

Textový editor obohatený o grafické prvky

Tímový projekt



Tím: UFOPAK (č. 5.)

Vedúci projektu: Ing. Peter Drahoš

Autori:

Alexandra Adamíková

Andrej Fogelton

Ondrej Kallo

Peter Ondruška

Martin Palo

Jakub Ukrop

Akademický rok: 2009/2010

Obsah

1	Úvod	1
2	Analýza	2
2.1	Existujúce riešenia	2
2.2	Implementačné technológie	8
2.2.1	Qt toolkit	8
2.2.2	Scintilla	12
2.2.3	QScintilla	13
2.3	Doplnkové technológie	13
2.3.1	Lua	13
2.3.2	Rich Text Format	16
2.4	Zhrnutie	19
3	Špecifikácia	21
3.1	Funkcionálne požiadavky	21
3.2	Nefunkcionálne požiadavky	22
4	Návrh	24
4.1	Základné východiská	24
4.2	Alternatívy jadra aplikácie	25
4.3	Syntaktický analyzátor	28
4.4	Modul pre využitie značiek RTF	29

OBSAH

4.5	Architektúra systému	30
5	Záver	31
	Zoznam použitej literatúry	32

Kapitola 1

Úvod

Vytvorený dokument slúži pre projekt *Textový editor obohatený o grafické prvky* v rámci predmetu Tvorba softvérového/informačného systému v tíme. Keďže téma projektu stále poskytuje možnosti na ďalší výskum, výsledok nášho snaženia bude veľmi pravdepodobne využiteľný v praxi. Cieľom vzniknutej dokumentácie je popísať nami poňatý návrh riešenia pre danú oblasť.

Dokument je rozdelený na jednotlivé časti opisujúce analýzu, špecifikáciu a návrh. Časť analýza poskytuje pohľad na už existujúce aplikácie, rozoberá prostredia, v ktorých bude daný projekt vytvorený. Kapitola špecifikácie uvádza požiadavky na nami vytváraný editor a v návrhu nás dokument oboznamuje s našimi predstavami o funkcionalite a realizácii projektu.

Kapitola 2

Analýza

V tejto kapitole najskôr opíšeme existujúce editory zdrojových kódov, potom bližšie preskúmame technológie, ktoré plánujeme využiť pri návrhu a implementácii požiadaviek.

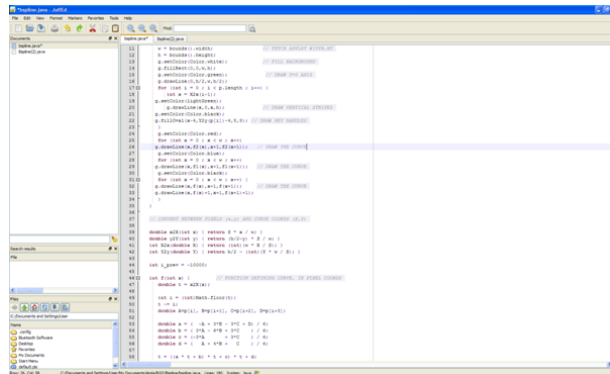
2.1 Existujúce riešenia

Táto časť je určená krátkemu a stručnému prehľadu niekoľkých editorov. Stručne popisuje ich základné vlastnosti. Menší prieskum vznikol na základe výberu témy pre tímový projekt ako pohľad na niečo už existujúce, no stále ešte doplniteľné. Medzi editormi sa nájdú jednoduchšie, s použitím len základných funkcií, ktoré sú kladené ako hlavné požiadavky potom sa nájdú aj projekty, ktoré sú väčšieho rozmeru a ponúkajú tak rozličné množstvo vlastností a možnej nadstavby.

JuffEd

Pokročilejší textový editor so zvýraznením syntaxe pre mnoho jazykov (C++,C#, CSS, Java, Lua, HTML, Perl, Ruby, SQL, Python, PHP, ...). Podporuje vyhľadávanie a nahradzovanie pomocou regulárnych výrazov, vrátane „multiline regexps“, výber blokov, dopĺňovanie kódu, označovanie riadkov, viac pohľadov na dokument a rôzne znakové sady. Funkcie editora je možné ďalej rozšíriť pomocou zásuvných modulov. Má príjemné používateľské

prostredie, je voľne dostupný, multiplatformový. je alternatívou k PsPad a Notepad++ editoru. JuffEd používa Qt4 knižnicu a editačnú komponentu Scintilla.



Obr. 2.1: JuffEd

qPEditor

Editor pre programátorov. Má dostupný kód pod GPL licenciou. Je viacplatformový, napísaný je úplne celý v Qt, takže minimálnou požiadavkou je Qt 4.2.x. Úprava štýlov alokácie pre rôzne programovacie jazyky, číslovanie riadkov, podpora undo/redo, má panel záložiek. Použitie v Qt Designer-i ako plugin.

eTextEditor (e)

Textový editor pre MS Windows s výkonnými funkciami pre úpravu. Umožňuje rýchlu a jednoduchú manipuláciu s textom, automatizuje všetku manuálnu prácu, čím vám napomáha lepšiemu sústredeniu sa na písanie. Môžete to pretiahnuť do akéhokoľvek jazyka. Medzi jeho pozoruhodné vlastnosti patrí osobný systém pre správu revízií, rozvetvené, viacstupňové, grafické undo, možnosť prevádzkovať TextMate zväzkov pomocou Cygwin. Významný prvok propagácie a marketingu „e“ je jeho schopnosť púšťať mnoho TextMate zväzkov priamo z repozitára MacroMates CVS.

„E“ podporuje viacnásobný výber textu. Ak je podržaný kláves Ctrl, potom dvojklik/viacnásobný výber slov, je vtedy možné editovať všetky tieto slová naraz. Vlastnosť

nájsť a premiestniť dáva okamžitú vizuálnu spätnú väzbu, zvýraznenie požiadaviek, ktoré sú písané. Táto vlastnosť je užitočná najmä pri používaní regulárnych výrazov. Keďže väčšina zväzkových príkazov sa spolieha na Unixové príkazy, ktoré nie sú k dispozícii pre Windows, e používa sadu nástrojov Cygwin. Zlou vlastnosťou je trochu pomalé načítanie, je to síce len pár sekúnd, no ak chcete rýchlo otvoriť súbor, tak budete nemilo prekvapení.



Obr. 2.2: eTextEditor

SciTE

Na prvý pohľad nevyzerá ako priateľský editor založený na Scintille. V SciTE [8] nenájdete žiadneho správcu súborov, Project Manager či integrovaného FTP klienta, je to teda čistý editor. SciTE môže držať viac súborov v pamäti naraz, pričom len jeden súbor bude viditeľný. SciTE zvýrazňuje syntax a podporuje množstvo jazykov (HTML, PHP, SQL, CSS, Java, ...).

Má všetky funkcie pre vytváranie kódu, ktoré sú potrebné a ďalšie je možné vďaka otvorenému zdrojovému kódu dopísať. Má dve sekcie, sekciu pre editáciu a výstupnú sekciu. Výstup sa nachádza na pravej strane panela úprav alebo pod ním. Spočiatku má nulovú hodnotu, ale môže narastať ťahaním deliča medzi ním a editačnou časťou. Obdĺžnikové bloky textov je možné vybrať podržaním klávesy Alt, zatiaľ čo je myš ťahaná ponad text. Používajú sa rôzne funkcie ako skratky, nápoveda, editačné možnosti, vyhľadávanie, pohyb

kurzora, kompilácia, dopĺňanie textu, makrá, komentáre, zobrazenie výstupu.

Tu uvádzame krátky prehľad základných a často používaných vlastností:

Skratky Napíšete slovo, stlačíte klávesu `Ctrl+B` a rozvinie sa skratka, napr. `if` môže byť namapované, ako `if (|) {\n\t|\n}`. Ich využitie je efektívne z hľadiska času, ak označíme kus kódu, stlačíme klávesy `Ctrl+Shift+R`, napíšeme `if` a kód sa obalí kompletnou konštrukciou `if`.

Nápoveda Kláves `F1` zobrazí nápovedu k funkcii, na ktorej je kurzor. Aj tu je možnosť namapovať si pre ľubovoľný jazyk to, čo vám najviac vyhovuje.

Editačné možnosti Základné editačné možnosti sú samozrejmosťou. Duplikácia riadka pomocou `Ctrl+D` či jeho prehodenie s predchádzajúcim riadkom `Ctrl+T`.

Vyhľadávanie `Ctrl+F3` vyhľadá slovo pod kurzorom alebo označený text. `Ctrl+Shift+F` vyhľadáva vo viac súboroch štandardnými nástrojmi `grep` alebo `findstr`. Je možné doplniť si aj vlastnú funkciu na vyhľadávanie, teda môžete napríklad vyhľadávať len v reťazcoch a text nájdený inde sa odignoruje.

Pohyb kurzora Klávesová skratka `Ctrl+E` presunie kurzor k odpovedajúcej zátvorke. Šikovná je aj funkcia pre prechod medzi časťami slov, na rozdiel od `Ctrl+šípky` zohľadňuje aj podčiariť a zmeny veľkosti písmen v slove či odseku (bloky textu oddelené prázdny riadkom).

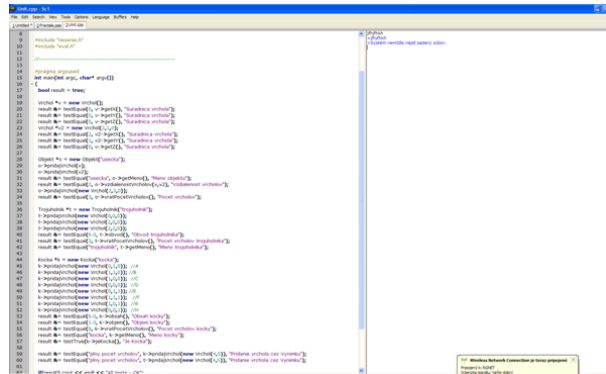
Kompilácia Skontrolovanie syntaxe a prenesenie na riadok, kde sa daná chyba nachádza.

Dopĺňanie textu `Ctrl+Space` doplní slovo z pevného zoznamu a `Ctrl+Enter` potom zo slov obsiahnutých v zozname.

Makrá Funkčnosť je možné rozširovať makrami písanými v jazyku Lua.

Komentáre `Ctrl+Q` prehodí zakomentovanosť označených riadkov, `Ctrl+Shift+Q` zakomentuje označený text.

Zobrazenie výstupu Výstup externých programov sa zobrazuje v samostatnom okne priamo v rámci editora. Okno sa dá zapnúť či vypnúť pomocou klávesy F8.



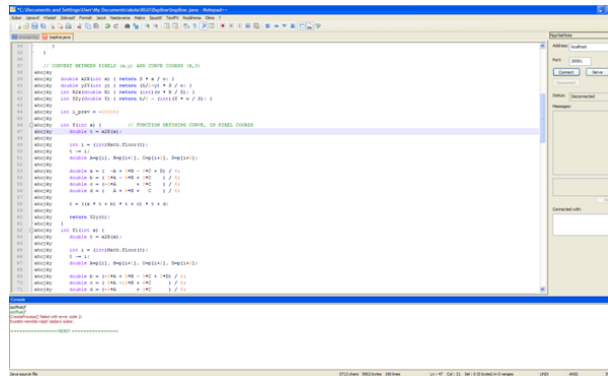
Obr. 2.3: SciTE

Notepad++

Voľne dostupný editor zdrojového kódu [4], ktorý aj podporou viacerých jazykov nahrádza Notepad. Beží v prostredí MS Windows pod licenciou GPL. Avšak stane sa viacplatformovým pomocou využitia softvéru, napr. WINE. Je založený na komponente Scintilla a napísaný v jazyku C++ a využíva čisté Win32 API a STL, ktoré zabezpečuje vyššiu rýchlosť a menšiu veľkosť programu.

Zaujímavosťou je, že optimalizáciou postupov (bez straty užívateľskej prívetivosti) sa Notepad++ snaží znížiť emisie oxidu uhličitého. Pri menšom použití napájania procesora, môže počítač ubrať a znížiť spotrebu energie, čo má za následok zdravšie prostredie. Cieľom programu je ale ponúknuť jednoduchý a efektívny spôsob úpravy kódov s úplne používateľnými GUI.

Podporuje zvýraznenie syntaxe pre 44 jazykov, skriptovacie a značkovacie jazyky. Užívateľia môžu tiež definovať svoj vlastný jazyk pomocou zabudovaného plugin-u. Pre väčšinu podporovaných jazykov môže užívateľ urobiť svoj vlastný zoznam API (alebo stiahnuť API súbory zo sekcie). Akonáhle je API súbor pripravený, zadajte Ctrl+Space na začatie tejto akcie.



Obr. 2.4: Notepad++

Podporuje multi-dokument, čo umožňuje úpravu viacerých dokumentov naraz. Poskytuje dva pohľady v rovnakom čase. To znamená, že môžete zobraziť dva rôzne dokumenty súčasne. Môžete vizualizovať (editovať) v dvoch náhľadoch jeden dokument a v dvoch rôznych pozíciách. Úprava dokumentu v jednom zobrazení sa bude vykonávať v inom náhľade.

Hľadanie a nahrádzanie reťazca v dokumente pomocou regulárnych výrazov. Úplná podpora drag-and-drop. Môžete otvoriť dokument pomocou tejto funkcie, presunúť tak dokument z pozície. Užívateľ si môže nastaviť pozíciu pohľadov dynamicky (len v režime dvoch zobrazení: oddeľovač môže byť nastavený horizontálne alebo vertikálne). Ak máte upraviť či vymazať súbor, ktorý sa otvoril v Notepad++, ste upozornení na aktualizáciu dokumentu (reload súbor alebo odstránenie súboru). Možnosť funkcie priblíženia a oddialenia, ktorá je zložkou Scintilly.

Podporuje viacjazyčné prostredie. Takže je možné používať aj čínštinu, hebrejčinu, kórejčinu či arabčinu, no ruštinu už tak nepodporuje. Poskytuje funkciu záložky, kde si užívateľ môže kliknúť na rozpätie alebo pomocou Ctrl+F2 prepínať návestia. Pre dosiahnutie záložiek stačí stlačiť F2 (ďalšie záložky), alebo Shift+F2 (predchádzajúca záložka). Vymazanie všetkých záložiek sa koná pomocou Menu, kde kliknete na Hľadať -> Odstrániť všetky záložky. Ak vsuvka zostane pri jednom zo symbolov { } () [] , symbol vedľa vsuvky a jeho opak budú zvýraznené, rovnako ako smernice za účelom ľahšieho nájdenia bloku.

2.2 Implementačné technológie

V tejto časti si predstavíme dva implementačné nástroje, ktoré prichádzajú do úvahy pri vývoji editora. Sú nimi nástroj *Qt toolkit 4.0* a *Scintilla*, respektíve *QScintilla*. Najskôr predstavíme jednotlivé technológie, pričom sa zameriame najmä na funkcie, techniky a metódy, ktoré by sme mohli využiť pri implementácii. Pri tejto analýze sa budeme venovať najmä nástrojom pre spracovanie textu a prácu s grafikou, keďže tieto oblasti budú kľúčové pri vývoji aplikácie.

Predbežne by sme chceli použiť nástroj Qt toolkit, ktorý by v prípade nedostatočnej funkcionality interných tried pre prácu s textom mohol byť doplnený o funkcie nástroja Scintilla pomocou portu QScintilla. Podľa získaných poznatkov z analýzy teda určíme, či použijeme pre vývoj editora len nástroj Qt alebo obidva nástroje skombinujeme.

2.2.1 Qt toolkit

Qt toolkit [6] je implementačný nástroj založený na jazyku C++. Je to technológia, pomocou ktorej je možné vyvíjať aplikácie pre rôzne platformy. Qt umožňuje vytvárať a jednoducho nasadzovať aplikácie pre počítače, mobilné telefóny, ale aj vnorené systémy (MP3 prehrávače), bežiace pod operačnými systémami Windows, Linux, MAC OS, Symbian. Multiplatformovosť je práve jedna z rozhodujúcich výhod, kvôli ktorým chceme implementovať editor pomocou tohto nástroja. Qt toolkit je v súčasnosti dostupný pod týmito licenciami:

- Komerčná
- Qt GNU LGPL v 2.1
- Qt GNU GPL v 3.0

Pre potreby školského projektu nám postačuje licencia GNU GPL v 3.0 (General Public License). Táto licencia umožňuje tvoriť aplikácie s otvoreným kódom (open-source), mimo komerčného využitia.

Nástroj Qt ponúka okrem množstva tried a knižníc pre tvorbu GUI aplikácií aj vlastné vývojové prostredie (IDE) *Qt Creator*. Možnosti práce s nástrojom Qt sú nasledovné:

- Qt plugin pre vývojové prostredie Eclipse
- Qt plugin pre vývojové prostredie Visual Studio
- knižnice Qt a zdrojové kódy v nejakom inom vývojovom prostredí
- integrované vývojové prostredie Qt Creator

Rozhodli sme sa pre použitie prostredia Qt Creator.

Práca s obohateným textom

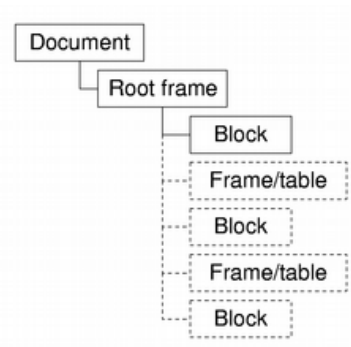
Qt toolkit poskytuje množstvo užitočných tried pre prácu s textom. Na oficiálnej stránke Qt [6] je práca s obohateným textom (rich text processing) uvedená ako kľúčová technológia. Tímto je daný predpoklad, že spomenutá funkcionálna bude v Qt bohate zastúpená. Qt obsahuje triedy pre čítanie a manipuláciu s dokumentami so štruktúrovaným obohateným textom. Údaje v rámci dokumentu môžu byť sprístupnené pomocou dvoch rozhraní:

- **Cursor-based interface** používa sa pre upravovanie textu, štruktúra dokumentu je pritom zachovaná.
- **Read-only hierarchical interface** poskytuje celkový pohľad na štruktúru dokumentu a umožňuje operácie nad textom spojené s vyhľadávaním, ukladaním alebo tlačením dokumentu.

Štruktúra dokumentu s obohateným textom poskytuje pohľad na to, aké elementy spolu-vytvárajú samotný dokument, a akým spôsobom sú tieto elementy usporiadané v dokumente. Základný model štruktúry dokumentu je načrtnutý na Obrázku 2.5.

Každý dokument má základný element **Root frame**, ktorý obsahuje najmenej jeden textový blok. Elementy **Frame** (rámy) a **Table** (tabuľky) sú vždy oddelené elementom **Block** (blok textu), aj keď ten nemusí obsahovať žiadne informácie. To umožňuje vkladanie nových elementov medzi už existujúce elementy.

Rozsiahle nástroje pre prácu s textom využiteľné v editore poskytujú najmä nasledovné triedy:



Obr. 2.5: Základná štruktúra dokumentu s obohateným textom

`QTextEdit` poskytuje funkcie pre zobrazovanie a editáciu obyčajného (plain text) a obohateného textu (rich text). Samotný text môže byť vkladajú použitím triedy `QTextCursor` alebo použitím funkcií triedy `QTextEdit` (napríklad `insertHtml()`, `insertPlainText()`, `append()` alebo `paste()`). Pomocou triedy `QTextCursor` je možné napríklad vkladať objekty ako tabuľky, zoznamy alebo obrázky do dokumentu, vyhľadávať a označovať text.

`QTextDocument` predstavuje kontajner pre dokumenty s formátovaným obohateným textom. Je to základná trieda pre prácu s obohateným textom.

`QSyntaxHighlighter` umožňuje definovať pravidlá pre zvýrazňovanie textu. Zvýrazňovanie textu je užitočné najmä pri práci so zdrojovými kódmi, čo je aj prípad nášho editora. Zlepšuje čitateľnosť kódu a pomáha jednoduchšie odhaliť syntaktické chyby v kóde.

Práca s grafikou

Pri implementácii editora bude dôležité vyriešiť problémy ohľadom vykresľovania grafických elementov v aplikácii. Keďže cieľom je vytvoriť editor obohatený o grafické prvky, bude táto oblasť jedna z najdôležitejších. Riešených bude hneď niekoľko problémov, ako napríklad reprezentovanie blokov kódu pomocou grafických elementov (napr. obdĺžniky), presúvanie celých blokov (drag-and-drop) alebo skrývanie blokov (folding). Qt toolkit po-

skytuje pre vývojárov niekoľko tried pre prácu s grafikou. Teraz stručne predstavíme tie, ktoré by sme mohli využiť.

Základnými triedami v Qt toolkit pre vykresľovanie geometrických útvarov a prácu s nimi sú `QPainter`, `QPaintDevice`, `QPaintEngine`, `QGraphicsScene`, `QGraphicsView` a `QGraphicsTextItem`.

`QPainter` poskytuje optimalizované funkcie pre vykresľovanie v GUI aplikáciach od kreslenia čiar, až po komplexné tvary ako diagramy a grafy.

`QPaintDevice` predstavuje podklad, na ktorý je možné pomocou triedy `QPainter` kresliť.

Takýmto podkladom môže byť napríklad inštancia triedy `Widget` (základný grafický element každej GUI aplikácie v QT – okno) alebo `QImage` (reprezentuje obrázok).

`QPaintEngine` je abstraktná trieda, ktorá poskytuje definíciu toho, ako môžeme pomocou triedy `QPainter` kresliť na určité zariadenie na určitej platforme. Qt 4.0 poskytuje niekoľko implementácií tejto triedy pre OpenGL či PostScript. Programátor si však môže vytvoriť aj svoju vlastnú implementáciu, napríklad pre PDF.

`QGraphicsScene`, `QGraphicsView` a `QGraphicsTextItem` Tieto tri triedy poskytujú funkcionality, ktorá by mohla byť využiteľná pre grafickú reprezentáciu blokov textu. Trieda `QGraphicsScene` poskytuje možnosti pre riadenie a dohľad nad veľkým množstvom 2D grafických prvkov. Služi ako scéna pre grafické prvky. Tie môžu byť vytvorené, buď volaním jednej z metód tejto triedy (`addEllipse()`, `addLine()`, ...), alebo pridaním inštancie triedy `QGraphicsItem` do scény. Spolu s triedou `QGraphicsView` sa `QGraphicsScene` používa napríklad aj pre vizualizáciu a riadenie textových prvkov.

V `QGraphicsScene` môžeme určovať napríklad pozíciu jednotlivých prvkov, ich viditeľnosť, priehľadnosť, poradie na pozadí/popredí a ošetrovať udalosti (event handling). Hlavná sila tejto triedy spočíva v tom, že dokáže veľmi rýchlo lokalizovať objekt aj na scéne s obrovským množstvom objektov (rádovo milióny). Táto trieda však nemá žiadne vizualizačné schopnosti.

Pre vizualizáciu objektov slúži trieda `QGraphicsView`. Tá zobrazuje grafické objekty v rolovacom okne a jej funkcionálnosť spočíva napríklad aj v selektovaní, priblížení objektu alebo v rolovaní scény. Inštancia triedy `QGraphicsTextItem` predstavuje textový formátovateľný objekt, ktorý je možné pridať do scény (na inštanciu `QGraphicsScene`).

Drag-and-drop

Technológia drag-and-drop (potiahni a pušť) bude použitá napríklad pri premiestňovaní blokov kódu v editore, ale aj na presun textu medzi oknami, respektíve medzi editorom a inými aplikáciami. Funkcie pre túto technológiu poskytuje už základná trieda všetkých aplikácií `QApplication` (nastavenie času držania tlačidla myši, po ktorom sa inicializuje ťahanie, atď.), avšak primárnymi nástrojmi pre túto technológiu v Qt sú trieda `QDrag` a funkcie triedy `QWidget`.

2.2.2 Scintilla

Scintilla [8] je nástroj, založený na jazyku C++ vyvinutý Neilom Hodgsonom. Je určená pre vývoj aplikácií, ktoré sa zaoberajú editáciou textu, a preto je vhodná aj pre implementáciu editora zdrojových kódov.

V rámci práce so zdrojovými kódmi poskytuje Scintilla možnosti pre editáciu a odstraňovanie chýb (debugging) kódu. V tom je zahrnutá podpora pre formátovanie syntaxe, indikácia chýb, dopĺňanie kódu a automatického pomocníka (editor zobrazí tipy na volanie funkcie). Okrem toho umožňuje použiť značky pre označenie aktuálneho riadku alebo bod prerušenia (breakpoint). Umožňuje tiež zvýraznenie textu pomocou tučného písma, kurzívy, nastavenia pozadia alebo fontu písma, čo u väčšiny editorov zdrojových kódov nebýva možné.

Nevýhodou tohto nástroja je, že na rozdiel od Qt nepodporuje v takej miere multiplatformovosť a nasadenie aplikácií vytvorených pomocou nej na rôznych platformách si vyžaduje viac práce vývojára. V náš prospech však hrá fakt, že v súčasnosti existuje vstupná

brána (port) Scintilly pre Qt — QScintilla (popísaná v ďalšej časti), takže je možné použiť podstatnú časť jej funkcionality v aplikáciách Qt.

Pre demonštráciu možností nástroja Scintilla bol vytvorený editor zdrojových kódov SciTE. Funkcionalita tohto editora je bližšie popísaná pri analýze existujúcich editorov v časti 2.1.

2.2.3 QScintilla

Nástroj QScintilla [5] poskytuje široké možnosti v oblasti práce s textom, najmä so zdrojovými kódmi. Chceme zanalyzovať jej možnosti a zistiť, či by nebolo vhodné skĺbiť funkcie nástrojov QScintilla a Qt v našom editore.

QScintilla ponúka pre vývojárov rovnaké typy licencií ako Qt, je teda prístupná aj pod licenciou GNU GPL a vo verzii 2 podporuje Qt v3 aj Qt v4.

QScintilla obsahuje viac ako 30 tried — tzv. lexerov, ktoré umožňujú lexikálnu analýzu jednotlivých programovacích jazykov. Pre každý jazyk je vytvorená samostatná trieda, napríklad `QsciLexerJava`. QScintilla teda značne uľahčuje vývoj editorov, ktoré podporujú lexikálnu analýzu veľkého množstva jazykov.

Okrem lexerov poskytuje QScintilla aj niektoré triedy implementujúce prostriedky pre tvorbu GUI a komplexnejšiu prácu s textom (napríklad triedy `QsciDocument` — reprezentácia dokumentu, `QsciScintilla` – API pre Scintilla na báze Qt, `QsciPrinter` — podtrieda Qt triedy `QPrinter` pre tlač dokumentov). Tie majú však menšiu funkcionality ako paralelné nástroje v Qt.

2.3 Doplnkové technológie

2.3.1 Lua

Lua [9] je rýchly procedurálny skriptovací jazyk, určený hlavne na vnorené používanie. Programátorské rozhranie (API) je navrhnuté tak, aby umožňovalo integráciu s programami napísanými v iných jazykoch (C, C++, Java, C#, ...) vrátane skriptovacích (Perl,

Ruby).

Filozofiou jazyka Lua je *jednoduchosť a rozšíriteľnosť*, obsahuje základnú funkcionálnu a (meta)mechanizmy ako dodefinovať čokoľvek, čo považujeme za potrebné. Týmto spôsobom je možné získať aj schopnosti objektovo-orientovaných (rozhrania, dedenie) alebo funkcionálnych jazykov. Lua je dynamicky typovaná a obsahuje niekoľko atomických dátových typov doplnených o jednu dátovú štruktúru – tabuľku. Tabuľka funguje ako asociatívne pole a jej pomocou je možné simulovať iné štruktúry (pole, množina, hash tabuľka, strom, atď.) a tiež objekty v zmysle OO paradigmy.

Lua patrí medzi najrýchlejšie skriptovacie jazyky [2]. Je implementovaná v štandardnom ANSI (ISO) C, čo sa prejavuje na jej vysokej prenositeľnosti - funguje pod všetkými známymi platformami. Výhodou Lua je jej veľkosť (aktuálna verzia Lua 5.1.4 má 860K aj s dokumentáciou), vďaka ktorej nie je problém pripojiť ju celú k aplikácii, ktorá to potrebuje.

Lua je vyvíjaná pod voľnou licenciou (MIT) a môže byť používaná zdarma na akékoľvek (aj komerčné) účely. Lua sa dnes často používa pri skriptovaní počítačových hier, ale využívajú ju aj iné programy, ako napríklad Wireshark alebo VLC media player.

LPeg

LPeg [3] je knižnica jazyka Lua určená na hľadanie vzoriek v texte (*pattern matching*). Snaží sa odstrániť problémy spojené s používaním regulárnych výrazov, ktoré môžu byť pri komplikovanejších úlohách neprehľadné. Je postavený na gramatikách typu PEG (*Parsing Expression Grammar*), formalizme podobnom bezkontextovým gramatikám. Na rozdiel od bežných gramatík, PEG nedefinuje jazyk, ale algoritmus na jeho rozpoznanie.

LPeg poskytuje dva moduly s rozličným spôsobom práce. Stručne popíšeme rozdiely medzi nimi a uvedieme ukážky syntaxe. Kompletnú syntax uvádza domovská stránka [1]. V prvom module `re` (skratka z *regex*) sú vzory popisované reťazcami so syntaxou odvodenou z regulárnych výrazov. Ako príklad uveďme funkciu `match(text, pattern)`, ktorá sa

pokúša nájsť vzor v prefixe reťazca a vráti index prvého nezhodujúceho sa znaku¹.

```
print(re.match("more words", "[a-z]+"))
--> 5, hladame 1 a viac znakov od 'a' po 'z'}
```

Vzory môžeme opísať aj gramatikou, nasledujúci kód rozoberá aritmetický výraz:

```
p = [[
  expression <- <factor> ([+-] <factor>)*
  factor      <- <term> ([*/] <term>)*
  term        <- <number> / "(" <expression> ")"
  numer       <- [0-9]+
]]
print(re.match("13+(22-15)", p)) --> 11
```

Druhý modul `lpeg` pracuje so vzormi ako s premennými vlastného dátového typu a obsahuje viac spôsobov na ich vytváranie a spájanie. Kód pre spracovanie aritmetického výrazu bude teraz nasledovný:

```
p = lpeg.P{ "expression",
  expression = lpeg.V("factor") * (lpeg.S("+-") * lpeg.V("factor"))^0,
  factor      = lpeg.V("term") * (lpeg.S("*/") * lpeg.V("term"))^0,
  term        = lpeg.V("number") + "(" * lpeg.V("expression") * ")",
  number      = lpeg.R("09")^1
}
print(lpeg.match(p, "13+(22-15)")) --> 11
```

Neterminálne symboly, rozsahy a množiny znakov sú definované pomocou samostatných funkcií. Tento zápis je pre jednoduché gramatiky príliš komplikovaný, ale dovoľuje znovupoužitie už definovaných vzorov, rovnako ako využitie všetkej funkcionality jazyka Lua.

¹Lua indexuje od 1.

Obidva moduly podporujú vyhľadávanie (vyjadrené priamo vzorom) rovnako ako zachytávanie reťazcov na pokročilej úrovni. Vybraný text je možné ukladať do tabuliek, ľubovoľne zamieňať a inak transformovať. LPeg používa tzv. *limitovaný backtracking*, vďaka ktorému je veľmi rýchly a efektívny [3].

2.3.2 Rich Text Format

Rich Text Format(RTF) je metóda slúžiaca na zakódovanie formátovaného textu a obrázkov v textovom dokumente. RTF bolo vyvinuté pre prenášanie dokumentov medzi rôznymi platformami bez straty formátovania.

Syntax RTF

Každý RTF súbor obsahuje neformátovaný text, riadiace slová, riadiace symboly a grupy. Pre zjednodušenie prenositeľnosti štandardný RTF dokument obsahuje 7-bitové znaky. *Riadiace slovo* je špeciálne formátovaný príkaz, ktorý sa používa na označenie riadiaceho kódu a informácií používaných pri manažovaní zobrazenia dokumentov. Riadiace slovo má maximálnu dĺžku 32 znakov a jeho forma je:

```
\LetterSequence<Delimiter>
```

Každé riadiace slovo začína spätným lomítkom (backslash). Nasleduje postupnosť písmen (LetterSequence) tvorených malými písmenami v rozsahu „a“ až „z“ vrátane. RTF je citlivý na veľkosť písmen a každé riadiace slovo musí byť tvorené malými písmenami. Nakoniec nasleduje oddeľovač (Delimiter), ktorý označuje koniec riadiaceho slova. Môže ho tvoriť:

- medzera — v tomto prípade je medzera súčasťou riadiaceho slova
- číslica alebo pomlčka — ktoré indikujú následnosť číselného parametru.

Následná postupnosť čísel je oddelená medzerou alebo ľubovoľným znakom rôznym od písmena alebo číslice. Parameter môže byť pozitívne alebo negatívne číslo v rozsahu -32767 až 32767. Ak nasleduje číselný parameter bezprostredne za riadiacim slovom, stáva sa jeho

súčasťou. Riadiace slovo je potom oddelené medzerou alebo nepísmenovým či nečíselným znakom rovnakým spôsobom, ako je to u ostatných riadiacich slov.

Pokiaľ je medzera oddeľovačom riadiaceho slova, neobjaví sa v dokumente. Každý nasledujúci znak za oddeľovačom vrátane medzier sa v dokumente objaví. Z tohto dôvodu môžeme používať medzery len tam, kde to je naozaj potrebné, v žiadnom prípade na rozdeľovanie RFT kódu.

Riadiaci symbol pozostáva zo spätného lomítka a jedného nepísmenkového znaku. Napríklad `\~` reprezentuje nedeliteľnú medzeru. Riadiace symboly neobsahujú žiaden oddeľovač.

Grupy pozostávajú z textu a riadiacich slov alebo symbolov uzatvorených v zátvorkách (`{}`). Otváracia zátvorka udáva začiatok grupy a uzatváracia zátvorka udáva koniec grupy. Každá grupa špecifikuje text ňou ovplyvnený s rôznymi atribútmi textu. RTF súbor môže obsahovať grupy pre fonty, štýly, farbu, komentáre a podobne.

Vlastnosti niektorých riadiacich slov (tučné, kurzíva, ...) majú len dva stavy. Pokiaľ má riadiace slovo nenulový parameter, rozpoznáva sa, že dané riadiace slovo vlastnosť zapína. Keď má riadiace slovo parameter 0, rozpoznáva sa, že riadiace slovo vypína danú vlastnosť. Napríklad `\b` zapína tučné písmo a `\b0` ho vypína.

Riadiace slová, symboly a zátvorky tvoria riadiace informácie. Všetky ostatné znaky súboru sú chápané ako čistý text. V nasledujúcom príklade je príklad čistého textu, ktorý sa nenachádza v grupe:

```
{\rtf\ansi\deff0
  {\fonttbl
    {\f0\froman Tms Rmn;}
    {\f1\fdecor Symbol;}
    {\f2\fswiss Helv;}
  }
  {\colortbl;\red0\green0\blue0;\red0\green0\blue255;
    \red0\green255\blue255;\red0\green255\blue0;
```

```
\red255\green0\blue255;\red255\green0\blue0;
\red255\green255\blue0;\red255\green255\blue255;}
{\stylesheet
  {\fs20 \snext0Normal;}
}
{\info
  {\author John Doe}
  {\creatim\yr1990\mo7\dy30\hr10\min48}
  {\version1}{\edmins0}
  {\nofpages1}{\nofwords0}{\nofchars0}{\vern8351}
}
\widoctrl\ftnbj
\sectd\linex0\endnhere \pard\plain
\fs20 This is plain text.\par
}
```

Fráza „This is the plain text.“ nie je súčasťou grupy a chápe sa ako text dokumentu. Ako sme už skôr spomenuli, spätné lomítko a zátvorky majú pre RTF špeciálny význam. Ak ich chceme použiť v texte, musíme pred ne napísať spätné lomítko.

Komentáre v RTF

Komentáre v RTF sa skladajú z dvoch častí:

- ID autora — uvádza sa riadiacim slovom `\atnid`
- Text komentáru — uvádza sa riadiacim slovom `\annotation`

Presnú syntax komentáru si môžeme pozrieť v špecifikácii [7]. Spôsob použitia komentáru si môžeme pozrieť v nasledujúcom príklade:

```
{\insrsid8729657 An example of a paradigm might be Darwinian biology.}
{\cs15\v\fs16\insrsid8729657
```

```
{*\atnid JD}
{*\atnauthor John Doe}\chatn
{*\annotation
  {*\atndate 1180187342}
  \pard\plain \s16\ql \li0\ri0\widctlpar\aspalpha\aspnum\faauto
  \adjustright\rin0\lin0\itap0 \fs20\lang1033\langfe1033\cgrid
  \langnp1033\langfenp1033
  {\cs15\fs16\insrsid8729657 \chatn }
  {\insrsid9244585 How about some examples?}
}
}
```

V komentári sa môže nachádzať aj časová pečiatka (označená riadiacim slovom `\atntime`), dátumová pečiatka (`\atndate`) alebo ikona (`\atnicn`). Tieto údaje sú však voliteľné a komentár ich nemusí obsahovať.

2.4 Zhrnutie

Zistili sme, že existujúce editory síce disponujú veľkým množstvom rozličnej funkcionality, ale nevyužívajú, okrem farebného zvýraznenia textu, žiadne grafické prvky. Napriek tomu, že mnohé z nich sú rozšíriteľné pomocou zásuvných modulov, sme sa vzhľadom na povahu nášho riešenia rozhodli nevyužiť tieto možnosti.

Na základe bližšej analýzy nástrojov, ktoré prichádzajú do úvahy pre vývoj editora sme dospeli k niekoľkým záverom. V prvom rade sme sa rozhodli, že ako vývojový nástroj budeme používať Qt toolkit, pričom sa budeme snažiť využiť natívne triedy tohto nástroja v čo najväčšej možnej miere. V zálohe tiež budeme mať použitie nástroja QScintilla, najmä v oblasti lexikálnej analýzy a formátovania textu. Samotný nástroj Scintilla pri vývoji nebudeme využívať, vzhľadom na zníženú multiplatformovosť.

Jazyk Lua v spojení s knižnicou LPeg spĺňa naše požiadavky na rýchlosť aj vyjadrovaciu

silu (gramatiky) a umožní nám efektívne vytvoriť syntaktický analyzátor. Na ukladanie dokumentácie a jej formátovania do zdrojového kódu využijeme existujúce značky formátu RTF, ktoré postačujú našim potrebám.

Kapitola 3

Špecifikácia

V tejto kapitole sa venujeme požiadavkám na vytváraný editor. Kapitola je rozdelená na dve časti. V prvej časti sa zaoberáme jednotlivými funkcionálnymi požiadavkami, ktoré uľahčia a zefektívnia prácu používateľa nášho editoru. V druhej časti si predstavíme nefunkcionálne požiadavky na náš editor, ktoré tvoria tiež jeho neoddeliteľnú súčasť.

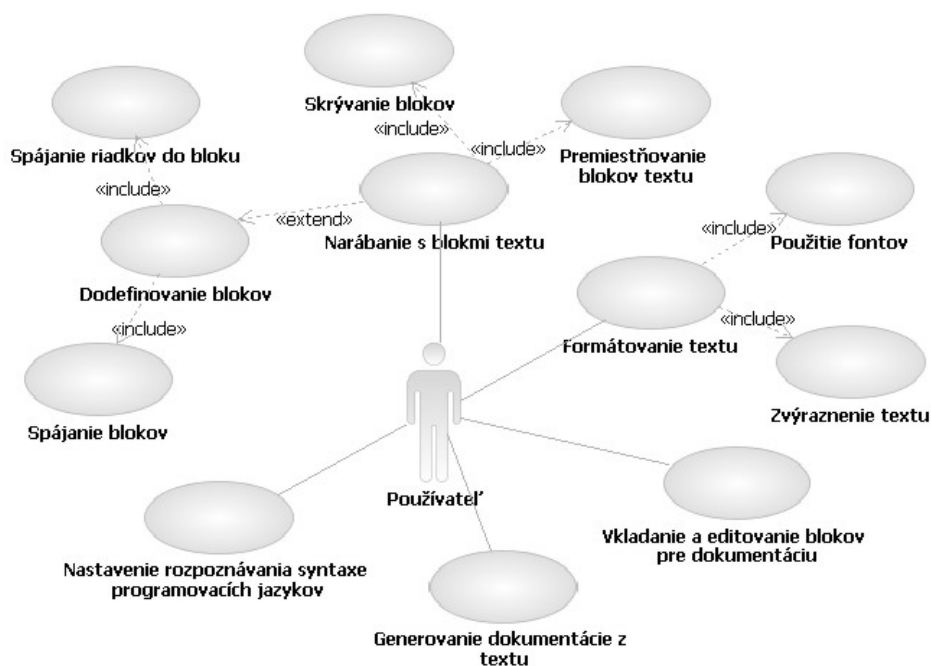
3.1 Funkcionálne požiadavky

Funkcionálne požiadavky vidíme na diagramoch prípadov použitia (use-case) uvedených nižšie (Obrázky 3.1, 3.2).

- Rozšírenie zvýraznenia textu základného editora o použitie fontov, farieb a grafických elementov sprehľadní celkovú čitateľnosť kódu a zefektívni prácu programátora pri editácii zdrojových súborov.
- Automatické rozpoznávanie syntaxe daného jazyka počas editácie textu umožní programátorovi ľahko odhaliť prípadné preklepy vzniknuté pri písaní tak, ako sme na to zvyknutí pri dnes používaných vývojových prostrediach.
- Okrem automatického rozpoznávania bežných kľúčových slov programovacích jazykov, bude editor rozpoznávať aj celé najčastejšie používané bloky kódu, ako sú napríklad funkcie, cykly, podmienky a graficky ich zvýrazní pomocou farebného bloku,

čo zvýši celkovú logickú čitateľnosť kódu.

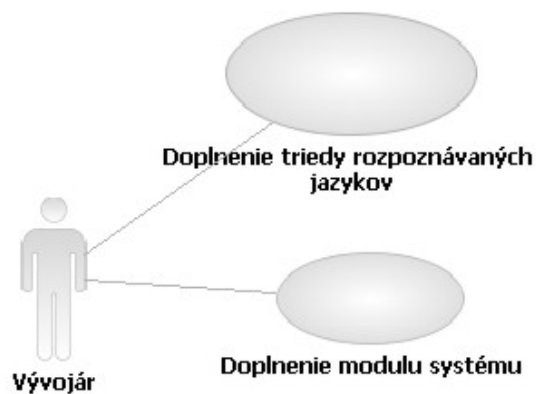
- Takto rozpoznané bloky kódu sa budú dať jednoducho zmenšiť na jeden riadok, čím zvýšime prehľadnosť veľmi dlhých a štruktúrovaných programov.
- Používateľ bude mať možnosť pracovať s celými blokmi kódu, a nielen so samotnými riadkami. Pre jednotlivé bloky môže programátor veľmi jednoducho využiť systém drag-and-drop a presunúť blok na inú pozíciu.
- Používateľ bude môcť vložiť do kódu špeciálne bloky slúžiace na dokumentovanie. V týchto blokoch sa bude písať dokumentácia priamo k zdrojovým kódom, a tá následne vygenerovať do dokumentu v požadovanej forme.



Obr. 3.1: Prípady použitia pre používateľa

3.2 Nefunkcionálne požiadavky

- Parser je nevyhnutný pre lexikálno-syntaktickú analýzu zdrojového kódu. Bude implementovaný ako skript a jeho úlohou bude rozpoznávať nielen kľúčové slová, ale aj



Obr. 3.2: Prípady použitia pre vývojára

celé funkčné bloky.

- Modulárnosť — systém musí byť dostatočne modulárny, aby umožňoval jednoduché doplnenie triedy rozpoznávaných jazykov o nový jazyk. Zavádzanie nových jazykov bude tvorené použitím skriptov jazyka Lua.
- Multi-platformovosť je základnou požiadavkou nášho editora. Pre jej zaistenie sa celý editor bude implementovať pomocou Qt toolkitu.
- Výstupom editora bude zdrojový kód obohatený o dokumentáciu. Aby bolo možné zdrojový kód skompilovať, všetka dokumentácia bude automaticky umiestnená v komentároch príslušného jazyka.

Kapitola 4

Návrh

V tejto kapitole predstavujeme návrh riešenia vypracovaný na základe špecifikácie požiadaviek. Pri návrhu riešenia sme sa sústredili hlavne na návrh alternatív pre reprezentáciu blokov, návrh syntaktického analyzátora a návrh modulu pre využitie značiek Rich Text Format (RTF). Jednotlivé prvky návrhu sú rozdelené do samostatných podkapitol.

Keďže projekt, ktorý riešime, má experimentálny charakter a v praxi niečo podobné ešte neexistuje, našou úlohou je preskúmať možnosti vytvorenia textového editora s danou funkcionalitou. Vzhľadom na tieto skutočnosti návrh nie je príliš detailný, zato sa zaoberá rôznymi alternatívami, ktoré môžu viesť k dosiahnutiu cieľa – vytvoreniu funkčného prototypu.

Pretože ešte nie je jasné, ktorú z alternatív sa nám podarí úspešne použiť, o detailnej architektúre zatiaľ nemôžeme hovoriť. Namiesto identifikácie jednotlivých súčiastok a ich rozloženia sa preto podkapitola venovaná architektúre zaoberá všeobecnejším opisom modelu architektúry na najvyššej úrovni.

4.1 Základné východiská

Ako bolo v zhrnutí na konci analýzy naznačené, rozhodli sme sa navrhnúť a vytvoriť vlastný textový editor, ktorý bude obsahovať požadovanú funkcionalitu. Možnosť vytvoriť

zásuvný modul (plugin) do niektorého z editorov podporujúcich takýto spôsob rozšírenia funkcionality sme zavrhlí, pretože by išlo o príliš hlboký zásah do jadra editora.

Vhodnejšie v prípade využitia už existujúceho editoru by bolo využiť ho len ako základ a modifikovať ho. Takto vytvorená nadstavba by potom bola naším finálnym produktom v rámci zadaného projektu. Túto alternatívu sme zamietli z dôvodu, že by to znamenalo nutnosť študovať zdrojový kód vytvorený niekým iným. Čas strávený štúdiom pritom môžeme rovnako stráviť aj vývojom vlastného textového editora s úplne základnou použiteľnosťou, doplnenou o funkcionlitu uvedenú v špecifikácii. Svoju kódu potom budeme lepšie rozumieť než cudziemu.

Pri vytváraní editora využijeme jednu zo základných tried, ktoré Qt toolkit ponúka – `QMainWindow`. Ide o hlavné okno aplikácie ako ho všetci poznáme, s priamou možnosťou využitia menu a panelov nástrojov. Jednotlivé položky menu a panelov nástrojov budú reprezentované pomocou triedy `QAction`, ktorá umožňuje aj priradenie ikony a klávesovej skratky. Čo sa ich funkcionality týka, vytvorenie ich prepojenia na príslušné obslužné funkcie je v Qt záležitosť jedného príkazu.

4.2 Alternatívy jadra aplikácie

Vychádzajúc zo špecifikácie, náš textový editor by mal ku zvýrazneniu textu používať aj fonty, farby a grafické elementy. Toto všetko sa dá spraviť využitím elementov priamo v rámci Qt, kde máme prvky s bohatou podporou práce s obohateným textom (rich text). Čo však robí tento projekt zaujímavým, netradičným a sťažuje situáciu je fakt, že nami vytvorený textový editor má umožniť používateľovi prácu na úrovni blokov.

Komponent, ktorým bude nakoniec reprezentovaný blok, musí nezávisle od konkrétnej implementácie spĺňať niekoľko podmienok:

- podpora práce s obohateným textom
- schopnosť pracovať na úrovni textu
- schopnosť pracovať na úrovni blokov

- možnosť zbalit/rozbalit blok
- upravený vzhľad

Úprava vzhľadu bude dôležitá bez ohľadu na to, ktorá alternatíva nakoniec povedie k cieľu. Na základe zvyšných podmienok sa nám ponúkajú nasledovné 4 možnosti ako pristúpiť k riešeniu:

Trieda `QGraphicsTextItem` ako základ

Trieda `QGraphicsTextItem` bola už spomínaná v rámci analýzy. Táto trieda je, čo sa týka práce s textom, veľmi podobná triede `QTextEdit`. Umožňuje spracovanie obohateného textu, taktiež aj prístup do dokumentu prostredníctvom triedy `QTextCursor`. Použitie tohoto komponentu na reprezentáciu bloku je teda logicky jednou z alternatív.

Každý dokument otvorený v našom textovom editore bude reprezentovaný scénou, ktorá je inštanciou triedy `QGraphicsScene`. Túto bude používateľ vidieť vďaka triede `QGraphicsView`, ktorá sa stará o vizualizáciu scény. Jednotlivé bloky budú potom do scény pridávané ako inštancie triedy odvodenej od `QGraphicsTextItem`. Scéna vie rozoznať objekty, ktoré jej boli pridané. Akékoľvek zmeny vo formátovaní, ktoré používateľ zvolí v editore, budú odoslané scéne a jej úlohou je, aby boli zmeny vykonané v príslušnom bloku.

Použitie triedy `QGraphicsTextItem` má z hľadiska želaného správania sa blokov výhodu v tom, že pôvodné reakcie triedy na zachytávanie udalostí stačí v niektorých prípadoch len trochu upraviť. Príkladom je presúvanie komponentu po scéne. V pôvodnej implementácii nám môžeme pohybovať ľubovoľne, stačí ho chytiť, presunúť a pustiť. Úprava bude v tomto prípade pozostávať z pridania obmedzení čo sa týka pohybov bloku po scéne. Zároveň treba upraviť správanie sa komponentu na kliknutie myšou, aby sme po kliknutí na text mohli tento editovať, no pri uchopení bloku s ním manipulovali ako s celkom.

Väčší problém nastane pri vytváraní hierarchie blokov. Keďže jednotlivé komponenty uložené v scéne sú pôvodne samostatné jednotky, je potrebné zabezpečiť synchronizáciu vnorených blokov. Pri presune vonkajšieho bloku sa vnorené bloky musia presunúť s ním, pri zbalení vonkajšieho bloku musia „zmiznúť“ spolu s ostatným textom. Synchronizovaný

presun by bolo možné zabezpečiť napríklad v metóde, ktorá sa stará o vykresľovanie bloku. Je potrebná synchronizácia vonkajšieho bloku s priamo vnorenými, a prešírenie po všetkých úrovniach. Skrývanie bloku sa dá zabezpečiť vypnutím zobrazovania bloku.

Ďalší problematický bod je presunutie bloku do, resp. z niektorého iného bloku. V tomto prípade je potrebné vytvoriť miesto pre príslušný blok a pripojiť ho k vonkajšiemu, aby spolu boli synchronizované. Pri odnímaní bloku platí opačný proces.

Možnosť zbalit/rozbalit blok možno pridať pripojením grafického komponentu, ktorý bude blok sprevádzať. V rámci tejto alternatívy by sme využili `QGraphicsPixmapItem`, ktorý by sme synchronizovali s príslušným blokom podobne ako je načrtnuté vyššie. Príslušné správanie bloku bude implementované v obsluhu kliknutia na príslušný komponent.

Trieda `QTextEdit` ako základ

Ako bolo uvednené v analýze (časť 2.2.1), `QTextEdit` je jednou zo základných tried integrovaných priamo v Qt toolkite, ktorá podporuje prácu s rich text prvkami. V rámci tejto alternatívy je blok reprezentovaný súčiastkou (widget), ktorá obsahuje tlačidlo na zbalenie/rozbalenie bloku a komponent odvodený od triedy `QTextEdit`. Takáto reprezentácia umožní veľmi ľahkú prácu s obohateným textom, rovnako ako aj kopírovanie, vkladanie obrázkov, vyhľadávanie v texte a pod. Pri takejto reprezentácii by aj problém so zbalením/rozbalením bloku bol triviálny, stačilo by nechať nezobrazovať/zobrazovať telo bloku.

Súčiastky ako také nie je možné do `QTextEdit`-u vkladať. Po vytvorení ich teda len necháme zobrazovať používateľovi, budeme odchytať ich vykresľovanie a v rámci neho ich po normálnom vykreslení prekreslíme do obrázka. Do `QTextEdit`-u je totiž možné vkladať obrázky. Presúvanie blokov teda bude založené na presúvaní obrázkov. Keďže základom blokov bude komponent odvodený od `QTextEdit`-u, tento spôsob vkladania zároveň umožňuje zo svojej podstaty vytváranie hierarchie blokov. Keď v rámci hierarchie príde k zmenie v niektorom z blokov, v celej hierarchii smerom hore by sa vynútila aktualizácia, aby používateľ mohol danú zmenu vnímať.

V rámci tejto alternatívy sa črtá problém, či nám možnosti Qt umožňujú po kliknutí

na príslušný obrázok reprezentujúci blok zistiť, na ktorý blok sme klikli (na ktoré miesto v rámci daného bloku) a následne predať príslušnú udalosť danému bloku na spracovanie.

Založiť riešenie na použití QScintilly

V prípade, že nás neuspokoja možnosti, ktoré ponúka samotný Qt toolkit, zvažili sme aj možnosť využiť QScintillu (viď 2.2.3). QScintilla priamo ponúka veľa vymožeností z hľadiska podpory práce programátora, no na strane druhej by bolo potrebné pridať určité správanie, ktoré je vyžadované a vyššie spomínané komponenty ho zvládajú.

Vytvorenie vlastného komponentu

Toto je posledná alternatíva, ku ktorej sa uchýlime len v krajnej núdzi. Znamenalo by to totiž, že sami musíme implementovať všetky užitočné funkcie pre prácu s textom a podporu práce s obohateným textom, čiže veci, ktoré vyššie spomínané komponenty priamo ponúkajú.

4.3 Syntaktický analyzátor

Modul syntaktického analyzátora (a tiež lexikálneho, v prípade nevyužitia (Q)Scintilly) bude postavený na knižnici LPeg (viď časť 2.3.1). Spracúvanie externých skriptov vyžaduje interpret jazyka Lua, ktorý bude priamo súčasťou aplikácie.

Podsystem načíta gramatiku požadovaného jazyka z dodaného skriptu a počas behu bude podľa potrieb iných podsystemov generovať abstraktný syntaktický strom (AST). AST bude priamo využívaný vizualizačným modulom pri vykresľovaní grafických blokov. Počas práce používateľa sa bude musieť automaticky (respektíve manuálne) spúšťať opakované generovanie kompletného, prípadne len parciálneho AST.

Dôležitou časťou je ošetrovanie vzniku syntaktických chýb, ktoré by nemali ovplyvňovať celý AST, ale len jeho minimálnu podmnožinu. Toto dosiahneme sledovaním bloku, ktorý

práve používateľ edituje a následnou analýzou obmedzenou len na tento blok a bloky v ňom vnorené.

AST bude reprezentovaný stromom pozostávajúcím s *textových* a *zložených* elementov (oboch reprezentovaných triedou `TElement`). Okrem spoločného atribútu `type` budú textové elementy používať atribút `text` (obsahujúci text) a zložené atribút `content` (obsahujúci zoznam ľubovoľných elementov) na ukladanie svojho obsahu. Každý zložený element reprezentuje jeden grafický blok, ktorého formátovanie určuje kľúč `type`.

Vzhľadom na možnosť pridávania nových jazykov máme tri možnosti definície formátovacích kľúčov:

1. Kľúče budú dané, jazykovo nezávislé s pevným formátovaním, napríklad `keyword`, `number`, `control_statement`, a podobne. Gramatika v skripte bude parsovať daný jazyk priamo do týchto všeobecných kategórií.
2. Kľúče budú definované spolu s príslušným formátovaním v skripte obsahujúcom gramatiku jazyka. Príklady pre C++ sú `include`, `while` alebo `function_parameter`. Vyžaduje použitie metajazyk na popis formátovania.
3. Kombinácia predošlých. Skript definuje jazykovo špecifické kľúče, ktoré následne namapuje na „vstavané“ základné kľúče (s pevným formátovaním). Zároveň má autor skriptu možnosť doplniť nové kategórie, prípadne modifikovať prednastavený spôsob formátovania.

Pri implementácii prototypu zvolíme pravdepodobne prvú, najjednoduchšiu cestu, ale neskôr zvážime aj možnosti niektorého z prispôbivejších variantov.

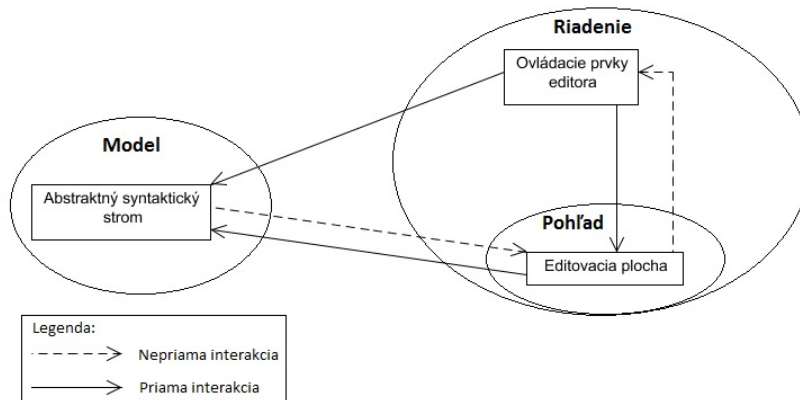
4.4 Modul pre využitie značiek RTF

Tento modul zabezpečí rozpoznávanie značiek RTF (viď 2.3.2) v našom editore. Značky, ktoré sa v dokumente budú využívať na označovanie blokov dokumentácie ako aj ostatné formátovacie značky, budú rozpoznávané, ale v editore sa nebudú zobrazovať. Čo sa týka zakomponovania RTF do nášho projektu, z danej špecifikácie plánujeme využiť najmä

komentáre.

4.5 Architektúra systému

Pri implementácii textového editora by sme chceli využiť štandardný návrhový vzor model-pohľad-riadenie. V našom prípade bude modelom abstraktný syntaktický strom, ktorý vznikne syntaktickou analýzou textu. Čo sa pohľadu týka, dosť závisí od nakoniec použitej alternatívy. Vo všeobecnosti to však bude komponenta, ktorá má na starosti vizualizáciu textu a jeho štruktúrovanie. Pohľad nám bude zobrazovať text usporiadaný do blokov v závislosti na tom, ako je tento uložený v syntaktickom strome (modeli). Riadenie bude zahŕňať ovládacie prvky editora. Vzhľadom na to, že plocha editora slúži nielen na zobrazovanie, ale aj na editáciu textu, môžeme povedať, že sa tu čiastočne zlučuje riadenie a pohľad. Situáciu možno vidieť na Obrázku 4.1.



Obr. 4.1: Návrh architektúry

Kapitola 5

Záver

Projekt textového editora obohateného o grafické prvky (pracovný názov *TrollEdit*) je svojím zameraním v súčasnosti jedinečný a poskytne používateľom zaujímavé možnosti. Vďaka poznatkom z analýzy sme vytvorili niekoľko alternatív pre finálnu integráciu uvažovaných technológií, najmä z ohľadom na požadovanú multiplatformovosť. V ďalšej práci bude naším cieľom priebežne zjemňovať prezentovaný návrh systému a postupne pristúpiť k implementácii prototypu, respektívne série prototypov.

Zoznam použitej literatúry

- [1] R. Ierusalimschy. LPeg – Parsing Expression Grammars For Lua. <http://www.inf.puc-rio.br/~roberto/lpeg/lpeg.html>. [posledný prístup 27.10.2009].
- [2] R. Ierusalimschy. *Programming in Lua*. Lua.org, 2003. ISBN 85-903798-1-7.
- [3] R. Ierusalimschy. A text pattern-matching tool based on Parsing Expression Grammars. *Softw. Pract. Exper.*, 39(3):221–258, 2009.
- [4] Notepad++. <http://notepad-plus.sourceforge.net/uk/site.htm>. [posledný prístup 1.11.2009].
- [5] QScintilla: What is QScintilla. <http://www.riverbankcomputing.co.uk/software/qscintilla/intro>. [posledný prístup 31.10.2009].
- [6] Qt 4.0: Qt Reference Documentation (Open Source Edition). <http://doc.trolltech.com/4.0/index.html>. [posledný prístup 30.10.2009].
- [7] Rich Text Format (RTF) Version 1.5 Specification. http://www.biblioscape.com/rtf15_spec.htm. [posledný prístup 1.11.2009].
- [8] Scintilla and SciTE. <http://www.scintilla.org/SciTE.html>. [posledný prístup 1.11.2009].
- [9] The Programming Language Lua. <http://www.lua.org>. [posledný prístup 27.10.2009].