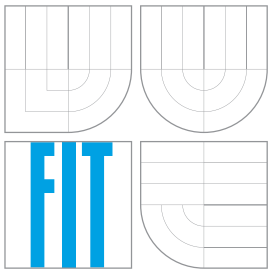
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

REKURZIVNÍ PROHLEDÁVAČ WEBU PRO KDE

RECURSIVE WEB SEARCH IN KDE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ HEFKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. RADEK BURGET, Ph.D.
SUPERVISOR

BRNO 2007

Rekurzivní prohledávač webu pro KDE

Zadání

1. Seznamte se s aplikačním rozhraním unixového pracovního prostředí KDE.
2. Prostudujte komponentovou architekturu tohoto prostředí.
3. Navrhněte aplikaci pro rekurzivní prohledání webových stránek s možností definovat různá omezení.
4. Implementujte navrženou aplikaci v jazyce C++.
5. Integrujte aplikaci do prostředí KDE.
6. Zhodnoťte dosažené výsledky a navrhněte možná rozšíření.

Licenční smlouva

Licenční smlouva je uložena v archivu Fakulty informačních technologií Vysokého učení technického v Brně.

Abstrakt

Tato bakalářská práce se zabývá problematikou prohledávání webových stránek z desktopového prostředí KDE. Součástí práce je seznámení s aplikačním rozhraním tohoto prostředí, popis návrhu a implementace aplikace. Ta podporuje řadu omezení, jak už na oblast hledání, tak na hledání samotné. Aplikace prochází internetem za pomoci rekurze a hypertextových odkazů stránek. V závěru práce jsou popsány výhody aplikace oproti jiným alternativám.

Klíčová slova

vyhledávač, KDE, Linux, rozhraní, vyhledávání, omezení, rekurze, Qt, C++, web, internet

Abstract

The bachelor thesis deals with problems related to web sites browsing from the KDE desktop environment. Part of this thesis is to introduce the application interface of this environment, suggestion description and application amplementation. The thesis supports range of restrictions both for searching area and searching itself. Application goes through the internet with the help of recursion and hypertext links. In conclusion there are descriobed application advatages in comparison with other alternative.

Keywords

finder, KDE, Linux, interface, searching, limitation, recursion, Qt, C++, web, internet

Citace

Lukáš Hefka: Rekurzivní prohledávač webu pro KDE, bakalářská práce, Brno, FIT VUT v Brně, 2007

Rekurzivní prohledávač webu pro KDE

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....

Lukáš Hefka

12. května 2007

Poděkování

Chtěl bych tímto poděkovat vedoucímu práce Ing. Radku Burgetovi, Ph.D., za jeho cenné připomínky a trpělivost, se kterou práci četl.

Obsah

Kapitola 1

Úvod

Tato práce si klade za cíl vytvořit aplikaci pracující pod linuxovým desktopovým prostředím KDE. Ta by měla sloužit k rekurzivnímu prohledávání webu s možností definovat řadu omezení, jak na oblast hledání, tak na hledání samotné. Implementačním jazykem je C++. Kapitola 2 obsahuje obecné informace týkající se internetového vyhledávání a vyhledávačů. V kapitole 4 je vysvětlen návrh aplikace v několika krocích. Především jsou zde nastíněné hlavní problémy a také objektový návrh. Kapitola 5 shrnuje implementaci aplikace a popisuje některé zajímavé implementační detaily. Celá práce i s dosaženými výsledky je shrnuta v kapitole 6.

Tento dokument byl vysázen systémem LaTeX.

Kapitola 2

Vyhledávání na webu

2.1 Vyhledávání dokumentů

Dnes už neodmyslitelná součást internetu, která s rostoucím počtem stránek nabývá na své důležitosti. Klasické získávání informací (information retrieval - IR) si klade za cíl získávání informace jako takové. Předpokládá **potřebu informace**. Na internetu nemusí být požadavek na vyhledávání čistě informační. Může jít např. o typ navigační (chci získat jednu požadovanou stránku) nebo transakční (najdi stránky, kde mohu učinit požadovanou činnost - transakci). Podrobnější informace lze najít v kapitole *3 A taxonomy of web searches*, [3].

Podle typu dotazů v internetových vyhledávačích je dělíme na tři druhy:

- Navigační
- Informační
- Transakční

Navigační dotazy – tyto typy dotazů použije uživatel v případě, že se chce dostat na nějakou **konkrétní stránku**. Má již představu o stránce, jen si ale nepamatuje konkrétní adresu nebo se chce na stránku dostat tímto způsobem (když se na ni tak dostal už dřív). Podobné typy dotazů se v klasickém IR označují také jako vyhledávání „známé položky“. Navigační dotazy se často používají pro vyhodnocování různých systémů. Tyto dotazy mají jako výsledek pouze jedinou správnou odpověď a dá se tak snadno zjistit, zda systém správně odpověděl či nikoli. Správná odpověď v podobě stránky z množiny odkazů (hub page) může být také uspokojivý výsledek, nicméně je již o něco nepohodlnější a horší než přesný odkaz.

Informační dotazy – důvodem použití informačních dotazů je snaha získat konkrétní informaci, která se na webu nalézá ve statické podobě. Takový výsledný dokument není odpovědí na konkrétní uživatelský dotaz. Informační dotazy jsou na webu často velmi obecné například: „sport“, „počítač“ nebo „Brno“, ale najdou se i dotazy

s konkrétnějším zaměřením jako: „Transkripční analýza“ nebo „Britský parlamentarismus“. A v téměř 15% takových dotazů je mnohem lepší odpovědí stránka, která vede na další informace (hub), než jeden určitý dokument.

Transakční dotazy – za tímto typem dotazu uživatel očekává ještě další akci. Mezi hlavní kategorie dotazů tohoto typu patří nakupování, stahování souborů (obrázků, písniček, atd.), přístup k nějaké databázi (poštovní směrovací čísla, zlaté stránky aj.) nebo vyhledání nějakého typu serveru s konkrétním zaměřením (herní server).

Z klasického pohledu IR se tento typ dotazu velmi těžko hodnotí. Nedá se jednoduše rozhodnout, zda je informace pro uživatele dostatečná nebo nedostatečná. U těchto typů dotazů se hodnotí také další externí faktory důležité pro uživatele (cena zboží, rychlost služby, kvalita obrázků apod.), které nejsou příliš dostupné obecným vyhledávačům.

Více o information retrieval si můžete přečíst např. v [1].

2.2 Internetové vyhledávače

Vyhledávací servery lze rozdělit do dvou základních typů:

- Fulltextové vyhledávače
- Katalogy

V následujícím textu je popsán rozdíl mezi fulltextovými vyhledávači a katalogy a způsob, jak fungují.

Fulltextové vyhledávače – pokud do fulltextového vyhledávače zadáte dotaz, vyhledávač nehledá daný výraz na internetu, nýbrž pouze v databázi stránek na svém serveru. Program takového vyhledávače (kterému se také říká **indexovací robot**) funguje následovně: prochází internetem a pomocí odkazů se dostává na další a další stránky, které následně ukládá do své databáze.[1] Robot prochází web v podstatě jako velkou „pavučinu“ po jejích vláknech, odkazech, které spojují jednotlivé stránky internetu mezi sebou.

Roboti tedy po příchodu na stránku „parsují “ (viz. kapitola *Vlastní HTML parser* 4.1) její zdrojový kód, zaindexují do své databáze a pokračují na vyhledaném odkazu, kde se činnost opět opakuje. Při indexování robot zpracovává a ukládá zdrojový kód stránky do své databáze na serveru vyhledávače. V této databázi je pak my, jako uživatelé, hledáme. Hledání tedy neprobíhá v reálném čase na internetu, ale pouze na serveru vyhledávače, a ten nám následně odešle do internetového prohlížeče výsledek tohoto hledání. Díky této metodě (indexace stránek na serveru) dostaneme výsledek hledání téměř okamžitě. Mezi největší nevýhody patří neaktuálnost dat.

[1] Někdy se také pro označení robota používá výraz *spider* (spider - pavouk) nebo *crawler* (crawl - lézt)

Katalogy – katalog odkazů je web, obsahující řadu odkazů na různé stránky internetu, řazené do tématických sekcí. Na rozdíl od fulltextových vyhledávačů se zápis stránek do katalogu provádí ručně. Na příslušné registrační stránce se stránka zaregistruje a odkaz je schválen pracovníky katalogu, kteří posuzují také jeho vhodnost pro danou kategorii. Při registrování odkazu do katalogu je obvykle třeba zadat URL stránky, titulek odkazu, který bude v katalogu sloužit současně jako odkaz na stránku, dále doplňující popis stránky, a zvolit vhodnou kategorii pro registraci. Většina katalogů nedovoluje v zápisu používat superlativy, vulgární nebo urážlivé výrazy a podobně.

2.3 Současný stav

V dnešní době moderních internetových vyhledávačů najde většina z nás na internetu to co potřebuje. Rok od roku se inteligence a funkcionalita těchto systémů zvyšuje. Klade se velký důraz na rychlost hledání a na porozumění zadaného dotazu.

Nyní se na internetu nachází obrovské množství nejrůznějších vyhledávačů. K nejznámějším z nich patří bez pochyby www.google.com, který si už dlouhou dobu udržuje pozici nejpoužívanějšího vyhledávače. Řada nezávislých měření ve světě uvádí podíl Googlu okolo 60 - 80%. Umožňuje prohledávat jen stránky ve zvoleném jazyce (jazykové mutace webu), hledat dle data změny internetové stránky či prohledávat jen určitou doménu. Mezi další světové konkurenty patří www.yahoo.com a www.msn.com. Více informací o světových vyhledávačích na searchenginewatch.com.

Na českém internetu je to převážně www.seznam.cz, který v současné době pro vyhledávání ve světě využívá služeb vyhledávače *Google*. Velmi moderní a ne příliš používané jsou u nás také jyxo.cz a morfeo.cz. Mají relativně dobré vlastnosti při vyhledávání českých výrazů. Dokáží ohýbání slov, umí rozpoznávat překlepy a také dobře pracují s diakritikou. Svou pozici na českém internetu posiluje také samotný Google.

Jak už jsem předeslal (kapitola *2. Teoretická část*, 2.2) vyhledávače se dělí do dvou základních skupin: **fulltextové** a **katalogy**. Obě kategorie mají problém s aktuálností dat. Vyhledávače fulltextové jsou závislé na prohledávacích robotech, kteří ne příliš často aktualizují indexační databázi. Může se tak stát, že vyhledávač obsahuje data o webových stránkách i několik týdnů či měsíců stará nebo je dokonce neobsahuje vůbec[2]. Katalogy jsou oproti fulltextovým vyhledávačům odkázány na uživatele internetu, kteří plní indexační databázi.

Jediný způsob zajišťující nalezení aktuálních dat je tzv. **real-time hledání**. Jedná se o vyhledávání informací na webových serverech v reálném čase. Tuto možnost nejvíce využijeme u hledání dat, které se často mění, jako např. zprávy, počasí, kurzy měn a jiné. Další možnosti využití jsou například u webů, které nepodporují vyhledávání (často to

[2]Tento fakt nastává v případě, že web je na internetu krátkou dobu a vyhledávač jej ještě nezaregistroval nebo na něj nevede žádný odkaz (robot se na něj nemá jak dostat).

bývají i emailové konference). Nepříjemnou vlastností těchto vyhledávačů je delší doba hledání. Oblast hledání se zpravidla zaměřuje na jeden konkrétní server a ne na celý internet.

V kategorii real-time vyhledávačů je softwarových produktů velmi málo. Jedním z nich je Web Finalist. Je určen pouze pro platformu Windows. Nabízí základní sadu vyhledávacích omezení jako jsou case sensitive a hledání celých slov. Na unixových systémech doposud žádný podobný software není. Tento fakt byl inspirací pro vytvoření této bakalářské práce - Rekurzivní prohledávač webu pro KDE.

Kapitola 3

Implementační prostředí

3.1 Desktopové prostředí KDE

KDE (K Desktop Environment) je desktopové prostředí pro Linux a jiné unixové operační systémy. V současné době patří k jednomu z nejpoužívanějších v této oblasti. První impuls pro vytvoření projektu KDE dal v roce 1996 Matthias Ettrich, který nebyl spokojen se současnou situací a především se mu nelíbila nejednotnost vzhledu jednotlivých aplikací. Jeho první verze (KDE 1.0) byla uvedena v červenci 1998. Nyní na projektu KDE pracují stovky vývojářů po celém světě, jedná se o svobodný software, který je šířen pod GPL (General Public License) licencí[1]. Významnou podporu vývoje software poskytují firmy MandrakeSoft, Trolltech a SuSE.

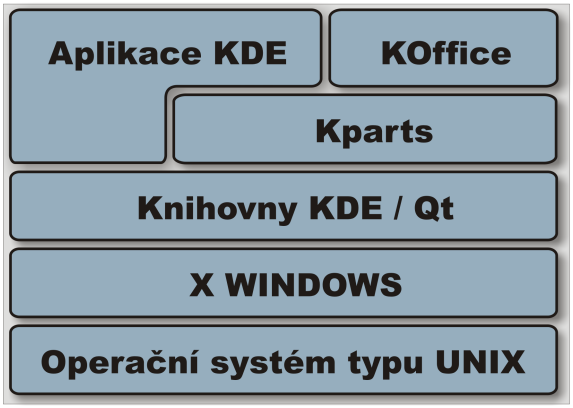
Toto prostředí je do značné míry podobné MS Windows. Obsahuje také řadu integrovaných aplikací, které jsou zahrnuty přímo do distribuce KDE. Jejich název většinou začíná písmenem K (KWrite, KPaint, KAlarm apod.). Souborovým manažerem a zároveň nativním WWW prohlížečem pro toto prostředí je Konqueror. KDE je také lokalizováno do češtiny. Prostředí se vyznačuje velkým množstvím konfigurací. Jiným všeobecně známým a používaným grafickým uživatelským prostředím pro unixové systémy je GNOME. Obě tato nejvýznamnější linuxová desktopová prostředí jsou v současnosti zapojena v projektu Freedesktop.org. KDE používá grafickou knihovnu Qt firmy Trolltech (viz 3.1.1).

3.1.1 Aplikační rozhraní KDE

V tomto desktopovém prostředí všechny aplikace vycházejí z objektu *KApplication* a odvozují se z objektu *KMainWindow*. Aplikace vytvořené touto cestou se pak automaticky přizpůsobují standardům KDE. Základní schéma KDE pro vývojáře je nastíněno na obrázku 3.1.

Pro vývoj korektních aplikací do tohoto prostředí je důležitých několik aspektů. Jedním z nich je seznam úkolů a vlastností KDE Application Developer's Checklist na [5], které

[1] Software pod licencí GPL může být svobodně používán a upravován. Šířen však opět pouze pod GPL, a to obvykle bezplatně.



Obrázek 3.1: Základní schéma KDE.

musí aplikace splňovat. Z nejzajímavějším bych uvedl správné umístění do CVS, napsat manuál, vytvořit standardní soubory `AUTHORS`, `ChangeLog`, `README`, `TODO`, `desktop`, `.lsm` (Linux Software Map), `.spec` (RPM balíčky), registrace na freshmeat.net, konfigurace uživatelského rozhraní v XML, definování standardů uživatelského rozhraní, kontrola portability, lokalizace všech textů.

Příkladem jednoduché aplikace pro KDE může být:

```
#include <kapp.h>
#include <kmainwindow.h>
int main (int argc, char **argv)
{
  KApplication a (argc, argv, ‘‘Hello KWorld!’’);
  KMainWindow *w = new KMainWindow();
  w->setGeometry(100, 100, 200, 100);
  a.setMainWidget(w);
  w->show();
  return a.exec();
}
```

Každý program určený pro KDE vyžaduje jednu instanci objektu *KApplication*. Této instanci pomocí metody *setMainWidget()* nastavíme hlavní okno aplikace - objekt z třídy *KMainWindow*. Metodou *setGeometry()* nastavíme umístění a velikost okna. Celou aplikaci spouští řádek *return a.exec();*.

Nativním vývojovým prostředím pro prostředí KDE je **KDevelop**[2]. Jedná se o projekt, který byl spuštěn v roce 1998 s cílem nabídnout programátorům, kteří potřebují

[2]Toto prostředí bylo použito pro vývoj této bakalářské práce - Rekurzivní prohledávač webu pro KDE

vyvíjet aplikace pro linux integrované vývojové prostředí pro jazyk C/C++[3]. Od té doby je publikován pod licencí GPL a podporuje projekty KDE, GNOME anebo programování v čistém C/C++. KDevelop poskytuje volně šiřitelné prostředí pro psaní programového kódu, zjednodušuje používání standardních nástrojů jako je autoconf a automake[4]. Mezi další výhody tohoto vývojového prostředí patří grafický dialogový editor pro rychlejší návrh a tvorbu grafického prostředí aplikace a přímý přístup k dokumentaci pro KDE a Qt API (Application Programming Interface - Rozhraní aplikačních programů). To programátorům umožňuje snížit dobu nutnou pro vývoj aplikace na minimum. I když KDevelop k práci potřebuje KDE a Qt, dá se použít pro vývoj jakéhokoli C++ projektu, protože je v první řadě navržen jako vývojové prostředí pro C/C++. Podrobnější informace lze najít na stránce projektu www.kdevelop.org.

3.2 Grafická knihovna Qt

Qt je multi-platformní C++ GUI systém, plně objektově orientovaný a jednoduše rozšiřitelný. Od jeho komerčního uvedení na trh v roce 1996 bylo na základech Qt vytvořeno tisíce aplikací. Jak už jsem předeslal dříve 3 tvoří základ linuxového desktopu KDE. I přesto, že přesná definice pro Qt je „C++ GUI system“, budu v této kapitole v souvislosti s Qt užívat pojem „knihovna Qt“.

V současné době je knihovna podporována na těchto platformách:

- MS Windows (od verze 95 po současné)
- Unix/X11 - Linux, Sun Solaris, HP-UX, IBM AIX, Digital Unix a další
- Macintosh - Mac OS X
- Embedded - Linux s podporou framebufferu

Knihovna Qt je produktem firmy Trolltech www.trolltech.com. Obsahuje sadu velmi užitečných nástrojů: `designer`, `assistent`, `uic`, `qt-config`, `qmake`, `moc`, `linquist`. Tyto nástroje pomáhají vývojáři zkrátit určité etapy vývoje aplikace. Domovskou stránku Qt lze nalézt na adrese www.trolltech.com/qt.

Programátor na středně pokročilé úrovni zvládne základy práce s knihovnou Qt velmi rychle. V následujícím textu se zaměřím na vlastnost, se kterou se programátor v C++ nesetkal, a která vyžaduje jistý čas na asimilaci programátorových technik. Jedná se o technologii signálů a slotů (viz. Signály a sloty 3.2.1) Další velmi zajímavou vlastností Qt jsou způsoby lokalizace aplikace, přenos na jiné platformy (nástroj qmake) atd. Popis těchto vlastností však výrazně přesahuje rámec této bakalářské práce.

[3] KDevelop jako částečná obdoba Visual Studia v prostředí Microsoft Windows

[4] V době psaní této bakalářské práce, tedy v době přechodu z KDE verze 3 na verzi 4 se zavádí nástroj cmake, který už taktéž KDevelop podporuje. Pro podrobnější informace o cmake bych odkázal na literaturu [6, Mastering Cmake 2.2 Edition]

Existují tři různé edice knihovny Qt:

Qt Enterprise Edition a Qt Professional Edition – edice sloužící pro vývoj komerčního software. Tyto edice zahrnují zdarma upgrade a technickou podporu. Informace o aktuálních cenách je možné nalézt na www stránkách firmy TrollTech nebo přímo kontaktovat *sales@trolltech.com*. Enterprise Edition nabízí oproti Professional Edition řadu rozšiřujících modulů.

Qt Free Edition – tato edice je dostupná pro Unix, k dispozici pouze pro vývoj „free software“ (Free and Open Source software). Je poskytována zdarma pod licencemi Q Public License[5] a GNU General Pulic License.

Qt/Embedded Free Edition – edice je určená pouze pro vývoj „free software“. Je poskytována zdarma pod licencí GNU General Public License.

3.2.1 Hlavní rysy

Knihovna Qt využívá standardní C/C++ třídy a funkce, sama ovšem nabízí velké množství vlastních tříd, jež definují objekty, které nám ve velké míře usnadní tvorbu aplikací (např. *QTime* pro práci s časem nebo *QRegExpr* pro práci s regulárními výrazy).

Největší zajímavostí knihovny Qt je vzájemná komunikace mezi objekty, dynamické identifikace typů a properties. Rozšíření jsou implementována pomocí maker a preprocesoru *moc* (Meta Object Compiler). Existují i rozhraní pro použití Qt z jiných jazyků jako například Perlu (PerlQt) a Pythonu (PyQt). Pokud chceme Qt aplikaci doplnit o skriptování, je možné použít nástroj QSA (Qt Scrtipt for Applications), který je založený na jazyce ECMAScript.

3.2.2 Signály a sloty

Signály a sloty slouží pro komunikaci mezi objekty. Mechanismus signálů a slotů je základní vlastností knihovny Qt a právě tato vlastnost dělá knihovnu jedinečnou. Při programování GUI je žádoucí, aby změna v jednom widgetu byla zaznamenána na widgetu druhém. Chceme tedy, aby objekt libovolné třídy byl schopen komunikovat s jiným objektem (opět libovolné třídy).

Příklad: Po stisku tlačítka objekt (pro vypsání textu např. Label) zachytí tuto činnost a vypíše nějaký text.

U starších grafických knihoven se ke komunikaci mezi objekty používaly tzv. „callback“ metody[6]. „Callbacks“ nejsou typově bezpečné a programátor nemá jistotu, zdali je „callback“ volán se správnými parametry. V české literatuře se termín „callbacks“ odborně

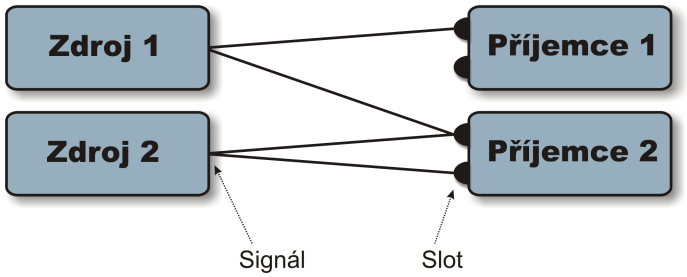
[5] Více o této licenci, viz. domovská stránka firmy Trolltech.

[6] „Callback“ metoda je v podstatě ukazatel na funkci.

překládá, jako „zpětné volání“. Prototyp funkce zpětného volání má tento tvar (X Toolkit, jazyk C K&R):

```
void (*XtCallbackProc)(Widget, XtPointer, XtPointer)
Widget w;                    // určuje widget, pro který byla funkce volána
XtPointer client_data;       // aplikačně závislá data
XtPointer call_data;         // data spec. pro zpětné volání
```

V Qt je tedy alternativa k volání technikou „callback“ - signály a sloty. Signál je vyslán, jestliže **nastane konkrétní událost**. Slot je funkce, která se volá jako **událost na určitý signál**. Pohled na signály a sloty se snaží přiblížit obrázek 3.2. Widgety v Qt mají mnoho předdefinovaných signálů a slotů, ale obvykle se přidávají signály a sloty vlastní. Tímto způsobem můžeme ovládat signály, které nás zajímají. Mechanismus signálů a slotů je typově bezpečný[7].



Obrázek 3.2: Komunikace objektů pomocí signálů a slotů.

Signály a sloty mohou obsahovat pouze ty třídy, které jsou odvozeny (děděny) od třídy *QObject*, nebo od libovolného následníka této třídy (např. *QWidget*). Objekty vysílají signály tehdy, jestliže se změní takový stav objektu, který může být zajímavý i pro „okolní svět“. Toto je způsob komunikace mezi objekty.

Sloty se převážně používají k přijímání signálů, ale jsou to také normální funkce a také se s nimi dá tak pracovat. Objekty nezajímá a nestarají se o to, zdali existuje objekt, který přijal vyslaný signál, či zdali byl na její sloty nějaký signál napojen. Objekty neví nic o komunikačním mechanismu. Do jednoho slotu je možno připojit i více signálů a naopak. Také lze jeden konkrétní signál spojit přímo s dalším signálem.

Mechanizmus signálů a slotů je velmi účinný. Za takovou flexibilitu si ale daň bere rychlost, která je nepatrně nižší než při použití „callbacks“ metod.

[7] Parametry signálů musí souhlasit s parametry přijímacího slotu.

Příklad: Jednoduchá ukázka předvádí použití signálů a slotů.

Mějme následující třídu:

```
class Foo
{
  public:
  Foo();
  int value() const { return val; }
  void setValue( int );
  private:
  int val;
};
```

Tuto třídu přepíšeme, aby podporovala signály a sloty:

```
#include <qobject.h>

class FooQt : public QObject
{
  Q_OBJECT
  public:
    FooQt();
    int value() const { return val; }
  public slots:
    void setValue( int );
  signals:
    void valueChanged( int );
  private:
    int val;
};
```

Tato třída má stejný privátní člen (int val) a stejné veřejné metody pro přístup k tomuto členu:

```
int value();
void setValue( int );
```

jako třída *Foo*. Třída *FooQt* má přidánu podporu pro využití slotů a signálů. Může tak oznámit okolnímu světu změnu svého vnitřního stavu *(int val)*, vysláním signálu *valueChanged()*. Má také definován slot *void setValue(int)*, kterému mohou ostatní objekty

posílat signály. Všechny třídy, které obsahují signály nebo sloty musí mít uvedené makro *Q_OBJECT* v deklaraci. Více o třídách v C++ v kapitole *Objekty a třídy*, [7].

Ukázka slotu z třídy FooQt:

```
void FooQt::setValue( int v~)
{
  if ( v != val )
  {
    val = v;
    emit valueChanged(v);
  }
}
```

Řádka *emit valueChanged(v);* vysílá signál *valueChanged(int)* čímž informuje okolní svět o změně hodnoty privátního členu *int val*. Z ukázky je patrné jakým způsobem lze vysílat signál:

```
emit signal(argumenty);
```

Možností jak propojit objekty dohromady je vícero. Jednou z nich je:

```
FooQt a, b;
QObject::connect(&a, SIGNAL(valueChanged(int)),
                 &b, SLOT(setValue(int)));
```

Preprocesor změní či odstraní klíčová slova *signals*, *slots* a *emit* tak, aby kompilátor jazyka C++ s nimi neměl problémy. Pokud spustíme program *moc*[8] na definici třídy, která obsahuje signály nebo sloty je zdrojový C++ soubor, který může být kompilován a linkován s ostatními objektovými soubory aplikace. (Zdroj [8])

3.2.3 Ladící techniky

Knihovna Qt obsahuje tři globální funkce pro výpis ladících a upozorňovacích textů.

QDebug – tato funkce slouží pro testovací výpisy při ladění programu.

qWarning() – funkce slouží pro upozornění, jestliže program způsobí chybu.

[8]Program moc (meta object compiler) analyzuje deklaraci třídy v C++ souboru a generuje C++ kód, který inicializuje meta object. Meta objekt obsahuje jména všech signálů a slotů včetně ukazatelů na tyto funkce. Více informací v kapitole *2 Creating Dialogs*, [2].

Příklad použití:

```
void f( int c )
{
  if ( c > 200 )
  qWarning( ‘‘f: chybný argument, c == %d’’, c );
}
```

qFatal() – při fatální chybě programu vypíše text a ukončí program
Příklad použití:

```
int divide( int a, int b )
{
  if ( b == 0 )                          // chyba v programu
  qFata( ‘‘dělení: nelze dělit nulou’’ );
    return a/b;
}
```

Kapitola 4

Návrh řešení

Aplikace bude umožňovat prohledávání webu za pomocí rekurzivního procházení stránek na základě obsažených hypertextových odkazů. Při vyhledávání bude možno definovat různá omezení. Důležitým aspektem je, že bude integrována do unixového desktopového prostředí KDE.

4.1 Požadavky na aplikaci

Z hlediska návrhu jsou požadavky následující:

- Efektivnost
- Vlastní HTML parser
- Omezení průchodu robota
- Parametry hledání
- Podpora různých znakových sad
- Hledání s diakritikou
- Uživatelské rozhraní

Efektivnost – jelikož aplikace bude prohledávat web v reálném čase, bude z tohoto důvodu hledání časově i výkonově náročné. Efektivita algoritmů bude hrát tedy nejdůležitější roli. Rychlost hledání musí být zatížena hlavně přenosovou rychlostí linky a ne zpracováváním obsahu.

Vlastní HTML parser – vytvoření vlastního HTML parseru má přímou souvislost s předchozím bodem, tedy efektivností. Parsování bude nejnáročnější operací celé aplikace, což by mohlo mít velký dopad na celkový výkon. Za předpokladu, že se budou zpracovávat soubory ve velikosti i několika megabajtů, nepřipadá použití technologií jako

např. *DOM*[1] v úvahu. Dalo by se uvažovat o obecných *HTML parserech*, které už jsou na tom výkonnostně podstatně líp, ale i přesto se snaží z HTML kódu vytěžit vše co se dá, což není žádoucí. Z tohoto důvodu jsem se rozhodl vytvořit parser vlastní, splňující přesně ty požadavky, které potřebuji. Ten jediným průchodem parsuje celý zdrojový kód HTML stránky.

Požadavky na parsování jsou tedy následující:

- Zjištění titulku stránky
- Uložení všech odkazů
- Uložení čistého textu
- Ignorování ostatních tagů, skriptů a komentářů

Omezení průchodu robota – průchod prohledávacího robota by měl být maximální mírou nastavitelný uživatelem. Jelikož se jedná o vyhledávání pouze v textovém obsahu, měl by umět rozpoznávat binární soubory a nezpracovávat je. Další velmi důležitou vlastností by mělo být ošetření proti zacyklení v rekurzi. Toho se dosáhne tím, že si bude pamatovat už prohledávané stránky a odkazy na tyto stránky ignorovat. Při stahování stránek by měl robot rozpoznávat *stavové kódy* vrácené webovým serverem. **Stavové kódy** jsou čísla, ke kterým přísluší stavová hlášení. Ta jsou jen slovním popisem daného kódu. Tato dvojice údajů v odpovědi klientovi hodnotí, jak se podařilo splnit jeho požadavek. Viz. tabulka 4.1.

Kategorie	**Rozsah stavových kódů**	**Popis**
Informační	100 - 199	Zprávy definované konkrétní aplikací.
Úspěch	200 - 299	Požadavek byl úspěšně zpracován.
Přesměrování	300 - 399	Klient musí pro konečné zpracování požadavků vykonat určitou činnost. O této činnosti se uživatel nemusí dovědět.
Chyba klienta	400 - 499	Problémy na straně klienta.
Chyba serveru	500 - 599	Problémy na straně serveru.

Tabulka 4.1: Stavové kódy a hlášení v odpovědi protokolu HTTP

O stavových kódech si můžete více přečíst na serveru Interval.cz [4].

[1] DOM (akronym anglického Document Object Model – objektový model dokumentu) je objektově orientovaná reprezentace XML nebo HTML dokumentu. DOM je API umožňující přístup či modifikaci obsahu, struktury nebo stylu dokumentu, či jeho částí. Definice modelu DOM podle Wikipedie[12].

Omezení budou definovat tyto podmínky:

- Počáteční adresa (startovací adresa, ze které bude robot vycházet)
- Hloubka rekurze (po kterou úroveň zanoření v odkazech má robot jít)
- Zda pouze v doméně nebo i mimo doménu
- Počet výsledků (počet výsledků, který uživateli stačí)

Parametry hledání – samotné vyhledávání na stránkách bude možno nastavovat několika parametry.

Parametry hledání:

- Hledaný řetězec
- Citlivost na velikost písma
- Ignorování diakritiky
- Celá slova (vyhledávání v celých slovech nebo jen v jejich částech)
- Pouze v titulcích
- Hledání regulárním výrazem

Podpora různých znakových sad – aby program uměl pracovat s různými znakovými sadami bude je muset z HTML dokumentu nějakým způsobem zjistit. Jsou dvě možnosti jak se dá znaková sada z HTML dokumentu vyčíst. Jedním ze způsobů je, že tento údaj pošle webový server **v http hlavičce**. Další možností je zjistit si znakovou sadu **v meta tagu HTML kódu**. První možnost má větší prioritu, což znamená, že pokud by webový server kódování v hlavičce neposlal, zjistí se kódování z meta tagu stránky. V případě, že by ani zde znaková sada uvedena nebyla, použije se *ASCII* kódování.

Po zjištění kódování HTML stránky se text převede do znakové sady *Unicode*. V této znakové sadě už bude bezproblémové hledání jakýchkoli znaků.

Hledání s diakritikou – aplikace bude umožňovat vyhledávání čistě např. českých výrazů. To znamená, že bude pracovat s diakritickými znaménky, pokud si tuto možnost uživatel zvolí. V případě že ne, bude se muset diakritika z řetězců odstranit. Pro takové nahrazení písmen s diakritikou za písmena bez ní se využije kódovacích tabulek. Jedna bude obsahovat výčet písmen s diakritikou a druhá bez ní. Přičemž písmena náležící k sobě se budou nacházet na stejných indexech. Tím se urychlí celý průběh konverze.

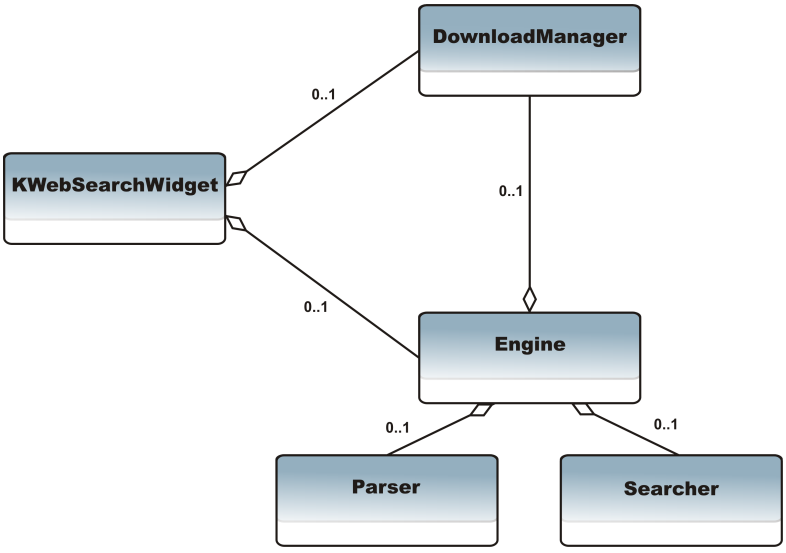
Export výsledků do XML souboru – aby se s výsledky aplikace mohlo dále pracovat bude aplikace umožňovat jejich uložení do XML souboru. Výsledky se myslí trojice hodnot pro každý nalezený záznam, což je:

- Adresa URL
- Titulek stránky
- Úryvek textu s nalezeným výrazem

Uživatelské rozhraní – ovládání aplikace bude jednoduché a intuitivní. Důraz bude kladen na přehledné zobrazování výsledků. Jelikož struktura webu představuje převážně „strom“, budou se výsledky zobrazovat ve stromovém uspořádáním. U jednotlivých výsledků by mělo být možné přejít na danou vyhledanou adresu nebo ji zkopírovat do schránky (clipboardu). Uživatelské rozhraní by mělo dále informovat o aktuálním stavu programu[2], počtu nalezených výsledků a době hledání. Komponenty pro nastavení parametrů hledání zaberou evidentně hodně místa, bylo by dobré panel s těmito volbami skrývat. Zvětší se tak prostor pro zobrazování výsledků, a tím se stane aplikace přehlednější. Při provádění časově náročných operací v průběhu hledání nesmí dojít k zablokování aplikace. Ta musí nadále komunikovat s uživatelem. Tomuto nežádoucímu jevu se předejde správným použitím signálů a slotů (viz. kapitola *Signály a sloty*, 3.2.1).

4.2 Objektový návrh

Stejně tak, jako ve většině projektů, tvoří správný objektový návrh velmi dobrý základ. U této aplikace to podle mého platí obzvláště. Asi největším problémem při objektovém návrhu této aplikace je rekurze, která tvoří jádro celého vyhledávacího systému.



Obrázek 4.1: Vazba mezi základními třídami aplikace

[2]Informace jako např. zda je program v inicializačním režimu nebo hledá nebo nastala chyba.

Hlavní třídou aplikace bude manažer celého hledání nazvaný *DownloadManager*, jehož největším úkolem bude správa rekurzivního průchodu internetem. Tato třída bude pro každou webovou stránku vytvářet objekt třídy *Engine*. Ten se bude starat o zpracování dané stránky. Stránku stáhne z internetu, za pomoci třídy *Parser* parsuje zdrojový kód na potřebné elementy a následně ve spolupráci s třídou *Searcher* vyhledá zadaný výraz v textu. Třída *DownloadManager*, kterou budou objekty třídy *Engine* plnit získanými odkazy ze stránek. Třída představující rozhraní aplikace se jmenuje *KWebSearchWidget*. V této třídě se bude inicializovat *DownloadManager* a spouštět celé hledání. Vazbu mezi těmito třídami lze vidět na obrázku 4.1.

Kapitola 5

Implementace

Aplikace byla implementována v jazyce *C++* za použití Qt toolkitu verze 3.3.6. Jako vývojové prostředí jsem použil nativní aplikaci prostředí KDE *KDevelop* verze 3.4, která splňovala veškeré mé požadavky. Pro optimalizování jsem využil nástrojů *Valgrind* a *KCachegring*[1].

5.1 Základní třídy

Zde bych se pokusil popsat jen několik základních tříd aplikace, jejich úlohu a také okrajově jejich implementaci. API celé aplikace včetně diagramů třídních vazeb lze nalézt na přiloženém CD.

Donwload manager – hlavní třída aplikace starající se o celé prohledávání webu. Obsahuje frontu požadavků[2], kterou plní objekty třídy Engine. Z této fronty vybere požadavek, vytvoří objekt třídy *Engine* a zavolá jeho metodu Download pro stažení a následné zpracování stránky. Aby nedošlo k zahlcení *socketu* (o této problematice více zde[10]) požadavky, má tato třída pevný počet slotů, které se mohou využívat. Sama si kontroluje obsazenost jednotlivým slotů a v případě, že jsou zaplněny, nedovolí stažení další stránky. Zacyklení v rekurzi ošetřuje tato třída tím, že obsahuje **hašovací tabulku** (viz. kapitola *14.1 Hašovací funkce*, [9]), do které ukládá již prohledané stránky. Při vstupu na novou stránku se tak zjišťuje, zda ještě nebyla stránka prohledána.

V průběhu zpracovávání požadavků musí třída kontrolovat, zda nedošlo k dokončení hledání. K tomu může dojít v následujících případech:

- Byl dosažen uživatelem zadaný počet výsledků
- Fronta požadavků a současně všechny sloty jsou prázdné (nečeká žádný požadavek ani žádný není zpracováván)

[1] Aplikace sloužící k zobrazení grafu volání. Používá se převážně v kombinaci s aplikací Valgrind

[2] Požadavky zde představují objekty třídy *Request*, což jsou ve své podstatě struktury, které v sobě nesou informace důležité pro stahování nové stránky.

- Přišel signál o explicitním ukončení hledání (uživatel hledání zastavil)
- Nastala chyba při spojení

Engine – hlavním úkolem této třídy je zpracování jedné stránky. Zpracováním se rozumí následující úkony

- Stažení webové stránky
- Parsování stránky
- Konverze textu z původní znakové sady do *Unicode*.
- Další úprava textu podle zadaných vyhledávacích parametrů (odstranění diakritiky, převod na malá písmena atd.)
- Vyhledání výrazu
- Podle nalezených odkazů se vytvoří požadavky (objekty třídy *Request*), které se přidají do fronty požadavků.

Parser – slouží k parsování zdrojového textu HTML stránky. Má největší podíl na efektivnosti celé aplikace. I přes veškeré mé snahy o optimalizování v ní program tráví nejvíce strojového času. Bázová metoda této třídy se nazývá *baseParser()*. Je tvořena převážně z jediného příkazu *switch*, kde každý *case* náleží jednomu stavu konečného automatu. Více v kapitole *Parsování HTML*, 5.1.

Searcher – jak už z anglického názvu vyplývá, tato třída se stará o vyhledávání výrazu v textu, jež byl za pomoci třídy Parser vytažen ze zdrojového kódu HTML. K tomuto hledání využívá knihovnu z Qt toolkitu `qregexp.h`. Na základě parametrů zadaných uživatelem vytvoří třída regulární výraz, kterým následně v textu vyhledává.

Příklad použití knihovny qregexp.h, která z textu zjistí hodnotu velikosti a jednotku délky:

```
QRegExp rxlen( ‘‘(\\d+)(?:\\s*)(cm|inch)’’ );
int pos = rxlen.search( ‘‘Length: 189cm’’ );
if ( pos > -1 ) {
    QString value = rxlen.cap( 1 ); // ‘‘189’’
    QString unit = rxlen.cap( 2 );  // ‘‘cm’’
    // ...
}
```

Základní metodou této třídy je metoda *Search()* vracející *true* nebo *false* podle úspěšnosti hledání.

KWebSearchWidget – je třída obsahující hlavní widget aplikace. Tvoří tedy rozhraní mezi uživatelem a aplikací. Stará se o správu komponent a zjišťování jejich stavu. Po spuštění vyhledávacího procesu, se v této třídě zjistí nastavené parametry hledání a předají se třídě *DownloadManager*, která se jimi bude řídit. Následně se vytvoří objekt třídy *Engine* a jeho metodou *Start* se spustí celý proces. Komunikace jednotlivých komponent se odehrává prostřednictvím **signálů a slotů** viz. 3.2.1. Jedním ze signálů jsou také nalezené výsledky, které jsou přímo zobrazovány v hierarchickém uspořádání. Pro tento výpis byla použita komponenta *ListView*, která nejvíce splňovala požadavky na přehlednost.

5.2 Parsování HTML

HTML parser je implementován jako *konečný automat*[3] obsahující 14 stavů. Ten pro každý znak vstupního řetězce přechází do jednoho ze stavů. Výsledkem parsování jsou dva řetězce obsahující titulek stránky a čistý text a datový kontejner **vector** z třídy *STL* naplněný nalezenými odkazy. Podrobnější popis konečného automatu spadá mimo rámec této technické zprávy, proto odkazuji alespoň na graf tohoto automatu v příloze A.1.

5.3 Konverze znakových sad

Aby program mohl pracovat se všemi texty (vrácenými *Parserem*) jednotně, bylo třeba jednotlivá kódování sjednotit do *Unicodu*. Protože Qt toolkit pracuje s *unicode* řetězci, byla tato volba jasná. Pro převod mezi kódováními jsem využil Qt knihovnu `qtextcodec.h`. Ta podporuje celou řadu kódování od klasického cp1250 až po nejrůznější japonské a čínské znakové sady. Používá se k překódovávání mezi Unicodem a danou sadou. Nejprve je nutno vytvořit codec pro znakovou sadu a následně použít buď funkci *fromUnicode()* nebo *toUnicode*. V našem případě to bude druhá varianta.

Příklad převodu řetězce v kódování *cp1250* do *Unicode*:

```
QTextCodec * codec;
QTextCodec::codecForName(‘‘cp1250’’);

QString unicodeString = codec->toUnicode(originalString);
```

O této problematice více zde[11].

[3] Definice **konečného automatu** podle Wikipedie[13]: Konečný automat (KA, též FSM z anglického finite state machine) je teoretický výpočetní model používaný v informatice pro studium vyčíslitelnosti a obecně formálních jazyků. Popisuje velice jednoduchý počítač, který může být v jednom z několika stavů, mezi kterými přechází na základě symbolů, které čte ze vstupu.

5.4 Export do XML

Export výsledků se neukládá do XML ve stromové hierarchii, tak jako tomu je při zobrazování v komponentě ListView. Všechny výsledky jsou jakoby v jedné úrovni. Podle uživatelem vybrané cesty a názvu souboru se výsledky uloží v následujícím formátu.

Element `results` obsahuje neomezené množství elementů `result`, které mají tyto atributy:

- title (titulek stránky)
- fragment (úryvek textu s hledaným výrazem)
- url (adresa stránky)

Příklad takového výstupu[4]:

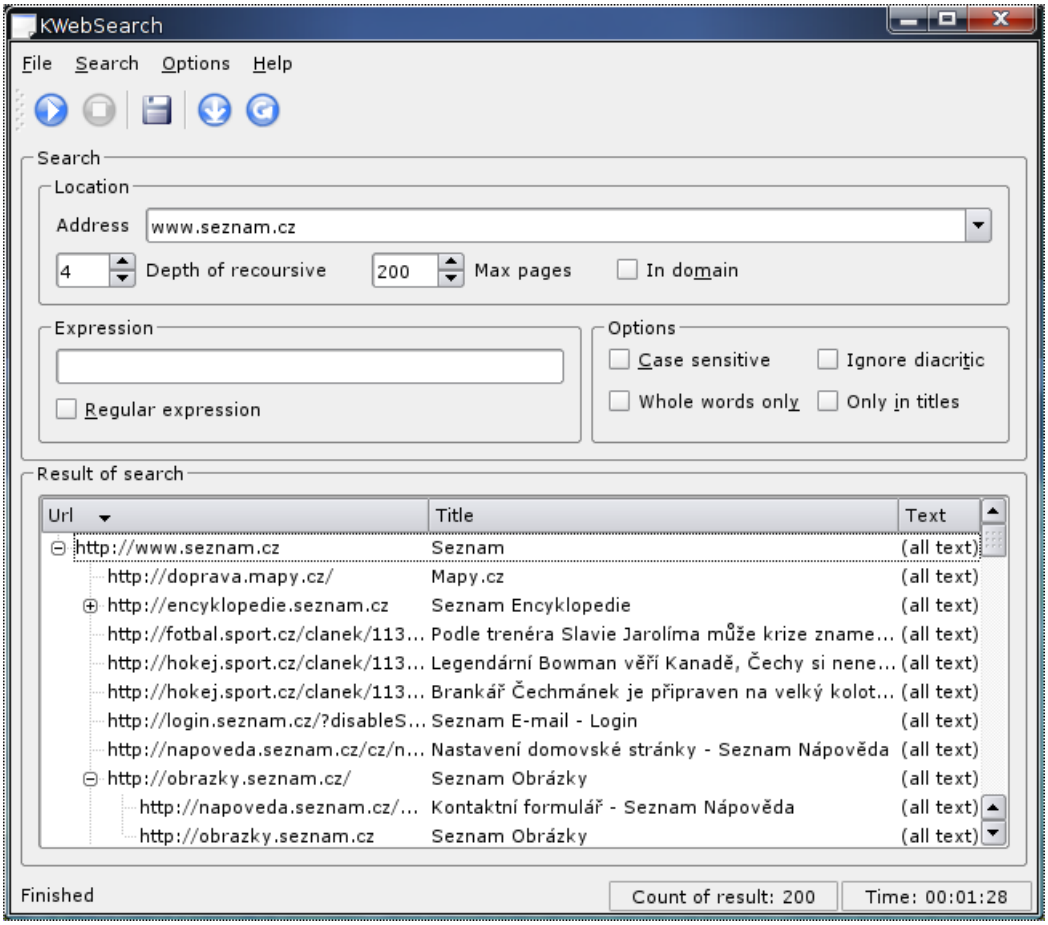
```
<!DOCTYPE XML>
<results>
 <result title=‘‘Úřední deska’’ fragment=‘‘(all text)’’
         url=‘‘http://www.fit.vutbr.cz/info/’’ />
 <result title=‘‘Aktuality’’ fragment=‘‘(all text)’’
         url=‘‘http://www.fit.vutbr.cz/lib/’’ />
 <result title=‘‘Odkazy’’ fragment=‘‘(all text)’’
         url=‘‘http://www.fit.vutbr.cz/links/’’ />
</results>
```

5.5 Grafické uživatelské rozhraní

Pro tvorbu uživatelského rozhraní bylo využito aplikace `Qt Designer`. Tento nástroj slouží k tvorbě rozhraní aplikací a formulářů založených na Qt toolkitu. Funguje buď jako samostatná aplikace nebo je integrována jakou součást IDE (KDevelop, Visual Studio a jiné.). Nabízí také jednoduchý manažer správy signálů a slotů nad formuláři. V aplikaci jsou použity jak komponenty samotného Qt toolkitu, tak prostředí KDE[5].

[4] Hodnota “all text” u atributu **fragment** znamená, že uživatel nezadal výraz pro vyhledání, tím se úryvkem textu stává text celý.

[5] Ve své podstatě komponenty prostředí KDE vycházejí z komponent Qt toolkitu.



Obrázek 5.1: Vzhled aplikace

Při rozložení komponent jsem kladl velký důraz na jednoduchost a přehlednost. Jak je vidět na obrázku5.1, panel pro nastavení parametrů hledání zabírá poměrně velkou část okna. To v případě zobrazení většího počtu výsledků hledání může být značně nepřehledné. Z toho důvodu jsem implementoval skrývání panelu pro nastavení parametrů hledání.

Kapitola 6

Výsledky

Cílem této práce bylo vytvořit aplikaci, která by prohledávala rekurzivním způsobem webové stránky. Do jaké míry se mi to podařilo bych chtěl nastínit v následující kapitole.

Aplikace pracuje korektně v desktopovém prostředí KDE. Dokáže prohledávat web podle nejrůznějších kritérií, která jednak vymezují pohyb robota, ale také upřesňují požadavek na hledání výrazu. Samotné ovládání je přesto velmi jednoduché a intuitivní. I když aplikace musí pro vyhledávání všechny stránky stahovat a následně s nimi pracovat, je rychlost hledání podle mého mínění dobrá. Pro srovnání na lince o rychlosti 2 MB/s prohledá průměrně 100 webových stránek za 18 sekund.

Rychlost hledání je ovlivňována několika faktory:

- Rychlostí linky
- Zadanými parametry pro hledání
- Výkonem počítače

Aplikace dokáže přehledně zobrazit výsledky hledání ve stromové hierarchii a umožňuje uživateli otevřít si vyhledanou stránku ve webovém prohlížeči nebo si odkaz zkopírovat do schránky. Všechny výsledky hledání také umí exportovat do XML souboru.

Pro názornost jsem provedl test, který udává, jak moc ovlivňují parametry pro hledání rychlost. Testovanými parametry jsou *Case sesitive* (citlivost na velká a malá písmena), *Ignore diacritic* (ignorování znaků s diakritikou) a *Whole words only* (hledání jen celých slov). Testování bylo prováděno na webových stránkách www.fit.vutbr.cz s rychlostí linky 2 MB/s a po dosažení 150 výsledků:

Číslo	**Case sensitive**	**Ignore diacritic**	**Whole words only**	**Čas**
1	TRUE	TRUE	TRUE	28s
2	FALSE	TRUE	TRUE	26s
3	TRUE	FALSE	TRUE	28s
4	FALSE	FALSE	TRUE	34s
5	TRUE	TRUE	FALSE	24s
6	FALSE	TRUE	FALSE	26s
7	TRUE	FALSE	FALSE	25s
8	FALSE	FALSE	FALSE	31s

Tabulka 6.1: Závislost nastavených parametrů na rychlosti hledání

Z tabulky lze vyčíst, že mezi nejhorším (24s) a nejlepším (34s) výsledkem je až 10s rozdíl, což je poměrně hodně. Nejvíce náročná volba je *Ignore diacritic* ve stavu TRUE. Musí se zde pro každý znak v řetězci vyhledat odpovídající hodnota v kódovací tabulce, což patří k náročnějším operacím.

Kapitola 7

Závěr

Práce se zabývala real-time vyhledávačem, který prochází web rekurzivním způsobem. Hlavním cílem byla snaha o co nejrychlejší hledání. Tento cíl byl z větší části splněn. Více než polovinu práce na aplikaci tvořila optimalizace a ladění.

Největším problémem bylo naučit aplikaci stahovat soubory tak, aby současně neblokovala uživatelské rozhraní. Pro stahování souborů jsem vyzkoušel nejrůznější knihovny. Nakonec jsem problém vyřešil knihovnou přímo od Qt toolkitu za pomoci signálů a slotů. Jedním z další problémů se ukázal HTML parser, jehož vytvoření bylo částečně problematické.

Aplikace by se do budoucna mohla rozšířit například o inteligentní odhad počtu prohledaných stránek. I přes tento fakt je aplikace plně použitelná.

Literatura

[1] Baeza-yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, ACM Press, 1999. ISBN 0-201-39829-X.

[2] Jasmin Blanchette and Mark Summerfield. *C++ GUI Programming with Qt 3*. Prentice Hall PTR; Pap/Cdr edition, 2004. ISBN 978-0131240728.

[3] Andrei Broder. *A taxonomy of web search*. ACM Press, 2002. ISSN 0163-5840.

[4] Interval.cz. Stavové kódy a hlášení v odpovědi protokolu HTTP. http://interval.cz/clanky/stavove-kody-a-hlaseni-v-odpovedi-protokolu-http/, 2002.

[5] KDE. KDE Application Developers' Checklist. http://developer.kde.org/documentation/other/checklist.html.

[6] Ken Martin and Bill Hoffman. *Mastering Cmake 2.2 Edition (Paperback)*. Kitware, Inc., 2006. ISBN 978-1930934160.

[7] Stephen Prata. *Mistrovství v C++*. Computer Press, 2004. ISBN 80-251-0098-7.

[8] Jan Pytel. Objektové uživatelské grafické prostředí pro vyrovnávání geodetických sítí. Master's thesis, České vysoké učení technické v Praze, 2001.

[9] Robert Sedgewick. *Algoritmy v C*. SoftPress, 2003. ISBN 80-86497-56-9.

[10] W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff. *UNIX Network Programming Volumen 1, Third Edition: The Sockets Networking API*. Addison Wesley, 2003. ISBN 0-13-141155-1.

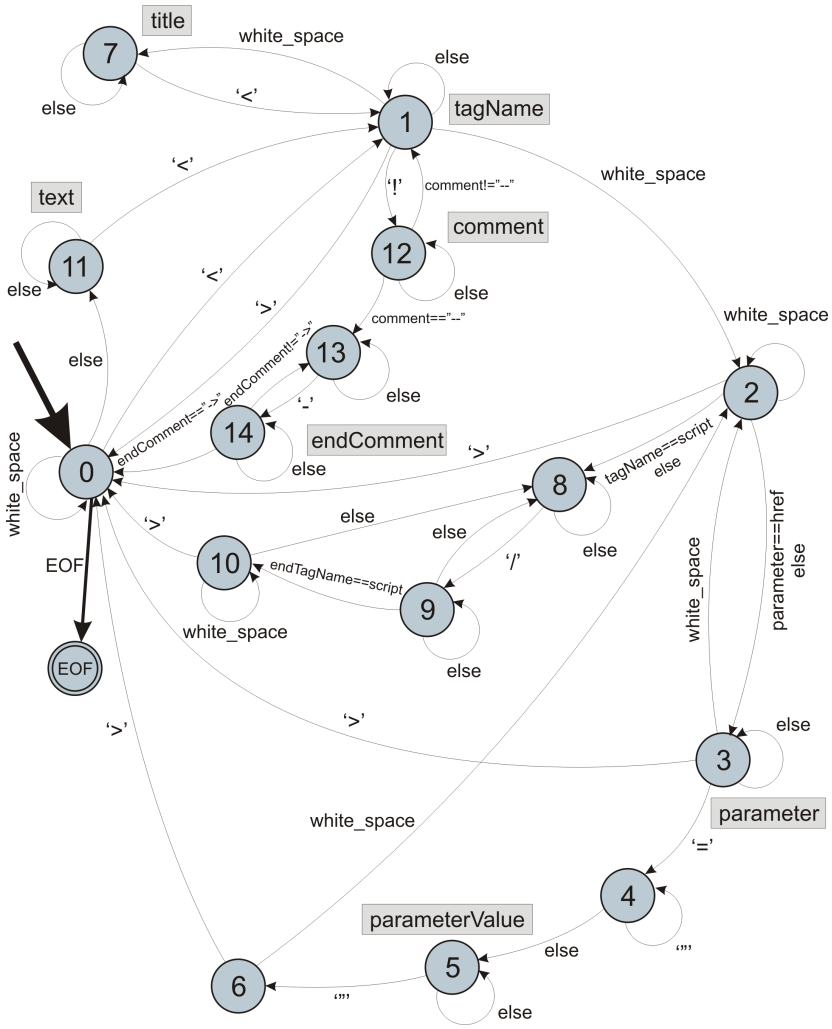
[11] Trolltech. Text Related Classes. http://doc.trolltech.com/3.3/text.html, 2005.

[12] Wikipedia. Document Object Model. http://cs.wikipedia.org/wiki/Document_Object_Model, 2007.

[13] Wikipedia. Finite state machine. http://en.wikipedia.org/wiki/Finite_state_machine, 2007.

Dodatek A

Konečný automat HTML parseru



Obrázek A.1: Graf konečného automatu