

Základným komponentom knižnice JGraph je trieda `JGraph`. V aplikácii PMLViewer je táto podedená triedou `PMLJGraph`. V nej sa nachádzajú metódy na inicializačný prevod štruktúry, ktorá vznikla počas načítania a spracovania dokumentu, na model komponentu `JGraph`. Okrem toho sa tu nachádzajú metódy, ktoré menia vzhľad celého grafu, nie len jeho častí (hlavne pozície uzlov).

Model grafu tvorí objekt triedy `DefaultGraphModel`. U jednotlivých objektov grafu je to `PMLNodeCell` pre uzol a `DefaultEdge` pre hranu (pôvodná trieda z knižnice JGraph). Bližší popis modelu a dôvod, prečo som pri implementácii nepoužil svoj vlastný model grafu, ale preddefinovaný, popíšem v časti 3.2.1.2.

Čo sa týka samotného výzoru grafu, okrem hlavnej triedy `PMLJGraph` sa o tzv. view (pohľad) starajú triedy zodpovedajúce objektom grafu, konkrétne `PMLNodeView` a `PMLCellView`. Mapovanie medzi modelmi a pohľadmi sprostredkováva trieda `GraphLayoutCache`.

Poslednou súčasťou MVC architektúry je tzv. controller (radič). Ten je v grafe aplikácie PMLViewer reprezentovaný triedou `PMLGraphUI` a stará sa o spracovanie požiadaviek od užívateľa. Bližšie bude popísaný v časti 3.2.1.3.

3.2.1.1 Pohľad grafu

3.2.1.1.1 Inicializácia grafu

V súvislosti s pohľadom grafu uznávam, že som sa dopustil malého prehrešku voči architektúre MVC. Celkový view grafu reprezentuje najmä trieda `PMLJGraph`. V nej sa však nachádza aj inicializácia a transformácia štruktúry, ktorá vzišla zo spracovania PML dokumentu, na model grafu. Takáto funkcionalita je priam predurčená na to, aby bola súčasťou triedy reprezentujúcej model grafu. Avšak podľa JGraph manuálu [9, s. 22] sa odporúča nové uzly a hrany vkladať do grafu prostredníctvom triedy `GraphLayoutCache`, ktorá je súčasťou pohľadu grafu. Umiestniť do modelu priame volanie metód z pohľadu by bolo ešte väčšie porušenie paradigmy MVC. Preto som sa rozhodol túto funkcionalitu umiestniť do triedy `PMLJGraph`, ktorá je podľa môjho názoru predsa len všeobecnejšia a zachycuje širšie spektrum než iba pohľad. Navyše pomocné štruktúry, ktoré sa počas inicializácie vytvoria, sú ďalej používané metódami umožňujúcimi zmeny tvaru celého stromu (zmena horizontálnej a vertikálnej vzdialenosti medzi uzlami, tri druhy usporiadania uzlov).

3.2.1.1.2 Triedenie grafu

Práve usporadúvanie uzlov je zaujímavá funkcionalita. O výslednom vzhľade stromu po aplikovaní týchto triedení som hovoril už v časti 2.5.3. Na tomto

mieste spomeniem v stručnosti, akým spôsobom sú tieto zmeny tvaru implementované. Všetky tri triedenia pri svojej činnosti používajú štruktúru `levels`. Je to zoznam zoznamov, kde položky vonkajšieho zoznamu sú všetky uzly ležiace na rovnakej hladine, čiže v rovnakej hĺbke začínajúc od koreňa stromu (tento je vytvorený pri inicializácii grafu). Z toho dôvodu sa vo vertikálnom smere tieto usporiadania neodlišujú.

Najjednoduchšie triedenie „bez poradia“ len postupne prejde všetky hladiny (tým určuje vertikálnu súradnicu pozície uzlu) a na rovnakej hladine sa uzly umiestnia od ľavej strany tesne za sebou.

Pri zotriedení v štýle závislostného stromu je nutné udržiavať poradie uzlov. To je dané rolou `#ORDER`, kde so stúpajúcou hodnotou atribútu s touto rolou sa horizontálna pozícia uzlov mení smerom doprava. Najprv sa rovnakým spôsobom ako v predošlom prípade nastavia vertikálne súradnice uzlov, následne sa využije zoznam uzlov vzostupne zotriedený podľa hodnôt atribútov s rolou `#ORDER` a postupne sa od uzlu s najmenšou hodnotou po ten s najväčšou určia zľava doprava horizontálne súradnice. Postupuje sa podobne ako pri triedení „bez poradia“, avšak horizontálne pozície sa neurčujú pre každú hladinu zvlášť, ale pre všetky uzly naraz.

Princíp triedenia v štýle zložkového stromu je odlišný. Vertikálne pozície sa však opäť určia pomocou štruktúry `levels`. Horizontálne polohy sú stanovené pri prechode stromom metódou postorder.[33] Prvý navštívený list je umiestnený najviac vľavo, vodorovná pozícia ďalších listov je určená postupne za sebou smerom doprava (listy nemusia ležať v jednej hladine). Poloha vnútorného uzlu je v polovici vzdialenosti medzi jeho najpravejším a najľavejším potomkom (samozrejme o hladinu vyššie). Ak však má iba jedného potomka, ich horizontálne pozície sú zhodné.

3.2.1.1.3 Pohľady hrany a uzlu. Grafická atribútová mapa

Najväčší vplyv na konečný výzor celého grafu majú čiastkové pohľady uzlu a hrany (reprezentované triedami `PMLNodeView` a `PMLEdgeView`, konkrétne triedami zodpovednými za samotné vykreslenie `PMLNodeRenderer` a `PMLEdgeRenderer`).

V architektúre knižnice JGraph je pre každú bunku (uzol, hrana) k dispozicii **grafická atribútová mapa**, ktorá je miestom, odkiaľ by pohľad mal čerpať informácie dôležité pre vykreslenie bunky. Prácu s touto mapou má na starosti trieda `GraphConstants`. Tá zabezpečuje, aby bolo do mapy možné uložiť len určenú množinu dát [9, kapitola 2.1.5]. V aplikácii PMLViewer túto funkciu na seba preberá trieda `ExGraphConstants` (potomok `GraphConstants`) a rozširuje ju o špecifickú prácu s vizuálnymi parametrami uzlu a hrany (objekty tried `NodeVisual` a `EdgeVisual`), zoznam viditeľných atribútov (zoznam inštancií `AttrVisualWrapper`) a o parametre grafických nastavení pre celý strom (miera nepriehľadnosti a šírka podfarbenia, vzdialenosť medzi symbolom uzlu a zoznamom viditeľných atribútov). Tým sa vlastne

[33] Prechod stromu do hĺbky metódou postorder je založený na tom, že po vstupe do nejakého uzla sa spracujú najprv všetky podstromy a až po nich tento uzol.

čiastočne prekrýva funkcionalita pôvodnej triedy grafických konštánt, preto mnohé z konštánt triedy `GraphConstants` nie sú pri vykresľovaní hrán a uzlov použité.

U hrany sa o veľké zmeny nejedná. Kreslenie grafických objektov akurát získava údaje z nových parametrov v grafickej atribútovej mape a spracúva nový vizuálny parameter – podfarbenie.

Komponent uzlu je naproti tomu zmenený úplne. Je to panel rozdelený na dve časti:

1. symbol uzlu
2. zoznam viditeľných atribútov

Symbol uzlu sa vykresľuje podľa nastavení pre uzol v mape grafických atribútov. Určí sa viditeľnosť, farba, parametre okraja. Rovnako ako aj pre hranu sa zistí, či má byť uzol podfarbený. Podfarbenie sa kreslí tak, že sa najprv zobrazí objekt rovnakého tvaru ako má mať uzol, s farbou podfarbenia. Tento objekt má nastavenú priehľadnosť a zväčšenú šírku. Obe vlastnosti má možnosť užívateľ zmeniť vo všeobecných grafických nastaveniach (viac v kapitole 2.8). Na tento podfarbujúci objekt sa doprostred nakreslí samotný uzol. U hrany je to podobné.

Zoznam viditeľných atribútov je panel s bielym pozadím umiestnený pod symbolom uzlu, do ktorého sa vkladajú jednotlivé atribúty ako komponenty triedy `JLabel`. Pre tieto komponenty sa opäť nastavia parametre pre vykreslenie pomocou zoznamu objektov `AttrVisualWrapper` z grafickej atribútovej mapy.

3.2.1.2 Model grafu

Ako som už spomenul, model celého grafu je v aplikácii PMLViewer reprezentovaný triedou `DefaultGraphModel`. Nebola to však jediná možnosť. Na to, aby mohli byť nejaké triedy modelom grafu, musia implementovať súvisiace rozhrania z knižnice JGraph (`GraphModel`, `GraphCell`, `Edge`). Preto by nebol problém implementovať tieto rozhrania pomocou tried `PMLTree`, `PMLNode` a `PMLEdge`. V takomto prípade by už nebolo nutné v inicializácii triedy `PMLJGraph` prechádzať celú štruktúru zavesenú na objekte typu `PMLTree` a vytvárať nové objekty do modelu grafu.

Takéto riešenie je istotne efektívnejšie, napriek tomu som sa mu chcel vyhnúť. Dôvodom bol môj záujem o úplné oddelenie časti starajúcej sa o parsing od tej, ktorá strom vykresľuje. Vychádzal som z toho, že môže nastať situácia, kedy bude niekto chcieť tento parsovací modul použiť v inom projekte, kde naopak nebude potrebovať prítomnosť knižnice JGraph. V alternatívnom riešení, kde by triedy `PMLTree`, `PMLNode` a `PMLEdge` implementovali rozhrania z knižnice JGraph, by sa modul spracovávajúci PML a VPML dokumenty musel nutne dodávať aj s knižnicou JGraph, čo je principiálne nesprávne.

Trieda `PMLNodeCell` je rozšírená oproti jej predkovi v dedičnosti (`DefaultGraphCell`) o funkciu priameho prístupu k potomkom v strome. Takáto funkcionalita sa zíde pri usporadúvaní uzlov do iného tvaru stromu v grafe. Bez tejto možnosti by bolo nutné pri každom prechode v smere

otec → syn získať najprv tzv. port[34] otcovského uzlu, potom hrany naň pripojené a nakoniec cez port synovských uzlov až samotné uzly potomkov.

3.2.1.3 Radič grafu

Controller grafu je v aplikácii PMLViewer reprezentovaný triedou `PMLGraphUI`. Tá je zodpovedná za spracovanie udalostí, ktoré sú generované inými komponentmi alebo po akcii užívateľa.

Takým spôsobom sa spracúva napríklad udalosť vyvolaná zmenou v grafických parametroch vnútornej reprezentácie stromu. Určí sa, či má zmena globálny alebo lokálny dopad a podľa toho sa pre dotyčné objekty (atribúty, uzly alebo hrany) zmení obsah grafickej atribútovej mapy, čo vyvolá prekreslenie grafu.

Radič grafu spracúva aj kliknutie na graf (presnejšie, uvoľnenie tlačidla myši). Podľa toho, či sa táto akcia vyvolala nad uzlom, hranou alebo nad prázdnym priestorom, zvolí sa vhodné kontextové menu a zobrazí sa.

Táto trieda okrem toho aj jednu udalosť generuje. Tá sa vyvolá práve po spracovaní akcie myši a indikuje vybratie jedného uzlu. To využíva panel vety a zvýrazní slová, ktoré vybraný uzol zahŕňa (viac o paneli vety v častiach 2.5.2 a 3.2.3).

3.2.2 Kontextové menu

V pravej časti hlavného okna aplikácie PMLViewer sa nachádza tzv. kontextové menu. Jeho obsah sa mení v závislosti na označenom objekte.

3.2.2.1 Editačné formuláre

Toto menu obsahuje niekoľko formulárov, ktoré som vytvoril za pomoci nástroja na rozvrhnutie užívateľského prostredia vo vývojovom prostredí NetBeans. Určené sú na modifikáciu vzhľadu uzlov a hrán. Rozdeľujú sa na lokálne a globálne (podľa toho, či je označený nejaký objekt alebo nie).

Pre formuláre globálneho nastavenia (napr. uzlu, u hrany to je analogické) slúžia ako model dva objekty z inštancie `GlobalVisualSettings`, ktorá vznikla v 4. fáze spracovania PML dokumentu, a to konkrétne typu `NodeNormalVisual` a `NodeHighlightedVisual`. Zmeny vo formulári okamžite vyvolajú zmeny v modeli, čo spôsobí vygenerovanie udalostí, na ktoré reaguje graf stromu pomocou triedy `PMLGraphUI`.

Formulár lokálneho nastavenia má za model vizuálnu informáciu z označeného uzlu (podobne aj pre hrany) čiže inštanciu triedy `NodeVisualWrapper`. Z tejto inštancie sa pri inicializácii formuláru alebo po aktivácii lokálnych nastavení pre daný uzol ešte vytiahne objekt `NodeNormalVisual` a použije sa na naplnenie dvoch premenných. Jedna určuje skutočné lokálne nastavenie, tá druhá obsahuje nastavenie, ktoré sa zobrazí užívateľovi. To môže byť zhodné s tým skutočným, avšak pri zvýraznenom uzle to je reálne nastavenie, na ktoré je aplikované nastavenie zvýraznenia pomocou metódy `highlight`.

---

[34] Port je miesto v uzle, kde sa k nemu pripájajú hrany. Takýchto portov môže byť v uzle viac a môžu mať aj svoju vlastnú grafickú interpretáciu [9, kapitola 3.5.4]

3.2.2.2 Komponent `JTreeTable`

Na zobrazenie atribútovej štruktúry, ktorá je hierarchická (dala by sa zobraziť pomocou Swing komponentu `JTree`) a zároveň každá z položiek má viacero parametrov (na túto funkciu by dobre poslúžil komponent `JTable`), nie je v štandardnej Jave žiadny adekvátny užívateľský ovládací prvok. Vývojárom toto chýbalo, preto sa v rámci Sun Developer Network[35] objavilo riešenie a implementácia užívateľského prvku, ktorý by bol tabuľkou, navyše by však boli jednotlivé riadky hierarchicky usporiadané. Tento komponent sa nazýva `JTreeTable`. Do projektu PMLViewer sú vložené priamo jeho zdrojové kódy (nie ako knižnica), opatrené hlavičkami poukazujúcimi na skutočného tvorcu týchto súborov.

Aplikácia PMLViewer takéto zobrazenie vyžaduje na viacerých miestach. Vždy istým spôsobom vykresľuje nejakú časť dokumentu. Na každé z týchto zobrazení som vytvoril vlastný model.

V kontextovom menu sa používa tento komponent na informačné zobrazenie hodnôt, ktoré nadobúdajú atribúty pre nejaký uzol, a taktiež na editáciu lokálnych vizuálnych nastavení týchto atribútov. Modelmi pre tieto použitia komponentu `JTreeTable` sú `AttributesInfoModel` a `AttributesLocalSettingsModel`. V oboch je základom pre ich vytvorenie atribútová štruktúra označeného uzla reprezentovaná inštanciou triedy `LocalAttributesTree`.

Ak nie je označený žiaden objekt v grafe, v kontextovom menu má užívateľ možnosť editovať globálne vizuálne nastavenia atribútov. V tomto prípade je opätovne použitý komponent `JTreeTable` s modelom `AttributesGlobalSettingsModel`, ktorý používa globálnu atribútovú štruktúru `GlobalAttributesTree`.

Podobne sa tento komponent používa aj na začiatku spracovania PML dokumentu, aby užívateľ doplnil chýbajúce role.

Okrem toho bolo potrebné naprogramovať spôsob zobrazenia špeciálnych typov buniek tabuľky. Toto zobrazenie sa ešte odlišuje podľa toho, či sa bunka edituje (kliklo sa na ňu) alebo nie. Vzniklo tak zaškrtávacie tlačidlo na zobrazovanie a editáciu hodnôt typu `boolean`, farebná bunka na zobrazenie farby (editácia sa prevádza cez dialógové okno na výber farby) a v neposlednej rade jednoduché zobrazenie štýlu písma, ktorého editácia prebieha takisto prostredníctvom dialógového okna. To však opäť v štandardnej Jave obsiahnuté nie je, takže som si ho musel implementovať sám.

3.2.3 Panel zobrazenia vety

Tento panel je určený na zmysluplné zobrazenie vety, ktorú vykresľovaný strom reprezentuje. Navyše obsahuje informáciu o počte viet v dokumente a poradí aktuálne zobrazovanej vety. Samotný panel vety správne funguje pri dokumentoch m-vrstvy, a-vrstvy a t-vrstvy PDT 2.0 a dokumentoch a-vrstvy

[35] http://java.sun.com/products/jfc/tsc/articles/treetable1/

a t-vrstvy z Prague English Dependecy Treebank[36] (PEDT). Pre ostatné dokumenty je neznáme buď umiestnenie slov, z ktorých sa konštruuje veta, alebo vetné poradie jednotlivých uzlov. Výsledkom je, že vetu nie je možné vypísať.

Panel vety je vytvorený pomocou špeciálneho komponentu, ktorého torzo tvorí abstraktná trieda `AbstractHighlightTextPane`. Tá je vlastne potomkom Swing komponentu `JTextPane`. Odlišuje sa pridanou funkcionalitou, a to zvýraznením aktuálne označeného uzlu v grafe. Konštruktor tohto komponentu požaduje mimo iného zoznam uzlov stromu. Tento zoznam sa následne transformuje na zoznam jednoduchých dvojíc: identifikátor uzlu, slovo (presnejšie token), ktorý je zotriedený podľa vetného poriadku. Tento komponent sa musí zaregistrovať v radiči grafu (objekte triedy `PMLGraphUI`) na príjem udalostí vygenerovaných radičom. Pri označení nejakého uzlu v grafe sa potom postupne do textového poľa komponentu vypisujú slová zo zotriedeného zoznamu. Tie, ktorých identifikátory zodpovedajú identifikátorom súvisiacim s označeným uzlom, sa vykreslia podfarbené. Úlohou potomka tejto abstraktnej triedy je implementovať spôsob, akým sa pretransformuje zoznam uzlov na zoznam zotriedených dvojíc, a spôsob, akým pre vybraný uzol získať množinu identifikátorov s ním súvisiacich.

M-vrstva sa líši od ďalších uvedených tým, že neobsahuje rolu `#ORDER`. Poradie tokenov tak nie je určené touto rolou, ale štruktúrou dokumentu. Zvyšné potrebné dáta sú priamo prístupné na m-vrstve.

Pre a-vrstvu je implementujúcim potomkom trieda `ALayerHighlightTextPane` a potrebné dáta získava z atribútov pripojených z m-vrstvy prostredníctvom role `#KNIT`.

V t-vrstve je získanie trojice identifikátor – token – poradie komplikovanejšie. Keďže sa v dokumente t-vrstvy nenachádza žiadny atribút s rolou `#KNIT`, sú všetky poddokumenty spracované iba do hladiny JDOM stromov, čiže neexistuje ich špeciálna reprezentácia ako pre hlavný dokument. Aby sa zabezpečilo rýchle získanie každého tokenu práve raz, aplikácia sa dostane z koreňa t-vrstvy na koreň a-vrstvy. Tam postupne za použitia dotazov jazyka XPath získa všetkých potomkov, potomkov potomkov atď. a u každého sa z a-vrstvy zistí identifikátor a poradie. Napokon sa zistí hodnota tokenu prechodom na m-vrstvu. Výsledný zoznam trojíc sa zotriedi podľa vetného poriadku.

Tokeny v zotriedenom zozname sú teda označené identifikátorom z a-vrstvy. Preto musí implementácia obstarania identifikátorov vracať zoznam tých z analytickej vrstvy PDT 2.0. Uzol na t-vrstve môže reprezentovať aj väčší počet slov. Tie sú určené referenciami do a-vrstvy v atribútoch `a/lex.rf` a `a/aux.rf`. Už nie je nutné pristupovať k poddokumentom, identifikátory a-vrstvy sa získajú odstránením prefixu poddokumentu z týchto referencií.

[36] http://ufal.mff.cuni.cz/~toman/pedt_manual/index.html

3.3 Uloženie a export

3.3.1 Priebeh uloženia a exportu

Aplikácia PMLViewer ukladá modifikované dokumenty vo formáte VPML (viac o VPML v časti 1.2). Dôležitý je fakt, že žiadny úsek z pôvodného čistého PML dokumentu, až na jednu výnimku, nie je v uloženom výsledku zmenený ani vymazaný. Tou výnimkou sú časti vložené pomocou role `#KNIT`, kde sa pôvodný obsah elementu označeného touto rolou zmaže a nahradí sa štruktúrou z poddokumentu. Tento prípad je výnimkou aj v inak striktnom pravidle, že vo výslednom VPML dokumente sú všetky novo vložené prvky identifikované menným priestorom `vpml`. Takisto platí, že pri ukladaní sa uchováva iba inštancia. Uložený výsledok stále prislúcha pôvodnej PML schéme. Z toho plynie, že zmeny uskutočnené v PML schéme počas parsingu nie sú zaznamenané. Ak napríklad nastane pri načítavaní PML dokumentu situácia, že je potrebné doplniť nejaké role, užívateľ ich doplní a potom dokument uloží, pri nasledujúcom načítaní tohto uloženého dokumentu bude musieť dopĺňať role znova. Toto riešenie súvisí so spôsobom spracovania inštancie, ktorej implementácia bude do nasledujúcej verzie prehodnotená okrem iného aj takým spôsobom, aby bolo možné upravenú schému uložiť.

Samotné uloženie prebieha podobne ako načítanie v inom vlákne, aby GUI bolo schopné reagovať na podnety od užívateľa. Stará sa o to trieda `SaveProvider`, ktorá je potomkom triedy `SwingWorker`. Jej akcia na pozadí spočíva jedine v serializácii JDOM stromu do XML súboru a jeho prípadnej komprimácii formátom gzip. Po uložení sa eventuálne zmení súbor, s ktorým sa pracuje, dokument sa prehlási za vizuálny a zmení sa titulok v hlavnom okne.

Na export do rôznych formátov sa používa knižnica FreeHEP VectorGraphics.[37] Táto knižnica je súčasťou projektu FreeHEP, ktorý zjednodušuje vývoj aplikácii zaoberajúcich sa časticovou fyzikou [10]. Tento balík knižníc obsahuje aj niektoré všeobecne použiteľné aplikačné rozhrania, mimo iných aj to, použité v aplikácii PMLViewer a slúžiace na export do veľkého množstva rastrových a vektorových formátov. PMLViewer umožňuje export do JPEG, PNG (rastrové formáty), SVG, PDF a EPS (vektorové formáty).

Pri výbere knižnice, ktorá by zabezpečila export do aspoň niektorých formátov, som zvažoval viaceré možnosti. Prvoradý bol pre mňa (a podľa špecifikácie) export do vektorového formátu SVG a rastrového PNG. Vytváranie PNG obrázkov umožňuje samotná Java 6 SE. Na prácu s SVG dokumentmi existuje knižnica Batik[38], ktorú som na začiatku aj používal. Keď som potom objavil FreeHEP VectorGraphics, rozhodnutie padlo práve na túto knižnicu a to z týchto dôvodov:

- umožňuje export do všetkých mnou požadovaných formátov

[37] http://java.freehep.org/vectorgraphics/index.html

[38] http://xmlgraphics.apache.org/batik/

- pre každý formát obsahuje už pripravené komponenty formulárov, ktoré slúžia užívateľovi na eventuálnu zmenu parametrov exportu
- na export sa používa jednotný spôsob – pre daný formát špecifický potomok triedy `Graphics` reprezentujúci grafické plátno, kam sa potom kreslí požadovaný obsah

Posledne spomenutý dôvod je pre aplikáciu PMLViewer veľmi dôležitý. Export prebieha tak, že sa zoberie aktuálne zobrazovaný graf, čiže inštancia triedy `PMLJGraph`, a metódou `print` sa vykreslí práve do potomka triedy `Graphics`, pre SVG to je `SVGGraphics2D`. Metóda `print` volá metódu `paint`, takže export prebieha rovnakými prostriedkami ako vykreslenie na obrazovku. Metóda `print` sa používa preto, aby sa zaistil vektorový charakter pri exporte do vektorových formátov. Keby sa v implementáciách metód `paint` v triedach `PMLNodeRenderer` a `PMLEdgeRenderer` nerozlišoval typ grafického plátna, v exportovaných obrázkoch by sa objavilo napríklad aj orámovanie označených uzlov, čo nie je žiaduce. Preto sa pri vykresľovaní zistí, či je grafické plátno inštanciou triedy `VectorGraphics` (tá je spoločným predkom pre grafické plátna všetkých formátov) a ak je, nevykreslia sa označenia objektov a zároveň sa vôbec nevykreslia objekty, ktoré sú úplne skryté (na obrazovke je skrytá hrana označená sivou prerušovanou čiarou a uzol malým sivým štvorcom s prerušovaným okrajom).

Pri kreslení rastrových obrázkov pomocou prostriedkov priamo dostupných v Jave sa takto jednoducho v metódach `paint` výstup kreslenia rozlíšiť nedá. V takom prípade sa totiž ako grafické plátno používa trieda `Graphics2D`, ktorá však nič nehovorí o charaktere výstupu. Môže byť použitá aj pri kreslení na obrazovku.

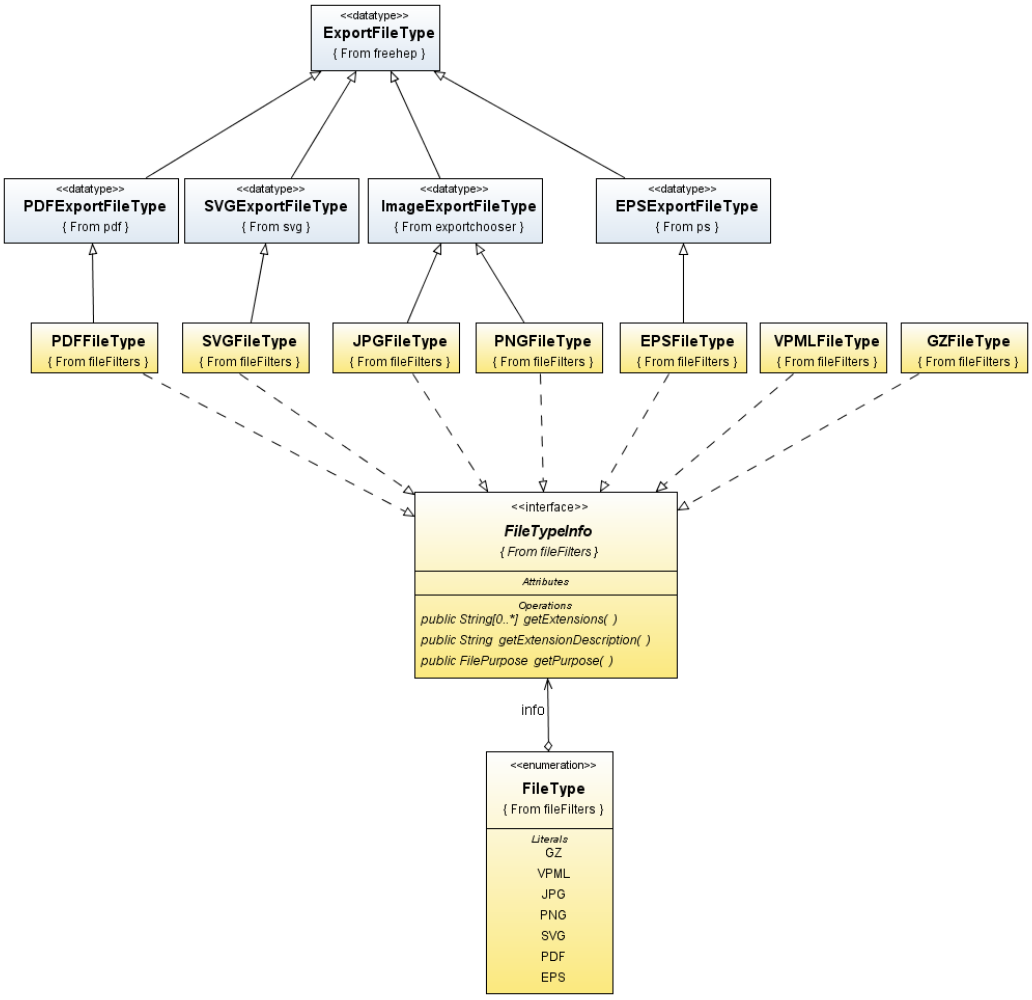
Export prebieha, podobne ako zvyšné dlhotrvajúce operácie načítania a uloženia, v inom vlákne. Zabezpečuje to potomok triedy `SwingWorker`, objekt triedy `PMLExport`. V dávkovom spracovaní sa export vykonáva priamo v hlavnom vlákne, preto je v tejto triede k dispozícii aj statické volanie exportu.

3.3.2 Práca s rôznymi typmi súborov

Jednotný prístup k vlastnostiam a funkciám rôznych typov súborov, s ktorými aplikácia PMLViewer pracuje, je zabezpečený enumeráciou `FileType`. Tá ako členy obsahuje práve označenia typov súborov. Každý takýto člen má okrem toho priradenú informáciu, ku ktorej sa pristupuje pomocou rozhrania `FileTypeInfo` a musí sa pre každý typ implementovať. Rozhranie `FileTypeInfo` poskytuje informáciu o koncovkách súborov tohto typu a o jeho popise pre dialógové okno uloženia alebo exportu. Rovnako udáva pomocou enumerácie `FilePurpose` aj účel súborového typu, či je určený na export alebo uloženie. Tieto funkcie sprostredkováva priamo trieda `FileType` a ponúka takisto statickú metódu, ktorá vráti typ pre zadanú koncovku.

Dôvod, prečo dáta dostupné cez `FileTypeInfo` nie sú uložené priamo v enumerácii `FileType`, teraz popíšem. Knižnica FreeHEP VectorGraphics definuje pre každý formát, do ktorého dokáže exportovať, podobné triedy

zhrňujúce informácie o formáte. Tie majú spoločného predka `ExportFileType` a okrem toho, že je v nich definovaná metóda vracajúca koncovky prináležiace danému formátu, obsahuje metódu, ktorá dokáže vrátiť komponentu typu `JPanel` reprezentujúcu formulár na parametrizáciu exportu do daného formátu. Keby boli informácie o type súboru priamo uložené v enumerácii `FileType`, museli by tu byť aj odkazy na tieto triedy z knižnice FreeHEP. Tie by však neboli definované pre tie členy enumerácie, ktorých účelom nie je export. Z toho dôvodu pre každý formát informácia o ňom implementuje rozhranie `FileTypeInfo` a navyše dedí od príslušných tried z knižnice FreeHEP. Diagram týchto vzťahov je prehľadne znázornený na obrázku 3.3.



Obrázok 3.3 Diagram hierarchie tried a rozhraní pre prácu s typmi súborov (modré obdĺžniky predstavujú triedy z knižnice FreeHEP VectorGraphics, obyčajné šípky dedičnosť, prerušované šípky implementáciu rozhrania a šípka s kosoštvorcom členskú premennú)

Enumeráciu `FileType` vo veľkom množstve využíva aj trieda `FileFilter`, ktorá je potomkom rovnomennej triedy zo štandardnej knižnice Swing. Tá slúži na filtrovanie podľa typu súborov v dialógových oknách pre výber súboru a po potvrdení sa podľa nej určí formát, do ktorého sa bude ukladať alebo exportovať. Jej rozšírenie implementuje aj statickú metódu, ktorá vráti všetky

svoje inštancie, ktoré slúžia pre daný účel (uloženie, export), a tým zjednodušuje a sprehľadňuje prácu pri zobrazovaní dialógového okna pre tieto dve výstupné operácie.

3.4 Nastavenia aplikácie

Doteraz boli všetky spomínané nastavenia určené priamo pre konkrétny dokument alebo nejakú časť dokumentu. PMLViewer však umožňuje meniť aj nastavenia, ktoré sú nezávislé na dokumente. Tieto nastavenia sú permanentne k dispozícii aj vtedy, ak nie je otvorený žiadny dokument. Podľa významu sa delia na dva druhy:

- nastavenia parsingu
- nastavenia grafu

Po technickej a implementačnej stránke medzi nimi výraznejšie rozdiely nie sú. Sú reprezentované triedami `PMLParseSettings` a `GraphSettings` a položky nastavení sú statické premenné. Na permanentné uloženie sa používa štandardná trieda Javy `Preferences`. Miesto, kam sa tieto nastavenia ukladajú, závisí od implementácie (`Preferences` je abstraktná trieda). Nastavenia sa načítajú v bloku `static`, ktorý sa volá na začiatku behu programu. Ich ukladanie prebieha vždy pri zavolaní „settera“ daného parametru. To sa pre všetky premenné oboch tried deje vždy pri potvrdení dialógového okna nastavení aplikácie (viac o jednotlivých nastaveniach v časti 2.8).

4 Porovnanie aplikácií PMLViewer a Tred

Toto porovnanie je aktuálne pre aplikácie PMLViewer vo verzii 1.0 a Tred vo verzii 1.3379.

Rozdiel medzi samotnými aplikáciami je napriek ich podobnosti značný. V prvom rade bola aplikácia PMLViewer vytvorená s cieľom čo najjednoduchšej intuitívnej obsluhy bez nutnosti programátorskej znalosti.

V ďalšom rade aplikácii PMLViewer absolútne chýba možnosť editácie načítaného dokumentu na významovej úrovni (nie prezentačnej), zatiaľ čo Tred takúto možnosť ponúka a za týmto účelom bol aj vytvorený. Naopak, PMLViewer bol vytvorený ako čisto vizualizačný nástroj. Z toho dôvodu by som sa venoval iba rozdielom v načítaní dokumentu, v možnostiach vizuálnych zmien a ich uložení, prípadne rozdielom v exporte. Najprv však porovnám výhodnosť použitia programovacích jazykov, v ktorých boli jednotlivé aplikácie napísané a náročnosť inštalácie.

4.1 Programovacie jazyky a spôsob inštalácie

Ako som už spomenul v úvode, pri programovaní aplikácie PMLViewer som kládol dôraz hlavne na jednoduchosť ovládania. Tá sa prejavuje už pri inštalácii, ktorá spočíva iba v prekopírovaní adresárovej štruktúry aplikácie PMLViewer na požadované miesto. Jedinou komplikáciou by mohla byť neprítomnosť Java Standard Edition Runtime Enviroment 6 (JRE), keďže aplikácia PMLViewer je vytvorená v programovaciom jazyku Java. Túto nutnú súčasť je však možné stiahnuť z oficiálnych stránok spoločnosti Sun a jednoducho doinštalovať. Aplikácií v jazyku Java je v dnešnej dobe už veľké množstvo a tento jazyk je veľmi využívaný v prostredí internetu, preto je vysoko pravdepodobné, že koncový užívateľ má JRE nainštalované.

Naproti tomu Tred, ktorý je vytvorený pomocou skriptovacieho jazyka Perl[39], má inštaláciu zložitejšiu. Nutnou prerekvizitou na spustenie programov napísaných v tomto jazyku je interpreter jazyka Perl. V operačných systémoch UNIX/Linux je väčšinou tento interpreter už obsiahnutý, v operačnom systéme Windows však nie, preto je ho nutné doinštalovať napríklad ako súčasť produktu ActivePerl[40] [11, kapitola 2.1]. Použitie aplikácií napísaných v jazyku Perl už oproti tým „javovým“ nie je natoľko rozšírené, takže pre užívateľov Windows je nutnosť inštalácie interpretera vysoko pravdepodobná.

Pri použití Tredu je takisto nutné doinštalovať potrebné knižnice Perlu. Aplikácii PMLViewer postačí samotný JRE, všetky potrebné knižnice sú súčasťou distribúcie aplikácie.

[39] http://www.perl.com/

[40] http://www.activestate.com/Products/activeperl/

Programovací jazyk Java je výhodnejší pri programovaní podobnej aplikácie aj preto, lebo jeho striktne objektový charakter dokáže dokonale zachytiť objektový model aplikácie.

Inou výhodou použitia jazyka Java je jednoduché možné prevedenie desktopovej aplikácie na applet.

4.2 Načítanie

Obe aplikácie dokážu bez problémov načítať závislostné stromy z PDT (súbory z a-vrstvy a t-vrstvy). Takisto si oba poradia so súbormi z m-vrstvy, avšak výsledne zobrazenie je nepatrne odlišné.

Výhodou aplikácie PMLViewer je možnosť doplniť chýbajúce role.

Obe aplikácie dokážu načítať aj dokumenty z PEDT. PMLViewer však potrebuje mať k dispozícii schémy, ktoré nie sú modulárne a obsahujú všetky potrebné pomenované typy, Tredu stačia aj modulárne a nekompletné schémy.

Oba programy si nedokážu dostatočne poradiť s novou aplikáciou PML – TectoMT[41]. Tá obsahuje pre každú vetu celý zväzok stromov, ktoré by mali byť zobrazené vedľa seba. PMLViewer ako aj Tred zobrazia v takomto prípade iba koreň.

Rýchlosť načítania je pri veľkosti dostupných súborov porovnateľná.

4.3 Ovládanie programu, vzhľad stromov a jeho zmena

Veľkou výhodou aplikácie PMLViewer je jeho intuitívne ovládanie. Všetky ponúkané zmeny idú vykonať niekoľkými kliknutiami myšou, nie je potrebné písať skripty alebo upravovať konfiguračné súbory. To však so sebou prináša nevýhodu nízkej variability.

V aplikácii Tred slúžia ako vizualizačná vrstva PML tzv. „stylesheety“. Sú to vlastne skripty jazyka Perl, ktoré sa pri zobrazovaní vyhodnotia a nastavia tak vizuálnu vrstvu zobrazovaného dokumentu. Výhoda takéhoto riešenia je taká, že môže existovať viac takýchto štýlov a dajú sa jednoducho zamieňať, pričom je možné zabezpečiť, aby sa všetky otvárané dokumenty zobrazovali s požadovaným štýlom. Nevýhoda je v nutnosti prepisovania „stylesheetu“ pri akejkoľvek požadovanej lokálnej zmene (napr. požiadavka na dočasné zvýraznenie uzlu).

Aplikácia PMLViewer má naproti tomu jeden implicitný štýl, ktorým zobrazuje všetky nevizuálne PML dokumenty (viac v časti 2.5.3). Umožňuje však prostredníctvom kontextového menu rýchlo meniť globálne a lokálne vizuálne nastavenia zobrazovaného stromu. Tieto zmeny je možné následne uložiť vo forme VPML dokumentu. Tak sa zachová špecifické vizuálne nastavenie, ale iba pre daný dokument. Ostatné dokumenty sa po načítaní zobrazia buď s vlastným vizuálnym štýlom, alebo s implicitným štýlom.

Výhoda formátu VPML je aj to, že je založený na XML, čiže akákoľvek iná aplikácia má možnosť takéto súbory spracovať. „Stylesheety“ pre Tred je

[41] Pre aplikáciu Tred však už existuje modifikácia, ktorá tieto dokumenty zobrazí.

možné jednoducho použiť iba pri aplikáciách napísaných v jazyku Perl, aj to však záleží od dátového modelu aplikácie.

Prínosom aplikácie PMLViewer je možnosť presúvaním uzlov rýchlo meniť vzhľad stromu a takisto možnosť automaticky zotriediť uzly troma rozličnými spôsobmi.

Na druhú stranu PMLViewer oproti Tredu nedokáže zobrazovať koreferenčné hrany.

4.4 Export a tlač

Aplikácia PMLViewer ponúka širšie pole obrázkových formátov, do ktorých umožňuje exportovať. Na rozdiel od aplikácie Tred je k dispozícii export do formátu SVG.

Aj funkcionalita exportu sa v aplikácii PMLViewer vyznačuje jednoduchým použitím, zatiaľ čo v Trede je trochu menej intuitívna (k exportu sa pristupuje cez položku tlače).

Práve neprítomnosť možnosti priamej tlače je jedným z nedostatkov aplikácie PMLViewer.

Záver

Cieľom tejto práce bolo vytvoriť aplikáciu, ktorá by bola schopná vizualizovať PML dáta. Toto sa mi podarilo dosiahnuť, pričom som splnil najdôležitejšiu požiadavku – požiadavky užívateľskej jednoduchosti a prehľadnosti.

V porovnaní so špecifikáciou aplikácie PMLViewer sa podarilo väčšinu požadovaných vlastností implementovať. Niektoré vlastnosti som však zámerne neimplementoval, napr.:

- možnosť tlače – zatiaľ postačuje široké spektrum formátov na export, tlač však bude v budúcnosti pridaná,
- stavový riadok – nedokázal som rozhodnúť, aké údaje by mal zobrazovať, neplnil by žiadny účel.

Naopak, nejaké vlastnosti oproti špecifikácii vo finálnej verzii pribudli, napr.:

- väčší počet formátov na export,
- možnosť spracovania zložkových stromov,
- možnosť doplnenia PML rolí pri parsingu.

Pri tvorbe programov vždy existujú časti kódu, ktoré by išli napísať lepšie, a takisto existujú vlastnosti, ktoré by bolo vhodné doplniť. Z tohto faktu sa nevymyká ani aplikácia PMLViewer. Niektoré nedostatky, ktoré budú do ďalšej verzie upravené som už spomenul v predchádzajúcom texte. Ide najmä o:

- lepší model PML schémy, aby umožňoval spracovanie vnorených schém a ukladanie zmien v schémach (napr. doplnenie rolí),
- kvalitnejšia chybová obsluha, vyššia vypovedacia schopnosť chybových správ.

Nakoniec je tu výber z vlastností, ktoré by mali byť do aplikácie PMLViewer implementované v budúcnosti:

- možnosť tlače,
- možnosť definovania pomenovaných štýlov a možnosť úpravy implicitného štýlu (štýl, ktorým sa zobrazujú čisté PML dokumenty),
- presunutie VPML rozšírenia PML do samostatného súboru a jeho následné spájanie s dokumentom pomocou XSLT transformácie,
- zobrazovanie koreferenčných hrán,
- atď.

Použitá literatúra

[1] Ústav formální a aplikované lingvistiky. (2006): Pražský závislostní korpus 2.0. *ÚFAL* [online]. [cit. 13. 4. 2008]. Dostupný z: <http://ufal.mff.cuni.cz/pdt2.0/index-cz.html>.

[2] Linguistic Data Consortium, University of Pennsylvania. (1995): Treebank-2. *LDC Catalog* [online]. [cit. 29. 3. 2008]. Dostupný z: <http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC95T7>.

[3] Pajas P., Štěpánek J. (2005): A generic XML-based format for structured linguistic annotation and its application to the Prague Dependency Treebank 2.0. ÚFAL/CKL, Praha.

[4] Pajas P. (2006): The Prague Markup Language (Version 1.1). *ÚFAL* [online]. [cit. 20. 4. 2008]. Dostupný z: <http://ufal.mff.cuni.cz/jazz/pml/doc/pml_doc.html>.

[5] Mlýnková I. a kol. (2006): Technologie XML. Nakladatelství Karolinum, Praha.

[6] World Wide Web Consortium. (2006): Extensible Markup Language (XML) 1.0 (Fourth Edition). *W3C* [online]. [cit. 20. 4. 2008]. Dostupný z: <http://www.w3.org/TR/REC-xml/>.

[7] Hunter J., McLaughlin B. (2007): JDOM - FAQ, Can an element or attribute name contain a colon? *JDOM* [online]. [cit. 21. 4. 2008]. Dostupný z: <http://www.jdom.org/docs/faq.html#a0250>.

[8] Wikipedia. (2008): Model-view-controller. *Wikipedia* [online]. [cit. 28. 4. 2008]. Dostupný z: <http://en.wikipedia.org/wiki/Model-view-controller>.

[9] Benson D. (2008): JGraph User Manual. *JGraph* [online]. [cit. 3. 5. 2008]. Dostupný z: <http://www.jgraph.com/pub/jgraphmanual.pdf>.

[10] Wikipedia. (2008): FreeHEP. *Wikipedia* [online]. [cit. 5. 5. 2008]. Dostupný z: <http://en.wikipedia.org/wiki/FreeHEP>.

[11] Pajas P. (2008): TrEd User's Manual. *ÚFAL* [online]. [cit. 23. 5. 2008]. Dostupný z: <http://ufal.mff.cuni.cz/~pajas/tred/ar01-toc.html>.