**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE**
Fakulta informatiky a informačných technológii

# Technická dokumentácia projektu

(Textový editor obohatený o grafické prvky)

*Tímový projekt*

**Vypracoval:** Innovators – tím č.10
**Akademický rok:** 2011/12

| | |
|---|---|
| **Autor:** | tím č.10 – Innovators |
| **Téma projektu:** | textový editor obohatený o grafické prvky (TrollEdit) |
| **Vytvorený:** | 02.10. 2011 |
| **Stav:** | predbežný |
| **Vedúci projektu:** | Ing. Peter Drahoš |
| **Vedúci tímu:** | Bc. Lukáš Turský |
| **Členovia tímu:** | Bc. Marek Brath<br>Bc. Adrián Feješ<br>Bc. Maroš Jendrej<br>Bc. Jozef Krajčovič<br>Bc. Ľuboš Staráček |
| **Kontakt:** | tp-team-10@googlegroups.com |

# Obsah

# 1 Úvod

Súčasné textové editory zdrojových kódov len minimálne využívajú možnosti grafickej reprezentácie, čo je veľká škoda vzhľadom na to, že práve obohatenie editorov o grafické prvky by mohlo v mnohých veciach uľahčiť prácu s takýmto editorom. Sprehľadnil by sa zdrojový kód, zjednodušila a zefektívnila nie len jeho tvorba, ale aj údržba a prezentácia, a vnieslo by to možnosť nového pohľadu na integráciu dokumentácie s programom.

Práve to by malo byť výsledkom tohto projektu, ktorého cieľom bude pokračovať vo vývoji multiplatformového editora „TrollEdit“ (ktorý bol riešení v roku 2009/10 tímom s názvom UFOPAK) pre editovanie najmä zdrojových kódov, ktorý bude využívať grafické prvky na zjednodušenie a zefektívnenie práce programátora. Našim zameraním bude rozšírenie stávajúcej funkcionality do podoby vhodnej pre reálne nasadenie editora do praxe.

Tento dokument obsahuje zhrnutie všetkých riešení nášho tímu na tomto projekte od analýzy až po implementáciu.

# 2 Analýza

## 2.1 Existujúce riešenia

V súčasnosti na trhu existuje mnoho editorov od jednoduchších až po zložitejšie, s rôznymi funkcionalitami a metódami ktoré uľahčujú prácu používateľa. Pri vytváraní projektu sa môžeme inšpirovať súčasnými ako sú eTextEditor (e), SciTE alebo Notepad++.

### 2.1.1 eTextEditor (e)

Textový editor pre Microsoft Windows s výkonnými funkciami pre úpravu textu. Vznikol ako alternatíva pre TextMate, pretože práve tento editor bol oslavovaný mnohými programátormi. Umožňuje rýchlu a jednoduchú manipuláciu s textom, automatizuje všetku manuálnu prácu, čím vám napomáha lepšiemu sústredeniu sa na písanie. 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. Menšou nevýhodou je trošku pomalé otváranie súborov.

### 2.1.2 SciTE

Editor založený na Scintille. V SciTE 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 otvorený zdrojový kód. 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 byt 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 namapovat 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 podtržník a zmeny veľkosti písmen v slove či odseku (bloky textu oddelené prázdnym 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:** Funkčnosti 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.

### 2.1.3 Notepad++

Voľne dostupný editor zdrojového kódu [4], ktorý aj podporou viacerých jazykov nahrádza Notepad. Beží v prostredí MS Windows pod licenciou GPL. Avšak môže byt viacplatformovým využitím 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.

Podporuje zvýraznenie syntaxe pre 44 jazykov, skriptovacie a značkovacie jazyky. Užívatelia môžu tiež definovať svoj vlastný jazyk pomocou zabudovaného zásuvného panelu. 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.

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 byt 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 napríklad aj čínštinu, hebrejčinu, kórejčinu či arabčinu. 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.

Tabuľka 1. Porovnanie funkcionalít

| | **eTextEditor (e)** | **SciTE** | **Notepad++** |
|---|---|---|---|
| **Spell checking** | plugin | nie | plugin |
| **Viacnásobné undo/redo** | áno | áno | áno |
| **Selekcie blokov** | áno | áno | áno |
| **Zvýraznenie syntaxe** | áno | áno | áno |
| **Automatické dopĺňanie** | áno | áno | áno |
| **Integrácia kompilátora** | áno | áno | áno |
| **Spoločné editovanie na viacerých počítačoch** | áno | nie | plugin |

## 2.2 Analýza predchádzajúceho riešenia nástroja TrollEdit

Vzhľadom na to, že pokračujeme na projekte, ktorý bol vyvíjaný v rámci minuloročného tímového projektu bolo nutné vykonať podrobnú analýzu predchádzajúceho riešenia. Výsledkom je porovnanie medzi reálnym stavom editora a technickou dokumentáciou minulého tímu. Správa o stave bola rozdelená podľa jednotlivých funkčných častí.

### 2.2.1 Inicializácia editora, otvorenie súborov

**Implementované:**

- pri načítaní súboru určenie správnej gramatiky a jej kontrola
- pri otvorení súboru automatická analýza a zobrazenie do blokov
    - komentáre sú prepojené v blokoch avšak umiestnené sú mimo riadku, na ktorý sa odvolávajú, bolo by vhodné ich umiestniť vedľa textu
- história naposledy otvorených súborov

- obsahuje modul pre syntaktickú analýzu
- novšia LuaJit verzia – rýchle spracovanie menších súborov

**Chýba:**

- veľké súbory stále dosť pomalé na prácu
- pri otvorení napr. Analyzer.cpp nevyrobí správne bloky častí súboru, všetko brané ako samostatný riadok

### 2.2.2 Práca s editorom

**Implementované:**

- zvýrazňovanie syntaxe až na úrovni blokov, teda je možné určiť grafické vlastnosti pre všeobecné prvky naprieč viacerým jazykom a gramatikám
- popis zvýrazňovania jednotlivých blokov, ktoré majú byť zvýraznené, je obsiahnutý v konfiguračnom súbore
- všetko v rámci editačného okna editora je možné presúvať
- existuje možnosť *„Edit plain text“* pre úpravu textu, vtedy je zobrazené v samostatnom okne všetko ako čistý text (tu prepínanie na dva módy, zabudovať priamo do editore)
- v súbore text_item.cpp implementovaný pohyb medzi blokmi po stlačení kláves
- samostatné zoomovanie každého otvoreného súboru nehľadiac na tie ostatné
- presúvanie blokov v editore
- vyhľadávanie v texte zobrazí bloky, v ktorých sa text nachádza

**Chýba:**

- klávesové skratky veľmi chýbajú
  - nejaké už sú implementované (v Menu -> File sa dajú vidieť)
  - priamo na začiatku v súbore main_window.cpp sa priraďujú skratky k akciám
- chýba možnosť Undo, Redo, Copy, Paste
  - len cez kontextovú ponuku cez pravé tlačítko je možná
- selekcia textu aj v rámci viacerých blokov
- malé možnosti vyhľadávania
  - chýba možnosť pri veľkých súboroch krokovania nájdených výskytov vyhľadávania
  - vyhľadáva len v aktuálnom súbore
  - lupa nie je klikateľné tlačítko
- naraz otvorená len jedna pracovná plocha (workspace)
  - triedu BlockGroup je v budúcnosti možné využiť na paralelné zobrazenie viacerých hierarchií (BlockGroup) na jednej scéne (DocScene) – viacero pracovných plôch

- konfiguračný súbor sa načíta len raz, pri spustení editora (sprevádza ho)
- základná štruktúra menu pre prácu s textom a options je zakomentovaná a neimplementovaná

### 2.2.3 Programovanie v editore

- analýza zdrojového kódu je časovo náročnejšia a preto sa spúšťa iba v čase prechodu písania na ďalší riadok.
- funguje automatické odsadzovanie pri analýze
    - medzery a tabulátory sa nezobrazujú a realizujú sa len ako prázdny priestor pred príslušným blokom
    - prebytočné zbavenie sa medzier
- znak konca riadku je nahradený nastavením príznaku v bloku

### 2.2.4 Komentáre

**Implementované:**

- posúvanie komentárov – plávajúce komentáre
- funguje zobrazenie jednoriadkových aj blokových komentárov ako samostatný blok
- šípka ku blokovému komentáru začína vždy na začiatku riadku
- šípka ku jednoriadkovému komentáru začína od konca daného riadku
- funguje CTRL + Ľavé tlačítko myší = vytvorí na danom mieste nový ale len bežný blok (podobný ako ten pre komentár), pričom ho prepojí šípkou z miestom kde sa nachádzal kurzor

**Chýba:**

- textové komentáre ako samostatné bloky, nie je možné napísať tvrdú medzeru
- šípka by mohla byť aj zmysluplnejšie ukazujúca na daný blokový komentár
- textové komentáre len ako bežné bloky, nie je možnosť rovno písať dokumentáciu ako bolo spomínané cez dokumentačné bloky
- chýba možnosť vytvárania dokumentačných blokov, ale funkčne je implementovaná

### 2.2.5 Bloky

**Implementované:**

- možnosť presúvať bloky, alebo časti blokov
- pomocná čiara pri presune

**Chýba:**

- plávajúce bloky sa nedajú zmazať priamo, jedine postupným vymazaním ich obsahu
- chýba možnosť samostatne vytvárať bloky a prepájacie šípky

- šípka odkazuje len na jeden blok
- vždy možné presúvať len jeden blok naraz, chýba výber viacerých blokov
- chýba skrývanie blokov, nezobrazuje možnosť na skrytie blokov

- pri inicializácii sa nedeteguje prekrývanie viacerých blokov na jednom mieste
- pri vkladaní bloku rozostupovanie ostatných blokov

### 2.2.6 Práca so súbormi, prílohami

**Implementované:**

- pridanie súboru ako prílohy v podobe bloku,
    - treba mať označený nejaký blok, aby bolo možné určiť časť súboru od ktorej sa priraďuje
- prílohy vkladá ako odkaz
- obrázky vie rovno zobraziť
- možnosť úpravy rozmerov obrázka
- ukladanie ako pôvodný súbor s komentármi, súbor bez komentárov, alebo ako PDF tlačiť (len printscreen v rámci ohraničenia pri tlači)
    - žiadne pokročilé prvky popisujúce obsah dokumentačných blokov ako bolo spomínané

### 2.2.7 Syntaktický analyzátor

**Implementované:**

- analýza realizovaná v jazyku LUA za pomoci knižnice LPeg
    - možnosť rozširovania o ďalšie gramatiky (načítavajú sa z priečinka grammars)
    - výstupom je LUA tabuľka obsahujúca ďalšie tabuľky a tento systém tabuliek zodpovedá syntaktickému stromu
- vytváranie AST na strane jazyka LUA a jeho prenos do C++

### 2.2.8 Gramatika

**Implementované:**

- základná gramatika default_grammar.lua
    - rozloženie ľubovoľného textu na slová a riadky
    - popísané povinné konštanty, ktoré musia gramatiky obsahovať
    - funkcie na testovanie gramatík
- gramatika pre C, LUA a XML

### 2.2.9 Literate programming

**Implementované:**

- možnosť vkladať ku kódu okrem klasických komentárov aj obrázky

**Chýba:**

- neukladajú sa pridané obrázky a iné formátovacie zmeny v dokumente
- ukladanie dokumentácie do RTF formátu

## 2.3 Analýza použitých technológií

Pri implementácií budeme používať nástroje a technológie, ktoré používal predchádzajúci tím UFOPAK počas vývoja editora. Nosnými technológiami sú Qt SDK, Lua a využitie RTF, ktoré v skratke predstavíme ako aj dôvod, prečo sme sa rozhodli pokračovať v ich používaní.

### 2.3.1 Qt

Qt 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 (MP3prehrávače), bežiace pod operačnými systémami Windows, Linux, MAC OS, Symbian. Multiplatformovosť je práve jedna z rozhodujúcich výhod, kvôli ktorým je editor implementovaný pomocou tohto nástroja. Qt je v súčasnosti dostupné pod komerčnou ale aj GNU GPL v3.0 licenciou.

Nástroj Qt ponúka okrem množstva tried a knižníc pre tvorbu GUI aplikácií aj vlastné vývojové prostredie Qt Creator. Uvažované možnosti práce s nástrojom Qt boli nasledovné:

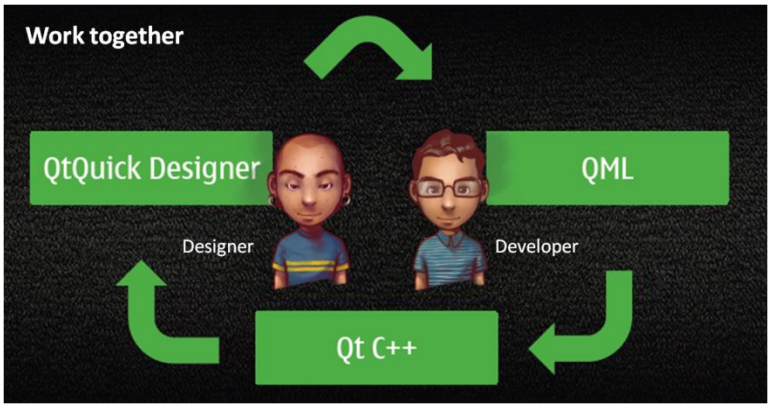
- Qt modul pre vývojové prostredie Eclipse
- Qt modul pre vývojové prostredie Visual Studio
- Integrované vývojové prostredie Qt Creator

Rozhodli sme sa pre použitie prostredia Qt Creator, keďže poskytuje samostatné vývojové prostredie, čiže pre naše potreby by malo byť ideálne, a taktiež integruje v sebe viacero novších technológií a prístupov, ktoré nám pomôžu pri vývoji. Napr. obsahuje novú technológiu Qt Quick.

### 2.3.2 Qt Quick a jazyk QML

Qt Quick je nová technológia určená pre rýchle vytváranie jednoduchých a bohatých používateľských rozhraní aplikácií pre rôzne platformy. Qt Quick obsahuje jazyk QML, ktorý je navrhnutý vychádzajúc z jazykov HTML, CSS, JavaScript, pričom spája ich výhody.

S použitím technológie Qt Quick je možne aby dizajnér navrhol UI podľa vlastnej fantázie a vývojár len doplnil logiku aplikácie, čo prináša obrovskú výhodu keďže vývojár a dizajnér majú každý iný pohľad na svet a nie vždy bolo možne nájsť konsenzus pri vytváraní aplikácie.




Obrázok 1 Pracovný cyklus s použitým Qt Quick

Qt Quick umožňuje vytvárať rôzne animácie, ktoré využívajú knižnicu OpenGL. Takisto umožňuje navrhnúť dizajn jednotlivých používateľských prvkov aplikácie ako napr. tlačítka v grafických editoroch Adobe Photoshop, Autodesk Maya, Gimp.

Jednoduchosť technológie Quick možno vidieť v rozdiele medzi definovaním jednoduchého tlačítka klasickým spôsobom cez actionscript a novým pomocou jazyka QML.

Actionscript: **MenuButton**.as

```
public class MenuButton extends MovieClip
        public function MenuButton() {
                this.x = 60;
                this.addEventListener(MouseEvent.MOUSE_DOWN, ClickBt);
        }
        function ClickBt(e:MouseEvent) {
                trace("clicked");
        }
}
```

QtQuick: **MenuButton**.qml

```
Item {
  x:60;
  MouseArea: {
      anchors.fill: parent;
      onClicked: print("clicked");
  }
}
```

Použitie technológie Qt Quick by mal pre nás veľký význam keďže nám umožňuje navrhnúť UI pre TrollEdit podľa našej potreby, ktorý by bol zaujímavejší ako súčasné UI riešenia editorov. To nám dáva možnosť v tomto smere vytvoriť kvalitnejší produkt.

Príklad dizajnu navrhnutého s použitím technológie Qt Quick je možno vidieť v takých aplikáciách ako Skype, VLC Media Player atď.

### 2.3.3 Jazyk Lua

Lua 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 funkcionalitu a mechanizmy ako definovať 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. 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 860KB aj s dokumentáciou), vďaka ktorej nie je problém pripojiť ju celú k aplikácii, ktorá ju používa.

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 Skype, Wireshark, VLC media player atď.

### 2.3.4 Knižnica LPeg

LPeg 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 postavená na gramatikách typu PEG (Parsing Expression Grammar) a 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. V prvom module re (skratka z regex) sú vzory

popisované reťazcami so syntaxou odvodenou z regulárnych výrazov. Druhý modul lpeg pracuje so vzormi ako s premennými vlastného dátového typu a obsahuje viac spôsobov na ich vytváranie a spájanie. 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.

### 2.3.5 RTF

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.

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.

## 2.4 Analýza spracovávania syntaktického stromu

Na špecifikáciu gramatiky v jazyku Lua je využitá knižnica LPeg. Je vytvorená gramatika pre jazyk C, pričom vychádza zo zápisu v Bakchus-Naurovej forme (BNF). Gramatika sa nachádza v skripte a je dynamicky kompilovaná za behu aplikácie. Spoluprácu s jadrom systému zabezpečuje C API (štandardná súčasť jazyka Lua) a funguje na báze zásobníka, z ktorého čítajú a zapisujú obe strany. Komunikácia prebieha nasledovne:

- aplikácia spustí skript s gramatikou a gramatika sa skompiluje
- aplikácia zavolá LPeg funkciu match, ktorej vstupom je gramatika a text (kód), ktorý chceme analyzovať
- výstupom je Lua tabuľka obsahujúca ďalšie tabuľky a tento systém tabuliek

zodpovedá syntaktickému stromu

- výstup je umiestnený na zásobník z ktorého je postupne čítaný, z Lua tabuliek sa zrekonštruuje AST v C++

Gramatika využíva funkcie na zachytávanie časti vstupu, ktoré zodpovedajú daným LPeg výrazom (podobným regulárnym výrazom). Ku každej zachytenej (lexikálnej) jednotke alebo skupine jednotiek je pripojený identifikačný kľuč (napr. storage_class_specifier, number_constant, parameter_list), ktorý v AST slúži na identifikáciu uzlov. Zachytené sú všetky znaky, teda aj tie, ktoré nie sú priamo lexémami, ako napríklad biele znaky (kľuč whitechar) alebo text, ktorý nezodpovedá gramatike jazyka (označený kľúčom unknown). Gramatika dosial nie je úplne kompletná, neobsahuje podporu inštrukcií preprocesora v tele funkcií a vyžaduje si ďalšie testovanie.

### 2.4.1 Gramatiky

Gramatiky jazykov sú písané v jazyku Lua a nachádzajú sa v priečinku /grammars. Pre fungovanie programu je nutná existencia základnej gramatiky default_grammar.lua. Táto gramatika slúži na rozloženie ľubovoľného textu na slová a riadky a obsahuje funkcie používané na testovanie gramatík. V súbore s touto gramatikou sú tiež popísané povinné konštanty, ktoré musí každý súbor s gramatikou obsahovať ako napríklad prípony spracúvaných súborov, zoznam párových znakov a podobne. Pomocou knižnice LPeg vytvára Lua interpret tabuľkovú reprezentáciu stromu ako systému hierarchicky vnorených tabuliek bez explicitných kľúčov v tvare:

uzol1; uzol1:1; uzol1:1:1; : : :; : : :; uzol1:2; : : :; uzol1:3; : : :; : : :

Názov uzla je vždy nasledovaný tabuľkami zodpovedajúcim jeho priamym potomkom. Zároveň by každá gramatika mala vrátiť len jednu tabuľku, ktorá ale nemusí nutne obsahovať len jeden koreň (napr. a; b je korektný výstup). Tabuľková štruktúra je definovaná priamo v syntaktických pravidlách gramatiky pomocou štandardných LPeg funkcií lpeg.C, lpeg.Ct a lpeg.Cc. Vo všeobecnosti sa predpokladá, že súbor s gramatikou obsahuje jednu kompletnú gramatiku (full_grammar) a ľubovoľný počet čiastkových gramatík (other_grammars). Plná gramatika sa využíva pri analýze celého súboru a ostatné gramatiky pri analýze menších časti. Napríklad, pre jazyk C sa používajú gramatiky program (analyzuje celý program v C), top_element (analyzuje funkcie, mimo-funkčné deklarácie alebo direktívy preprocesora) a in_block (analyzuje obsah bloku príkazov). Zmysel čiastkových gramatík je v tom, že

napríklad na vyhodnotenie syntaktickej správnosti skupiny príkazov nemôžeme použiť kompletnú gramatiku, pretože skupina príkazov nie je platným programom.

### 2.4.2 Rozhranie Lua — Qt

Celú komunikáciu medzi Lua a Qt/C++ zapuzdruje trieda Analyzer. Pri požiadavke na analýzu textu je vytvorená inštancia Lua interpretu a vykonaný príslušný skript. Daný text je potom spracovaný pomocou funkcie lpeg.match a výsledok je načítaný z Lua zásobníka.

Tabuľková hierarchia je paralelne prevádzaná do stromovej štruktúry reprezentovanej triedou TreeElement. Pomocnú funkciu má trieda LanguageManager, ktorá uchováva zoznam objektov triedy Analyzer pre všetky podporované jazyky.

Funkcie, ktoré sa starajú o analýzu kódu:

```
// analyze string, creates AST and returns root
```

TreeElement* Analyzer::analyzeFull(QString input)

- Táto funkcia analyzuje celý kód

```
// reanalyze text from element and it's descendants, updates AST and returns first modified node
```

TreeElement *Analyzer::analyzeElement(TreeElement* element)

- Funkcia analyzuje konkrétny podstrom AST, bez nutnosti prepisovať celu štruktúru

```
// analyze string by provided grammar
```

TreeElement *Analyzer::analyzeString(QString grammar, QString input)

- Analyzuje sa kód pomocou vybratej gramatiky, využíva pritom funkciu na prepis LUA tabuliek do AST

```
// creates AST from recursive lua tables (from stack), returns root(s)
```

TreeElement *Analyzer::createTreeFromLuaStack()

- Táto funkcia konvertuje tabuľky z jazyka LUA do stromovej štruktúry typu TreeElement

Trieda TreeElement obsahuje smerník na predchodcu a svojich potomkov, takto vytvára stromovú štruktúru.

```
class TreeElement
```

```
{
    ...
protected:
    TreeElement *parent;
private:
    QList<TreeElement*> children;
    QString type;
    Block *myBlock;
    TreeElement *pair;
    bool lineBreaking;
    bool selectable;
    bool paragraphsAllowed;
    bool paired;
    bool floating;
    ...
};
```

# 3 Špecifikácia požiadaviek

Keďže vychádzame z už existujúceho multiplatformového textového editora „TrollEdit“ obohateného o grafické prvky, popisujeme iba tie požiadavky na systém, ktoré chceme implementovať pripadne modifikovať v súčasnej verzií.

Hlavné ciele tohto projektu sú:

- Rozšíriť súčasnú - implementovanú funkcionalitu
- Modifikovať používateľské rozhranie - GUI
- Vytvoriť kvalitný produkt, ktorý bude úspešný a mohol by sa presadiť aj v praxi

## 3.1 Funkcionálne požiadavky

Pre TrollEdit boli identifikovane funkcionálne požiadavky na základe dôkladnej analýzy predchádzajúceho riešenia a taktiež na základe podnetov od nášho vedúceho tímu, ktoré sú spísané v Tab.1.

Tab. 1 Funkcionálne požiadavky

| ID | Požiadavka | Charakteristika | Priorita |
|---|---|---|---|
| F01 | Možnosť Undo/Redo | Možnosť vrátiť zmeny naspäť a opačne | vysoká |
| F02 | Podpora skratiek v editore (Shortcuts) | Možnosť spustiť funkcie programu pomocou klávesových skratiek | vysoká |
| F03 | Dopytovanie sa do Lua | | vysoká |
| F04 | Podpora paralelizmu | Syntaktický strom by mal bežať na pozadí pod vlastným vláknom a program pod vlastným | vysoká |
| F05 | 2 módy písania | Prvý by bol klasický editor na úpravu kódu a po prepnutí by editor prešiel do druhého grafického módu. | vysoká |
| F06 | Nastavenie programu | Možnosť rozšírených nastavení priamo v editore | stredná |
| F07 | Podpora intellisense | Rozpoznávania bežných kľúčových slov programovacích jazykov, ale aj najčastejšie používané bloky kódu (napr. funkcie, cykly, podmienky) | stredná |
| F08 | Rozšírenie funkcionality | Možnosť rozširovať funkcionalitu pomocou zásuvných modulov | stredná |
| F09 | Vyhľadávanie | Určitý druh fulltextového vyhľadávania s prípadnou optimalizáciou pre najčastejšie vyhľadávané výrazy | stredná |
| F10 | Export súborov | Možnosť exportovania súboru do iných formátov (.csv, .doc) | nízka |
| F11 | Podpora sw metrík | Schopnosť detegovať určité ukazovatele v zdrojovom kóde ako index udržateľnosti, cyklomatická zložitosť, hodnoty Fan in a Fan out, | nízka |

| | | | |
|---|---|---|---|
| | | ktoré by boli zobrazené v tabuľke, prípadne vizuálne v podobe grafov | |

**Legenda:**
Vysoká priorita – nevyhnutia funkcia systému, je základom funkcionality systému.
Stredná priorita – funkcia, ktorú možno implementovať neskôr netvorí základ funkcionality systému.
Nízka priorita – funkcionalita bude implementovaná v ďalších verziách

## 3.2 Nefunkcionálne požiadavky

Pre TrollEdit boli identifikovane nasledujúce nefunkcionálne požiadavky pre správne zabezpečenie fungovania programu.

Tab. 2 Nefunkcionálne požiadavky

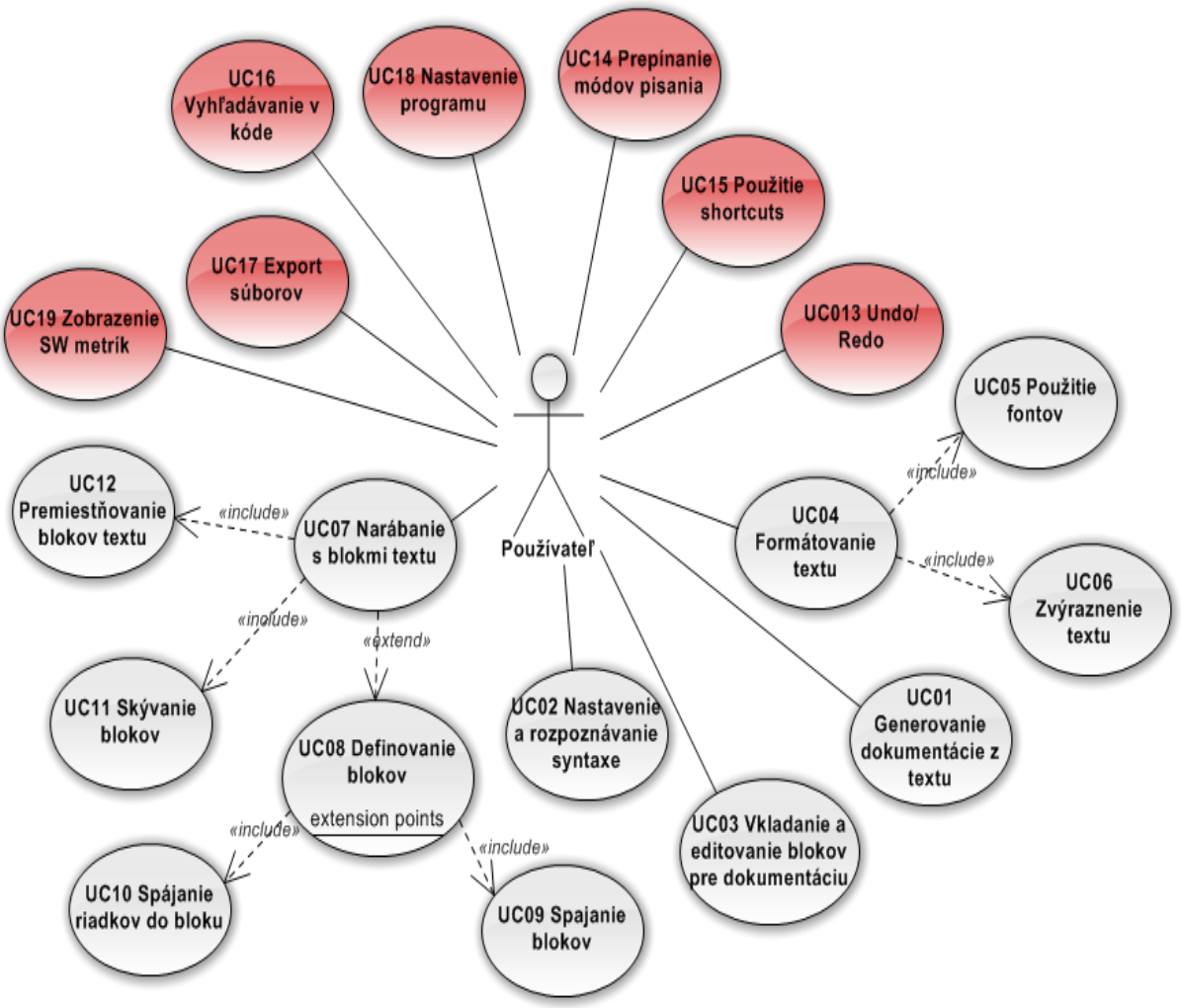
| ID | Požiadavka | Charakteristika |
|---|---|---|
| N01 | Rýchlosť a spoľahlivosť | Zrýchlenie programu hlavne čo sa týka parsovania. Program by mal byť schopný pracovať aj na menej výkonnom hardvéri |
| N02 | Modulárnosť | Možnosť rozširovania jeho funkcií pomocou dodatočnej implementácie nových modulov. Tým pádom nie je v zásade nutné zasahovať do samotnej implementácie systému pri rozširovaní jeho funkcionality |
| N03 | Redesign používateľského rozhrania GUI | Musí byť jednoduché a prehľadné, pričom najčastejšie funkcie systému by mali byť prístupné používateľovi bez náročného hľadania |

# 4 Návrh riešenia

V tejto kapitole je popísaný návrh programu TrollEdit podľa požiadaviek definovaných v predchádzajúcej kapitole. Funkcionálne požiadavky sa premietnu do diagramu prípadov použitia a nefunkcionálne do architektúry systému.

## 4.1 Diagram prípadov použitia

Na diagrame sú znázornené prípady použitia popisujúce funkcionalitu, ktorá je už implementovaná v programe TrollEdit a taktiež novu funkcionalitu, ktorú sme identifikovali na základe analýzy. Nové prípady použitia sú odlíšené od tých existujúcich červenou farbou.




Obr. 1 Diagram prípadov použitia

Tab. 3 Prípad použitia UC13 Undo/redo

| **Názov** | **Undo/ redo** | | | | |
|---|---|---|---|---|---|
| **ID** | UC13 | | | | |
| **Opis** | Možnosť voľby undo/ redo nad vykonanými zmenami v zdrojovom kóde | | | **Priorita** | vysoká |
| **Vstupne podmienky** | História vykonaných zmien | | | | |
| **Výstupne podmienky** | - | | | | |
| **Participant** | Používateľ (použ.) | | | | |
| **Základná postupnosť** | **Krok** | **Rola** | **Činnosť** | | |
| | 1. | Použ. | vyberie možnosť undo/ redo v pop menu na zdrojovým kódom | | |
| | 2. | Systém | urobí zmeny v kóde podľa histórie výkonných akcií | | |
| **Alternatívna postupnosť** | **Krok** | **Rola** | **Činnosť** | | |
| **-** | | | | | |
| **Poznámky** | - | | | | |

Tab. 4 Prípad použitia UC14 Prepínanie módov písania

| **Názov** | **Prepínanie módov písania** | | | | |
|---|---|---|---|---|---|
| **ID** | UC14 | | | | |
| **Opis** | Prvý mód pre klasický editor na úpravu kódu a po prepnutí by editor prešiel do druhého grafického módu | | | **Priorita** | vysoká |
| **Vstupne podmienky** | - | | | | |
| **Výstupne podmienky** | - | | | | |
| **Participant** | používateľ (použ.) | | | | |
| **Základná postupnosť** | **Krok** | **Rola** | **Činnosť** | | |
| | 1. | Použ. | V pop menu si zvolí možnosť prepnutia do druhého módu písania kódu | | |
| | 2. | Systém | prepne úpravu kódu do grafického módu | | |
| | 3. | Systém | rozšíri možnosti funkcionality pre graficky mód úpravy kódu | | |
| **Alternatívna postupnosť** | **Krok** | **Rola** | **Činnosť** | | |
| **-** | | | | | |
| | | | | | |
| **Poznámky** | - | | | | |

Tab. 5 Prípad použitia UC14 Použitie shortcuts

| **Názov** | **Použitie shortcuts** | | | | |
|---|---|---|---|---|---|
| **ID** | UC15 | | | | |
| **Opis** | | | | **Priorita** | vysoká |
| **Vstupne podmienky** | - | | | | |
| **Výstupne podmienky** | - | | | | |
| **Participant** | používateľ (použ.) | | | | |
| **Základná postupnosť** | **Krok** | **Rola** | **Činnosť** | | |

| | | | |
|---|---|---|---|
| | 1. | | |
| | 2. | | |
| | 3. | | |
| **Alternatívna postupnosť** | **Krok** | **Rola** | **Činnosť** |
| | | | |
| | | | |
| **Poznámky** | - | | |

Tab. 6 Prípad použitia UC16 Vyhľadávanie v kóde

| **Názov** | **Vyhľadávanie v kóde** | | | | |
|---|---|---|---|---|---|
| **ID** | UC16 | | | | |
| **Opis** | Vyhľadanie zvoleného výrazu v zdrojovom kóde | | | **Priorita** | stredná |
| **Vstupne podmienky** | - | | | | |
| **Výstupne podmienky** | Zobrazenie výsledku hľadaného výrazu | | | | |
| **Participant** | Používateľ (použ.) | | | | |
| **Základná postupnosť** | **Krok** | **Rola** | **Činnosť** | | |
| | 1. | Použ. | zadá hladný výraz do textboxu pre vyhľadávanie a potvrdí tlačidlom hľadať | | |
| | 2. | Systém | vyhľadá zvolený výraz v aktuálnom zdrojovom kóde | | |
| | 3. | Systém | zobrazí výsledky hľadaného výrazu | | |
| **Alternatívna postupnosť** | **Krok** | **Rola** | **Činnosť** | | |
| | 3.a | Systém | v prípade nenájdenia hľadaného výrazu zobrazí modálne okno s upozornením | | |
| **Poznámky** | - | | | | |

Tab. 7 Prípad použitia UC17 Export súborov

| **Názov** | **Export súborov** | | | | |
|---|---|---|---|---|---|
| **ID** | UC17 | | | | |
| **Opis** | Export súborov zdrojového kódu do iných formátov | | | **Priorita** | nízka |
| **Vstupne podmienky** | Parsovaný zdrojový kód | | | | |
| **Výstupne podmienky** | Vyexportovaný súbor | | | | |
| **Participant** | Používateľ (použ.) | | | | |
| **Základná postupnosť** | **Krok** | **Rola** | **Činnosť** | | |
| | 1. | Použ. | vyberie možnosť exportu súborov zo zdrojového kódu | | |
| | 2. | Systém | ponúkne možností do akých formátov ma exportovať súbory | | |
| | 3. | Použ. | vyberie formát súboru pre uloženie | | |
| | 4. | Systém | uloží súbory vo zvolenom formáte | | |
| **Alternatívna postupnosť** | **Krok** | **Rola** | **Činnosť** | | |
| **-** | | | | | |
| **Poznámky** | formát pdf, doc. | | | | |

Tab. 8 Prípad použitia UC18 Nastavenie programu

| **Názov** | **Nastavenie programu** | | | | |
|---|---|---|---|---|---|
| **ID** | UC18 | | | | |
| **Opis** | Podrobne nastavenie možností programu | | | **Priorita** | stredná |
| **Vstupne podmienky** | - | | | | |
| **Výstupne podmienky** | Zmena nastavenia programu | | | | |
| **Participant** | Používateľ (použ.) | | | | |
| **Základná postupnosť** | **Krok** | **Rola** | **Činnosť** | | |
| | 1. | Použ. | si zvolí možnosť nastavenia programu z hlavného menu | | |
| | 2. | Systém | zobrazí modálne okno s možnosťami nastavenia programu | | |
| | 3. | Použ. | vykoná zmeny v nastaveniach a uloží zmeny | | |
| | 4. | Systém | uloží vykonane zmeny a reštartuje program | | |
| **Alternatívna postupnosť** | **Krok** | **Rola** | **Činnosť** | | |
| | 4.a | Systém | v prípade nekorektného nastavenia oznámi používateľa varovaním oknom s popisom chyby | | |
| **Poznámky** | - | | | | |

Tab. 9 Prípad použitia UC19 Zobrazenie sw metrík

| **Názov** | **Zobrazenie sw metrík** | | | | |
|---|---|---|---|---|---|
| **ID** | UC19 | | | | |
| **Opis** | Zobrazenie sw metrík zdrojového kódu | | | **Priorita** | nízka |
| **Vstupne podmienky** | Zdrojový kód pre vygenerovanie metrík | | | | |
| **Výstupne podmienky** | Zobrazenie výsledkov metrík vo forme grafov | | | | |
| **Participant** | Používateľ (použ.) | | | | |
| **Základná postupnosť** | **Krok** | **Rola** | **Činnosť** | | |
| | 1. | Použ. | si zvolí možnosť zobraziť sw metriky z menu | | |
| | 2. | Systém | vygeneruje metriky zo zdrojového kódu a zobrazí výsledky vo forme grafov | | |
| **Alternatívna postupnosť** | **Krok** | **Rola** | **Činnosť** | | |
| - | | | | | |
| **Poznámky** | - | | | | |

## 4.2 Architektúra programu

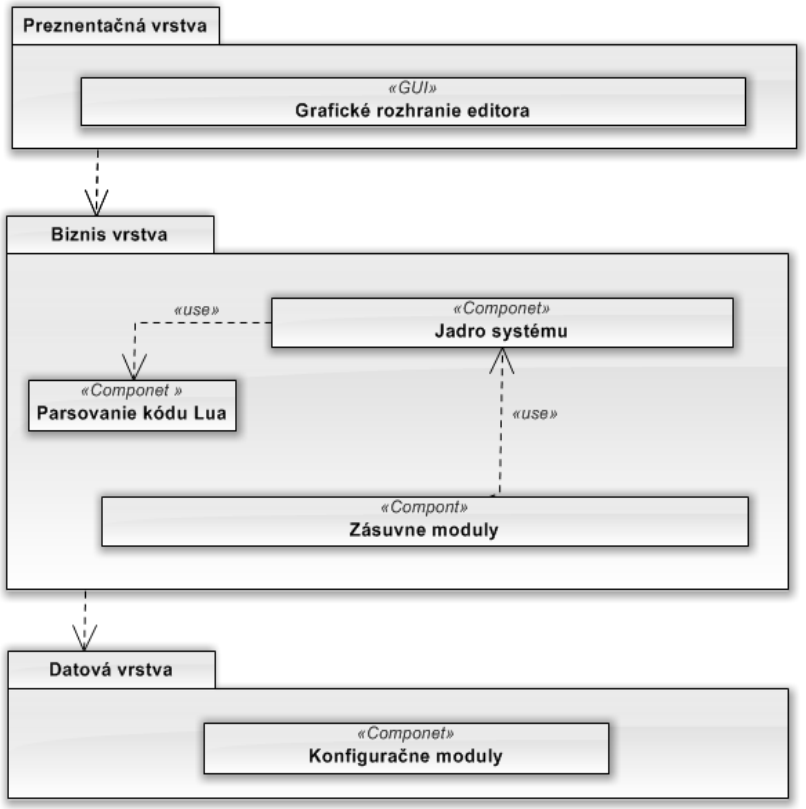
Architektúra programu bude postavená na klasickom trojvrstvom princípe, t.j. rozdelená na prezentačnú, biznis a dátovú vrstvu vid. Obr. 2.

V prezentačnej vrstve budú implementované triedy pre grafické rozhranie editora od hlavného menu až po nápovedu. Prezentačná vrstva bude komunikovať s biznis vrstvou, v ktorej bude spracovávaná aplikačná logika programu.

Biznis vrstva sa bude skladať z troch komponentov. Jeden pre jadro systému kde bude implementovaná základná funkcionalita programu. Druhý pre parsovanie zdrojového kódu,

kde sa bude vytvárať AST strom v skriptovacom jazyku Lua. A tretí pre pridávanie novej funkcionality, ktorá nenaruší základnú funkcionalitu jadra programu.

V dátovej vrstve budú dáta ktoré si bude program ukladať ako nastavenie programu dočasnú históriu zmien nad zdrojovým kódom a údaje o projekte.

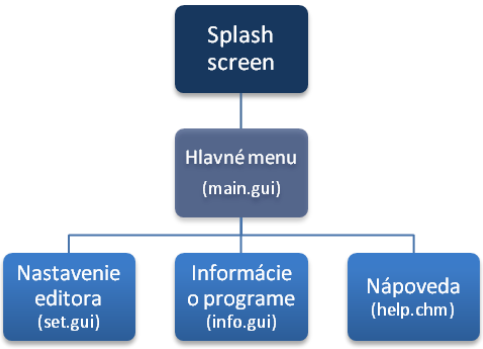



Obr. 2 Architektúra programu

//Refactoring  použiť nejake vzory ktore by sa nam tam hodily
//Nejaky diagram tried

## 4.3 Návrh GUI

Na základe analýzy sme identifikovali 5 okien, ktoré budú v programe implementované. Tieto okná budú mať hierarchicky význam, t.j. okno na vyššej úrovni môže volať iba okná na nižšej úrovni, nie však opačne.




Obr. 3 Hierarchické rozdelenie okien programu

**Hierarchia:**

- **Splash screen**
  - úvodné modálne okno, ktoré sa zobrazí vždy pri spustení TrollEditu
  - odstraňuje problémy studeného štartu
- **Hlavne menu**
  - nemodálne okno, ktoré slúži ako hlavné menu programu
  - z tohto okna je možné volať iné okna
- **Nastavenie editora**
  - modálne okno, ktoré slúži pre detailné nastavenie editora
- **Informácie o programe**
  - modálne okno, ktoré zobrazuje informácie o programe ako popis programu, dátum vytvorenia, verzia programu.
- **Nápoveda**
  - nemodálne okno pre zobrazenie nápovedy programu


Obr. 4 Splash screen programu


Obr. 5 Predbežný návrh hlavného menu programu

## 4.4 Analýza a návrh funkcionality UNDO/REDO

### 4.4.1 Qt Undo Framework

**Koncepcia**

Je to implementácia návrhového vzoru Command. Základnou myšlienkou tohto vzoru je, že každá zmena v aplikácií je vykonávaná pomocou vytvárania „Command" objektov. Tieto objekty aplikujú zmeny a sú uložené do „Command stack"-u. Vďaka tomu každý Command objekt vie ako vrátiť predchádzajúci stav dokumentu. Pomocou prechádzania stacku a zavolania funkcií undo() a redo() vieme vykonávať akcie UNDO/REDO.

**Návrh riešenia**

Do triedy MainWindow by sme vytvorili QUndoStack pre uchovávanie Command objektov a QUndoView na sledovanie a interakciu so stackom.

```
QUndoStack *undoStack;
QUndoView *undoView;
```

*Pomocou funkcie createUndoView() vytvoríme QUndoView*

```
void MainWindow::createUndoView()
 {
     undoView = new QUndoView(undoStack);
     undoView->setWindowTitle(tr("Command List"));
     undoView->show();
     undoView->setAttribute(Qt::WA_QuitOnClose, false);
 }
```

*Vo funkcií createActions() vytvoríme akcie UNDO/REDO*

```
 void MainWindow::createActions()
  {
      undoAction = undoStack->createUndoAction(this, tr("&Undo"));
      undoAction->setShortcuts(QKeySequence::Undo);
      redoAction = undoStack->createRedoAction(this, tr("&Redo"));
      redoAction->setShortcuts(QKeySequence::Redo);
 }
```

*Vytvoríme "Command" na zmazanie.*

```
 class DeleteCommand: public QUndoCommand
  {
  public:
      DeleteCommand (QGraphicsScene *graphicsScene, QUndoCommand
 *parent = 0);
      void undo();
      void redo();
  private:
      QGraphicsScene *myGraphicsScene;
  };
```

*Implementujeme funkcie undo() a redo()*

```
 void DeleteCommand::undo()
```

```
  {
      myGraphicsScene->addItem(myDiagramItem);
      myGraphicsScene->update();
  }
 void DeleteCommand::redo()
  {
      myGraphicsScene->removeItem(myDiagramItem);
  }
```

**Výhoda:** rýchla, prehľadná a škálovateľná implementácia funkcionality UNDO/REDO

**Nevýhoda:** vytváranie „Command" objektov pre každú akciu môže byť neefektívne.
potreba zmeny existujúcej implementácie.

### 4.4.2 QScintilla

Uloží akcie vykonávané na dokumente. Umožňuje kombináciu viacerých akcií do transakcií, na ktorými je potom možné vykonať UNDO/REDO.

Obsahuje rôzne príznaky, ktoré určujú či je možné alebo nie je možné vykonať UNDO/REDO.

**Výhoda:** poskytuje možnosť riadenia zmien na dokumentoch pomocou základných operácií UNDO/REDO.
QScintilla poskytuje aj ďalšie operácie vhodné pre prácu so zdrojovými súbormi. (Syntax highlighting, Searching,Replacing, Key bindings, Copy, Paste)

**Nevýhoda:** preštudovanie novej technológie a neprehľadná implementácia

### 4.4.3 QTextDocument

QTextDocument je grafický element, ktorý obsahuje formátovaný text.

Poskytuje sloty a funkcie na podporu funkcionality UNDO/REDO.

```
void redo ()
void setModified ( bool m = true )
void undo ()
```

Atribút modified poskytuje informáciu o tom či bol alebo nebol dokument modifikovaný. Okrem toho obsahuje funkcie aj na zistenie dostupnosti operácií UNDO/REDO.

**Výhoda:** nepotrebujeme nové technológie, malý zásah do existujúceho kódu

**Nevýhoda:** poskytuje len triviálne riešenie

## 4.5 Návrh funkcionality pre shortcuts

Spoločným použitím object model a action system v Qt môžeme vytvoriť prispôsobiteľné funkcie v editore akcií, ktoré sa dajú integrovať do už existujúcich aplikácií. Qt action system je založený na triede QAction, ktorá uchováva informácie o všetkých rôznych spôsoboch ako je možné spustiť určitý príkaz v aplikácií.

### 4.5.1 Dialógové okno

Vytvoríme si dialógové okno, ktoré umožní používateľovi prispôsobovať klávesové skratky v aplikácii. V jednom stĺpci budú vypísané akcie a v druhom stĺpci budú vypísané klávesové skratky, ktoré bude môcť používateľ editovať.

Definícia triedy ActionsDialog:

```
class ActionsDialog : public QDialog
  {
      Q_OBJECT

  public:
      ActionsDialog(QObjectList *actions,
                    QWidget *parent = 0);
  protected slots:
      void accept();

  private slots:
      void recordAction(int row, int column);
      void validateAction(int row, int column);

  private:
      QString oldAccelText;
      QTable *actionsTable;
      QValueList<QAction*> actionsList;
  };
```

Keď sa text klávesovej skratky edituje, tak sa využijú funkcie recordAction() and validateAction(). Funkcia accept() sa využije, keď dialógové okno akceptuje skratku.

Konštruktor plní úlohu vytvorenia užívateľského rozhrania . Na minimalizovanie vplyvu na aplikáciu sa trieda vytvorí v QObjectList a použijeme QTable na zobrazenie informácií.

```
ActionsDialog::ActionsDialog(QObjectList *actions,
                                QWidget *parent)
        : QDialog(parent)
    {
        actionsTable = new QTable(actions->count(), 2, this);
        actionsTable->horizontalHeader()->setLabel(0,
            tr("Description"));
        actionsTable->horizontalHeader()->setLabel(1,
            tr("Shortcut"));
        actionsTable->verticalHeader()->hide();
        actionsTable->setLeftMargin(0);
        actionsTable->setColumnReadOnly(0, true);
```

V tabuľke je povolené len jeden stĺpec editovať a odstránime z nej aj vertikálnu hlavičku na ľavej strane.

Každá akcia je taktiež pridaná do zoznamu, ktorý nám umožní vyhľadať akciu zodpovedajúcu danému riadku v tabuľke. Toto ešte budeme používať na modifikovanie akcií.

```
QAction *action =
                static_cast<QAction *>(actions->first());
        int row = 0;

        while (action) {
            actionsTable->setText(row, 0, action->text());
            actionsTable->setText(row, 1,
                                  QString(action->accel()));
            actionsList.append(action);
            action = static_cast<QAction *>(actions->next());
            ++row;
        }
```

Dialógové okno bude mať dve tlačidlá: OK a Cancel.

```
QPushButton *okButton = new QPushButton(tr("&OK"), this);
        QPushButton *cancelButton = new QPushButton(tr("&Cancel"), this);
        connect(okButton, SIGNAL(clicked()),
                this, SLOT(accept()));
        connect(cancelButton, SIGNAL(clicked()),
                this, SLOT(reject()));
```

Dva signály z tabuľky slúžia na editačný proces:

```
connect(actionsTable, SIGNAL(currentChanged(int, int)),
                this, SLOT(recordAction(int, int)));
        connect(actionsTable, SIGNAL(valueChanged(int, int)),
                this, SLOT(validateAction(int, int)));
        ...
        setCaption(tr("Edit Actions"));
    }
```

### 4.5.2 Proces editovania

Keď používateľ začne upravovať bunku, actionsTable vyšle signál currentChanged() a zavolá sa recordAction() s riadkom a stĺpcom bunky.

```
void ActionsDialog::recordAction(int row, int col)
    {
        oldAccelText = actionsTable->item(row, col)->text();
    }
```

Predtým, než používatľ dostane šancu modifikovať obsah, uložíme si aktuálny text bunky. Keď nebude vyhovovať zmenený text, tak sa text bunky vráti na pôvodný.

```
void ActionsDialog::validateAction(int row, int column)
    {
        QTableItem *item = actionsTable->item(row, column);
        QString accelText = QString(QKeySequence(
                                    item->text()));

        if (accelText.isEmpty() && !item->text().isEmpty()) {
            item->setText(oldAccelText);
        } else {
            item->setText(accelText);
        }
    }
```

Použijeme QKeySequence na kontrolu nového textu. Ak nový text nemôže byť použiteľný QKeySequence, tak sa do bunky uloží pôvodný text. Keď používateľ z bunky zmaže starý text a nezadá nový, tak bunka zostane prázdna. Keď nový text môže byť použiteľný, tak sa uloží do bunky a nahradí starý text.

```
void ActionsDialog::accept()
    {
        for (int row = 0; row < actionsList.size(); ++row) {
            QAction *action = actionsList[row];
            action->setAccel(QKeySequence(
                             actionsTable->text(row, 1)));
        }

        QDialog::accept();
    }
```

Pri stlačení tlačidla OK sa zmeny uložia a pri stlačení Cancel sa všetky zmeny stratia.

### 4.5.3 Editovanie, načítanie a ukladanie skratiek

V aplikácii musíme zabezpečiť otvorenie dialógu s používateľského rozhrania, načítanie nastavenia skratiek a ich ukladanie.

```
private slots:
        ...
        void editActions();
```

```
        void saveActions();

    private:
        void loadActions();
        ...


ApplicationWindow::ApplicationWindow()
        : QMainWindow(0, "example application main window",
                      WDestructiveClose)
    {
        ...
        QPopupMenu *settingsMenu = new QPopupMenu(this);
        menuBar()->insertItem(tr("&Settings"), settingsMenu);

        QAction *editActionsAction = new QAction(this);
        editActionsAction->setMenuText(tr(
                                  "&Edit Actions..."));
        editActionsAction->setText(tr("Edit Actions"));
        connect(editActionsAction, SIGNAL(activated()),
                this, SLOT(editActions()));
        editActionsAction->addTo(settingsMenu);

        QAction *saveActionsAction = new QAction(this);
        saveActionsAction->setMenuText(tr("&Save Actions"));
        saveActionsAction->setText(tr("Save Actions"));
        connect(saveActionsAction, SIGNAL(activated()),
                this, SLOT(saveActions()));
        saveActionsAction->addTo(settingsMenu);
```

Na prepísanie predvolených klávesových skratiek vložíme nasledovný kód na koniec konštruktorov:

```
...
        loadActions();
        ...
    }
```

Funkcia LoadActions () slúži na zmenu skratky každého QAction, ktorého názov zodpovedá záznamu v nastavení:

```
void ApplicationWindow::loadActions()
    {
        QSettings settings;
        settings.setPath("trolltech.com", "Action");
        settings.beginGroup("/Action");

        QObjectList *actions = queryList("QAction");
        QAction *action =
                static_cast<QAction *>(actions->first());

        while (action) {
            QString accelText = settings.readEntry(
                                          action->text());
```

```
            if (!accelText.isEmpty())
                action->setAccel(QKeySequence(accelText));
            action = static_cast<QAction *>(actions->next());
        }
    }
```

Táto funkcia závisí na predvolené hodnoty z QSettings:: readEntry (), aby zabezpečili, že každá akcia je zmeniť iba v prípade, že je vhodný vstup s platnou skratku k dispozícii.

editActions () vykonáva úlohu otvoriť dialóg so zoznamom všetkých akcií hlavného okna:

```
void ApplicationWindow::editActions()
    {
        ActionsDialog actionsDialog(queryList("QAction"),
                                    this);
        actionsDialog.exec();
    }
```

QObjectList vrátené QueryList () obsahuje všetky QActions v hlavnom okne, vrátane tých nových, ktoré sme pridali do menu.

saveActions() je volané vždy keď Save Action menu je aktivované:

```
void ApplicationWindow::saveActions()
    {
        QSettings settings;
        settings.setPath("trolltech.com", "Action");
        settings.beginGroup("/Action");

        QObjectList *actions = queryList("QAction");
        QAction *action =
                static_cast<QAction *>(actions->first());

        while (action) {
            QString accelText = QString(action->accel());
            settings.writeEntry(action->text(), accelText);
            action = static_cast<QAction *>(actions->next());
        }
    }
```

## 4.6 Návrh spracovania syntaktického stromu

Funkcia TreeElement *Analyzer::createTreeFromLuaStack() by sa zrušila a jej funkcionalita by bola nahradená čiastkovými volaniami z tabuľky na strane jazyka LUA. Tieto čiastkové volania by musela implementovať nová trieda, ktorá by plne nahradila triedu `TreeElement`.

Takáto zmena reprezentácie AST by znamenala značnú zmenu architektúry celej aplikácie, lebo ostatné triedy sú s triedou `TreeElement` silno previazané.

V C++ by sme už viac nevytvárali AST, len by sme sa dopytovali na strane LUA, kde by bol trvalo prístupný a menil sa len pri jeho modifikácií.

# 5 Implementácia prototypu

Táto kapitola popisuje implementáciu systému t.j. prevedenie návrhu do výsledného funkčného kódu.

## 5.1 Popis prototypu

# 6 Testovanie

V rámci tejto časti budú popísať spôsoby testovania a jednotlivé navrhnuté akceptačné testy.

## 6.1 Akceptačné testy pre overenie funkcionality

| **Názov** | **Použitie Undo/ redo** | | |
|---|---|---|---|
| **Rozhranie** | hlavne menu | **ID testu** | 01-13 |
| **Účel testu** | | **ID UC** | 13 |
| **Vstupne podmienky** | | | |
| **Výstupné podmienky** | | | |

| **Krok** | **Akcia** | **Očakávaná reakcia** | **Skutočná reakcia** |
|---|---|---|---|
| 1. | | | |

| | |
|---|---|
| **Úroveň splnenia testu** | splnil očakávanie |
| **Poznámka** | - |

**Inkrementálny a iteratívny vývoj**

- Analýza, špecifikácia požiadaviek a hrubý návrh (zimný semester)
  - *Úvod - o čom je tento dokument, ciele, ohraničenia.*
  - *Analýza problému a špecifikácia riešenia*
    (pre tvorbu softvérového systému typicky zahŕňa tieto časti: Kontext systému, Špecifikácia funkcií systému (určí sa aj priorita pre jednotlivé funkcie), Špecifikácia údajov v systéme, Špecifikácia správania systému)
  - *Hrubý návrh riešenia*
  - *Ďalšie požiadavky a ohraničenia*
- Prototyp (zimný semester)
  - Cieľ prototypovania, dosiahnuté výsledky
  - Podľa dohody s pedagógom, odporúča sa používateľská príručka (pre celý systém)
- Produkt a dokumentácia k produktu (letný semester)
  - Stanoví sa podľa povahy projektu. Štandardne zahŕňa tieto časti:
    - *Používateľská príručka*
    - *Systémová príručka* (spolu s návodom na inštaláciu)
- Návrh, implementácia a overenie riešenia (letný semester)
  - *Zapracovanie nedostatkov špecifikácie a hrubého návrhu*
  - *Návrh systému*
    (pre tvorbu softvérového systému typicky zahŕňa tieto časti: Architektúra systému, Fyzický model údajov systému, Návrh algoritmov spracovania)
  - *Ohraničenia, zmeny špecifikácie, priority riešenia*
  - *Výber implementačného jazyka a prostredia*
  - *Opis realizácie (implementácie jednotlivých modulov, napr. zaujímavé veci, optimalizácia, doplnenia oproti návrhu,...)*
  - *Overenie výsledku (určenie spôsobu overenia výsledku, postup, testovacie údaje, ak sa zmenili oproti návrhu)*
  - *Záznam o používaní systému*
  - *Čo sme nestihli*
  - *Čo sme sa naučili*