VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

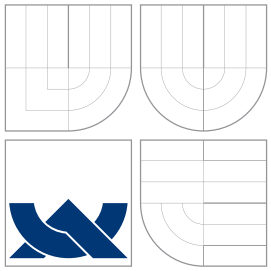
GUI PRE KONFIGURÁCIU FTP SERVERU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

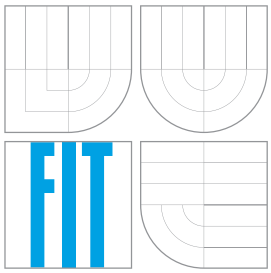
AUTOR PRÁCE
AUTHOR

MAROŠ BARABAS

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

GUI PRE KONFIGURÁCIU FTP SERVERU

A GUI FOR CONFIGURING AN FTP SERVER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MAROŠ BARABAS

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ VOJNAR, Ph.D.

BRNO 2007

Zadanie bakalárskej práce

1. Prostudujte protokol FTP a seznamte se s principy existujících implementací FTP serverů.
2. Detailně se seznamte s Vsftpd FTP serverem a s možnostmi jeho konfigurace. S ohledem na možnost vzájemného provázání konfigurací se také seznamte se způsoby konfigurace uživatelských účtů v systému společnosti Red Hat.
3. Navrhněte grafický nástroj pro konfiguraci Vsftpd serveru, který umožní nastavit jeho základní i rošířené volby v uživatelsky přehledném prostředí, a to s možností provázání s dalšími konfiguračními nástroji systému Red Hat.
4. V prostředí OS Linux naimplementujte navrženou aplikaci pomocí jazyka Python a grafického toolkitu PyGTK tak, aby vzhledově odpovídala prostředí Gnome.
5. Zhodnoťte vytvořenou aplikaci a diskutujte možnosti jejího dalšího rozvoje.

Licenčná zmluva

Licenčná zmluva je uložená v archíve Fakulty informačných technológií Vysokého učenia technického v Brne.

Abstrakt

Obsahom tejto bakalárskej práce je návrh a implementácia grafického konfiguračného nástroja pre ftp server vsftpd, distribuovaný do operačných systémov Red Hat Linux. Dôraz je kladený na jednoduchosť prístupu užívateľov ku konfigurácii servera, komplexnosť v prístupe ku konfiguračným možnostiam a ich škálovateľnosť. Program je integrovaný do desktopového prostredia GNOME.

Klíčová slova

vsftpd, python, PyGTK, GTK+, Gnome, grafický konfiguračný nástroj, system-config-vsftpd

Abstract

The subject of this document is concept and implementation of graphical configuration tool for vsftpd ftp server, which is distributed to Red Hat Linux operating systems. Mainly, the document puts accent on simplicity of user's access to server configuration, complexity of access to configuration options and their scalability. The program is integrated to GNOME desktop enviroment.

Keywords

vsftpd, python, PyGTK, GTK+, Gnome, graphical configuration tool, system-config-vsftpd

Citace

Maroš Barabas: GUI pre konfiguráciu FTP serveru, bakalářská práce, Brno, FIT VUT v Brně, 2007

GUI pre konfiguráciu FTP serveru

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Tomáše Vojnara. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Maroš Barabas
15. května 2007

Poděkování

Na tomto mieste by som chcel poďakovať svojmu vedúcemu Ing. Tomášovi Vojnárovi, Ph.D. za jeho trpezlivosť a ochotu, technickému vedúcemu Ing. Radkovi Vokálovi, ako aj firme Red Hat Czech za umožnenie pracovať na projekte pod ich vedením.

Obsah

1 Úvod **2**

2 Použité nástroje **3**
2.1 Python . . . 3
2.1.1 Porovnanie s inými jazykmi . . . 3
2.2 GTK+ / Python . . . 4
2.3 Glade . . . 4
2.4 Vsftpd . . . 4
2.4.1 XFerlog . . . 5

3 Návrh **6**
3.1 Analýza . . . 6
3.1.1 Funkcionálne požiadavky . . . 6
3.1.2 Požiadavky na beh systému . . . 7
3.1.3 Požiadavky na výsledný systém . . . 7
3.1.4 Základný funkčný model . . . 7
3.2 Nastolenie problémov . . . 8
3.3 Návrh štruktúry . . . 9
3.3.1 Štruktúra dát . . . 10
3.4 Grafický návrh . . . 12

4 Implementácia **15**
4.1 Modul Main . . . 15
4.2 Modul Render . . . 15
4.3 Modul FUtils . . . 15
4.4 Modul Logs . . . 16
4.5 Modul Parser . . . 17
4.5.1 Objekt ako zoznam . . . 17
4.5.2 Metóda LoadConfig . . . 18
4.5.3 Metóda SaveConfig . . . 19
4.6 Výsledná štruktúra aplikácie . . . 19

5 Porovnanie s existujúcimi nástrojmi **21**

6 Testovanie **23**

7 Záver **24**

Kapitola 1

Úvod

V 21. storočí v čase IT revolúcie a neustále stúpajúcej konkurencii je nutné ponúkať zákazníkom stále lepšie produkty, stabilnejšiu podporu a dbať na maličkosti, ktoré môžu rozhodnúť pri konečnom výbere produktu. Softwarové spoločnosti sa snažia vychádzať v ústrety zákazníkom pre ich pohodlie a spokojnosť a získať si tým ich dôveru. Aj z tohto dôvodu sa v operačných systémoch od firmy Red Hat snažia vývojári vyvinúť nástroje na konfiguráciu systému v užívateľsky čo najpríjemnejšej forme a samozrejme pokrývajúcej čo najväčší rozsah konfiguračných možností.

Jedným z takýchto programov by mal byť `system-config-vsftpd`, grafická pomôcka na konfiguráciu „Very Secure FTP“ serveru.

Táto práca sa snaží navrhnúť a vytvoriť konfiguračné grafické prostredie, ktoré by bolo intuitívne a uľahčilo by prácu mnohým, i skúseným administrátorom.

V nasledujúcich kapitolách budú objasnené aspekty návrhu a implementácie projektu. V kapitole 2 budú uvedené princípy a dôvody použitia vybraných technológií a princípy fungovania vsftpd serveru. V kapitole 3 sa zameriam na stručné priblíženie návrhu a analýzy a v kapitole 4 samotnej implementácii programu. Na záver bude krátke porovnanie s existujúcimi nástrojmi na konfiguráciu rôznych, väčšinou serverových aplikácií.

Projekt je vyvíjaný v spolupráci s vývojovým centrom firmy Red Hat Czech v Brne a Fakultou informačních technologií VUT v Brne.

Kapitola 2

Použité nástroje

V tejto kapitole budú zhrnuté hlavné princípy, klady a zápory vybraných programovacích jazykov, toolkitov a nástrojov, ich stručná charakteristika a pri programovacom jazyku python aj porovnanie s inými jazykmi, nad ktorými sa pri tvorení zadania projektu uvažovalo.

2.1 Python

Python je skriptovací objektovo-orientovaný jazyk vyvinutý z programovacieho jazyka C. Jednou z jeho najdôležitejších vlastností je rýchlosť akou je program napísaný, tzv. vyjadrovacia a konštrukčná schopnosť jazyka. Ďalšou veľkou výhodou tohto jazyka je jeho zameranie na prácu s reťazcami a regulárnymi výrazmi.

2.1.1 Porovnanie s inými jazykmi

V tomto odstavci je zdôvodnený výber jazyka python a porovnanie s ostatnými programovacími jazykmi. Tradičné jazyky ako C, C++ a Java majú spoločné charakteristiky, ako silné, statické typy, zložité cykly a pomerne dlhý kód na vykonanie relatívne jednoduchých úloh. Python obsahuje všetky zaužívané konštrukcie, ako cykly, podmienky, typy: pole, reťazce, atď, ale ich použitie je oveľa jednoduchšie.

- Riadenie pamäte je plne automatické, nie je potrebná alokácia, či dealokácia pamäte a o všetky objekty sa stará python. Jediný jazyk okrem pythonu, ktorý podporuje abstrakciu pamäte je Java.
- Operácie a práca s jazykom sú na vysokom stupni abstrakcie, čo je dôsledkom rozsiahleho kódu štandardnej knižnice.
- Všetky typy sú spojené s objektami, nie s premennými. Nie je potrebné definovať typ premennej, či typ položiek zoznamu. Dokonca každá položka zložených typov, ako je zoznam, slovník, či ntica môže byť iného typu.
- Produktivita kódu v porovnaní s ostatnými jazykmi je pri niektorých aplikáciách až päťnásobne rýchlejšia (u numerických aplikáciách je približne rovnaká).
- Pravidlá syntaxe sú jednoduché. Nie je napr. potrebné používať zložené zátvorky pri oddeľovaní funkčných blokov, ako je tomu v hore zmienených jazykoch. Python rozpoznáva bloky podľa odsadenia, takže každý kód, nech ho písal ktokoľvek, má približne rovnakú formu.

- V prípade potreby je jednoduché spojiť program v Pythone s modulmi jazyka C alebo C++

Nevýhodou jayzka je rýchlosť vykonávania samotného programu. Python, tak isto ako Java je jazyk interpretovaný. Kód programu je najprv preložený do vnútorného bytového kódu a následne vykonaný interpretom jazyka. Pri dnešnej výpočtovej rýchlosti a zložitosti nášho programu je toto spomalenie zanedbateľné.

Z dôvodu prístupu aplikácie ku konfiguračným súborom bol práve python ako programovací jazyk vybraný pre jeho špecializáciu na prácu s reťazcami a jeho nasadenie na projektoch rovnakého zamerania.

2.2 GTK+ / Python

GTK+ je multi-platformový toolkit pre tvorbu grafických aplikácií. Je to voľne šíriteľný software pod licenciou GNU LGPL ako časť projektu GNU. GTK+ pôvodne vzniklo ako toolkit pre grafický program Gimp. Dnes tvorí základ desktopového prostredia GNOME. GTK+ je napísaný v jazyku C a ponúka funkcie na prácu s objektami, tzv. widgetmi[1]. Keďže jazyk C nepodporuje objektovo orientované programovanie, bolo nutné do GTK+ implementovať prostriedky pre prácu s objektmi. Napríklad triedy OOP jazykov[2] sú definované pomocou štruktúr a virtuálne metódy sú riešené pomocou interných signálov. Navonok sa teda GTK+ javí ako objektový, preto použitie objektovo orientovaných jazykov nespôsobí problémy. Pre GTK+ existujú API pre jazyky C, C++, Ada, Perl, Python a iné.

GNOME je intuitívne grafické pracovné prostredie pre užívateľov. Tak ako GTK+ je časťou GNU projektu a implicitné grafické prostredie v systémoch firmy Red Hat. Jedným z bodov zadania tejto práce bola integrácia s užívateľským prostredím GNOME a práve preto je práca naprogramovaná s využitím GTK+.

2.3 Glade

Galde je vizuálny grafický nástroj, používaný na návrh vzhľadu aplikácie pre GTK+ a GNOME. Výsledný návrh aplikácie je uložený vo formáte XML nezávisle na programovacom jazyku, v ktorom je napísaný výsledný program. Do programu sa tento návrh dá načítať dynamicky pomocou knižníc pre daný jazyk (pre jazyk python je to knižnica libglade).[3]

2.4 Vsftpd

Vsftpd (Very Secure FTP Daemon) je FTP server pre Unixové systémy vydaný pod licenciou GPL. Je navrhnutý s prioritou bezpečnosti (viď názov) stability a extrémnej rýchlosti. Vsftpd taktiež podporuje nastupujúcu Ipv6 technológiu, PAM autentifikáciu, spojenie s využitím SSL a konfiguráciu povolení a zákazov pripojení pomocou „tcp wrappers“. Vsftpd nepoužíva na prihlasovanie do systému vlastnú databázu užívateľov, ale systémové účty, ktorým dokáže definovať oddelene rôzne konfiguračné nastavenia, takže každý užívateľ má vlastné nastavenie prístupu k serveru. Ďalšou vymoženosťou serveru je

[1] Widget je objekt v grafickom ponímaní
[2] OOP jazyky - Objektovo Orientované Programovacie jazyky
[3] Aktuálny zoznam podporovaných jazykov nájdete na adrese http://glade.gnome.org.

nastavenie aktívneho a pasívneho spojenia, čas potrebný na pripojenie, časové hranice na odpovede klientov a pod.

FTP server poskytuje možnosť pripojenia pomocou aktívneho a pasívneho spojenia (tak isto sa dajú zakázať) a ich oddelenú konfiguráciu. Dokáže vymedziť porty, z ktorých má vsftpd náhodne vyberať pre dátové i riadiace spojenia, obmedziť počet užívateľov pripojených v jednom okamihu, ako aj počet súbežných pripojení od jedného klienta. Server má pomocou regulárnych výrazov prepracované filtre na špecifikáciu súborov, ktoré sú užívateľovi skryté, alebo zakázané. Podľa potreby je možné užívateľom skryť skutočných vlastníkov súborov (užívateľov i skupiny) a priradiť novým súborom preneseným do systému špecifické práva a vlastníka. Veľkou výhodou je tzv. „chroot jail“ - užívatelia po prihlásení na server sú uzamknutí vo vybranom adresári, ktorý vnímajú ako koreňový adresár bez možnosti zmeny pracovného adresára na nadradený.

Existuje ešte množstvo iných špecializovaných možností, prepínačov alebo špecifikácií, ktorým sa tu nemôžeme venovať. Pri porovnaní s inými servermi je vsftpd flexibilný a pri striktnom použití tých správnych možností aj veľmi bezpečný. Toto sú asi tie najhlavnejšie dôvody vo výbere serveru pre tento projekt.

Vsftpd na túto konfiguráciu používa konfiguračný súbor (implicitne /etc/vsftpd/vsftpd.conf), ktorý môže obsahovať viac ako sto rôznych prepínačov a práve na konfiguráciu tohto súboru je potrebné grafické rozhranie.

2.4.1 XFerlog

XFerlog je štandardizovaný log pre rôzne služby. Daemon vsftpd má možnosť používať túto formu zaznamenávania prenosov medzi serverom a klientmi. Štruktúra XFerlogu je však komplikovaná:

Fri Apr 27 21:09:54 2007[4] 2[5] 88.102.192.93[6] 617819[7] /ples/foto1.jpg[8] b[9] _[10] o[11] r[12] guest[13] ftp[14] 0[15] *[16] c[17]

Viac na `http://bsdftpd-ssl.sc.ru/doc/unix/xferlog.5.txt`

Pre bežného užívateľa dosť neprehľadný formát logu, ktorý bude treba upraviť a zobraziť užívateľovi v prehľadnejšom formáte (viď kapitolu 4.4).

[4] dátum
[5] čas prenosu súboru
[6] IP pripojeného užívateľa (alebo hostname)
[7] počet prenesených bytov
[8] meno súboru s cestou
[9] typ prenosu: b - binárny, a - ascii
[10] špeciálne značky napríklad zabalený, a pod. (_ - žiadna značka)
[11] smer prenosu: o - output(von), i - input(dnu)
[12] identifikácia typu užívateľa: r - reálny užívateľ, a - anonymný užívateľ
[13] meno užívateľa
[14] typ služby
[15] autentifikačná metóda
[16] autentifikovaný užívateľ (* - každý)
[17] stav dokončenia prenosu: c - dokončený, i - nedokončený

Kapitola 3

Návrh

Návrh aplikácie sa odvíja od logiky fungovania ftp serveru a princípov tvorby užívateľských grafických rozhraní. Návrh sa člení na `analýzu`, v ktorej sa budem venovať požiadavkom na systém, ďalej na architektonický, podrobný a objektový návrh, ktoré budú zhrnuté do kapitoly 3.3 - `návrh štruktúry`. Táto kapitola sa venuje návrhu systému s dekompozíciou na podsystémy a definíciou vzťahov medzi nimi, ujasňuje logickú a fyzickú štruktúru dát a výber správnych postupov pri implementácii.

Dobrý návrh nám ušetrí veľa práce pri samotnej implementácii.

3.1 Analýza

Je úvodnou etapou pri vývoji softwaru, v ktorom sa zameriavam na požiadavky, ktoré sú kladené na systém. Pozornosť je venovaná len požiadavkom a nie samotnej implementácií problémov, alebo ich riešenie. V tejto kapitole sú zhrnuté požiadavky na výsledný program rozdelené do niekoľkých sekcií podľa typu požiadavkov, ktoré budú potrebné v návrhu programu.

3.1.1 Funkcionálne požiadavky

Požiadavky, ktoré majú určiť, čo má daný systém robiť, aké sú základné kritéria pri jeho priebehu. Na aplikáciu môžu byť vyčlenené rôzne požiadavky. Snažil som sa vystihnúť tie najzákladnejšie a zhrnúť ich do nasledujúcich bodov, z ktorých čerpám pri funkčnom návrhu výslednej aplikácie.

- Rozčlenenie konfiguračných možností do menších zrozumiteľnejších intuitívnych celkov
- Zobrazenie momentálneho stavu servera s možnosťou jeho zapnutia, či vypnutia
- Dynamické menenie konfigurácie podľa závislosti na konfiguračných premenných
- Možnosť návratu k pôvodnej konfigurácii pred štartom aplikácie
- Uloženie konfigurácie a reštart serveru bez nutnosti vypnutia programu
- Zobrazenie logov serveru
- Znovu vytvorenie implicitnej konfigurácie

- Otvorenie akéhokoľvek konfiguračného súboru

Pre užívateľa je dôležité pochopenie rozčlenenia samotnej aplikácie pre orientáciu v programe. Užívateľ nezvykne konfigurovať server ako celok, vždy hľadá konkrétnu možnosť. Do úvahy sa berie možnosť zmeny konfiguračného súboru, zmenu ciest a názvov súborov s logmi, ako aj ich formát.

3.1.2 Požiadavky na beh systému

Definuje, za akých podmienok bude aplikácia pracovať, aké sú požiadavky na výkon systému, koľko užívateľov bude pristupovať k systému, odozva systému, nutnosť obmedzenia práv aplikácie pri behu a ostatné požiadavky, ktoré ovplyvňujú beh systému.

- Aplikáciu bude v reálnom čase obsluhovať iba jeden užívateľ
- Nie sú známe žiadne požiadavky na výkon alebo dobu odozvy
- Aplikácia nepúšťa žiadne externé programy, okrem konzolových príkazov - nie je potrebné obmedzenie práv

Nie je známy žiaden dôvod na špeciálny prístup k aplikácii, ani žiadne zásahy do normálneho behu štandardnej užívateľskej aplikácie.

3.1.3 Požiadavky na výsledný systém

Definujú predovšetkým požiadavky súvisiace s výkonom a nasadením aplikácie. Požiadavky na hardwarové a softwarové vybavenie, bezpečnosť, spoľahlivosť, odolnosť voči chybám a pod.

- Aplikácia nie je závislá na type architektúry ani špecifikácii hardwaru
- Ošetrenie výnimiek v prístupe k súboru (neautorizovaný prístup, existencia súboru)
- Aplikácia musí byť prenositeľná medzi operačnými systémami

Ošetrenie výnimiek pri behu systému zaručí identifikovateľnosť chyby bez nutnosti metód typu backtrace a pod. Úplná prenositeľnosť aplikácie nie je možná, pokiaľ nevznikne vsftpd pre iný operačný systém ako Unixový.

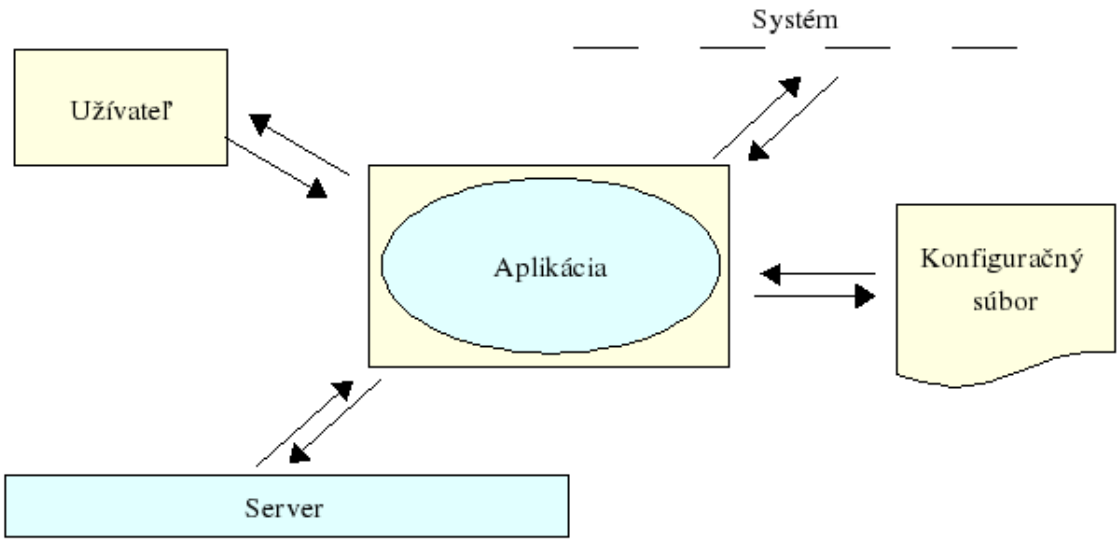
3.1.4 Základný funkčný model

Základom funkčného modelu je proces užívateľa, tok dát medzi užívateľom a konfiguračným súborom a tok dát medzi vlastným serverom a aplikáciou.

Priebeh procesov (informácií) v systéme bude vznikať pri interakcii s užívateľom: užívateľ zmení hodnotu v systéme, vykoná akciu. Aplikácia zaznamená zmenu v konfigurácii, ktorá sa prenesie do konfiguračného súboru alebo sa zmena prejaví na stave serveru (vypnutie, zapnutie serveru).

Základným kameňom modelu je samozrejme samotná aplikácia. Keďže ide o užívateľskú aplikáciu, musíme do modelu zahrnúť užívateľa. Ďalšou časťou modelu je konfiguračný súbor, na ktorom je aplikácia postavená. Dáta budú teda tiecť po dráhach užívateľ - aplikácia a aplikácia - konfiguračný súbor. V prípade, že ide o laického užívateľa a aplikácia

je navrhnutá, tak aby sme mohli užívateľovi umožniť konfiguráciu mimo konzoly, musí byť program schopný server na požiadanie vypnúť, zapnúť, prípadne reštartovať. A nakoniec, ak budeme uvažovať o prípadnej spolupráci so systémom, ako napríklad prezeranie logov, musíme do modelu zahrnúť tok dát medzi aplikáciou a systémom. Z tohoto úsudku dostávame diagram na obrázku 3.1. Tento model nám poslúži na nastolenie hlavných problémov pri špecifikácii návrhu[1].



Obrázok 3.1: Funkčný model aplikácie

3.2 Nastolenie problémov

Z hore uvedeného modelu (Obr. 1) sú viditeľné štyri hlavné uzly, pri ktorých môžu nastať problémy v toku informácií. Vyriešenie problémov pred samotnou implementáciou zamedzí vzniku nepríjemných kruhových závislostí. Tie vznikajú pri riešení problémov presunom časti kódu, jeho špecializáciou alebo generalizáciou, pri ktorých vznikajú obdobné problémy vyžadujúce rovnaké riešenia, s rovnakým výsledkom.

Prvý problém môže nastať pri poskytovaní adekvátnych informácií užívateľovi. Intuitívne rozmiestnenie objektov v rozhraní je kľúčovým aspektom v spokojnosti užívateľa. Konfiguračné premenné pospájané vo vzájomných závislostiach sa budú ťažko rozdeľovať do logických skupín. Grafický návrh musí byť vopred prepracovaný do detailu.

Druhý problém môže nastať pri prístupe ku konfiguračnému súboru. Pri načítaní je zo súboru nutné vyradiť komentované riadky, postupne vyberať každú premennú s jej hodnotou a vkladať ich do vhodnej štruktúry, ktorá prenesie na grafické rozhranie aplikácie a zobrazí užívateľovi konfiguráciu v akej sa nachádza konfiguračný súbor. Pri prístupe k súboru za účelom zápisu novej konfigurácie nastáva hneď niekoľko menších problémov. Prvým a najhorším je zachovanie pôvodnej konfigurácie aj s komentovanými riadkami a pridanie nových premenných s odstránením redundancie. Ďalším problémom je zmena hodnoty

[1] viď kapitolu 3.2

pôvodnej premennej a posledným problémom je vymazanie premenných, ktoré obsahujú implicitné hodnoty servera.

Tretí problém viditeľný z modelu, je komunikácia medzi serverom a aplikáciou. Počas vykonávania programu bude nutné zistiť stav serveru, prípadne ho spustiť, zastaviť, či reštartovať.

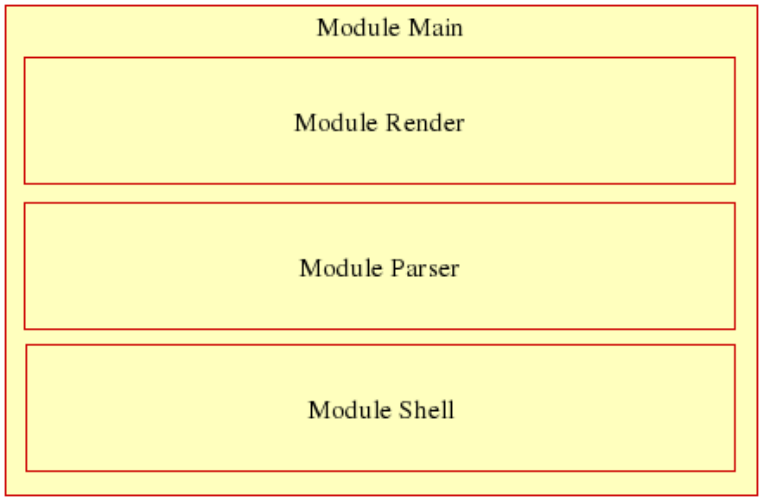
Štvrtým problémom je realizácia samotnej komunikácie so systémom. Potrebné bude čítanie logov, zistenie existencie súboru (možné vyriešiť aj výnimkou pri otváraní súboru), zistenie práv užívateľa a pod.

Medzi problémy môžme zaradiť aj požiadavky na aplikáciu z časti 3.1.

3.3 Návrh štruktúry

Návrh projektu sa delí na architektonický, podrobný a návrh objektovej štruktúry aplikácie. Architektonický návrh nadväzuje na prechádzajúcu analýzu a slúži na dekompozíciu projektu. Architektonický návrh obyčajne nemusí presne odrážať skutočný už hotový projekt, má poskytnúť predstavu o budúcom vývoji. Podrobný návrh je zameraný na stanovenie logickej a fyzickej štruktúry a na spôsoby ošetrovania chybných stavov a výnimiek. V objektovom návrhu je uvedená štruktúra aplikácie z pohľadu rozčlenenia kódu na objekty, moduly a popis vzťahov medzi nimi. V nasledujúcom texte budú pre názornejšie zhrnuté tieto tri typy návrhov do jedného.

Postup pri návrhu aplikácie sa sústredil na logické usporiadanie aplikácie. Aplikácia by mala byť rozdelená do logických celkov, ktoré budú poskytovať funkcie na zobrazenie aplikácie, na komunikáciu so systémom a funkcie na narábanie s konfiguračným súborom.



Obrázok 3.2: Modulárna štruktúra aplikácie z pohľadu návrhu

Návrh projektu sa skladá zo štyroch hlavných modulov (viď obr. 3.2), ktoré obsahujú rovnomenné objekty. Z hľadiska objektovej štruktúry budeme v nasledujúcom texte chápať tieto moduly ako objekty :

Modul Main – volá sa po spustení aplikácie a riadi celé konanie aplikácie, až po jej ukončenie. Volá metódy objektov z ostatných modulov podľa interakcie s užívateľom.

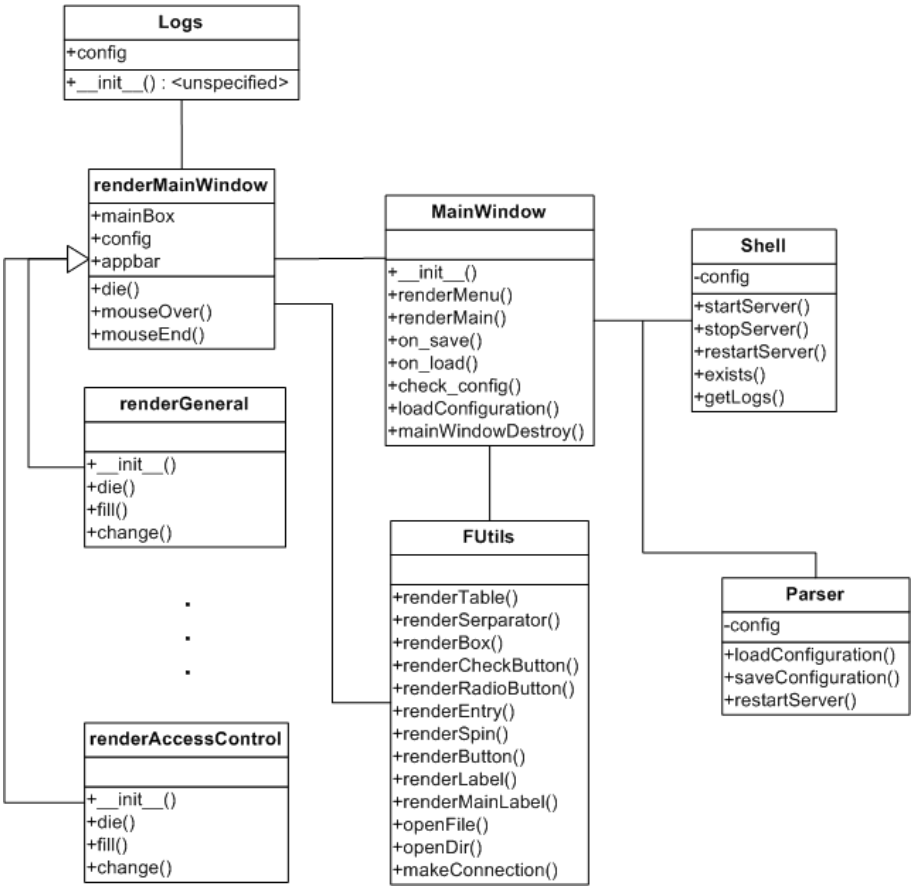
Modul Render – objekt, ktorý má na starosti vykresľovanie aplikácie, prideľovanie funkcií widgetom a implementáciu chovania týchto widgetov pri interakcii s užívateľom

Modul Parser – objekt, ktorý obaľuje konfiguračný súbor metódami, prostredníctvom ktorých poskytuje abstrakciu v prístupe k súboru.

Modul Shell – objekt na abstrakciu komunikácie so systémom a serverom. Modul by mal poskytovať metódy na prácu s okolím.

Komunikácia medzi modulmi bude zabezpečená nadradenosťou modulu Main, ktorý zabezpečuje volania metód z ostatných modulov.

Na obrázku 3.3 je vidieť už výsledný diagram tried[2]. Objekty v diagrame približne zodpovedajú názvom modulov, v ktorých sú umiestnené. V module Render sa nachádza objekt `renderMainWindow`, od ktorého dedia objekty, ktoré vykresľujú jednotlivé položky menu (viď časť 3.4).



Obrázok 3.3: Diagram tried

3.3.1 Štruktúra dát

Pri návrhu štruktúry dát musíme vziať do úvahy vhodnosť rôznych typov pre rôzne účely tak, aby výsledná aplikácia nebola príliš náročná na pamäť systému a bola pri tom do-

[2]Tento diagram je upravený zmenami pri implementácii.

statočne zrozumiteľná.

Prvým závažnejším problémom je výber štruktúry, do ktorej umiestnime konfiguračný súbor. Naskytli sa dve hlavné možnosti.

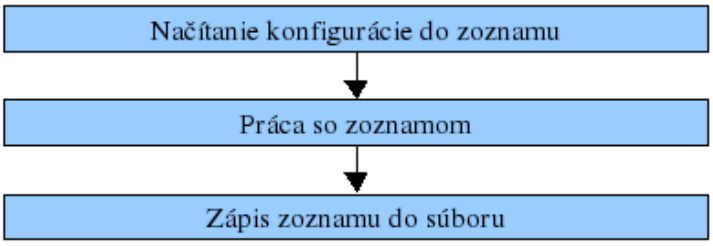
Prvou z nich je nenačítať súbor do štruktúry, ale zmeny robiť priamo do súboru. To znamená, že by sme načítali zo súboru riadky, ktoré nie sú komentované, rozparsovali[3] by sme ich a funkčný obsah súboru zobrazili užívateľovi. Každá zmena by sa odrážala v systémovej premennej (zoznam alebo slovník) a uloženie súboru by prebiehalo v dvoch cykloch, v ktorých by sme hľadali danú premennú, vymazali riadok s premennou a vložili novú hodnotu, alebo by sme riadok zakomentovali.

Druhou možnosťou je načítať celú obsah súboru do zoznamu (zoznam riadkov) a následne pracovať iba s touto premennou. Do súboru by sme vložili zmeny až pri ukladaní. To znamená, že celý proces aplikovania zmien by prebiehal počas behu program v načítanom zozname a pri ukončení programu by sme súbor prepísali našou verziou.

Iné možnosti vznikali len rôznymi kombináciami spomenutých algoritmov zápisu. Pri hlbšej analýze tohto problému som dospel k nasledujúcemu záveru:

Pri práci s konfiguračným súborom v jazyku python stojí na jednej strane rýchlosť aplikácie, ktorá je znižovaná interpretačnou vlastnosťou jazyka, častými zásahmi do periférnych zariadení a dlhými a zložitými cyklusmi a na druhej strane práca s pamäťou, ktorú (možno zbytočne) využíva nadmerne. Štandardne nakonfigurovaný súbor s implicitnými hodnotami systému Red Hat a pridanými 10-timi riadkami s vlastnými úpravami dosiahol veľkosť 4683 bytov čistého ASCII textu, ktorý nám zaberie tú istú veľkosť pamäte po jeho načítaní (cca 5KB). Pri týchto číslach nemôžeme hovoriť o zbytočnom mrhaní pamäte a súbor načítaný do zoznamu nám môže dopomôcť k rýchlosti aplikácie, ktorá je v prípade jazyka python rozhodujúca (viď kapitolu 2.1).

Práca s dátami v aplikácii bude teda po implementácii vyzerať nasledovne:



Obrázok 3.4: Diagram postupu práce s dátami z konfiguračného súboru

Práca so zoznamom vyzerá celkom jednoducho, ale náš problém to nevyrieši. Zoznam nám poskytuje náhradu za súbor v prípade, ak chceme ušetriť čas programu na úkor pamäte, ale druhý problém, ktorý potrebujeme vyriešiť, je práca s konfiguračnými premennými a ich hodnotami. Zoznam je postupnosť riadkov súboru, takže prvým zákrokom musíme odstrániť komentáre. V prípade, že ich jednoducho vymažeme, ich nadobro stratíme. Server po načítaní konfigurácie síce bude fungovať bez zmeny, ale v konfiguračnom súbore zmiznú

[3]Parsovanie - rozdelenie na jednotlivé logické zložky

komentáre a to nie je želaný stav. Z toho vyplýva, že budeme musieť pracovať len s riadkami, ktoré nezačínajú znakom komentára „#“. Druhým problémom je spôsob prístupu k premenným a hodnotám. Konfiguračný súbor má nasledujúcu syntax:

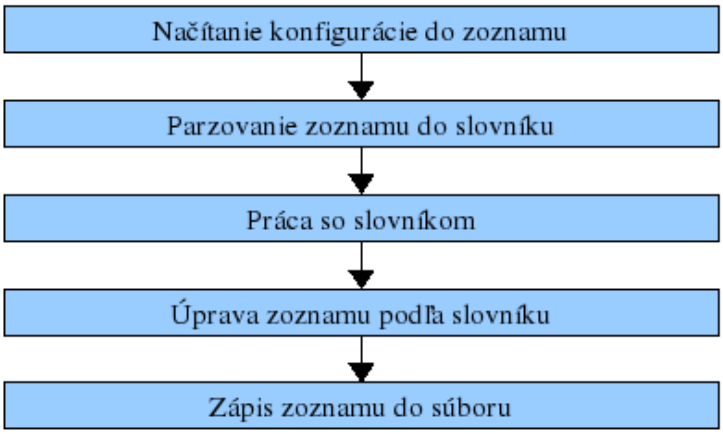
```
premenna=hodnota
```

Premenná musí začínať na začiatku riadku a medzi premennou a znakom „=“, ako aj medzi ním a hodnotou nesmie byť žiaden znak (V prípade, že táto syntax nie je dodržaná, program by mal konfiguračný súbor náležite opraviť a skončiť bez chyby). Takto „čisté“ dáta je možné uložiť do slovníku[4] a pristupovať k nim cez abstrakciu objektu, ktorý bude zabaľovať prácu so súborom.

Pred uložením novej konfigurácie do súboru sa preklopia nové hodnoty do zoznamu. Následne zoznam prepíše pôvodný súbor a program sa ukončí.

Ďalšou dátovou štruktúrou, ktorá je potrebná pri behu aplikácie je slovník implicitných hodnôt, porovnaním s ktorým dokážeme určiť, ktoré hodnoty nepotrebujeme v konfigurácií definovať. Pomocou rozdielu tohto slovníka so slovníkom, v ktorom prebiehajú počas behu aplikácie zmeny, odstránime premenné, ktoré sú nepotrebné (inak by nám súbor rástol, kým by neobsahoval všetky možé premenné).

Práca s dátami v aplikácii bude teda po implementácii vyzerať nasledovne:

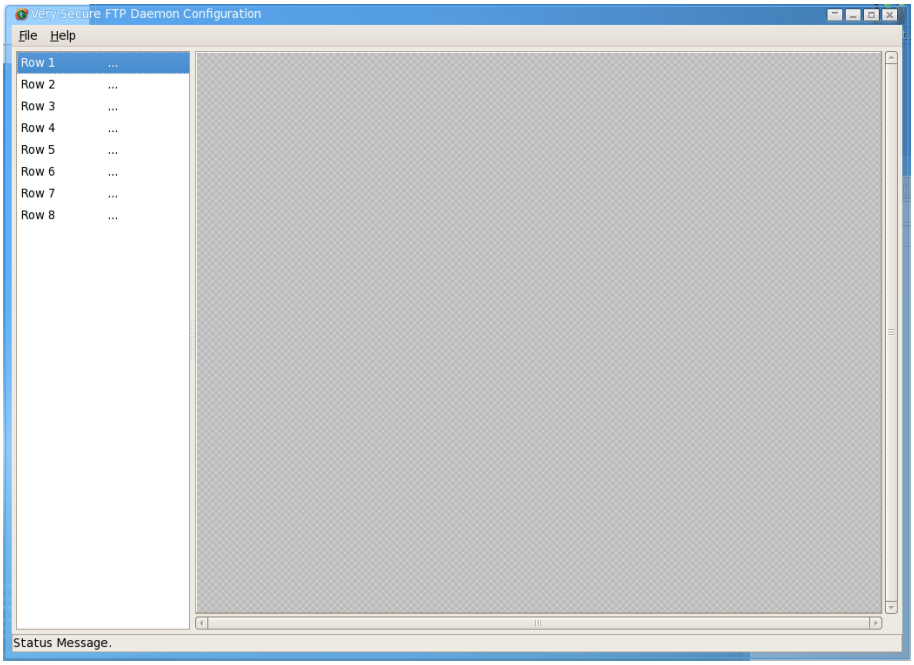


Obrázok 3.5: Diagram postupu práce s dátammi po implementácií slovníka

3.4 Grafický návrh

Grafický návrh je postavený na štyroch hlavných objektoch. Do prvých dvoch (menu tvorené objektom gtk.TreeView a zobrazovacia plocha, ktorú tvorí objekt gtk.VBox) sa budú generovať ďalšie objekty dynamicky podľa vzniknutých udalostí. Druhé dva objekty sú viac menej statické: horné menu obsahujúce základné položky ako tlačítko na otvorenie súboru, uloženie a ukončenie programu a dolná lišta, do ktorej budeme zobrazovať pomocné správy (viď obr. 3.6).

[4] Špecifická štruktúra jazyka Python, každá položka poľa kľúčov obsahuje hodnotu (ako slovník)



Obrázok 3.6: Grafický návrh aplikácie z nástroja Glade

V návrhu je ešte zakomponovaný dialóg, ktorý sa spúťša pri uložení konfigurácie a pýta sa užívateľa, či má aplikácia reštartovať server.

Aplikačné menu bude pozostávať z častí, kde každá časť tvorí logický celok, v ktorom sú združené prepínače, ktoré s ňou súvisia[5]. Nižšie sú uvedené časti:

- **General** – obsahuje základné možnosti behu servera, protokol nad ktorým má bežať[6].
- **Server Control** – kontrola aktuálneho stavu servera v aplikácii, možnosť zapnutia, či vypnutia servera a výpis hlavných logov.
- **Acccess Control** – obsahuje možnosti, ktoré kontrolujú prístup k serveru, menia výpis uvítaní a ovplyvňujú žívateľov v prístupe.
- **Users** – obsahuje záložku pre lokálnych užívateľov a pre anonymných užívateľov. Možnosti v tejto časti ovplyvňujú chovanie užívateľov v systéme, redukujú rýchlosť prístupu,...
- **Directory Options** – táto časť narába s chovaním výpisov zložiek zo servera. Dokáže zakázať zobrazenie alebo stiahnutie vybraných súborov pomocou regulárnych výrazov, schovanie reálnych vlastníkov súborov,...
- **Logging** – základné možnosti na konfiguráciu zaznamenávania prístupov, chýb, prenosov a pod.

[5] Viď obrázok 7.1

[6] Vsftpd nedokáže naraz bežať nad Ipv4 aj Ipv6. Východiskom z tejto situácie je mať spustené dve inštancie servera s rozdielnymi konfiguračnými súbormi

- **Transfer log** – časť, v ktorej užívateľ nájde grafickú interpretáciu tzv. xferlog logu, do ktorého sa zaznamenávajú prenosy zo/do serveru. Grafické rozhranie poskytuje prehľadný strom prístupov v poradí podľa dňa prístupu, IP klienta, ktorý pristupoval k serveru a zoznamu súborov ku ktorým pristupoval s podrobným výpisom daného prenosu

Hlavné okno, ktoré sa nachádza na pravej strane od menu, slúži na vykresľovanie rôznych prepínačov. Generuje sa dynamicky v závislosti na tom, ktorá časť z aplikačného menu bola vybraná. Toto okno sa nachádza v takzvanom `scrollWindow` widgete, ktorý zaručí, že sa automaticky na spodnej (pravej) strane objaví posuvník, ak text v okne pretečie za nastavenú šírku (výšku).

Spodná lišta, tzv. `application status bar`, je určená na zobrazovanie nápovedy. Vždy, keď užívateľ prejde myšou cez akýkoľvek prepínať v hlavnom okne, zobrazí sa na tejto lište krátky text, ktorý charakterizuje daný prepínač.

Kapitola 4

Implementácia

V tejto kapitole budú priblížené implementačné metódy a riešenie problémov z návrhu z kapitoly 3, podrobnejší výklad štruktúry aplikácie z pohľadu modulov a objektového modelu.

4.1 Modul Main

Základná časť aplikácie, ktorá obsahuje jeden objekt `MainWindow`, ktorého konštruktor spúšťa celú aplikáciu, volá metódy na generáciu ľavého menu aplikácie (viď obr. 7.1) a prideľuje volania metód každému objektu menu. Po kliknutí na konkrétnu položku v menu sa zavolá metóda konštruktoru konkrétneho objektu z modelu render. Ďalšou metódou objektu MainWindow je metóda, ktorá sa volá pri ukončení aplikácie a zodpovedá za uloženie zmien.

4.2 Modul Render

V module render sa nachádza všetok kód ktorý sa stará o vytváranie grafiky a o jej interakciu s užívateľom. V module Render sa rozdelil rovnomenný objekt z návrhu na viacero menších objektov, každý z nich zobrazujúci jednu kartu, ktorá predstavuje logickú časť konfigurácie (napr. karta `Users` pre skupinu prepínačov ovplyvňujúce hlavne užívateľské účty). Pri kliknutí užívateľa na konkrétnu položku z menu aplikácie sa zavolá konštruktor objektu, ktorý je položke pridelený. Následne vygeneruje všetky objekty a zobrazí ich do hlavného okna. Nakoniec spojí metódy so signálmi od zobrazených objektov a naplní ich príslušnými hodnotami.

Každý objekt zobrazujúci jednu zo skupín dedí od objektu `renderMainWindow`, ktorá obsahuje metódu `die(self, parent)`, ktorá vyčistí obsah hlavného okna[1] a metódy `mouseOver(self, widget, value)` a `mouseEnd(self, widget, value)`, ktoré majú za úlohu napĺňať status bar vetami[2], ktoré vysvetľujú daný objekt, ktorý metódu zavolal. Tieto metódy volá každý objekt (widget), cez ktorý prejde užívateľ myšou.

4.3 Modul FUtils

Postupom času do aplikácie pribúdali objekty, ktoré vizuálne odzrkadľovali stav konfiguračného súboru. Pri približne 70% z celkového počtu prepínačov mal modul render 7 ka-

[1] volá sa vždy, keď je nutné vyčistiť hlavné okno - pred volaním konštruktoru niektorého z objektov
[2] Slúži ako jednoduchá stručná nápoveda

riet (skupín) a 2800 riadkov. Kód bol neprehľadný pre príliš veľké množstvo opakujúcich sa volaní metód na vkladanie rôznych objektov.

Príklad: Objekt „Frame“ - obaľuje iné objekty s vizuálnym ohraničením a nadpisom. Pri tvorbe tohto objektu musel byť volaný nasledujúci kód:

Časť kódu:

```
...
frame = gtk.Frame( )
frame.set_label( ' Label ' )
frame.set_label_align( 0, 0.5 )
frame.get_label_widget( ).set_use_markup( True )
frame.set_shadow_type( gtk.SHADOW_ETCHED_IN )
...
```

Z tohto dôvodu pribudol do projektu modul FUtils, ktorý neobsahuje žiaden objekt, ale funkcie, ktoré volajú konštruktor daného objektu, viažu ich s inými objektami a nastavujú im vlastnosti. Ukážka volania funkcie je uvedená pod textom.

```
[frame, box] = futils.renderFrame( 'Server status' )
```

Podobnou abstrakciou sme zaručili menší kód a v prípade jednej zmeny v obsahu funkcie sa zmeny premietnu do celej aplikácie.

4.4 Modul Logs

Tento modul vznikol pre parsovanie `xferlog` logu (slúži na zaznamenávanie prenosov zo/do servera). Do xferlogu sa zapisuje štandardizovaným tzv. wu-ftpd kompatibilným formátom[3]

Je potrebné užívateľovi poskytnúť prehľadnejšie riešenie zobrazenie logov. Objekt Log načíta xferlog súbor do zoznamu riadkov a prechádza ich v cykloch po riadkoch. Ako prvý načíta dátum (prvý záznam), vyjme z neho rok, vloží ho do slovníku, kde kľúčom je rok a hodnotou je vnorený slovník. Každý riadok, ktorý začína týmto rokom uloží do tohto slovníku, kde kľúč je dátum a položka je zoznam s riadkami. Taká istá metóda je použitá pre IP, takže dostávame dátovú šrtuktúru, ktorá vyzerá nasledovne:

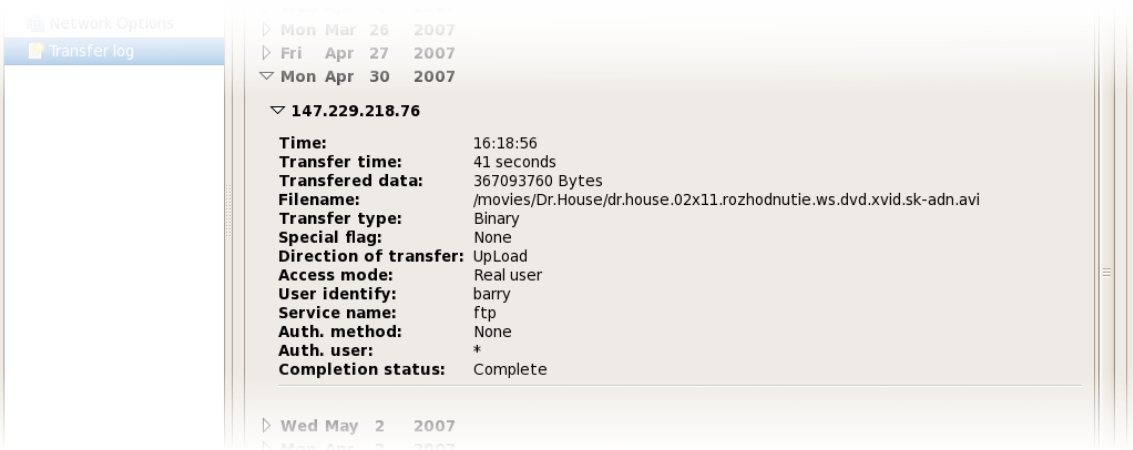
```
// roky {
   // datumy {
      // IP [
         //  zoznam riadkov
              ],..
    },..
         }
```

Pri generovaní zobrazenia takejto štruktúry, metóda ktorá je volaná z modulu Render rozparsováva pre každý dátum konkrétne riadky a tie sú zobrazené do konečnej štruktúry,

[3] viď 2.4.1

ako je vidieť na obrázku 4.1. Tento postup je nutný z dôvodu obmedzenia jazyka python, ktorý nedovolí zanoriť viac ako 5 štruktúr typu zoznam.

Výsledná štruktúra po zobrazení:



Obrázok 4.1: Zobrazenie logu v aplikácii

4.5 Modul Parser

Modul parser je programová časť aplikácie, v ktorej sa odohrávajú všetky operácie s konfiguračným súborom. Obsahuje jeden objekt `Configuration`, s metódami pre uloženie, načítanie a zmenu konfiguračného súboru (Obr. 4.2).

Na parsovanie konfiguračných súborov je napísaný do jazyka python modul pod názvom `ConfigParser`. Tento modul však vyžaduje, aby konfiguračný súbor obsahoval tzv. sekcie. Pre úplnosť uvádzam ukážku:

```
[Sekcia BC]
premenna=hodnota
```

Použitie sekcií v konfiguračnom súbore vsftpd nie je korektné voči parseru servera. Z tohto dôvodu bol parser jazyka python nepoužiteľný a bolo nutné naprogramovať vlastný. V objekte sa nachádzajú dve štruktúry typu `slovník`. Prvý z nich slúži na načítanie a úpravu konfigurácie zo súboru a druhý obsahuje všetky možnosti, ktoré majú implicitné nastavenie servera – `default`.

4.5.1 Objekt ako zoznam

Pre zjednodušenie práce s objektom, ktorý sa stará o konfiguráciu existuje možnosť preťaženia dvoch funkcií **__getitem__** a **__setitem__**. Predpokladajme, že máme objekt `object`.

Prvá z menovaných metód je vyvolaná vždy, keď sa snažíme čítať z objektu položku.

```
P = object[0]
```

Záleží na implementácii, čím vyplníme položku `P`.

```
def __getitem__( self, item ):
    return 'Hodnota položky ', item, ' je prazdna'
```

Druhá funkcia je vyvolaná, keď do objektu zapisujeme.

```
 object[0] = 'hodnota'

def __setitem__( self, item, value ):
    self.zoznam[item] = value
```

V tomto prípade do svojej premennej `zoznam` prvej položke priradí hodnotu „`hodnota`“.

Toto chovanie môže byť užitočné v prípade, že potrebujeme nejakým spôsobom ovplyvniť udalosť, pri ktorej zapisujeme alebo čítame z objektu. V našom prípade potrebujeme vedieť, kedy meníme hodnoty (pri zápise) v štruktúre. To môže sledovať premenná `change`, ktorá bude nastavená na hodnotu `True` v prípade, že sme niečo zmenili a server by mal byť reštartovaný.

```
def __setitem__( self, item, value ):

    if self.new[item] != value:
       self.change = True

       self.new[item] = value
```

V opačnom prípade nemusíme nič meniť, ani otravovať užívateľa s otázkami. Ak je však hodnota nastavená na `True`, musíme sa užívateľa spýtať, či má aplikácia server reštartovať, alebo nie.

```
if self.change: self.restartServer( )
...

def restartServer( self ):
    mbox = gtk.MessageDialog( None, gtk.DIALOG_MODAL, gtk.MESSAGE_QUESTION,
                              gtk.BUTTONS_YES_NO, 'New configuration was save.
                              \nDo you want to restart the server now ?' )
    response = mbox.run( )
...
```

4.5.2 Metóda LoadConfig

Metóda na načítanie konfiguračného súboru do programu. Metóda najskôr overí pomocou odchytenia výnimky čítania súboru, či daný súbor ide otvoriť (či sú postačujúce práva na čítanie súboru a pod.). Následne do zoznamu riadkov načíta súbor a každý neprázdny riadok, ktorý začína na iný znak, ako je znak komentára „`#`“, prejde podmienkami na zistenie o akú premennú sa jedná (pravdivostná, alebo iná), vyčlení z nej názov a hodnotu a zapíše ich do štruktúry „slovník“. V prípade, že sa vyskytne výnimka (nekorektný konfiguračný

súbor), je vypísaná správa o poškodení súboru, slovník je zničený, čím sa do súboru zapíšu len správne hodnoty.

V prípade, že sa jedná o pravdivostnú hodnotu (za znakom priradenia je buď „YES“ alebo „NO“), je prevedená hodnota na veľké písmená a až tak zapísaná. Vyhneme sa tým neustálemu volaniu funkcie na prevedenie reťazca na veľké alebo malé písmena pri porovnávaní, či „YES“ sa rovná „yes“.

Návratovou hodnotou tejto metódy je pravdivostná hodnota „True“ v prípade, že bola naplnená štruktúra objektu, odpovedá reálnej konfigurácii serveru a nevyskytla sa žiadna chyba. Hodnota „False“ je vrátená v prípade chyby.

4.5.3 Metóda SaveConfig

Metóda, ktorá slúži na zápis konfigurácie do konfiguračného súboru servera.

Pre každú premennú zo slovníka zistí, či nie je nastavená na implicitnú hodnotu servera. V prípade, že áno, v zozname (zoznam riadkov súboru) sa zakomentuje. Inak nájde v tomto zozname riadok s premennou a zmení jej hodnotu, prípadne zruší značku komentára. Ak sa premenná v zozname vôbec nenachádza, vloží ju na koniec súboru.

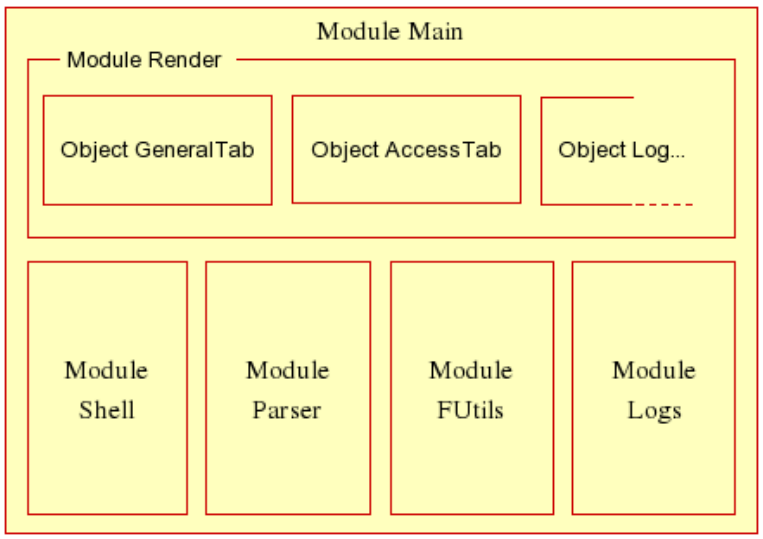
Nakoniec upravený zoznam zpíše do súboru a ak je nastavená hodnota `change`[4] na „True“, požiada o reštart serveru.

4.6 Výsledná štruktúra aplikácie

V porovnaní so štrukturálnym návrhom[5] pribudli dva moduly - Modul FUtils, ktorý zjednodušil prácu s objektami pomocou abstrakcie vytvárania týchto objektov a modul Logs, ktorý sa stará o parsovanie `xferlog`u a poskytuje nástroj k jeho príjemnejšiemu zobrazeniu užívateľovi. Modul Render sa rozčlenil na viacero objektov, každý prislúchajúc jednej položke menu. Viď obrázok 4.2

[4] Premenná udržuje stav konfigurácie. Ak je konfigurácia zmenená, premenná change sa nastaví do hodnoty True

[5] Viď obr 3.2



Obrázok 4.2: Modulárna štruktúra aplikácie po implementácii

Kapitola 5

Porovnanie s existujúcimi nástrojmi

Tento projekt vznikol ako zadanie projektu z firmy Red Hat. Projekt má slúžiť ako konfiguračný nástroj `system-config-vsftpd` a spadať do skupiny konfiguračných nástrojov `system-config`. Je to sada pre gnome väčšinou písaných v Pythone.

Momentálne sa bežne používajú

- **system-config-httpd** – konfiguračný nástroj na nastavenie webového servera
- **system-config-authentication** – konfigurácia autentifikácie - Kerberos, LDAP, Winbind, SMB, Smart Card
- **system-config-boot** – nastavuje čas výberu a výber štartovacieho systému
- **system-config-cluster** – konfigurácia clusterov
- **system-config-lvm** – konfigurácia logických diskov cez LVM
- **system-config-network** – konfigurácia siete
- **system-config-packages** – balíčkovací manažér
- **system-config-printer** – konfigurátor tlačiarní
- **system-config-services** – manažér procesov
- **system-config-securitylevel** – firewall a SELinux
- **system-config-soundcard** – konfigurácia zvukových kariet

Zo spomenutých nástrojov som vybral jeden, ktorý uvediem podrobnejšie v nasledujúcej časti.

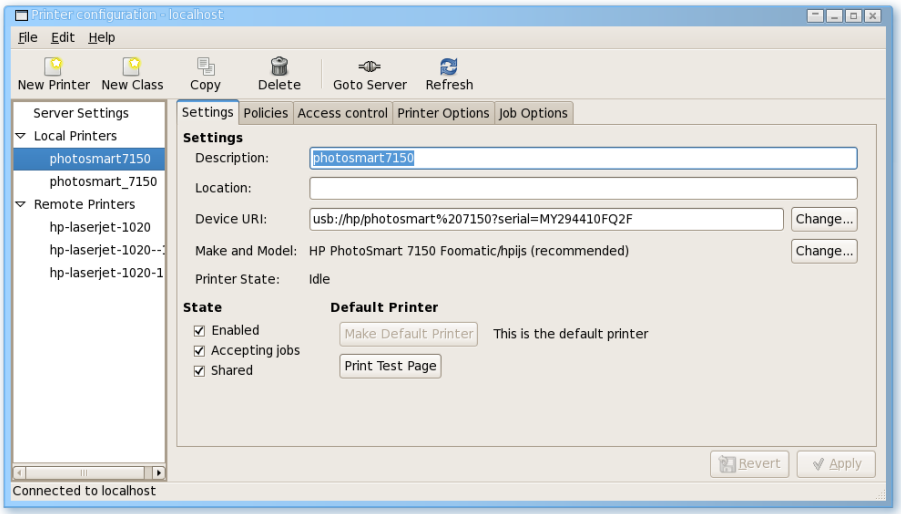
system-config-printer

Asi najvydarenejší z konfiguračných nástrojov. Slúži na administráciu lokálnych aj vzdialených tlačiarní. Tak ako by mal byť tento projekt konfiguračným programom pre vsftpd, tak system-config-printer je konfiguračným programom pre `CUPS`[1].

System-config-printer je písaný v jazyku Python s využitím Gnome/GTK+.

Grafická stránka programu je jednoduchá, ale prehľadná s ľavým `stromovým boxom`[2], v ktorom sa nachádzajú nájdené tlačiarne a pravou stranou, kde sú konfiguračné možnosti. Nástrojová lišta slúži na narábanie s objektami tlačiarní - v prípade projektu system-config-vsftpd nie je táto lišta opodstatnená. Čo sa týka generovania widgetov je väčšina grafickej štruktúry robená v nástroji Glade. U system-config-vsftpd sa tieto objekty generujú za behu aplikácie.[3]

Z pohľadu implementácie má tento nástroj taktiež naimplementovaný vlastný parser konfiguračného súboru, ale príliš zakomponovaný do ostatného kódu, aby som z neho mohol čerpať.



Obrázok 5.1: System-config-printer

[1] CUPS - Common UNIX Printing System, štandardný unixový „tlačiarenský“ systém
[2] Myslený objekt TreeView v jazyku Python
[3] Viď 4.2

Kapitola 6

Testovanie

Testovanie je jedna z veľmi dôležitých častí vývoja softwaru. Jednou z možností bolo urobiť testovacie skripty, ktoré odhalia určité percento chýb, ktoré sú pri bežnom odlaďovaní programátorovi z nejakých príčin ukryté. Pre projekt, ktorý bude nasadený do bežného užívania je to jedna z najlepších metód. Aj napriek tomu som sa rozhodol nepodrobiť program priamo týmto testom, ale urobiť testy nepriamym zavedením programu do praxe.

Firma Red Hat mi poskytla výbornú základňu nie len na ladenie chybových stavov, ktoré sa vo vývojovom centre Red Hatu v Brne deje každodenne, ale poskytli mi priamu väzbu užívateľ (administrátor) – program a tým radu pripomienok a návrhov na vylepšenie. Taktiež som oslovil laických užívateľov a zaradil ich pripomienky do zoznamu. V priemere mi každým dňom behu aplikácie pribudli do zoznamu približne 4 body, ktoré som priebežne dorábal, alebo opravoval. Niektoré požiadavky na zlepšenie som musel kvôli spojitosti projektu s bakalárskou prácou odložiť na neskôr a to hlavne pre ich rozsiahlosť a časovú náročnosť.

Uvádzam z nich aspoň pár pre predstavu, čo užívatelia požadujú:

- Vylepšiť zobrazenie logu prenosov[1] napríklad tabuľkou
- Pridať možnosť konfigurácie pre každého užívateľa zvlášť
- Zaradiť konfiguráciu „TCP wrappers“ do aplikácie

Aplikácia prešla testovacím obdobím, v ktorom by sa mali nájsť všetky závažnejšie problémy a po ich odladení bude aplikácia poskytnutá komunite a následne širšej verejnosti.

[1] Viď časť 4.4

Kapitola 7

Záver

Tento projekt bol zameraný na implementáciu nástroja, ktorý by zjednodušil prácu mnohým užívateľom a administrátorom. Osobne som na podobnom projekte pred tým nikdy nepracoval a vývoj mi priniesol veľa nových znalostí z oblasti programovania v jazyku python a GTK a hlavne z oblasti tvorenia kvalitného návrhu na základe daných požiadavkov. V tejto záverečnej časti sa pokúsim zhrnúť všetky etapy vývoja a ich hlavné úskalia.

Na samom začiatku práce bola najťažšia predstava, ako bude výsledná aplikácia vyzerať - ako docieliť, aby bol program pútavý, jednoduchý a hlavne pre užívateľa zrozumiteľný. Na tieto otázky sa nedalo odpovedať bez spätného vyjadrenia užívateľov, ktorí budú program používať a bez znalosti užívateľského myslenia. Preto samotnej implementácii i návrhu predchádzala dlhá štúdia nástrojov z praxe, ktoré majú podobné zameranie, ako navrhovaný projekt. Taktiež som sa spočiatku sústredil na štúdium princípov tvorby a návrhu grafických užívateľských rozhraní.

Na konci tejto etapy vývoja bol kvalitný návrh, ktorého základ tvoril štrukturálny a funkčný model presne popisujúci dané požiadavky. Návrh spĺňal všetky požiadavky a projekt bol na najlepšej ceste k začiatku implementácie. V časti implementácie sa však ukázali rôzne problémy vyplývajúcej hlavne z nejednotnosti konfiguračných premenných, čo sa ukázalo až na konci, takže návrh musel byť mierne upravený. Ku zmenám v návrhu na konci implementácie prispeli možnosti zlepšiť a sprehľadniť kód aplikácie rozdelením objektov a pridaním rôznych modulov. Väčšie sústredenie na návrh projektu z implementačného hľadiska mohol zamedziť podobným chybám.

Projekt je dokončený a zaregistrovaný na www.sourceforge.net[1] ako projekt pod licenciou GPL. Následne bude aplikácia vydaná do linuxovej distribúcie `Fedora` firmy Red Hat. Na projekte plánujem naďalej pracovať, vylepšovať a hlavne vyvýjať, čo je pre dosiahnutie spokojnosti užívateľa a udržanie zákazníka najdôležitejšie.

Projekt by mohol inšpirovať ďalších programátorov k vytvoreniu podobných grafických konfigurátorov, ktoré by spríjemnili prácu menej skúseným užívateľom a možno tak prispeli k rozrasteniu hlavne laickej základne open-source riešení.

[1] http://vsftpd-config.sourceforge.net

Úplne na záver uvádzam citácie ľudí, ktorí prišli do styku s projektom pred jeho finálnym ukončením.

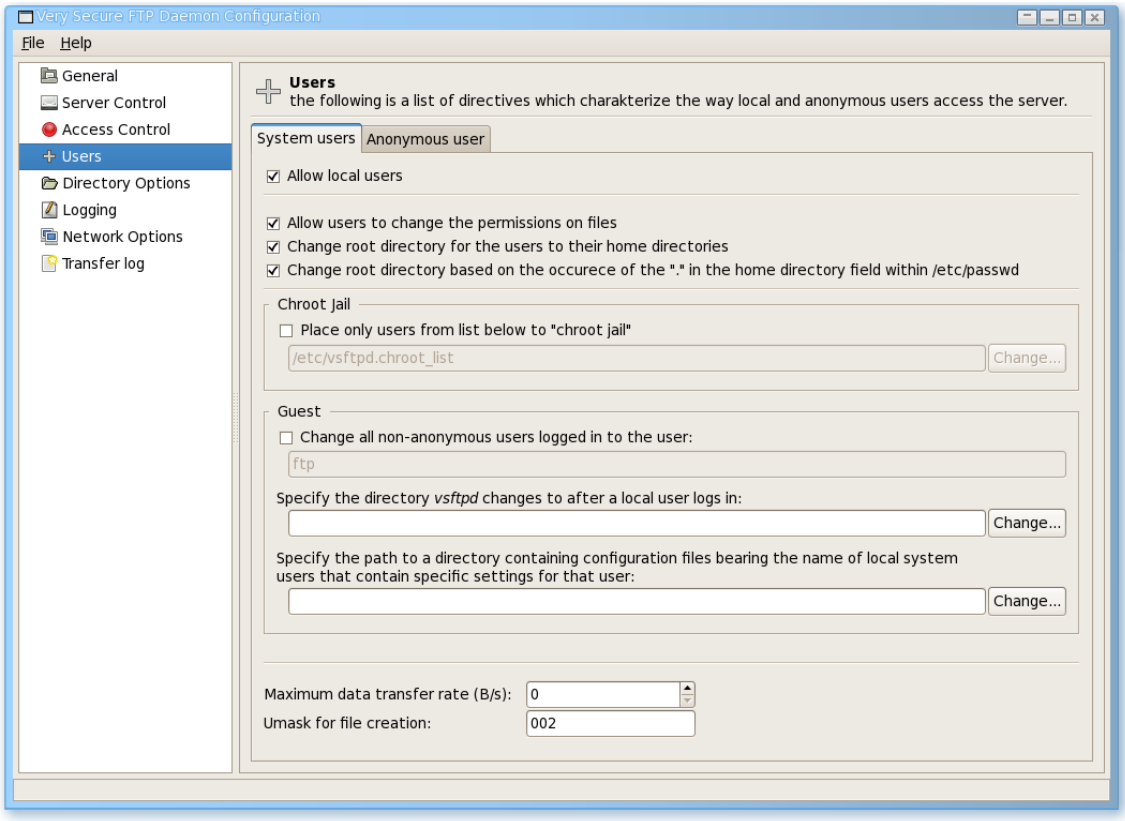
„Velice pěkné. Jediné, co mi vadí, je, že to není součástí integrovaného nástroje pro celou fedoru, což ale není tvá chyba. Jak by byl svět krásný, kdyby vše bylo jako system-config-vsftpd. Obzvlášť prohlížeč logu se mi zdál naprosto cool.“

Michal Marciniszyn, Quality assurance engineer, Red Hat, Inc.

„Už jsem ani nedoufal, že nějaký konfigurační nástroj pro vsftpd vznikne a pokud ano, tak jen s velmi omezenými možnostmi. Avšak již beta verze system-config-vsftpd mě překvapila svou propracovaností a intuitivním ovládáním. Pak přibyla i užitečná a elegantní nápověda. Jedná se určitě o nejlepší system-config nástroj“

Radek Bíba, Quality assurance engineer, Red Hat, Inc.

Výsledný program



Obrázok 7.1: Výsledná aplikácia

Literatúra

[1] Daryl Harms, Kenneth McDonald. Začínáme programovat v jazyce Python. Computer press, 2006. ISBN 80-7226-799-X.

[2] Mark Lutz, David Ascher. Pohotová příručka, Naučte se Python.

[3] John Finaly. PyGTK 2.0 Reference manual. `http://www.pygtk.org/docs/pygtk/`

[4] John Finaly. PyGTK 2.0 Tutorial.
`http://www.pygtk.org/pygtk2tutorial/index.html`

[5] Manual page for vsftpd.conf. `http://vsftpd.beasts.org/vsftpd_conf.html`.

[6] Red Hat Enterprise Linux 4. Reference Guide
`http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/ref-guide/s1-ftp-vsftpd-conf.html`

[7] Benson Calum, Elman Adam, Nickell Seth. GNOME Human Interface Guidelines.
`http://developer.gnome.org/projects/gup/hig/`

[8] WWW stránka GTK `http://www.gtk.org`