VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ANALÝZA SYSTÉMOVÝCH ZÁZNAMŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. JAN ŠČOTKA
AUTHOR

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ANALÝZA SYSTÉMOVÝCH ZÁZNAMŮ

SYSTEM LOG ANALYSIS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. JAN ŠČOTKA

AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEŠ SMRČKA

SUPERVISOR

BRNO 2008

Abstrakt

Práce zkoumá IDS systémy. Pokusí se nalézt jejich omezení a možnosti rozšíření. Zabývá se obecnější analýzou záznamových souborů než jen v zaměření na IDS systémy. Zkoumá možnost zápisu regulárních výrazů jednoduchým způsobem, tak aby i uživatel bez jejich znalosti byl schopen pracovat v takovém systému.

Klíčová slova

Systémové záznamy, analýza záznamů, uživatelsky přívětivé regulární výrazy, Unix, Python, IDS, HIDS, statistiky přístupů

Abstract

The goal of this master thesis is to make possible to perform system log analysis in more general way than well-known host-based instrusion detection systems (HIDS). The way how to achieve the goal is via proposed user-friendly regular expressions. This thesis deals with making regular expressions possible to use in the field of log analysis, and mainly by users unfamiliar with more detail aspects of computer science.

Keywords

Log, system log analysis, user-friendly regular expression, Unix, Python, IDS, HIDS, access statistic

Citace

Jan Ščotka: Analýza systémových záznamů, diplomová práce, Brno, FIT VUT v Brně, 2008

Analýza systémových záznamů

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením pana Ing. Aleše Smrčky. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Ščotka
16. května 2008

Poděkování

Děkuji vedoucímu mé diplomové práce za ochotu pomáhat mi s řešením nejrůznějších problémů s touto diplomovou prací spojených a ochotu vést mou práci. Dále děkuji oddanosti mé přítelkyně za psychickou podporu z její strany. Dále děkuji systému LaTeX za vzornou sázecí práci, bez kterého by tato práce nemohla vzniknout.

Obsah

Kapitola 1

Úvod

Bezpečnost a monitorování systémů je jedním ze základních kamenů dnešních *informačních systémů*. Tato práce se bude zabývat problematikou *analýzy systémových záznamů* a pokusí se navrhnout rozšíření stávajících systémů o obecnější funkce, zaměřující se na analýzu záznamů a vyhledávání informací v datech. Základním kamenem je studium stávajících systémů a hledání možností, které systémům chybí, a navrhnout řešení daného problému.
V dnešní době informačních dálnic a bohatých možností předávání informací je velmi důležité mít systémy na úrovni systémů *detekce průniků*. Chyby v softwarových produktech se budou nacházet ještě hodně dlouho. Proto jsou hodně důležité systémy detekce průniku IDS (angl. Intrusion Detection System), které mají za úkol monitorovat neobvyklé aktivity, či činnost, která může být považována za možný útok. Takovým chováním může být např. několikanásobné přihlašování uživatelů, kdy uživatel, případně útočník, opakovaně zadává heslo a je systémem odmítnut kvůli zadání špatného hesla. Takové chování může být považováno za podezřelé. Typů útoků existují celé řady a vytváří se tím vzory chování, které se odlišují od běžné uživatelské činnosti.
Další oblastí kterou zkoumáme, je obecná analýza záznamů a jejich možnosti. Každý, kdo je připojen do sítě, může mít spuštěn vlastní server, na kterém bude poskytovat jisté služby. Tyto služby obvykle mívají možnost záznamu činnosti serveru. Ať již pozitivní, či negativní odpovědi. Úkolem je navrhnout systém, který bude řešit danou problematiku a bude umět získávat informace v daných datech. Jako příklad můžeme uvést *FTP* a jeho záznamový soubor. Může nás zajímat, kolik daný uživatel identifikovatelný v systému jménem přenesl dat za jisté období.
Oblast, kterou využijeme pro analýzy systémových záznamů a záznamů služeb, se opírá o regulární výrazy. Ovšem regulární výrazy jsou pro uživatele bez informatických znalostí formálních jazyků a automatů jen stěží pochopitelné. Proto se pokusíme navrhnout nějaké řešení tohoto problému a přiblížení regulárních výrazů běžně zapisovanému výrazu. Vytvoříme uživatelsky přívětivou reprezentaci a tvorbu regulárních výrazů tak, aby člověk neznalý formálních jazyků dovedl takový výraz zkonstruovat a upravovat. Tím by se značně umožnilo zkoumat záznamy a mít možnost podle daných výsledků možnost zareagovat. Nebo-li budou se muset v záznamech nalézt sémantické významové závislosti, tak jak by je pochopil uživatel. Tím se regulární výrazy zjednoduší na přiřazení jmen slovům, či různým strukturám jako je např. datum.
Po implementaci aplikace, která bude vytvořena v jazyce Python, provedeme otestování a navrhneme možná rozšíření budoucích verzí. Aplikaci vytvoříme grafické uživatelské rozhraní s pomocí knihovny GTK.
Diplomová práce navazuje na výsledky ze semestrálního projektu řešeného v bezprostředně

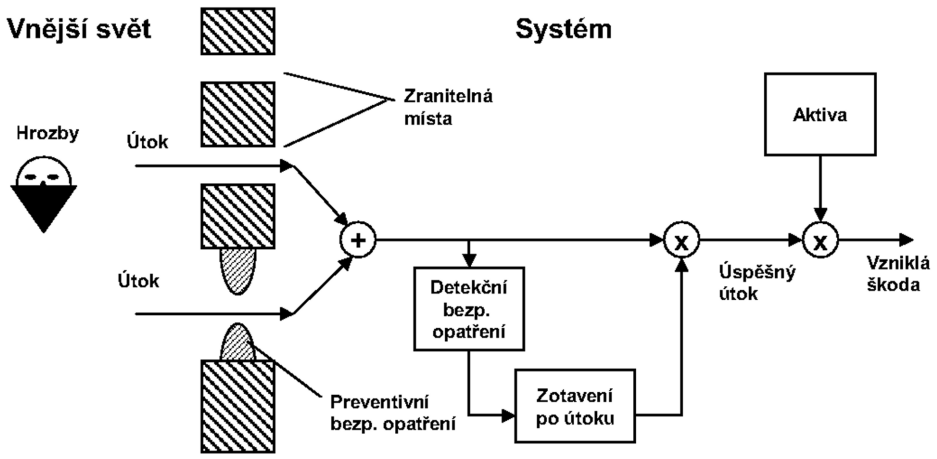
předcházejícím semestru. Obsahem semestrálního projektu bylo prozkoumat obecnou problematiky analýzy systémových záznamů a prozkoumat existující nástroje a navrhnout rozšíření nástrojů v podobě návrhu rozšiřující aplikace zahrnující obecnější analýzu. Diplomový projekt vychází z navrženého modelu aplikace ze semestrálního projektu a implementuje navrženou aplikaci. Všechny kapitoly a sekce, včetně návrhu, jsou převzaty ze semestrálního projektu a je jich využito jako výchozího materiálu pro implementaci modelu.

Kapitola 2

Problematika současného stavu

2.1 IDS – Systémy detekce narušení

Obecně detekují nechtěnou manipulaci s počítačovým systémem. Většinou pocházející z internetu. Taková manipulace je vedena obvykle ze strany útočníka. [2]



Obrázek 2.1: Obrázek ukazuje, jak vypadají hrozby a jejich zpracování a možnosti ochrany. Obrázek převzat ze skript předmětu Bezpečnost informačních systémů. [10]

Systémy mohou reagovat na mnoho typů chování, které je v rozporu se správným chováním. Toto chování mohou vyvolat různé viry, trojské koně, přímé útoky, botnety[1]. Mezi možné typy útoků patří:

- Útok ze sítě na nezabezpečenou službu, případně službu obsahující chybu v protokolu

[1] Sítě využívané pro distribuované činnosti, útoky, rozesílaní spamu, které jsou vytvořeny pomocí jiných technik z počítačů obyčejných uživatelů, které může útočník ovládat najednou. Tím je využit např. k DoS útoku (Denail of Service – odepření služby)

- Pokus o neoprávněné získání přístupu k systému, či získání přístupu k citlivým datům.
- Útok na aplikační logiku
- Získání vyšších práv v systému.

Základními typy IDS systémů jsou:

Network Intrusion Detection System – síťový systém detekce. Takovýto systém funguje na principu analýzy síťového provozu a dokáže v tomto provozu hledat vzorce chování, které signalizují nechtěné vzory. Systémy umožňují prohledávat síť jako celek, nejsou omezeny na jeden počítač. Můžeme nalézt útoky, které směřují i na jiný prvek v síti a samotný systém nijak neohrozí.

Host-based Intrusion Detection System – Detekční systém, který je určen přímo pro jeden daný objekt a dokáže monitorovat interní informace a je schopen detekovat všechny dynamické aspekty, ne jen analýzu síťového provozu. Takovouto informací může být např. detekce změn souborového systému, či analýza záznamových souborů, či může být monitorován stav běžících služeb a hlídat, zda-li běží.

Důležitou vlastností sytému je, jak se zareaguje na příchozí *incident*. Existují dva typy systémů. První z nich je pasivní, jenž je chopen zaznamenávat činnost, posílat výstrahy, ovšem sám o sobě nereaguje nijak na příchozí událost tím, že by se ji snažil eliminovat. Naproti tomu na druhé straně existují reaktivní, jenž na danou výstrahu (upozornění systému) jsou schopny automaticky reagovat, a to tak, že vytvoří firewallové pravidlo, či vypnou poskytování služby, či v případě napadení konkrétního účtu zablokují účet daného uživatele.

2.2 Systémy detekce narušení na lokálním počítači

Jsou systémy, které pracují na odlišném principu, než síťové detekční systémy NIDS, které pracují na externím rozhraní systému. Systémy HIDS pracují na vnitřním rozhraní systému a řeší např. problémy typu, zda-li program přistupuje k souborům, které má k dispozici, tím dokáže zachytit i útok, který zatím nebyl znám. Např. pokud textový procesor zpracovává soubor `/etc/passwd` můžeme se domnívat, že je něco špatně. [1] Těchto principů je využito i v bezpečnostním mechanizmu pro operační systém Linux *SElinux*.
Teoreticky jde o to, že útočník, který chce napadnout systém, chce využít systém i v budoucnu, a proto se bude snažit upravit si systém ke své potřebě, aby měl cestu, jak se do systému opětovně dostat. Těmito prostředky jsou tzv. zadní vrátka, která zná pouze útočník. Ovšem, aby tyto zadní vrátka do systému aplikoval, musí obvykle pozměnit soubory, což sebou nese zanechání stop. V případě běžícího systému HIDS je možné tyto stopy nalézt právě monitorováním změn souborového systému.

Tyto systémy pracují obvykle na principu objektů, které jsou uchovány v informační databázi (zaneseny údaje o jejich parametrech). Taková databáze může obsahovat:

- Velikosti souborů
- Časy přístupu do souboru, jeho změn a vytváření.
- Dále obsahuje různé kontrolní součty, mezi hlavní patří MD5, či SHA1, které zaručují, že soubor nebyl změněn oproti této databázi. Jelikož MD5 kontrolní součty už nejsou

úplně bezpečné a dají se nalézt kolize [13], používá se více těchto hash algoritmů najednou, aby se zamezilo podobným problémům v případě rozlomení jednoho z těchto algoritmů.

Ovšem ne všechny změny jsou takto lehce zaznamenatelné pomocí kontrolních součtů. Jako příklad uveďme pokusy o uhodnutí vstupního hesla. Pro tyto případy jsou využity techniky jako monitorování systémových záznamů a sledování běžících procesů.
Důležitou zbraní útočníků, kteří se pokusí napadnout systém je, že se pokusí ochranu po vniknutí do napadeného systému vypnout tak, aby nezanechali stopy. Technikou postupuje mnoho virů a červů, které se po vniknutí do systému pokusí vypnout antivirovou kontrolu, a další kontrolní mechanizmy. Systémy IDS musí s touto možností počítat, a proto by měly mít možnosti jak samy sebe chránit. Jednou z možností je uložit tyto kontrolní objekty mimo dosah samotného počítače, třeba na média pouze pro čtení (jako jsou CD, či DVD disky), či jednosměrné přenosové kanály tak, aby se nedaly vymazat zásahy do systému. Další možností jsou souborové systémy stavěné jako *append only*, to znamená, že data nemohou být vymazána. V ideálním případě, by bylo vhodné propojit systémy HIDS se systémy NIDS, které by podle změn interních struktur byly schopny reagovat na systémy NIDS a podle potřeby filtrovat síťový provoz.

Systémy IDS jsou důležitou součástí ochrany systémů. V operačních systémech typu Microsoft Windows tuto funkci obstarávají různé druhy *antivirových programů* a firewallů, které mívají zabudovány kontroly integrity souborového systému, či kontrolují vzájemné spouštění programů v sytému.

V unixovém světě je situace poněkud odlišnější. Nástrojů na tyto činnosti existuje celá řada. Unixový svět má svá specifika a mnoho nástrojů se drží filozofie dělat dobře jednu věc. Existují proto nástroje na kontrolu souborového systému a jeho integrity jako jsou nástroje typu Tripwire[2] či OSSEC[3]. Pak jsou tu nástroje na kontrolu a hledání informací v souborech systémových služeb jako např. Logcheck[4] či OSSEC. Další kategorií jsou nástroje na sledování statistik služeb jako je třeba Awstat[5], který je schopen generovat grafy ze souborů záznamů přístupu Apache web serveru.

2.2.1 Tripwire

Důležitou součástí systému je kontrola integrity souboru. Jedním z prověřených nástrojů provádějící tuto činnost je Tripwire. Tento nástroj byl jedním z prvních v Linuxovém světě, který takové možnosti poskytoval[6]. Společnost Tripwire dále poskytuje, ovšem již v komerční verzi nástroje pro GUI administraci, či přístup přes webové rozhraní.

Tento nástroj nejprve sestaví databázi souborů a vytvoří soubor, který tuto databázi obsahuje. S touto databází je možné kdykoliv srovnávat aktuální stav a kontrolovat změny

[2] http://www.tripwire.com/
[3] http://www.ossec.net/
[4] http://sourceforge.net/projects/logcheck/
[5] http://awstats.sourceforge.net/
[6] Samozřejmě pomíjíme možnost si takovýto nástroj napsat samostatně udělat kontrolní součty souborů v systému a uložení na externí médium, a následné srovnání těchto dat. Takto vytvořit databázi můžeme např. příkazem `find` s provedením příkazu exec s nějakým Výstupem hashe souboru, tento výstup pak uložit do souboru, pro možnost budoucího porovnání. Takový to příkaz by mohl vypadat obdobně jako následující: `find / -type f -exec sha1sum {} \;`. Kontrola může být prováděna obdobně např. vytažením jmen souborů z tohoto výstupu a vygenerování součtu a jejich porovnání.

o proti tomuto stavu. Je možné kontrolovat změny přístupových práv, změny velikosti, integritu a nalézt smazané objekty. Administrátorovi je pak umožněno provádět obnovení databáze s daty o souborech. Může provádět jejich update, aby tím aktualizoval změny v systému. Systém Tripwire u jednotlivých souborů sleduje následující atributy:

- Přístupová práva
- Číslo i-uzlu
- Počet odkazů na soubor
- UID (ID uživatele - vlastníka souboru)
- GID (ID skupiny - vlastníka souboru)
- Typ souboru
- Velikost souboru
- Velikost souboru pouze roste (záznamové soubory apod.)
- Číslo blokového zařízení, na kterém se soubor nalézá
- Čísla zařízení u speciálních souborů
- Počet alokovaných datových bloků souboru
- Čas modifikace souboru
- Čas modifikace i-uzlu
- Čas přístupu

a poskytuje 4 typy kontrolních součtů v závislosti na požadované rychlosti výstupů. Mohou být aplikovány různé kombinace těchto součtů a také sdružovaní souborů do jednoho objektu a určení, na kterou emailovou adresu se má posílat upozornění s případnou změnou.

- CRC-32 (nízká bezpečnost, rychlý)
- MD5 (vyšší bezpečnost)[7]
- SHA (vyšší bezpečnost)
- HAVAL (128 bitový klíč, nejvyšší bezpečnost)

Inicializace tohoto systému a podobných je jednoduchá. Jediným potřebným krokem inicializace je základní vytvoření databáze informací o souborech [8]. Pouhé instalování Tripwire (nebo jakéhokoliv jiného systému kontroly integrity), bez pečlivé konfigurace a dodržování rozumných zásad úroveň zabezpečení systému nezvýší, pouze povede k falešnému „pocitu bezpečí“. Jaké zásady by tedy měl administrátor při nasazení takového systému pro kontrolu integrity dodržovat? Minimálně alespoň databázové soubory je potřeba uchovávat odděleně na důvěryhodném médiu, nejlépe digitálně podepsané. Optimální je mít na odděleném médiu i spustitelné soubory Tripwire (jako doplněk k „záchranné“ sadě startovacích médií) a kontrolu provádět po nastartování systému z důvěryhodného média [8].

[7] Dnes se již jednocestná funkce MD5 nepovažuje za bezpečnou. [13]

Tripwire je nástrojem pro kontrolu integrity souborů, a pokud je dobře nakonfigurován, výrazně může pomoci nejen při detekci neautorizovaných zásahů do systému, ale i při odstraňování následků bezpečnostního incidentu.
Nástroj obsahuje funkčnost HIDS systémů na úrovni detekce průniků změnou stavu souborového systému. Neposkytuje vlastnosti potřebné pro analýzu souborů se záznamy.

2.2.2 Logcheck

Pravidelná kontrola záznamových souborů by měla být nedílnou součástí údržby každého systému, i když v praxi tomu tak bohužel mnohdy nebývá. Navíc systémové záznamy bývají rozsáhlé a nepřehledné, takže prohlížení takových záznamů řádek po řádku pochopitelně není reálné. V unixových systémech máme k dispozici spoustu nástrojů určených pro práci s textem, které pracují s regulárními výrazy a pro zkušenějšího administrátora není problém vytvořit krátký skript pro zpracování záznamů spouštěný třeba přes démona cron. Na místo psaní vlastních skriptů můžeme použít nástroj Logcheck [7].

Program Logcheck slouží k analýze systémových záznamů. Tím způsobem, že odfiltrovává uživatele (správce) od zbytečných hlášení systému a posílá jen zprávy, které mají *potenciálně důležitý charakter*. Program se postará i o problém rotování souborů, protože si pamatuje inode uzly souborů. Další důležitou vlastností je, že po ukončení analýzy si uloží pozici v souboru, aby při příští analýze nemusel jít opět od začátku a tím odesílat zvolenému uživateli již odeslané zprávy.

Logcheck funguje na principu filtrování zpráv podle klíčových slov a je možno použít i regulární výrazy. Tato činnost se pak dá naplánovat do nějaké cyklické úlohy, např. přiřadit do plánovače *cron*. Události se dělí do 3 typů hlášení, které jsou odděleny v různých adresářích. Tyto 3 typy patří do jednoho ze dvou typů seznamů klíčových slov. Druhý seznam obsahuje slova, která jsou běžná, a proto budou ve výsledku ignorována a nebudou odesílána uživateli.

Aktivní útok – cracking útok, přímo cílený útok, který ohrožuje systém na bezpečnosti.

Porušení bezpečnosti – toto může být útok způsobený neúmyslně, nemusí se jednat o útok na systém.

Neočekávané události – jsou události, které nebyly zařazeny do vyšší kategorie, ale zároveň nejsou filtrovány jako nepodstatné pomocí adresáře se slovy pro ignorování hlášení.

Důležité pro tento systém je, aby byly vhodně zvoleny filtry, a tím se omezilo množství posílaných zpráv jen na únosnou míru a to takových, které jsou pro systém potenciálním rizikem.

Systém pro kontrolu systémových záznamů Logcheck je systém velmi jednoduchý pracující na základě filtrování řádků pomocí klíčových slov a regulárních výrazů. Při dobrém nastavení je schopen velmi dobře informovat uživatele o možných rizicích a napadeních serveru. Tento systém je ale omezen pouze na analýzu každého samostatného řádku a pracuje pouze se vzory na jednom řádku, tudíž nezachytí útoky které jsou rozpoznatelné až při analýze více řádků. Také musí být uživatel seznámen s prací regulárních výrazů ve své obecné podobě.

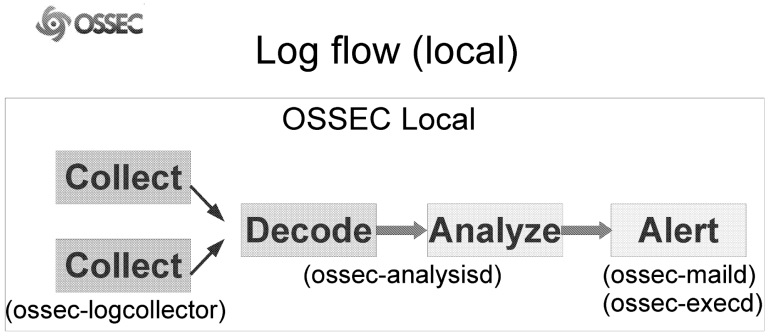
2.2.3 OSSEC

OSSEC je systém, který kombinuje funkce předchozích programů a přidává nové možnosti analýzy souborů systému a služeb. Tento systém je již při základní instalaci přednastavený na základní typy útoků, které jsou zaneseny v konfiguračních souborech. Tento systém nemá jen pasivní funkci sledování systému, ale umožňuje i aktivní odpovědi na události. Systém obsahuje velmi propracovanou podporu záznamů, jsou podporovány záznamy napříč operačními systémy a rovněž spousty služeb. Systém navíc nabízí kontrolu integrity systému. Další vlastností je, že umožňuje běh v několika režimech: lokálním, či klient server módu.

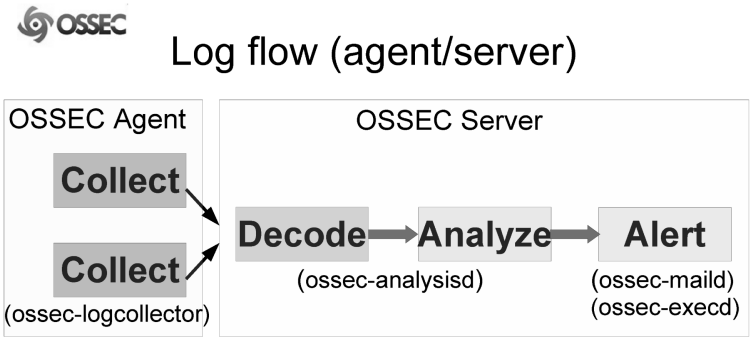
Základním konceptem systému OSSEC spočívá v analýze souborů se záznamy a ukládání upozornění na události. Nepamatuje si celé soubory se záznamy. Celý systém je velmi flexibilní, a díky tomu, že je postaven na základech HIDS systémů, je nezávislý na šifrovaných protokolech jako jsou SSHD či SSL [6]. Rovněž díky flexibilní konfiguraci jde použít na téměř libovolnou aplikaci, která používá nějaký výstup do souborů se záznamy o činnosti. Možnosti které OSSEC nabízí:

- Analýza záznamů
- Kontrola integrity souborů
- Kontrola registrů (týká se klienta pro operační systém Microsoft Windows)
- Host-based detekci anomálií (*rootkit* v Unixovém prostředí)
- Aktivní reakci na podněty

Důležité je, že systém reaguje real-time, a proto je schopen následně i generovat odpověď, což v předchozím systému nebylo. Ten umožňoval pouze analýzu záznamů. Dále velkou předností oproti předchozímu systémů je, že již v implicitní konfiguraci obsahuje stovky pravidel a reakcí na podněty. Což ovšem může být i problémem v případě systému, který má neobvyklé požadavky na zátěž, a tento systém může při této konfiguraci např. vytvářet Odmítnutí služby, protože ji po předchozí analýze pravidel vypne.
Systém jak již bylo zmíněno, pracuje na dvou principech. Buď v lokálním módu, kdy vše běží jen na jednom počítači a nepočítá se s jakoukoliv komunikací, a analýzy dat z více serverů, či může běžet v *klient-server* módu, kdy klient posílá na server záznamy činnosti ze stanice. Následně sever je agreguje a spravuje (viz. obrázek 2.2) a obrázek 2.3).



Obrázek 2.2: Ukázka činnosti analýzy OSSEC v lokálním režimu (převzato z [6]).



Obrázek 2.3: Ukázka činnosti analýzy OSSEC v režimu klient/server (převzato z [6]).

Základem celého systému jsou pak *dekodéry a pravidla*. Dekodéry jsou založeny na regulárních výrazech. Prvním krokem, co systém dělá je tak zvaný predecoding, kdy se snaží rozpoznat známé záznamy v souborech. Takže je schopen rozpoznat počáteční řetězce v souborech se záznamy, protože bývají v podobném formátu začínající např. datem, takže je schopen určit datum, stroj, službu a podobně (Tento záznam musí být ve známém formátu). Následuje činnost samotného dekódování, kdy se do systému zanesou pravidla, jak dekódovat tu část, která následuje po první fázi a pomocí regulárních výrazů se ze záznamů vybírají a pojmenovávají části. Dekodéry se řadí do stromové struktury [6]. Takže výsledné dekódování probíhá podle struktury služby a pro případ, že neuspěje již dekodér ve vyšší vrstvě nepokračuje se dále. Ilustrační příklad dekodéru pro telnet, převzato přímo z originálních dekodérů dodávaných k programu:

```
<decoder name="telnetd">
  <program_name>^telnetd|^in.telnetd</program_name>
</decoder>

<decoder name="telnetd-ip">
  <parent>telnetd</parent>
  <regex>from (\d+.\d+.\d+.\d+)\$</regex>
  <order>srcip</order>
</decoder>
```

Takto vytvořené dekodéry pak generují události, které jsou využity pravidly, na které se pak mohou navazovat události. Existují 2 typy pravidel:

Atomické – postavené na jediné události. Ty jsou obdobou funkcionality programy Logcheck, který fungoval také na úrovni jednotlivých událostí.

Složené – Události dovolují zachytit složení více událostí, dovolují zachytit časové závislosti událostí. Ty nám dovolují vyjádřit např. to, že po 3 neúspěšných pokusech o přihlášení v rozmezí 60 sekund zablokuje účet.

Tímto se dostáváme k poslední části nástroje OSSEC, a to jsou posílání upozornění, či aktivní odpovědi. Pravidla mohou obsahovat odpověď na to co dělat v případě, že pravidlo projde.

Systém OSSEC je velmi propracovaný systém s velmi bohatou funkčností. Sdružuje funkcionalitu 2 předchozích systémů a rozšiřuje jejich možnosti. Psaní vlastních dekodérů a pravidel není pro informatika obtížné. Neumožňuje ovšem toto psaní pro uživatele, který neumí regulární výrazy a není schopen se vyznat ve stromové struktuře pravidel. Tato stromová struktura je výhodná pro zpracovávání počítačem, ale složitější na pochopení závislostí. Systém, který navrhneme by měl sdružit funkčnost tohoto nástroje na úrovni analýzy souborů, a zároveň zjednodušit psaní regulárních výrazů, nabídnut lidsky čitelnou formu regulárních výrazů, a umožnit další funkce spojené se získáváním informací ze souborů se záznamy.

2.3 Analýza záznamů systému

Prostudováním nástrojů v předchozí sekci a zjišťováním jejich funkčnosti, vyplynuly jisté závěry o možnostech analýzy systémových záznamů. Analýza systémových záznamů je samotným základem IDS systémů. Problematika této analýzy je důležitá pro definici samotné funkčnosti IDS aplikace, či podobných. Zjistily jsme techniky, které nástroje používají. Pokusíme se ji rozšířit prostudováním obecnějších principů.

2.3.1 Časové závislosti

Základním požadavkem na analýzu je podívat se jakým způsobem potřebujeme reagovat na události objevené v systému. Máme následující 2 možnosti analýzy:

Intervalově založená analýza – (Dávkové zpracování) analýza mezi jednotlivými body není nikterak provázána a události jsou generovány dopředným uložením a následným spuštěním analyzátoru. Většina IDS prvních systémů tímto způsobem fungovala. Tato analýza se hodí hlavně pro systémový audit či pro statistické vyhodnocování, což jsou záznamy které potřebují znalost velkého množství informací najednou. Od systémů se nepožaduje okamžitá reakce [5].

Real-time analýza – tento typ provádí kontinuální analýzu. Tyto systémy jsou pak vhodné pro aktivní odpovědi na události vyskytnutí v systému. Je zvláště vhodné pro systémy NIDS, které kontinuálně zpracovávají pakety, generují události při jejich průchodu a tím se může real-time generovat odpověď [5].

Příkladem systému prvního typu je systém Tripwire nebo OSSEC se svou částí kontroly integrity souborového systému. Výhodou je nižší náročnost na dávkové zpracování. Z toho plyne malé zatížení systému a možnost odhalit útok na stejný cíl, což vyžaduje delší časové informace. Nevýhodou jsou velké objemy dat, které je potřeba uložit a není možná bezprostřední interakce.

2.3.2 Analýza problémů

Na analýzu můžeme pohlížet ze dvou pohledů. Prvním je, že víme co je špatně, či-li jaké vzorky hledáme a ty nás zajímají a provádíme analýzu podle vzorů. Druhou skupinou jsou analýzy, kde nevíme, co je v záznamu špatně, ale víme, jaké je správné chování a snažíme se nalézt anomality od „dobrého“ chování.

Systémy prvního typu – systémy detekce zneužití. Umožňují nám analyzovat soubory na základě detekce vzorů v souborech: může se jednat o dříve zmíněné regulární výrazy. Hledá se vzorek, či skupina vzorků v daném souboru a v případě nalezení se generuje událost. Tyto systémy jsou majoritně zastoupeny. Jejich velkou výhodou je právě to, že reagují na známé útoky, tudíž nepotřebují vysokou *inteligenci* při rozhodování, zda se jedná o útok.

Analýzy druhého typu, založené na hledání anomálií v záznamech. Tyto systémy jsou problematičtější. Vyžadují znalost legitimního chování, respektive legitimních záznamů v souboru. Tyto záznamy se mohou získat z historie, či se adaptovat různými formami učení, či užitím expertních systémů. Velikou výhodou takto prováděné analýzy je možnost nalézt problémy, které ještě nebyly objeveny, což může být přínosné, protože tím získáváme nové znalosti. Systémy rovněž mohou generovat velké množství falešných událostí, protože se např. událost v systému vyskytuje sporadicky, a systém ji určí jako anomálii [5].

Každá z výše uvedených technik má své výhody a nevýhody. Důležité je jakou analýzu chceme provádět, jestli si vystačíme se vzory špatného chování, či chceme nalézat anomálie, vzory nezachytitelnými, jaké máme výpočetní možnosti, a jestli požadujeme okamžité reakce. V rámci diplomového projektu je využita první možnost, protože primárně půjde o analýzu záznamů.

2.3.3 Uživatelsky přívětivé regulární výrazy

Regulární výrazy a jejich tvorba patří k základním dovednostem zkušených informatiků. Problém nastává v situaci, kdy se k problému vytvořit regulární výraz dostane uživatel, který neví, co regulární výraz je a má jen povrchní znalosti o vzorech. Takový systém se budeme snažit si nadefinovat a posléze i vytvořit. Základem jsou regulární výrazy definované induktivně takto:

- a je regulární výraz
- Pokud a, b jsou regulární výrazy pak také $a + b$ je regulární vyraz.
- Pokud a, b jsou regulární výrazy pak také $a.b$ je regulární vyraz.
- Pokud a je regulární výraz, pak také a^* je regulární výraz.
- Každý regulární výraz vznikne konečnou aplikací předchozích pravidel.

Tyto induktivně definované výrazy poskytují velmi silný aparát na analýzu záznamů. Pro analýzu systémových záznamů nepotřebujeme jejich plnou sílu. Uveďme si příklad, jak takovéto záznamy vznikají, či spíše jakým stylem vzniká jejich formát. Tento příklad si můžeme uvést ve formátu formátovacího řetězce příkazu `printf`, zde uvádím fiktivní řetězec, podobný záznamům z záznamů

```
printf("%d %s %s[%i]:accept from %i.%i.%i.%i", date, name, process, pid,
                                                ip1, ip2, ip3, ip4);
```

Kde první část vyjadřuje samotný formátovací řetězec. Znaky `%x`, $x \in d, s, i, ...$ vyjadřují formátovací typy, d – date, s – string, i – integer

date je typ nesoucí záznam o datu.

name je jméno uživatele.

process je jméno procesu který je volán.

pid je identifikační číslo procesu.

ip1 až ip4 vyjadřuje první až poslední část IPv4 adresy.

Příklad ukazuje jakým stylem bychom měli umět vyjádřit regulární výrazy, které nás budou zajímat. Pro lepší pochopení ukážeme výstupy z několika záznamů služeb:

```
/var/log/messages:
Dec 31 11:06:44 localhost acpid: notifying client 5287[0:100]
Dec 31 11:06:44 localhost su[5383]: Successful su for scottyh by root

/var/log/apache2/acces.log:
127.0.0.1 - - [10/Jul/2007:00:26:44 +0200] "GET /icons/image2.gif HTTP/1.1"
       404 289

/var/log/apache2/error.log:
[Thu Apr 26 22:48:35 2007] [error] [client 89.29.26.70] File does not exist:
       /usr/share/apache, referer: http://89.29.26.70/~scottyh/

/var/log/vsftpd.log:
Tue Dec 11 19:50:39 2007 [pid 25837] [ondra] OK DOWNLOAD: Client
       "195.113.168.254", "/home/ondra/107_pana/p1070489.jpg",
       545469 bytes, 34.41Kbyte/sec
```

Z výše provedené analýzy a příkladů z reálných systémů je vidět, jakým stylem budeme potřebovat analyzovat záznamy. Záznamy jsou tvořeny oddělovači a shluky písmen či číslic. Tyto shluky poté vytváří další sémantické celky jako jsou datum, IP adresa, cesta v souborovém systému, pid procesu, různé chybové hodnoty a velikosti souborů.

Uživateli by proto mělo být nabídnuto toto rozdělení na shluky písmen a číslic, aby si dále mohl určit jejich význam. Také by heureistycky mělo být určeno složení jednodušších celků na typy s vnitřním významem. Uživatel potom bude mít možnost si jen určit důležitost jednotlivých typů, jestli se jedná o konstantní výraz, to znamená, že se musí vyskytovat ve všech záznamech. Může se také jednat o proměnnou, která se obsahuje jméno a je využíto tohoto pojmenování v dotazech nad záznamem. Dále můžeme říci, že nás daná hodnota nezajímá, to znamená, není důležité, co se v dané části vyskytne, a tato hodnota ve výsledku nebude ani zaznamenána.

Takže uživatel by pak měl být schopen používat tyto výrazy na základě jejích vnitřního významu tak jak je chápe, a nemusí přemýšlet na tím, jak se který výraz zapíše ve formě regulárního výrazu. Toto je důležitou vlastností uživatelsky přívětivých regulárních výrazů, používat je tak, jak uživatel chápe význam záznamů v souboru.

2.4 Tvorba statistik přístupů informačních systémů

Užitečnou funkcí pro uživatele a získávání informací z informačních systémů je tvorba statistik přístupů. Jedná se o funkci, která umožňuje z systémových záznamů vytažení statistických informací. Dovoluje dělat součty hodnot, které vyjadřují velikost přenesených dat, počet přístupů za období a podobné funkce.

Většina existujících nástrojů je schopna provádět statistiky nad webovými servery, či podporují tyto statistiky nad FTP či mail servery. Provádí statistické informace ve stylu statistik přístupu k serverům za určitá období, dokáže analyzovat unikátní přístupy z adres, určovat přístupy podle světového rozložení IP adres (geoip). Umožňují získat vše, co se v souborech se záznamy nachází např. nástroj AWStats nacházející se na adrese: `http://awstats.sourceforge.net/`.
Všechny nástroje tyto statistiky se provádí dávkově tak, že se přepočítává celý soubor se záznamy. Jednou za časové období se provede opětovný výpočet. Tato operace je náročná a jsou zde doporučení na velikost volné paměti a rychlost výpočtů (tyto problémy nastávají u matematických výpočtů náročnějších na paměť, kde dochází k ukládání mnoha stavů). Pro ukázku vypočtené záznamy uvedené na stránkách nástroje AWstat tabulka 2.4.

Návštěvnost	Doporučená frekvence rotace záznamů	Užití paměti	Čas výpočtu
$0 - 10^3$	1/den 0-1MB log	4 MB	1 s
$10^3 - 10^4$	1/den 0-1MB log	4 MB	2 min
$10^4 - 10^5$	1/den 1-0MB log	4-8 MB	1s - 10s
$(2 - 4) \cdot 10^6$	každých 6 hodin, 384-768MB log	256-512 MB	12- 24 min

Tabulka 2.1: Náročnosti nástroje AWstat na generování statistických informací o přístupu k webovým serverům

Z tabulky je vidět, jak stoupá náročnost provádění této operace, a to jak na úrovni času, tak na úrovni spotřeby paměti, proto se tento nástroj nehodí pro více zatížené servery. Pro takové je výhodnější použít nástroj Webalizer[8] či Analog[9], který provádí výpočty řádkově orientovaným způsobem a ukládá jen obecnější statistiky, které vyžadují konstantní velikost paměti.

Statistiky pro web stránky jsou velmi důležitým pomocníkem, ovšem jsou zde problémy spojené s paměťovou náročností u složitějších statistik. Statistiky proto musí být zvoleny uvážlivě vzhledem výpočetní a paměťové složitosti. Analýzou existujících systémů jsme došli k názoru, že pro činnost systému jsou vhodnější statistiky, které se dají provádět jednoduchým způsobem a nejsou paměťově náročné a také takové, které se nepřepočítávají a umožňují proudový výpočet. Většina těchto systémů neumí klást na statistiky podmínky a dělat dotazy obdobné takovým, které se provádí v databázi. Systémy na této úrovni dále neumí interaktivní zadávání podmínek, omezení jsou vždy zadány v konfiguračních souborech.

[8] http://www.mrunix.net/webalizer/
[9] http://www.analog.cx/

Kapitola 3

Návrh systému

Naším úkolem je rozšířit systém pro analýzu záznamů, či IDS systémy o další funkčnost. Z předchozí analýzy plyne, že existují systémy na analýzu souborů, i na statistické výpočty. Ovšem tyto systémy jsou přímo určeny pro danou činnost. To znamená, že existují nastroje, které umí kontrolovat integritu souborových systémů, pak také, co umí provádět kontrolu souborů se záznamy, či dokáží provádět statistické výpočty. Jsou to většinou vysoce specializované nástroje, které by nebylo jednoduché upravit pro obecnější funkce. Proto vytvoříme nástroj, který bude dané systémy rozšiřovat. Vhodné body rozšíření byly zmíněny. Navrhneme sytém, který bude umět vytvářet uživatelsky přívětivé regulární výrazy. Bude potom nad nimi provádět statistické výpočty a zároveň bude mít možnost rozšíření o signalizaci událostí událostí a jejich možností reakce. Výsledný systém bude napsán v programovacím jazyce Python, který je předurčen přímo v zadání. Nabízí vysokou flexibilitu při zpracování textu.

3.1 Neformální specifikace

Naším úkolem je navrhnout systém, který umí analyzovat soubory se záznamy. Systém je postaven na základě uživatelsky přívětivých regulárních výrazů. Bude vytvořeno uživatelské rozhraní, které umožní jednoduchou práci s těmito výrazy, jejich vytváření a ukládání. Jako vstup vloží uživatel soubor, který chce analyzovat. Z tohoto souboru uživatel vybere jeden řádek, pro který chce vytvářet uživatelsky přívětivý regulární výraz. Tento řádek systém zpracuje a rozdělí na shluky písmen a číslic, které poté systém nabídne pro další zpracování uživatelem. Shluky číslic a písmen se ještě heuristickou analýzou spojí do sémantických celků, jaké bude schopen systém nalézt. Uživatel zůstane odstíněn od faktu, jakým způsobem jsou regulární výrazy vnitřně zapsány.

3.1.1 Funkční požadavky

- Interaktivní práce v grafickém rozhraní.
- Dávkové zpracování řádkovým klientem.
- Automatické vytváření sémantických jednotek.
- Výběr analyzovaného řádku ve zvoleném souboru.
- Změna analyzovaného souboru.

- Změna jména vybrané sémantické jednotky.
- Změna důležitosti u zvolené sémantické jednotky.
- Hromadné změny důležitostí pokud je vybráno více jednotek najednou.
- Přidávání omezení na řádek (zadáváno v disjunktivní normální formě).
- Odebírání omezení.
- Práce s agregačními funkcemi (součty, průměry, minima, maxima, počty) u vhodných sémantických jednotek.
- Výstup všech vyhovujících řádku.
- Výstup agregačních funkcí.
- Ukládání dotazů do souborů pro zpracování řádkovým klientem.

3.1.2 Sémantické jednotky

Datum – data mohou být v různých formátech a systém by je měl umět rozeznávat, alespoň základní typy. Datum bude obsahovat i časovou informaci.

IP – IPv4 adresa uzlu, jedná se o 4 čísla, které jsou odděleny tečkami.

Cesta – cesta v souborovém systému, cesta je určena jako spojení shluků číslic i písmen spojených lomítky jako oddělovači.

URI – identifikátor URI, jak je definován ve standardech. Bude rozeznávat alespoň základní typ URI identifikátoru jako je HTTP, FTP. Tento identifikátor bude navrhnut obecně a tudíž bude použitelný i pro jiné protokoly.

Číslo – tento identifikátor bude označovat obecné číslo, které nemá další vnitřní význam, ovšem dá se použít na porovnávání velikosti.

Velikost – tento identifikátor označuje velikost přenesených dat, či jinak vyjádřený objem dat.

Řetězec – spojení více textových shluků, či shluků číselných znaků.

Tyto typy bude možné v případě potřeby rozšířit o další uživatelské typy dle toho jaké operace, či potřeby budou kladeny na systém.
Po rozpoznání a spojení různých sémantických typů se pro jednotlivé určí jejich důležitost. Důležitost je rozdělena do 3 a více typů:

Nepodstatné – tento výraz bude ve výsledku ignorován a nebude do něj zahrnut.

Konstantní – v případě použití typu konstanta, se tento výraz v přesné podobě bude nacházet i ve výsledku.

Proměnná – Proměnné mohou nabývat libovolných hodnot, mohou být pojmenované, a tyto jména se pak využijí při následujících dotazech.

Vytvořený uživatelský výraz je předán do analyzátoru. Analyzátor pracuje s výše uvedenými proměnnými, které byly pojmenovány. Systém umožňuje klást dotazy na jednotlivé záznamy, psát omezení na velikosti a porovnávat a ptát se na podřetězce. Mezi funkce patří dotazy statistické, které se budou moci ptát na součty, průměry a maxima hodnot, u kterých je to proveditelné.

3.2 Analýza požadavků

Model je analyzován systémem diagramu případů užití. Jsou zachyceny všechny důležité funkce, které uživatel bude potřebovat (viz diagram 3.1). Systém musí poskytnout funkce které umožní plnohodnotnou práci s programem. Z nich se odvíjí diagram případů užití.

3.3 Architektura aplikace

Aplikace bude obsahovat dva typy zpracování, jeden vhodný pro interaktivní analýzu a druhý vhodný pro dávkové zpracování. Navržená architektura bude vypadat následovně (viz. obrázek 3.2). Samotná analýza není závislá na použitém rozhraní. Prvním typem je grafické rozhraní, které nabízí vyšší funkcionalitu včetně úprav a vytváření uživatelsky přívětivých regulárních výrazů. Umožní nám uložení uživatelsky přívětivého regulárního výrazu, tak aby se dal použitk analýze v řádkovém klinetu.

3.4 Konceptuální návrh tříd

Pokusíme se navrhnout model tříd, nad kterými bude systém pro analýzu pracovat. Jsou zde zachyceny pouze třídy zajištující funkce systému, nikoliv třídy, které by měly charakter zajišťující uživatelské rozhraní. Návrh tříd je uveden v diagramu 3.3.

3.4.1 Popis tříd

File – třída obstarávající práci se souborem, obvyklé operace jako otevírání, čtení.

Row – práce s řádkem v souboru.

Word – základní třída pro práci s logickými jednotkami textu, použita je pro odvozování dalších tříd konkretizující jejich funkci.

InterWord – Implementace třídy Word, pro znaky oddělovačů.

AlphaWord – Třída vyjadřující shluk písmen.

DigitWord – Třída vyjadřující shluk číslic.

Parser – Třída, která je schopna rozdělit řádek podle vstupních oddělovačů na výše zmíněné shluky, písmen či číslic, či oddělovačů.

REsemantic – sémanticky významné spojení jednoho a více předchozích typů tříd odvozených od třídy Word. Tato třída použita pro odvozování dalších tříd, které již nabývají konkrétního významu.

Dat – typ vyjadřující datum a čas.

IP – IPv4 adresa.

Size – velikostní jednotka.

Phrase – spojení vice slov, do věty.

Value – obecná číslicová hodnota.

Path – cesta v souborovém systému Unixového typu.

URL – URL identifikátor.

UserFriendlyRE – je třída, která z typů odvozených od třídy REsemantic a vstupního rozděleného řádku je schopna tvořit spojováním a rozdělováním, významové celky. Tato třída obsahuje i heuristickou metodu, která je schopna tyto typy automaticky poznat (pro jistou množinu formátování).

Query – třída odpovídající dotazu na soubor se záznamy, využívající jména. zvolená pro sémantické celky typu variable.

Podmínky – na výše uvedenou třídu mohou být kladeny ještě omezující podmínky, které vyjadřuje tato třída.

Analyser – je samotnou třídou provádějící analýzu nad souborem,a produkující výsledky dotazů, které mohou být vytvořeny.

Třídy nejsou podrobným výčtem všech, které se použijí ve výsledném projektu. Vystihují však základní funkčnost nad tím, jak je systém navržen a dává možnost pracovat s *uživatelsky přívětivými regulárními výrazy*. Umožní nám následně s jejich pomocí provádět dotazy nad zadaným souborem se záznamem.

3.4.2 Diagramy sekvence tříd

Při analýze jsou zajímavé dva diagramy sekvence ukazující posloupnost kroků pro dosažení cíle analýzy textu. Diagramy nastiňují, jak by měla probíhat komunikace mezi třídami a ukazují, jak třídy koncepčně spolupracují.
Obrázek 3.4 ukazuje posloupnost kroků pro vytvoření uživatelsky přívětivého regulárního výrazu z řádku souboru a práci s vkládáním omezení. Jako první krok uživatelské rozhraní vytvoří instanci třídy UserFriendlyRE, reprezentující uživalsky přívětivý výraz. Konstruktoru třídy je dán parametr odpovídající zvolenému řádku. V konstruktoru je vytořena instance třídy Parser, která se stará o rozdělení řádku na shluky znaků, použije při tom třídy jednotlivých shluků, pro zjištění jaký výraz odpovídá konkrétní tříde. Tím je vytvořeno pole obsahující třídy odpovídající posloupnosti shluků. Následně rozhraní zavolá metodu heuristic() třídy UserFriendlyRE, která provede sloučení shluků podle sémantických významů. Sekvenčně se projdou všechny třídy sémantických jednotek od nejsložitějších. Potom již je možné s jednotlivými jednotkami pracovat. Rozhraní muže libovolně měnit jména, důležitosti a vytvářet omezení. Omezení je reprezentováno speciální třídou, která obsahuje pole starající se o jednotlivé podmínky.

Druhý diagram sekvence (viz. obrázek 3.5) ukazuje práci analyzátoru pro vytvořený regulární výraz. Nejdůležitější je třída Analyser, která se stará o vlastní analýzu textu. Prvně musí rozhraní získat regulární výraz reprezentující daný řádek. Zavolá se metoda get_python_re(). Navrácen je text regulárního výrazu. Potom vytvoří rozhraní instanci

třídy Analyser, které předá při inicializaci regulární výraz, pole obsahující omezení a pole agregačních funkcí do kterého budou uloženy výsledky, dalším parametrem je soubor který chceme analyzovat. Posledním krokem je provádění samotné analýzy. Voláním metody analyse() je zanalyzován jeden řádek souboru. Analýza je provedena pro všechny řádky souboru.

3.5 Návrh grafického uživatelského rozhraní

V zadání diplomové práce bylo vytvořit grafické uživatelské prostředí pro program, které by umožňovalo jednoduchou formou s uživatelsky přívětivými výrazy pracovat. Návrh grafického subsystému je řešen pomocí knihovny GTK s dynamicky nahrávaným vzhledem. GTK knihovna byla zvolena proto, že je velmi rozšířená, dostupná a také proto, že to bylo požadováno v zadání diplomové práce. Pro návrh oken jsme použili návrhové prostředí Glade verze 3, které nám umožňuje vytvořit vzhled oken nezávislých na programovacím jazyku a operačním systému. Tento sytém je vybaven vizuálním nástrojem na návrh vzhledu. Pro většinu tlačítek v aplikaci byly zvoleny „Stock“ tlačítka, která jsou přímo v knihovně GTK, takže popis tlačítek a ikon je závislý na interpretaci knihovnou GTK podle nastaveného jazyka v systému.[1] Návrh grafického rozhraní není přímo závislé na použití knihovny GTK. Posloupnost funkcionality uvádíme v následujících stavových diagramech (viz. obrázeky 3.6 a 3.7). Podrobnější popis se nachází v podsekcích popisující jednotlivá okna.

3.5.1 Okno výběru řádku

První okno 3.8 je určeno pro zvolení souboru a volbu řádku z daného souboru, který bude následně analyzován. Pro otevření souboru slouží v horní liště v nabídce File tlačítko Otevřít. Pokud je v dialogu pro výběr souboru stisknuto Open, je otevřen jiný soubor. Před sebou vidíme okno, ve kterém můžeme rolovat a můžeme vybrat řádek, který budeme chtít analyzovat. Pro pokračování k dalšímu kroku zmáčkneme tlačítko Převést. (Viz obrázek 3.8.)

3.5.2 Hlavní okno analyzátoru

Do okna analyzátoru se dostaneme z předchozího okna za použití tlačítka Převést. Toto okno 3.9 slouží k samotnému nastavení parametrů pro analýzu a umožňuje změnu analyzovaného souboru a uložení regulárního výrazu pro pozdější použití v řádkovém klientu. Jak je vidět z obrázku 3.9, hlavním členem okna je GTK objekt Treeview starající se o zpřístupnění jednotlivých sémantických celků. na každém řádku máme možnost nastavit, jméno dané sémantické jednotky, a její důležitost. Důležitost může nabývat hodnot:

- Variable
- Constant
- None
- Hidden
- Unique

[1] Jazyk může být v systému nastaven např. proměnnou prostředí `LC_ALL`.

Dalším členem je samotný text, který reprezentuje sémantickou jednotku. Následují zadaná omezení, které můžeme pomocí okna 3.10 nastavovat, stejně jako agregační funkci.
Pro hromadnou změnu důležitosti je v okně vytvořen objekt Combobox, kdy se změní důležitost u vybraných řádků.
Jakmile je člověk spokojen s tím, jak daný uživatelsky přívětivý výraz vypadá může si ho uložit do souboru.[2] Následně může být tlačítkem analyse spuštěna analýza daného souboru a v případě potřeby může být analýza tlačítkem přerušena. Po dokončení analýzy je zobrazen výsledek ve výstupním okně 3.11.

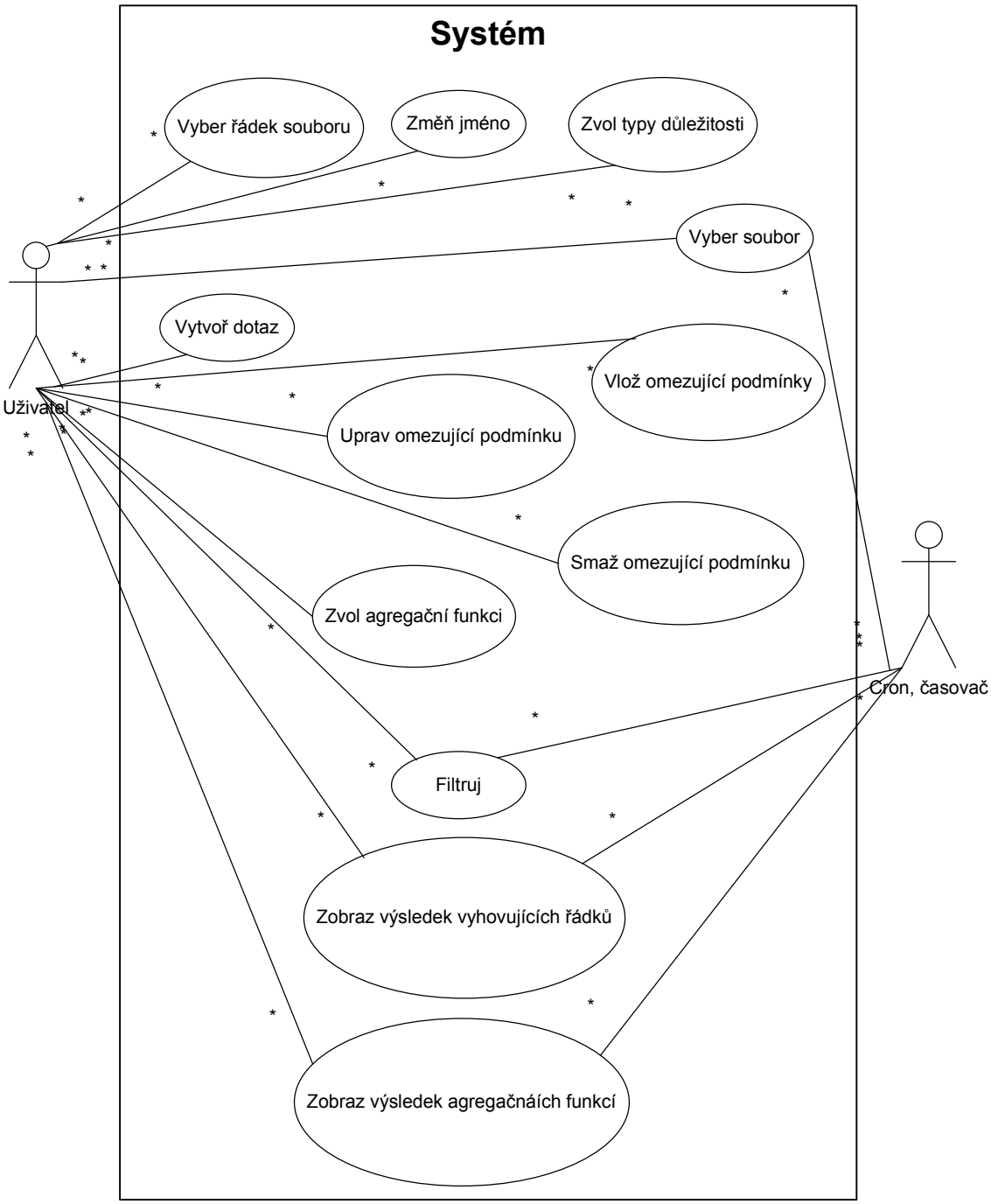
3.5.3 Okno pro změnu omezení

Dialog pro změnu omezení nám umožňuje zadávat podmínky pro sémantickou jednotku v disjunktivní normální formě. Objekt treeview v pravo nám dává možnost přidávat a ubírat jednotlivé členy disjunkce, vlevo nám umožňuje přidávat jednotlivé konkrétní podmínky. Podmínek je několik typů. které jsou závislé na tom, zda-li se jedná pouze o textovou formu, či formu s dalším v nítřním významem, na který můžeme klást podmínky na uspořádání.

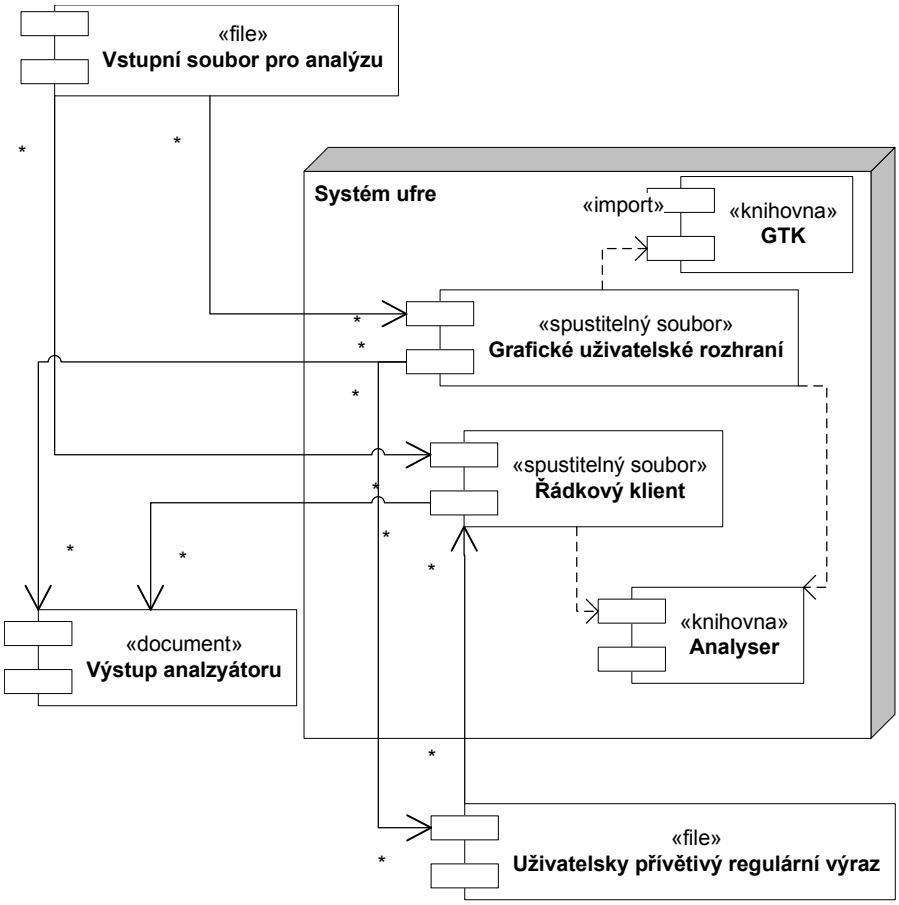
3.5.4 Výstupní okno programu

V okně zobrazeném na obr. 3.11 vidíme výstupní dialog se zobrazenými výsledky. Uloženy jsou v textových polích, takže můžeme v případě potřeby výsledky kopírovat. pro další použití. v horní části můžeme vidět řádky které vyhověly všem podmínkám a důležitost ním typům. Ve spodní části je zobraze celkový počet analyzovaných řádků a případné funkce které byly zvoleny.

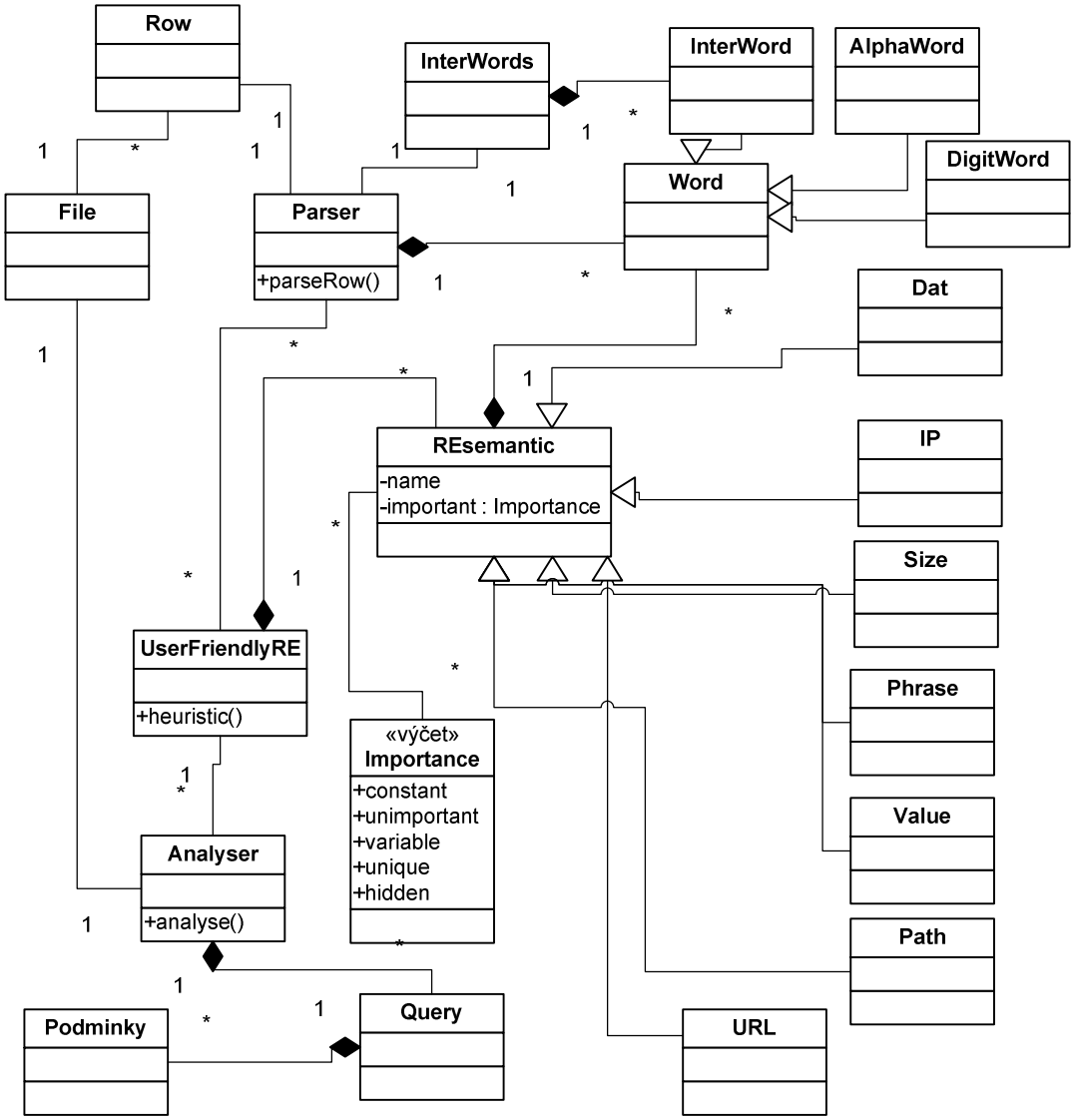
[2] Formát uložení je dán reprezentací objektů při použití modulu pickle. Formát není jednoduše uživatelsky čitelný ovšem to nebylo ani účelem.



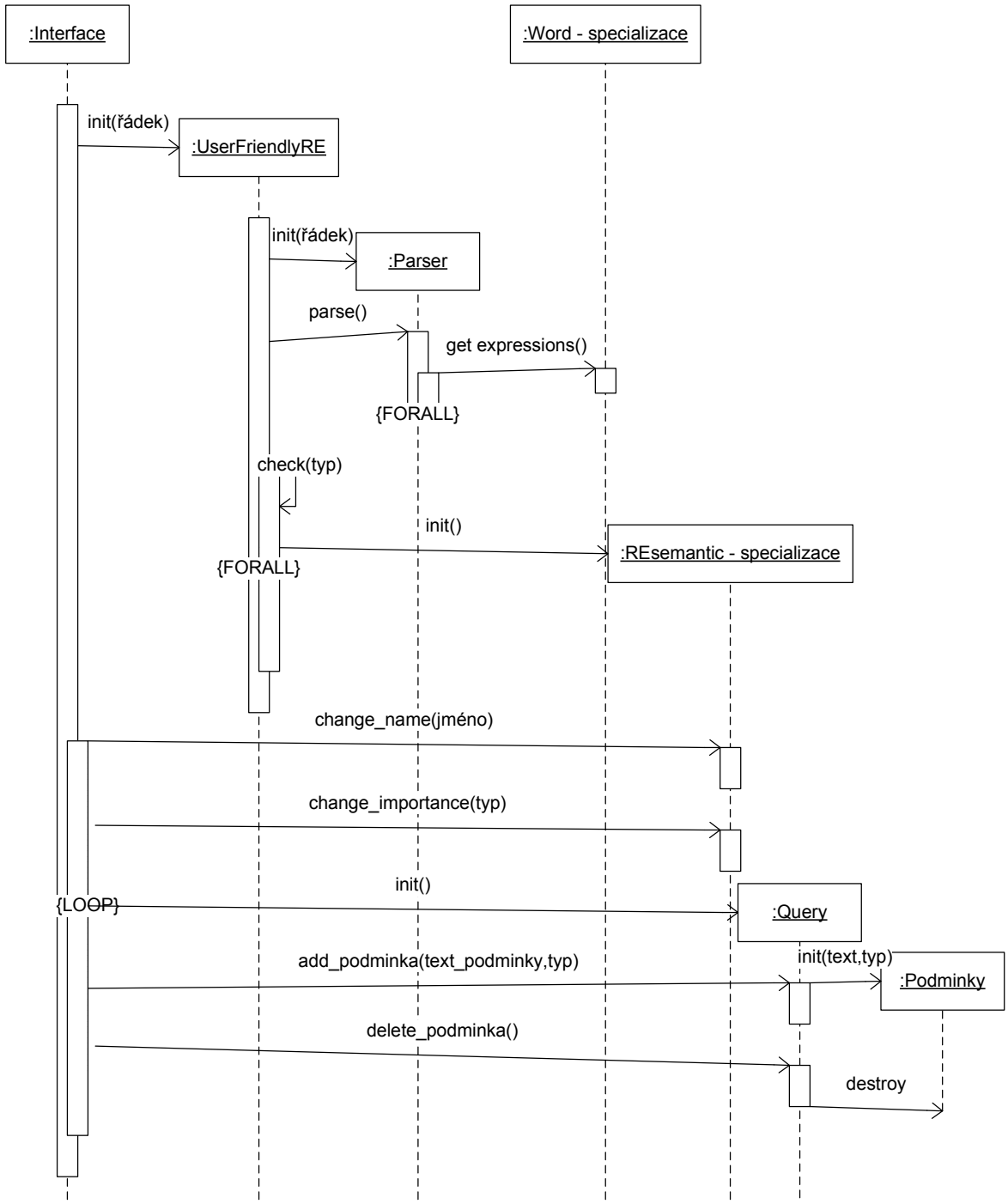
Obrázek 3.1: Diagram případů užití pro systém. Popisuje jaké funkce budou uživateli poskytnuty.



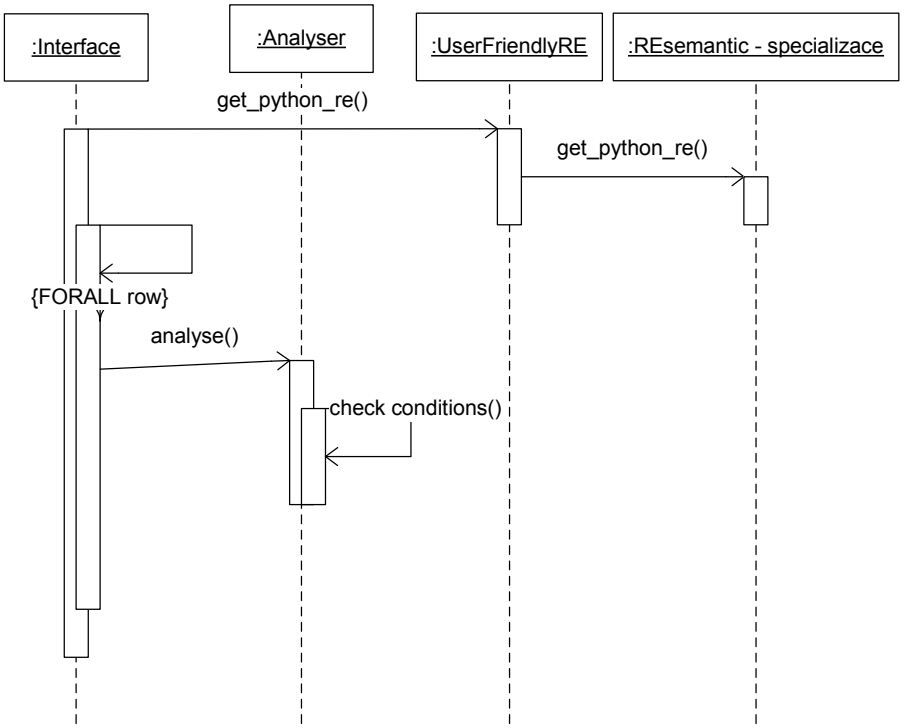
Obrázek 3.2: Diagram architektury aplikace, popisuje dva typy rozhraní, které jsou použitelné pro ovládání analyzátoru. Vstupem je externí soubor se záznamy, výstupem je pak výstupní dokument analyzátoru a u grafického rozhraní také možnost uložení regulárního výrazu.



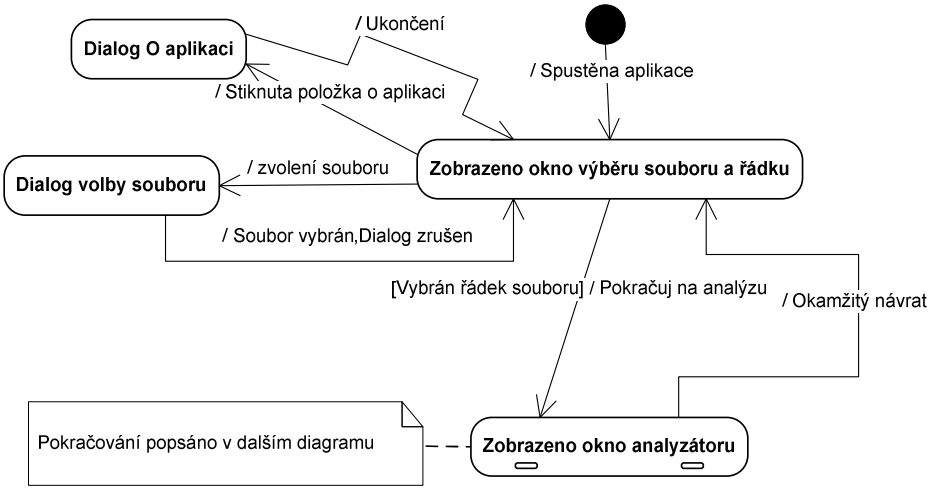
Obrázek 3.3: Diagram konceptuálního návrhu tříd.



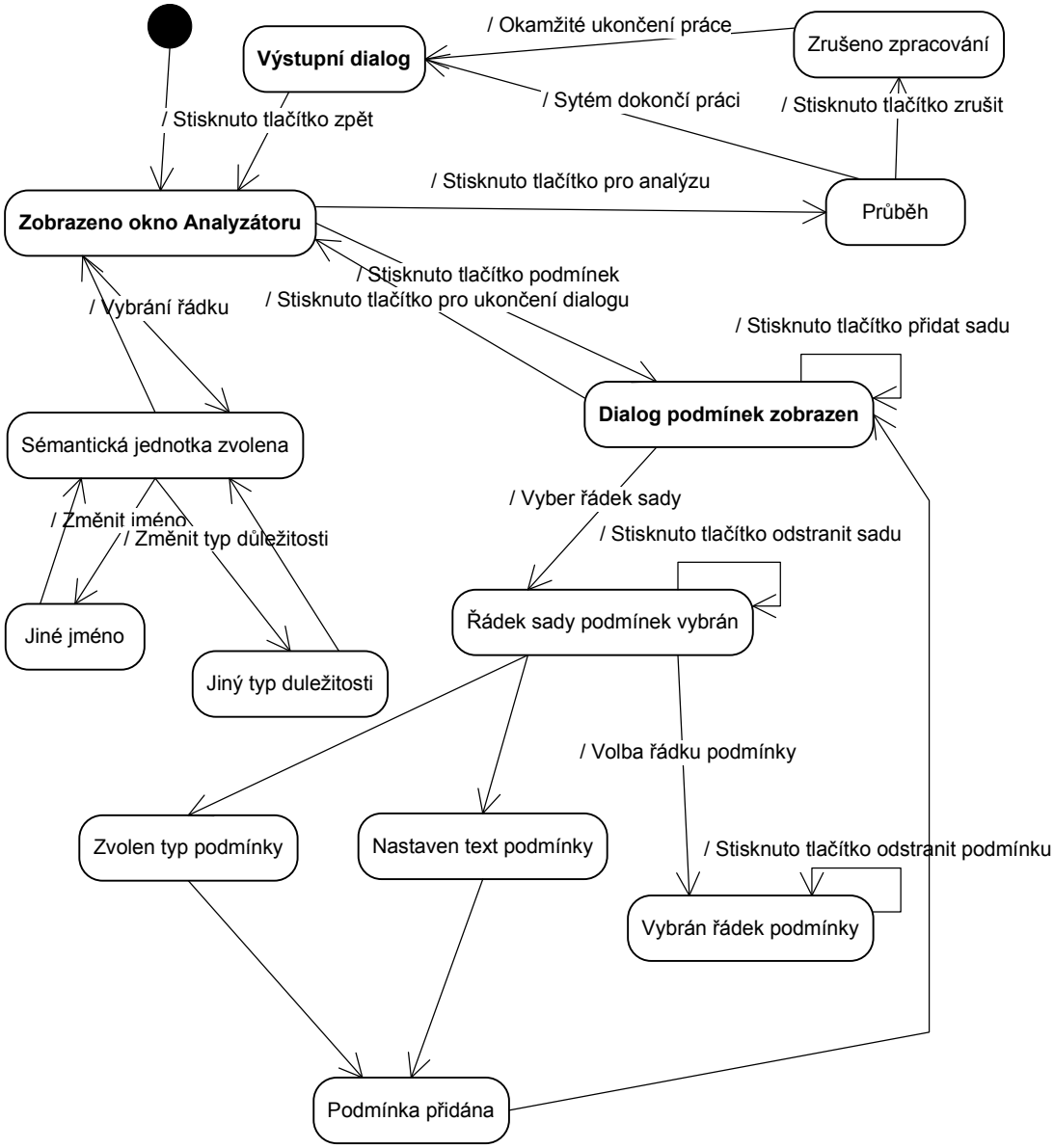
Obrázek 3.4: Diagram sekvence, ukazující posloupnost předávání zpráv a tvorby tříd pro tvorbu uživatelsky přívětivého regulárního výrazu s možností úprav jména, důležitosti a práci s omezeními.



Obrázek 3.5: Diagram sekvence, pro analyzátor regulárního výrazu. Ukazuje, jak je provedena analýza. Nejpodstatnější je inicializace a předání parametrů odpovídajících omezením, agregačním funkcím a analyzovanému souboru. Potom je v cyklu volána metoda analyse(), při každém spuštění je analyzován jeden řádek.



Obrázek 3.6: Návrh postupu kroků pro výběr řádku.



Obrázek 3.7: Návrh postupu kroků prováděných v analyzátoru pro daný zvolený řádek. Šipky do stejného stavu zdurazňují, že je provedena akce, ale nedojde ke změne stavu. Ze všech stavů úpravy podmínek je možno vést šipku zpět na stav dialog podmínek zobrazen.

File Help

Line no.	Text
31785	Mon Feb 4 13:11:31 2008 [pid 12037] CONNECT: Client "212.96.178.82"
31786	Mon Feb 4 13:12:17 2008 [pid 12036] [ondra] OK LOGIN: Client "212.96.178.82"
31787	Mon Feb 4 13:12:46 2008 [pid 12505] [ondra] OK MKDIR: Client "212.96.178.82", "/home/ondra/Ples zah
31788	Mon Feb 4 13:13:31 2008 [pid 12505] [ondra] OK MKDIR: Client "212.96.178.82", "/home/ondra/Ples ?ko
31789	Mon Feb 4 13:15:15 2008 [pid 12505] [ondra] FAIL MKDIR: Client "212.96.178.82", "/home/ondra/Ples ?k
31790	Mon Feb 4 13:16:29 2008 [pid 12505] [ondra] OK MKDIR: Client "212.96.178.82", "/home/ondra/Photos"
31791	Mon Feb 4 13:16:47 2008 [pid 12505] [ondra] OK MKDIR: Client "212.96.178.82", "/home/ondra/Photos.j
31792	Mon Feb 4 13:22:30 2008 [pid 18809] CONNECT: Client "212.96.178.82"
31793	Mon Feb 4 13:22:49 2008 [pid 18808] [ondra] FAIL LOGIN: Client "212.96.178.82"
31794	Mon Feb 4 13:23:15 2008 [pid 18808] [ondra] OK LOGIN: Client "212.96.178.82"

Převést

Obrázek 3.8: Úvodní okno systému, umožňující výběr souboru a řádku.

Otevřít Uložit jako O aplikaci

Type	Name	Importance	Expression	Podminky	Agregace
Date	name2	variable	Mon Feb 4 13:22:30 2008		
Other	name7	variable			
Other	name4	variable	[		
Other	name8	variable	pid		
Value	name3	unique	18809		
Other	name5	variable	]		
Other	name9	variable	CONNECT		
Other	name6	variable	:		
Other	name10	variable	Client		
Phrase	name1	variable	"212.96.178.82"	{contain(212) contain(96) } {contain(127) }	count

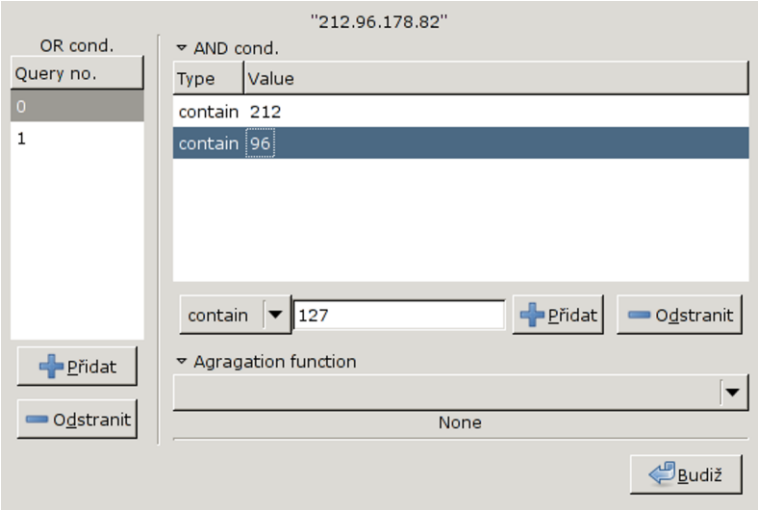
importance:

Vlastnosti

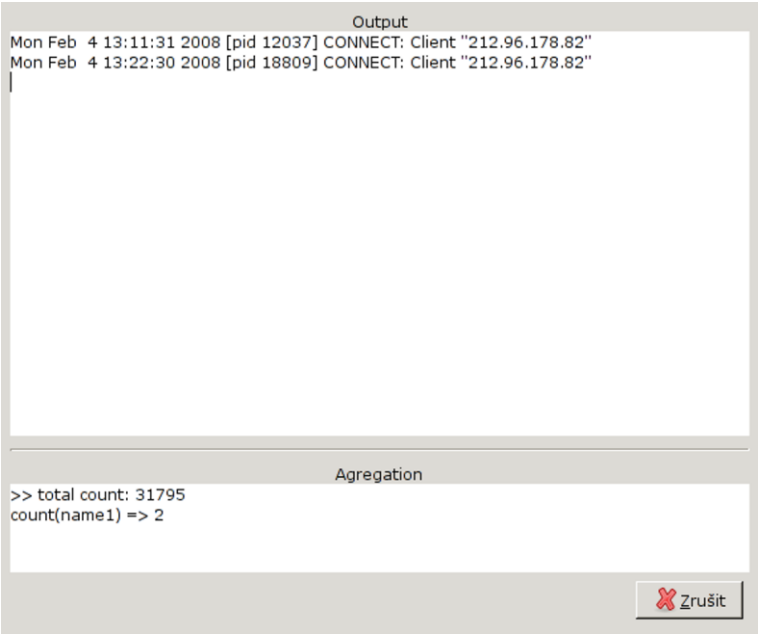
/a/diplomka/prog/vsftpd.log

Analyze Zastavit

Obrázek 3.9: Hlavní okno, které umožňuje úpravy podmínek a výrazů



Obrázek 3.10: Okno úpravy podmínek, přidávání, rušení. Změnu agregační funkce.



Obrázek 3.11: Okno s výstupem, zobrazuje vyhovující řádky a výsledky funkcí.

Kapitola 4

Implementace

4.1 Opravy navrženého modelu

V předchozí kapitole jsme navrhli model, který měl implementovat danou funkcionalitu. Návrh byl revidován a opraven oproti návrhu, který byl součástí semestrálního projektu. Narazili jsme na problémy s rozdělováním řádku na shluky souvisejících znaků. Dále bylo upraveno navázání podmínek, které sice patří k analyzátoru, ale musí být spojeny s konkrétní sémantickou jednotkou, kvůli vnoření operací do sémantických jednotek.

4.1.1 Opravy dělení řádku

Při prvních pokusech při dělení řádku na dále již nedělitelné jednotky jsme zjistili, že potřebujeme jemnější dělení. Původně bylo navrženo dělení na shluky písmen číslic a oddělovače.
Při spuštění se nám řádek rozdělil na tyto celky správně, ale pro následné skládání do sémantických jednotek jsme potřebovali ještě některé z mezislovních oddělovačů aby měli významnější váhu. Proto jsme vytvořili významné oddělovače, které dále dělí množinu oddělovacích znaků na další pod celky. Mezi významné byli zařazeny:

Bílé znaky – Všechny znaky patřící do třídy kterou obsahuje regulární výraz \s.

Uvozovky – Význačné pro vytvoření složení dále neděleného obsahu v uvozovkách, který může obsahovat jakýkoliv jiný znak

Lomítko – Potřeba pro rozdělování cest v adresářích

Tečka – Mívá speciální význam, např. v IP adrese, či v URL

Vytvořili jsme potřebné třídy odvozené od třídy Word, která reprezentuje tyto shluky. Tím jsme dosáhli větší granularity a podařilo se nám implementovat dělení, uvnitř kterého jsme následně mohli používat opakování interních znaků, např. u sémantické jednotky v uvozovkách jsme mohli říci, že se uvnitř může vyskytnout libovolný jiný znak než uvozovka.

Náhrada regulárních výrazů interní reprezentací

V projektu jsme vytvořili vlastní implementaci regulárních výrazů. Implementace je použita na úrovni dělení a spojování dále nedělitelných jednotek.

K vytvoření vlastní zjednodušené implementace regulárních výrazů nás vedla myšlenka budoucí potřeby práce s těmito výrazy. Obecné regulární výrazy jsou pro většinu lidí nepoužitelné, protože mají příliš expresivní syntaxi. Potřebovali jsme zachovat nedělitelné jednotky jako je sluk znaků či číslic, tak aby uživatel nemohl pracovat až na úrovni jednotlivých znaků. Toho můžeme dále využít ke zjednodušenému zápisu, kdy např. za IP adresu označíme vše, co obsahuje čtyři čísla oddělená tečkami. Tato práce s regulárnímy výrazy je použita pouze na úrovni vnitřního rozdělování řádku, pro samotnou analýzu jsou výrazy následně převedeny do klasických regulárních výrazů jazyka Python. Implementace neklade důraz na optimalitu oproti implementaci regulárních výrazů v modulu `Re` jazyka Python, ale pomohla nám zjednodušit výrazy.

Výsledkem zjednodušení je zachování shluků znaků, které by v případě potřeby rozšíření aplikace dovolovali jednodušší manipulaci s jednotkami, jejich slučování a rozdělování na jednodušší uživatelské úrovni.

4.1.2 Navázání podmínek

Podmínky v původním návrhu měly být navázány na analyzátor s využitím uživatelsky přívětivého regulárního výrazu. Zvážením možností jsme dospěli k názoru, že takové to omezení je až moc volné a nebylo by možnost kontrolovat jestli podmínky jsou vůbec smysluplně zadané. Do tříd starající se o danou sémantickou jednotku je vložena funkčnost obstarávající kontrolu toho, zda-li text daný parametrem je výrazem dané jednotky, v případě neúspěchu je vyvolána výjimka patřící k danému typu, která může být na vyšší úrovni odchycena a dle toho se zachová uživatelské rozhraní.

Zásadní funkčností je rovněž přidání porovnávání dvou řetězců na úrovni sémantické jednotky. Funkčnost musela být na této úrovni, protože jsme potřebovali porovnávat typy vzhledem k jejich sémantickému významu. V případě sémantické jednotky reprezentující datum jsme museli konvertovat řetězce na datum a porovnávat jejich význam vzhledem k významu datu, obdobně u číselných hodnot.

4.2 Závislosti

Program byl navržen pro implementaci v jazyce Python, implementace a testování probíhalo s verzí 2.4.4. Byly použity standardní knihovny pro jazyk Python.

re – modul pro práci s regulárními výrazy jazyka Python.

os – modul využit pro práci se soubory, kontrola jejich existence, získání absolutních cest z symbolických odkazů.

sys – modul využit pro nestandardní ukončování aplikace.

pickle – modul pro ukládání a čtení dat a tříd do/ze souboru. V naší aplikaci je použito pro uložení potřebných dat uživatelsky přívětivého výrazu pro další analýzu.

locale – modul pro lokalizaci, nastavuje lokalizaci LC_TIME na hodnotu C, kvůli správné práci s daty. Systémové záznamy toto kódování využívají implicitně.

datetime – modul pro práci s daty, poskytuje funkce rozdělení textového data do interní reprezentace.

GTK – základní modul pro práci s GTK grafickou knihovnou, ve které je aplikace tvořena, minimální verze knihovny GTK je 2.6.

GTK.glade – rozšiřující modul knihovny GTK pro dynamické nahrávání uživatelských rozhraní z externích XML souborů. Pro návrh jsme použili verzi `Glade 3`.

gobject – nadtřída prvků z knihovny GTK.

Tyto verze není třeba zcela nutně dodržet, ovšem doporučujeme je jako testovací prostředí. Verzi PyGTK je nutno dodržet poněvadž program využívá GTK.CellRendererCombo, který je nový od verze 2.6 a je programem využíván. Grafický návrhový systém Glade 3 není potřeba v případě používaní programu. Nutný je pro návrh. Testováno bylo i Glade 2, ve kterém rovněž okenní prostředí je možné modifikovat.

4.3 Implementace modelu

Model jsme implementovali rozložením funkcionality do mnoha souborů, kde každý soubor obsahuje pouze jednu třídu, mimo tříd starajících se o okenní subsystém. Tento přístup byl zvolen pro lepší orientaci mezi třídami a rovněž aby kopíroval navržený model. Model je dělán pokud možno robustní cestou, tak aby bylo možné libovolnou třídu nahradit, či přidat funkcionalitu.

4.3.1 Rozložení funkcionality podle souborů

File.py – Je obalující knihovnou pro práci se soubory, obsahující metody na inicializaci souboru, načtení řádku textu ze souboru a z dané pozice. Jde o obalující třídu nad standardními funkcemi v jazyce Python pro práci se soubory. Byla vytvořena pro budoucí potřebu nahradit standardní funkci za jinou, kvůli možné optimalizaci.

Row.py – Obsahuje funkce pro práci nad třídou File o kterou se stará. Umožňuje provádět možné konverze ještě před předáním samotnému zpracování. V našem případě je vždy odstraněn znak konce řádku kvůli chybnému zobrazování v okenním prostředí. Obsahuje metody, která vrátí pozici a text z daného řádku načteného metodou read() třídy File.

Word.py – Základní nadtřída od které jsou odvozeny třídy znamenající shluky znaků, číslic, mezislovních oddělovacích znaků. Obsahuje metody starající se o práci se základními celky, navrací regulární výraz reprezentující danou hodnotu. Do Třídy je uložen reprezentant daného výrazu, pozice na které se nachází a přímo definicí třídy je zabudován na začátku, i jak vypadá reprezentace daného celku. takže např. pro číslo je interním reprezentantem : \d.

AlphaWord.py – Třída odvozená z třídy Word, obsahuje metody a předefinovává třídu, tak aby mohla daná třída reprezentovat shluk normálních abecedních znaků.

BlankWord.py – Třída odvozená od třídy Word, a reprezentující shluk bílých znaků, jejím reprezentantem je znak v Python regulárních výrazech \s.

DigitWord.py – Třída odvozená od třídy Word reprezentující shluk číselných znaků jazyka Python: \d.

DotWord.py – Třída odvozená od třídy Word reprezentující shluk znaků tečka. Tuto třídu jsme museli dodělat, byla nutná pro oddělení znaku tečka od ostatních oddělovacích znaků.

SlashWord.py – Třída odvozená od třídy Word reprezentující shluk znaků lomítko. Potřebovali jsme vytvořit Strukturu reprezentující adresářovou cestu, proto jako v předchozím případě musela být tato třída vyčleněna z ostatních oddělovacích znak.

QuoteWord.py – Třída odvozená od třídy Word reprezentující shluk znaků uvozovek. Tuto třídu jsme museli vyčlenit z důvodu potřeby najít párový znak uvozovek při vytváření sémantického celku v uvozovkách.

InterWord.py – Třída odvozená od třídy Word reprezentující znaky oddělovačů slov. Takže musí obsahovat zbylé znaky, které se nezařadili do ostatních tříd.

InterWords.py – Obalovací třída nad třídou InterWord, která má za úkol provést možné konverze mezislovních znaků, v našem případě není této možnosti využito. Třída pouze předává mezislovní znaky dále do třídy reprezentující Parser.

Parser.py – Soubor obsahuje třídu starající se o rozdělení řádku na dále nedělitelné celky odvozené od třídy Word, tato metoda pomocí metody findall objektu re nalezne všechny takto definované celky. a určí do jaké třídy patří daný shluk. Třída obsahuje metodu, která vrátí pole uložených rozdělení na třídy odvozené od třídy Word.

REsemantic.py - Obecná třída reprezentující sémantickou jednotku, jednotku, která je reprezentována svým jménem, důležitostí, reprezentačním textem a pozici v textu a číslem vzoru podle kterého byl reprezentant nalezen. Třída obsahuje pole reprezentantů, které je možno použít na hledání na shodu v analyzovaném textu. Třída obsahuje následující důležité metody, které jsou využívány při analýze:

- is_unique – Metoda která testuje zda-li je daná hodnota unikátní, slouží na uložení hodnot do množiny a test unikátnosti. Pokud je unikátní, je vstupní hodnota vložena do množiny, a navrácena hodnota True, jinak False.
- is_contain – Jedna z metod pro testování podmínek pro danou sémantickou jednotku, testuje zda-li řetězec je obsažen v daném textu. V našem programu je tato hodnota testována vždy jako jediná vzhledem k textové reprezentaci. Nezdálo se užitečné pro testování na obsaženost testovat konvertovanou hodnotu.
- is_equal – kontrola na shodnost daných hodnot, je porovnáváno v konvertované podobě, takže např. hodnota typu Size $1kb$ je konvertovaná na číselnou hodnotu $1024,0$ a musí být proto testována na shodu s konvertovanou hodnotou.
- is_bigger – Test převedené podmínky na to zda-li je hodnota stejná nebo větší, tyto testy jsou povoleny pouze u typů u kterých lze takto porovnávat, proto je porovnání omezeno pouze na číselné a datumové typy.
- is_smaller – Test převedené podmínky na to zda-li je hodnota stejná nebo menší. Omezení je stejné jako u podmínky *is_bigger*.
- get_python_reg – metoda vrátí regulární výraz v jazyce python, který reprezentuje sémantická jednotka. Výraz je vracen dle nastavené proměnné důležitosti, která slouží k nastavení důležitosti.
- get_ret – Konverze vstupního řetězce na hodnotu odpovídajícího vzoru.

Phrase.py – Obsahuje třídu odvozenou od třídy REsemantic reprezentující libovolný text v uvozovkách, tato třída je analyzována jako první, a vnitřní smysl textu v uvozovkách se dále neanalyzuje.

Dat.py – Obsahuje třídu odvozenou od třídy REsemantic reprezentující datum a a čas, pro testování podmínek a návratové typy je text převeden na typ Datetime reprezentovaný v jazyce.

IP.py – Obsahuje třídu odvozenou od třídy REsemantic reprezentující IP adresu. Vnitřní význam není dále interpretován, je určeno, že se jedná o IP adresu, všechny operace porovnávání jsou prováděny v textové podobě.

Size.py – Reprezentace třídy Size, starající se o reprezentaci a operace spojené s velikostí která je udána v bytech, kilobytech, megabytech a gigabitech, hodnoty jsou převedeny do interní reprezentace odpovídající velikosti v bytech.

URL.py – Obsahuje třídu odvozenou od třídy REsemantic reprezentující URL adresu, tento typ se v analyzovaných systémových záznamech nevyskytoval, proto není důsledně otestován. Je připraven pro budoucí použití.

Path.py – Reprezentace cesty v souborovém systému unixového typu, který musí začínat znakem lomítka.

Value.py – Reprezentace obecné hodnoty, bez dalšího vnitřního významu. Je nad ním možno provádět stejné operace jako nad typem Size.

Zbytek.py – Obsahuje třídu odvozenou od třídy REsemantic reprezentující zbytek, spojení a symboly které nemají další vnitřní smysl, který se nepodařilo zachytit v předchozích třídách. Musí být testováno na výskyt až v posledním kroku, tak aby nedošlo k zachycení všeho tímto typem.

UserFriendlyRE.py – Třída reprezentující kompletní uživatelsky přívětivý výraz, a nástroje na jeho tvorbu. Třída obsahuje 2 zásadní metody, které vykonávají veškerou funkčnost.

- heureistic – Pokusí se nalézt všechny sémantické jednotky v daném řádku. Hledání probíhá sekvenčně, podle subjektivní důležitosti typů a jejich složitosti. Nejsložitější typy by se měli hledat první. tato metoda interně použije metodu `hledani`, která se postará o průchod přes všechny vzory pro daný sémantický typ. Tato metoda následně použije pro interní použití metodu `check` pro kontrolu a vyhodnocení zdali se daná sémantická jednotka pro konkrétní vzor v řádku nalézá, či nikoliv. V případě nalezení se označí a pokračuje se v prohledávání. Tato funkce je napsána dosti složitě, protože jsme museli implementovat hledání v námi vytvořených regulárních výrazech.
- get_python_reg – Metoda vytvoří kompletní regulární výraz jazyka python. Výraz je složen z jednotlivých regulárních výrazů pro sémantické jednotky.

Query.py – Obsahuje třídu starající se o jednu sadu podmínek. podmínky pro sémantickou jednotku tvoří disjunktivní normální formu. Třída query je jednou částí. Vyjadřuje jednu konjunkci, disjunktivní normální formy:

$$pominka_1 \wedge pominka_2 \wedge \cdots$$

Podminka.py – Obsahuje třídu jedné podmínky obsažené které se využívají ve třídě Query.

Analyser.py – Třída analyzátoru. Jako vstup jsou použity sémantické jednotky uživatelsky přívětivého regulárního výrazu a přidané podmínky dané na analyzátor. a obsahuje jedinou důležitou metodu která se jmenuje `analyse`. Ta provede analýzu jednoho řádku a výstup je poslána hodnota odpovídající danému řádku, pokud řádek vyhovuje. Jinak je navrácena hodnota None. V této funkci se rovněž počítají agregační funkce zadané přes uživatelské rozhraní.

ufre_gui.py – Hlavní soubor pro spouštění grafického uživatelského rozhraní. Obsahuje navázání na dynamicky nahrávaný vzhled pře knihovnu GTK-glade. Soubor vytváří třídy obsluhující události, které nastanou v oknech a provádí analýzu ve vláknu, tak aby zůstalo v provozu hlavní vlákno a nezůstala program v nepoužitelném stavu. V souboru se nachází třída fungující jako wrapper nad třídami pro ovládání událostí a automaticky navazuje všechny metody jako ovladače událostí v okně.

ufre_cli.py – Klient pro příkazovou řádku. Soubor obsahuje funkčnost tohoto řádkového klienta. Stará se o obsluhu příkazové řádky a pokud se nevyskytne chyba při zpracování, provádí analýzu zadaného souboru pomocí uloženého regulárního výrazu. Jako regulární výraz je brán soubor, který je vyexportován, po tvorbě v grafickém uživatelském rozhraní.

Třídy jsou rozčleněny do souborů dle jejích významu. Základní třídy z pohledu modelu se nachází každá v jediném souboru kvůli přehlednosti a návaznosti na model. Tím jsme ušetřili čas pro pochopení modelu a nalezení tříd.

4.4 Možnosti použití

Aplikace je určena pro analýzu systémových záznamů. Od toho se odvíjí možnosti použití aplikace. Vytvořili jsme dvě možnosti, jak s aplikací pracovat. Buď interaktivně přes grafické uživatelské prostředí, či přes program pro příkazovou řádku. Program pro Grafické uživatelské prostředí nám nabízí možnosti, jak vybírat řádky a klást omezení na výběry. Klienta pro příkazovou řádku můžeme spouštět dávkově a navázat jeho vyhodnocení na případné další akce.

4.4.1 Grafického uživatelské rozhraní

Grafické uživatelské rozhraní poskytuje komfort okenního prostředí a interaktivní práce. Při práci můžeme doladit omezení a vizuálně kontrolovat výstupy a snažit se o jejich korekce, např. podle očekávané výstupní hodnoty počtu nalezených řádků.
Grafické prostředí jsme vytvořili pro snadnou tvorbu uživatelsky přívětivých regulárních výrazů. Uživateli je nabídnut seznam sémantických jednotek daného zvoleného řádku a uživatel má možnost je upravovat. Pro každý celek má možnost nastavit jméno. Jméno je použito při vytváření agregačních funkcí, kde funkce bude svázána s tímto jménem. V budoucí verzi může být jméno použito pro lepší specifikaci výstupu, kdy by si uživatel mohl zvolit i pořadí daného sémantického celku. V této verzi programu není implementováno, poněvadž tato funkce by měla pouze kosmetický charakter seřazení podle zvolených jmen. Další funkcí, co je uživateli nabídnuta, je možnost nastavit důležitost dané sémantické jednotky. Ta ovlivňuje, jak je jednotka zpracována. Pak může uživatel nastavit podmínky pro

jednotku, které omezují výběr zadaného výrazu v disjunktivní normální formě.
Hlavním důvodem existence a možností použití je právě interaktivní analýza založená na tom, že uživatel potřebuje zjistit danou věc okamžitě. To znamená, že si zvolí řádek a omezující podmínky a okamžitě po stisku tlačítka je mu nabídnut výsledek, podle kterého se může zachovat. Případně podle uvážení zareagovat. Druhou možností použití je vytvoření vhodného výrazu tak, aby byl použitelný pro klienta příkazové řádky. Takže je vytvořen např. výraz který má za úkol vypsat unikátní IP adresy v daném souboru. Ten pak bude spouštěn např. každý týden a mohou být takto vytvářeny statistiky IP adres, což odráží požadavek na řádkového klienta.

4.4.2 Řádkový analyzátor

Řádkový analyzátor je velmi jednoduchý. Neposkytuje interaktivní práci s uživatelsky přívětivými výrazy. Neumožňuje ani změnu výrazů. Vše musí být zvoleno již při konstrukci výrazu v grafickém prostředí.
Ačkoliv nemá možnosti grafického prostředí, jeho síla tkví v jednoduchosti a dávkovém zpracování. Možností použití by bylo spojení s aplikací `cron`, či podobné aplikace na spouštění příkazů v nastavený čas, Použití by mohlo být, jak je naznačeno v předchozím příkladě (např. kontrola každotýdenní testy unikátních IP adres). Zvláště ve spojení s rotací systémového záznamu, kdy by rotace následovala pro provedení těchto výrazů, tak by mohlo být sledováno např. jaké IP adresy se připojují, ve které týdny v roce a mohl by se provádět nějaký druh statistické analýzy jako např. dolování v datech OLAP technikami.
Dalším způsobem použití bude spojení s dalším programem rourou, či jiným způsobem spojené aplikace, kde muže být výsledek spojen např. s tvorbou firewallového pravidla. Mohlo by být nastavena závislost povolení či zakázání přihlášení daného uživatele v systému po splnění určitých podmínek.

4.5 Práva uživatele

Spouštění programu není nijak omezeno na uživatelská práva. Program může používat každý uživatel systému s právy spouštět program. Uvedeme si omezení plynoucí ze systémových záznamů.
Důvodem omezení jsou bezpečnostní omezení systémových služeb. Mnoho systémových služeb běží pod právy root uživatele, či pod jiným uživatelem, a muže nastat, že se v systémovém záznamu mohou vyskytnout citlivé údaje např. při nesprávném přihlášení do systému se mohla vyskytnout situace, že uživatel napsal omylem heslo místo uživatelského jména a toto jméno se tím pádem dostalo do souboru se špatně přihlášenými uživateli. Což představuje velké bezpečnostní riziko, proto většina systémových záznamů se nachází v adresáři `/var/log/` kde jsou nedostupné pro běžného uživatele.
Samozřejmě uživatel může spustit službu pod svým uživatelským účtem a nastavit systémový záznam do adresáře, kde se dostane (obvykle nastavitelné v konfiguračních souborech). Tím mu bude umožněno systémový záznam analyzovat.
Všeobecně jsou dvě možnosti, jak postupovat. Můžeme analyzovat záznam tzv. offline, neboli máme soubor, ke kterému máme přístup a není svázán s konkrétní službou a my ho chceme analyzovat. Druhou možností je mít dostatečné oprávnění na čtení z daného souboru. Toho lze dosáhnout několika možnostmi. Buď spustit program pod právy oprávněného uživatele, či oprávněný uživatel udělí právo nám s tímto souborem pracovat. (V operačním systému Linux např. přidáním uživatele do skupiny, pod kterou běží daná služba, či přidáním

do rozšiřujících ACL listu[1]). Není vhodná možnost nastavení spuštění pod jiným efektivním ID uživatele v systému (např. pomocí nastavení set UID bitu programu s právy uživatele `root`. Tím by se vytvořila bezpečnostní díra, uživatel by si mohl zobrazit libovolný soubor systému, což v žádném případě není vhodné. Také to není vhodné z důvodu, že tyto programy musí být maximálně zabezpečeny a poskytovat co nejmenší možnosti, tak aby se minimalizovalo nebezpečí nalezení chyby.

Omezení tedy odpovídá nastavení operačního systému a přístupu k daným souborům. Samotný program není nijak výjimečný a nemá udělena zvláštní oprávnění. Může být spouštěn libovolným uživatelem a ten s ním může provádět analýzu souborů, ke kterým má přístup.

[1] ACL (angl. Access Control List) – poskytuje rozšíření standardních práv v systému. Rozšiřuje model vlastníka a skupiny. Umožňuje mít větší kontrolu nad přístupem k souboru, až na úroveň jednotlivých uživatelů.

Kapitola 5

Testování

5.1 Verze

Testování probíhalo na počítači s operačním systémem Linux s následující konfigurací:

```
Linux
Jádro:      2.6.23.12
Platforma:  i686 Mobile AMD Sempron(tm) 3100+ AuthenticAMD
Velikost operační paměti: 512Mb (SO-DIMM DDR 333Mhz)
Disk: 160Gb (5400 otáček/s)
Distribuce: Gentoo
Správce oken: Openbox 3.4.6.1
```

Python – verze 2.4.4 jako provozní prostředí.

PyGTK – verze 2.6 pro základní grafický subsystém.

5.2 Porovnání rychlosti zpracování regulárních výrazů

Při testování aplikace jsme se rozhodli otestovat rychlost zpracování regulárních výrazů. Porovnali jsme rychlosti zpracování různých typů regulárních výrazů nad stejným souborem. Použili jsme výrazy, které vybíraly ze souboru stejné řádky
Pro testování jsme zvolili soubor `vsftpd.log` o velikosti 5361064 byte (cca 5,2Mb), 31795 samostatných řádků.
Jako regulární výraz jsme zvolili následující:

```
^[ a-zA-Z]+.*[\d]+[.][\d]+[.][\d]+[.][\d]+
```

Testovali jsme několik typů, které se odlišovaly tím, že se ukládaly řetězce, či se neukládalo nic. Porovnávali jsme rychlosti průběhu aplikace pomocí systémového programu `time`, který navrací reálný, uživatelský a systémový čas strávený při vykonávání programu. Takulky obsahují naměřené hodnoty.
Testovací program:

```
#!/usr/bin/python

import re
fi=open("../../prog/vsftpd.log")
```

```
count=0

while True:
    line=fi.readline()
    if line==="":
        break
    # a1
    #a=re.search( "^[ a-zA-Z]+.*[\d]+[.][\d]+[.][\d]+[.][\d]+", line)

    # a2
    #a=re.search( "(^[ a-zA-Z]+)(.*)([\d]+[.][\d]+[.][\d]+[.][\d]+)", line)

    # a3
    #a=re.search( "(?P<name1>^[ a-zA-Z]+)(?P<name2>.*)
    (?P<name3>[\d]+[.][\d]+[.][\d]+[.][\d]+)", line)

    # b1
    #a=re.search( "(?P<name1>^[ a-zA-Z]+)(?P<name2>.*)(?P<name3>[\d]+)
    (?P<name4>[.][\d]+)(?P<name5>[.][\d]+)(?P<name6>[.][\d]+)", line)

    # b2
    #a=re.search( "(?P<name1>^[ a-zA-Z]+.*[\d]+[.][\d]+[.][\d]+[.][\d]+)",
                  line)

    # c1
    #a=re.search( "(^[ a-zA-Z]+.*[\d]+[.][\d]+[.][\d]+[.][\d]+)", line)

    if a!=None:
        count=count+1
print count;
```

Programem jsme testovali jednotlivé výrazy, které jsme upravovali tím, že jsme odkomentovali dány řádek, pro který jsme prováděli testování. Nad každým je dáno označení konkrétního testu.

a1	real	user	system
1.	0m1.252s	0m0.936s	0m0.008s
2.	0m1.245s	0m0.924s	0m0.016s
3.	0m1.246s	0m0.912s	0m0.024s
4.	0m1.240s	0m0.704s	0m0.152s

Tabulka 5.1: Test a1 testující rychlost zpracování souboru bez ukládání dat, pouze porovnání na shodu, bez ukládání.

Po otestování daných regulárních výrazů jsme dospěli k výsledku, že nejrychlejší je hledání bez ukládání hodnot a nejhůře z toho vychází ukládání pod jménem, ačkoliv rozdíl mezi jménem a ukládáním bez jména již nebyl tak velký.

$$a_1 < a_2 < a_3$$

a2	real	user	system
1.	0m1.402s	0m1.052s	0m0.012s
2.	0m1.459s	0m0.972s	0m0.088s
3.	0m1.410s	0m1.100s	0m0.020s
4.	0m1.478s	0m1.104s	0m0.020s

Tabulka 5.2: Test a2 testující rychlost zpracování souboru s ukládáním dat, ukládá se pouze do číselně označeného pole.

a3	real	user	system
1.	0m1.449s	0m1.064s	0m0.004s
2.	0m1.500s	0m1.116s	0m0.004s
3.	0m1.466s	0m1.116s	0m0.008s
4.	0m1.505s	0m1.144s	0m0.008s

Tabulka 5.3: Test a3 testující rychlost zpracování souboru s ukládání dat, ukládá se do pojmenovaných proměnných.

Dále jsme nalezli očekávanou závislost, mezi počtem ukládaných členů a rychlostí, kde jsme nalezli markantní rozdíly. Tato výrazná závislost byla pozorována i při bezejmenném ukládání i při ukládání pod jménem.

$$b_2 < a_3 < b_1$$

$$c_1 < a_2$$

Závislost výrazně ovlivňovala rychlost zpracování souboru. Poněvadž při naší analýze se nachází mnoho ukládaných jednotek, tak naše analýza probíhá relativně pomalu. Možným vylepšením by bylo použití nepojmenovaného ukládání, ovšem takto ušetřený čas by nebyl natolik význačný.
Zjistili jsme, že analýza textu pomocí regulárních výrazů s jejím ukládáním do proměnných výrazně prodlužuje čas zpracování úměrně s počtem členů.

5.3 Testování parametrů a podmínek

5.3.1 Parametry programy ufre_gui.py

1. ***Test spustitelnosti programu ufre_gui.py***
   Vstup: Přidán adresář se soubory programu, do proměnné PATH, a provedlo se spouštění z jiného adresáře než se program nachází.
   Očekávaná hodnota: Standardní spuštění aplikace.
   Výstup: FAIL. Aplikace zhavarovala na nemožnosti najít soubor s glade popisem programu.
   Oprava: Upraveno aby se glade file našel podle cesty k programu.

2. ***Vstupní parametry příkazové řádky ufre_gui.py***
   Vstup: Libovolné další parametry.
   Očekávaná hodnota: Neovlivnění funkčnsoti programu.
   Výstup: OK.

b1	real	user	system
1.	0m1.650s	0m1.360s	0m0.036s
2.	0m1.690s	0m1.312s	0m0.012s
3.	0m1.667s	0m1.316s	0m0.016s
4.	0m1.664s	0m1.232s	0m0.024s

Tabulka 5.4: Test b1 testující rychlost zpracování souboru s ukládání dat, ukládá se do pojmenovaných proměnných. Testované ukládání do šesti oddělených proměnných

b2	real	user	system
1.	0m1.167s	0m0.968s	0m0.012s
2.	0m1.245s	0m0.952s	0m0.000s
3.	0m1.170s	0m0.936s	0m0.012s
4.	0m1.192s	0m0.924s	0m0.008s

Tabulka 5.5: Test b2 testující rychlost zpracování souboru s ukládání dat, ukládá se do pojmenovaných proměnných. Testované ukládání všeho do jedné proměnné

5.3.2 Parametry programy ufre_cli.py

1. ***Test spustitelnosti programu ufre_cli.py***
   Vstup: Přidán adresář se soubory programu, do proměnné PATH, a provedlo se spouštění z jiného adresáře než se program nachází.
   Očekávaná hodnota: Standardní spuštění aplikace, zobrazena manuálová stránka.
   Výstup: OK.

2. ***Parametry - zadany 1***
   Vstup: ufre_cli.py -i vsftpd.log -r 1.o: Soubor vsftpd.log existuje, soubor 1.o existuje a obsahuje zkonstruovaný regulární výraz.
   Očekávaná hodnota: Aplikace vypisuje analyzované řádky, výstup nezajímavý, nesmí skončit chybou.
   Výstup: OK.

3. ***Prametry - zadany 2***
   Vstup: ufre_cli.py -i vsftpd.log -r 1.o -a: Soubor vsftpd.log existuje, soubor 1.o existuje a obsahuje zkonstruovaný regulární výraz.
   Očekávaná hodnota: Výstup agregovaných dotazů, bez výpisu řádků, ve formátu funkce (jméno) = hodnota Nesmí skončit chybou.
   Výstup: OK.

4. ***Parametry - pořadí***
   Vstup: Parametry jako v předchozích dvou, akorát změněno pořadí parametrů.
   Očekávaná hodnota: Stejný výstup jako v předchozích příkladech.
   Výstup: OK.

5. ***Parametry - nápověda***
   Vstup: ufre_cli.py -h: zadán libovolý počet parametrů, ovšem jeden z nich je -h.
   Očekávaná hodnota: Vypíše se nápověda programu.
   Výstup: OK.

c1	real	user	system
1.	0m1.180s	0m0.944s	0m0.004s
2.	0m1.181s	0m0.940s	0m0.008s
3.	0m1.120s	0m0.936s	0m0.008s
4.	0m1.222s	0m0.932s	0m0.008s

Tabulka 5.6: Test c1 testující rychlost zpracování souboru s ukládání dat, ukládá se celý výraz jako nepojmenovaný celek, indexovatelný pouze číslem

6. ***Parametry - chyba 1***
   Vstup: ufre_cli.py -i vsftpd.log -r 1.o: neexistující soubor vsftpd.log, ostatní parametry nepodstatné.
   Očekávaná hodnota: Vypsání chyby a ukončení programu. Missing input file vsftpd.log.
   Výstup: FAIL. Nevypsáno jméno souboru.
   Oprava:

7. ***Parametry - chyba 2***
   Vstup: ufre_cli.py -i vsftpd.log -r re.in: neexistující re.in, parametr -i musí být zadán, jinak skončí s předchozí chybou.
   Očekávaná hodnota: Vypsání chyby a ukončení programu. Missing file of regular expression re.in.
   Výstup: OK.

8. ***Parametry - chyba 3***
   Vstup: Zadán parametr, který nepatří k parametrům programu (ostatní existující nejsou postatné) ufre_cli.py -i vsftpd.log -r re.in -d.
   Očekávaná hodnota: Výstup chyby option -d not recognized a vypsání nápovědy a ukončení programu.
   Výstup: OK.

9. ***Parametry - chyba 4***
   Vstup: ufre_cli.py -i vsftpd.log -r re.in: kde re.in neobsahuje výraz odpovídající zkonstruovanému a uloženému regulárnímu výrazu.
   Očekávaná hodnota: Cant read data from regular expression file.
   Výstup: OK.

5.3.3 Test grafického uživatelského rozhraní

1. ***Window 1 - otevření souboru***
   Vstup: Otevření existujícího souboru vsftpd.log, zmáčknuto tlačítko otevřít.
   Očekávaná hodnota: Soubor otevřen a přiřazeny řádky, tlačítko převést neaktivní, poněvadž není vybrán žádný řádek.
   Výstup: FAIL. Tlačítko aktivní, soubor otevřen správně.
   Oprava: Nastavení citlivosti, při výběru řádku, nikoliv při otevření souboru.

2. ***Window 1 - otevření souboru***
   Vstup: Zmáčknuto tlačítko zrušit.
   Očekávaná hodnota: Žádný soubor neotevřen, hlavní okno prázdné, tlačítko převést neaktivní.
   Výstup: OK.

3. ***Window 1 - zavření aplikace***
   Vstup: Stisknuta položka v menu file Ukončit.
   Očekávaná hodnota: Aplikace ukončena, včetně všech otevřených oken.
   Výstup: OK.

4. ***Window 1 - tlačítko převést***
   Vstup: Zmáčknuto tlačítko převést v hlavním okně.
   Očekávaná hodnota: Otevřeno Window 2 s rozděleným řádkem, který je vybraný, na sémantické části.
   Výstup: OK.

5. ***Window 1 - about dialog***
   Vstup: V menu help zvolena položka O aplikaci.
   Očekávaná hodnota: Otevřeno dialogové okno O aplikaci.
   Výstup: OK.

6. ***Window 1 - otevření souboru***
   Vstup: Jeden soubor již otevřen, otevření jiného souboru.
   Očekávaná hodnota: Načtení jiného souboru do okna, při nevybraném řádku neaktivní tlačítko pro převod.
   Výstup: FAIL. Soubor načten, ale tlačítko i přes vybraný řádek zůstalo aktivní.
   Oprava:

7. ***Window 2 - výběr řádku***
   Vstup: Nevybrán žádný řádek v sémantických jednotkách.
   Očekávaná hodnota: Tlačítko vlastnosti neaktivní, vše ostatní aktivní.
   Výstup: OK.

8. ***Window 2 - otevevření souboru***
   Vstup: Zmáčknuto tlačítko na otevření souboru a zvolen nový soubor.
   Očekávaná hodnota: Otevřen jiný soubor na analýzu, jeho jméno zobrazeno v levém dolním rohu okna.
   Výstup: OK.

9. ***Window 2 - uložení***
   Vstup: Zmáčknuto tlačítko uložit jako.
   Očekávaná hodnota: Uložen soubor regulárního výrazu, pokud soubor existuje, zobrazí se dialog jestli se má soubor přepsat.
   Výstup: OK.

10. ***Window 2 - o aplikaci***
    Vstup: Stisknuto tlačítko O aplikaci.
    Očekávaná hodnota: Zobrazen dialog O aplikaci.
    Výstup: OK.

11. ***Window 2 - změna jména***
    Vstup: Klepnuto na políčko ve sloupečku name, zadáno jméno obsahující symboly co nesmí například obsahuje mezeru 'jmeno x'.
    Očekávaná hodnota: Zobrazen dialog řetězec obsahuje nepovolené znaky, jméno nezměněno.
    Výstup: OK.

12. ***Window 2 - změna jména***
    Vstup: Klepnuto na políčko ve sloupečku name, zadáno jméno bez mezer a dalších speciálních symbolů řetězec "jmeno".
    Očekávaná hodnota: Jméno změněno a uloženo.
    Výstup: OK.

13. ***Window 2 - změna jména 2***
    Vstup: Klepnuto na políčko ve sloupečku name, zadáno jméno bez mezer a dalších speciálních symbolů řetězec "jmeno", toto jméno je stejné jako jiný prvek.
    Očekávaná hodnota: Jméno nelze změnit, poněvadž by došlo ke kolizi jmen v python regulárním výrazu.
    Výstup: FAIL. Jméno se změnit povedlo.
    Oprava: Přidána kontrola na shodu s jinými jmény. V případě kontroly zobrazeno dialogové okno a jméno nezměněno.

14. ***Window 2 - změna důležitosti***
    Vstup: Klepnuto na políčko ve sloupečku Importance, zvolen jiný typ důležitosti z výběru s následným klepnutím na jiný sloupec či položku pro změnu.
    Očekávaná hodnota: Typ důležitosti změněn.
    Výstup: OK.

15. ***Window 2 - hromadná změna důležitosti***
    Vstup: Zvolen 1 a více řádků (více náhodných výběrů) a vybrána důležitost z combo boxu pod tabulkou.
    Očekávaná hodnota: Změněna důležitost pro zvolené řádky.
    Výstup: OK.

16. ***Window 2 - hromadný výběr***
    Vstup: Zvoleny 2 a více řádků takulky.
    Očekávaná hodnota: Tlačítko Vlastnosti se deaktivuje, jinak je aktivní.
    Výstup: OK.

17. ***Window 2 - tlačítko vlastnosti***
    Vstup: Zmáčknuto tlačítko vlasnosti.
    Očekávaná hodnota: Otevřen dialog na změny podmínek rodičovské okno neaktivní do ukončení práce s dialogem.
    Výstup: OK.

18. ***Dialog podmínky - odstranění podmínek***
    Vstup: Stisknuto tlačítko odstranit u sady podmínek (OR) na odstranění pokud není zvolen žádný řádek.
    Očekávaná hodnota: Zobrazen dialog, že není vybrán řádek na odstranění.
    Výstup: OK.

19. ***Dialog podmínky - odstranění podmínky***
    Vstup: Stisknuto tlačítko odstranit u podmínky (AND) na odstranění pokud není zvolen žádný řádek.
    Očekávaná hodnota: Zobrazen dialog, že není vybrán řádek na odstranění.
    Výstup: OK.

20. ***Dialog podmínky - přidání sady podmínek***
    Vstup: Stisknuto tlačítko přidat na levé straně.
    Očekávaná hodnota: Přidána další sada podmínek, číslo o 1 vyšší než předchozí přidávaná.
    Výstup: OK.

21. ***Dialog podmínky - přidání podmínky***
    Vstup: Stisknuto tlačítko přidat na pravé straně okna, nevybrán typ podmínky ani hodnota.
    Očekávaná hodnota: Zobrazen dialog o nemožnosti vložit prázdnou podmínku.
    Výstup: OK.

22. ***Dialog podmínky - přidání podmínky 2***
    Vstup: U číselné hodnoty, či datumu pokus o vložení ve formátu který není vhodný.
    Očekávaná hodnota: Zobrazen dialog s tím, že daný formát nelze vložit, opraveno již ve fázi vývoje.
    Výstup: OK

23. ***Dialog podmínky - přidání podmínky***
    Vstup: Stisknuto tlačítko vložit, zvolen typ podmínky a zadán text ve správné formátu.
    Očekávaná hodnota: Podmínka vložena do tabulky v pravo.
    Výstup: OK

24. ***Dialog podmínky - Smazání podmínky***
    Vstup: Vybrán řádek v pravé tabulce a stisknuto tlačítko odstranit.
    Očekávaná hodnota: Zvolená podmínka odstraněna.
    Výstup: OK.

25. ***Dialog podmínky - Změna agregace***
    Vstup: Zvolena agregace v combo boxu pod napisem agregaion.
    Očekávaná hodnota: Typ zvolené agregace zobrazen pod výběrem.
    Výstup: OK.

26. ***Dialog podmínky - Zrušení agregace***
    Vstup: Zrušení agregace.
    Očekávaná hodnota: Agregace zrušena.
    Výstup: FAIL tato funkce nepodporována.
    Oprava: Přidána do menu volby agregace položka none znamenající její zrušení.

27. ***Dialog podmínky - Tlačítko potvrdit***
    Vstup: Tlačítko potvrdit.
    Očekávaná hodnota: Ukončí dialog, jinak bez efektu.
    Výstup: OK.

28. ***Window 2 - analyze***
    Vstup: Zmáčknuto tlačítko analyze.
    Očekávaná hodnota: Začne analýza dat, ve druhém threadu, Window 2 neaktivní mimo tlačítko Stop pro zastavení.
    Výstup: OK.

29. ***Window 2 - analyze***
    Vstup: Analýza v běhu, stisknuto tlačítko zastavit.
    Očekávaná hodnota: Thread zastaven, vypsány analyzované záznamy, které se stihly provést.
    Výstup: OK.

30. ***Dialog Output***
    Vstup: Analýza provedena.
    Očekávaná hodnota: Zobrazeno okno s analyzovanými záznamy.
    Výstup: OK.

5.3.4 Test funkcionality

1. ***Nastaveni unique pro dano jednotku***
   Vstup: Analyzován soubor access.log, a kontrolována kolik unikátních IP adres se vyskytlo v souboru.
   Očekávaná hodnota: 17 - stejného výsledku dosaženo i za pomoci složeného příkazu `cat access.log | awk "{print $1}"| sort | uniq | wc -l` .
   Výstup: OK.

2. ***Nastavení none pro danou jednotku***
   Vstup: Nastaveno none pro danou semantickou jednotku.
   Očekávaná hodnota: Nezobrazena ip adresa ve výstupu, počet řádků výstupu musí zůstat zachován, nepoužijí se omezující podmínky ani agregační dotazy ve výsledku.
   Výstup: OK.

3. ***Nastavení constant pro danou jednotku***
   Vstup: Nastaveno constant pro zobrazování IP adresy.
   Očekávaná hodnota: Vypsány pouze řádky obsahující danou IP adresu, žádné jiné se již ve výsledku neobjeví.
   Výstup: OK.

4. ***Zvolen typ dulezitosti constant u dlouheho retezce***
   Vstup: "Mozilla/5.0 (X11; U; Linux i686; en-US; rv1.8.1.4) Gecko/20070604 Firefox/2.0.0.4" nastaveno na constant.
   Očekávaná hodnota: Zustanou ve výsledku jen hodnoty ohsahující tento řetězec.
   Výstup: FAIL. Nenalezen ani jeden řádek.
   Oprava: Musela se přidat náhrada speciálních symbolů u konstatního výrazu pomoci substituce regulárním výrazem: `re.sub( r'([.*+?\\()])', r'\\\1', slovo)`.

5. ***Nastavení variable pro danou jednotku***
   Vstup: Variable nastaveno v daném řádku.
   Očekávaná hodnota: Neomezeny hodnoty, všechny se objeví ve výsledku.
   Výstup: OK.

6. ***Nastavení hidden pro danou jednotku***
   Vstup: Hidden nastaveno v daném řádku, použita podmínka ip adresa obsahuje 127.0.0.1 a jako agregační dotaz počet řádků.
   Očekávaná hodnota: Hodnoty neomezeny, neobjeví se ve výsledku, ale naproti none se budou počítat agregační dotazy a použijí se podmínky. Výsledek musí odpovídat

výsledku při použití variable. Výsledek 142 v obou případech.
Výstup: OK.

7. ***Porovnání výstupů podmínky contain, equal a constant***
Vstup: Soubor access.log dáno omezení na číselnou hodotu odpovědi, contain(200), podruhé to samé avšak s podmínkou $equal(200)$, tyto hodpoty porovnány mezi sebou a srovnány pokud se nastaví tato sémantická jednotka na constant.
Očekávaná hodnota: Počet řádků 1036 ve všech třech případech.
Výstup: OK.

8. ***Agregace - výstup***
Vstup: Spuštěn test, s nastavenou agregací count u jednoho ze sémantických celků, tak aby výstupen byl > 0, v druhé iteraci dána podmínka, tak aby do výsledku nepadl ani jeden řádek.
Očekávaná hodnota: Count $== 0$.
Výstup: FAIL. zobrazen předchozí výsledek.
Oprava:

9. ***Agregace - výstup, použití***
Vstup: Nastavena agregace u prvního prvku, a nastaven typ důležitosti none.
Očekávaná hodnota: Na výstupu se neobjeví tato agregace.
Výstup: OK.

10. ***Testování numerické podmínky 1***
Vstup: Test podmínek větší nebo rovno a menší nebo rovno, jako vstup použit soubor access.log. První část, definována podmínka $>= 500$ na navratovou hodnotu Apache serveru, druhý test nastaven na $<= 499$ a v třetí části zrušena podmínka. Nastaven agregace count, jako kontrola správnosti.
Očekávaná hodnota: Očekává se, že počet řádků bez podmínek se bude rovnat součtu řádků u obou podmínek, dále, jako vizuální kontrola, musí u všech řádků $>= 500$ hodnota $>= 500$ $(3580 = 13 + 3567)$.
Výstup: OK.

11. ***Agregace - count***
Vstup: Nastavena podmínka jako v předchozím testu na $>= 500$ a nastavena agregační funkce count.
Očekávaná hodnota: Výstupem bude 13 řádků a v agragačním výsledku bude $count(name) = 13$.
Výstup: OK.

12. ***Agregace - sum***
Vstup: Nastavena podmínka jako v předchozím testu na $>= 500$ a nastavena agregační funkce count, dále nastavena agregační funkce součtu u další hodnoty.
Očekávaná hodnota: Výstupem bude 13 řádků a v agragačním výsledku bude $sum(name1) = 8042$.
Výstup: OK.

13. ***Agregace - min***
Vstup: Nastavena podmínka jako v předchozím testu na $>= 500$ a nastavena agregační funkce count, dále nastavena agregační funkce minima u další hodnoty.
Očekávaná hodnota: Výstupem bude 13 řádků a v agragačním výsledku bude

$min(name1) = 617$.
Výstup: OK.

14. ***Agregace - max***
Vstup: Nastavena podmínka jako v předchozím testu na >= 500 a nastavena agregační funkce count, dále nastavena agregační funkce maxima u další hodnoty.
Očekávaná hodnota: Výstupem bude 13 řádků a v agragačním výsledku bude $max(name1) = 638$.
Výstup: OK.

15. ***Agregace - avg***
Vstup: Nastavena podmínka jako v předchozím testu na >= 500 a nastavena agregační funkce count, dále nastavena agregační funkce průměru u další hodnoty.
Očekávaná hodnota: Výstupem bude 13 řádků a v agragačním výsledku bude $avg(name1) = 618.615$.
Výstup: OK.

16. ***Správnost unique s podmínkou***
Vstup: Nastavena podmínka jako v předchozím testu na >= 500 a nastavena agregační funkce count.
Očekávaná hodnota: Výstupem bude u hodnoty count 2 a ve výsledku 2 řádky s odlišnou hodnotou druhého číselného parametru.
Výstup: OK.

17. ***Podminky - typ Date <=***
Vstup: Analyzovaný soubor, přidána podmínka menší než na zadané datum.
Očekávaná hodnota: Vypsány pouze řádky obsahující záznamy, menší než zvolené datum.
Výstup: OK.

18. ***Podminky - typ Date >=***
Vstup: Analyzovaný soubor, přidána podmínka větší než na zadané datum.
Očekávaná hodnota: Vypsány pouze řádky obsahující záznamy, větší než zvolené datum.
Výstup: OK.

19. ***Podminky - Contain***
Vstup: Zvoleny různé sémantické typy, u nich dána podmínka na contain, tato podmínka je testována jako porovnání na obsah podřetězce jako textu u všech typů, proto bude chování stejné. Testováno na souboru access.log a testována podmínka na IP adresu $contain("127")$.
Očekávaná hodnota: Vypsány pouze řádky vyhovujíví dané podmínce. Ve výsledku řádky obsahující hodnotu 127 $127.0.0.1, 89.56.127.50)$.
Výstup: OK.

20. ***Podmínka Equal***
Vstup: Testovány podmínky na equal. obdobne typu constant, ovšem na úrovni podmínek. Otestováno na typu v souboru vsftpd.log, dána podmínka "LOGIN FAIL".
Očekávaná hodnota: Na výstupu jen řádky shodující se s řetězcem LOGIN FAIL.
Výstup: OK.

21. ***Více podmínek AND***
    Vstup: Vloženy 3 podmínky contain(1) AND contain(9) AND contain(8) na IP adresu v souboru vsftpd.log u řádku connect.
    Očekávaná hodnota: Do výsledku zařazeny řádky vyhovující dané podmínce.
    Výstup: OK.

22. ***Více podmínek OR***
    Vstup: Vloženy 2 podmínky equal("scottyh") OR equal("jura") na jméno uživatele v souboru vsftpd.log.
    Očekávaná hodnota: Do výsledku zařazeny řádky vyhovující dané podmínce, takže zažazeny záznamy v kterých se vyskytuje jako uživatelské jméno buď "scottyh" nebo "jura".
    Výstup: OK.

5.3.5 Test nápovědy

1. ***Řádkový klient***
    Vstup: ufre_cli.py -h a ufre_cli.py –help.
    Očekávaná hodnota: Vypsána napověda.
    Výstup: OK.

2. ***Klient pro grafické uživalelské rozhraní***
    Vstup: Test všech prvků oken, na přítomnost tooltip nápovědy.
    Očekávaná hodnota: U každého prvku vypsána nápověda jak jej použít, či k čemu slouží.
    Výstup: OK.

Kapitola 6

Rozšíření stávajícího systému

Při vývoji aplikaci jsme narazili na několik cest vhodných pro rozšíření aplikace. Můžeme aplikaci rozšířit o hlubší dotazy, které uživatel může provádět, či změnit pozadí analýzy. Dále by mohl být směr vývoje grafická konstrukce uživatelsky přívětivých výrazů, aby si je uživatel mohl modifikovat a podrobněji s nimi pracovat. Nabízí se i jednoduchá rozšíření, mající základ v práci rozšíření zadávání podmínek (např. přidání operátoru negace).

6.1 Změna pozadí aplikace a hlubší dotazy

Mohli bychom změnit pozadí analyzovaného systému. Jedna z cest vývoje by mohla být nahradit třídu analyzátoru a převést funkčnost dotazů na databázový server v pozadí. O tomto rozšíření jsme uvažovali již při návrhu současné aplikace. Návrh jsme zavrhli kvůli problémům s optimalitou dotazů. Museli bychom změnit pozadí zpracování souborů a ukládání výrazů. Problém nastává v tom, že by při každém dotazu musela být znovu naplněna databáze nad nějakým typem dočasné tabulky a nad ní provedeny operace. Samotné naplnění by bylo pomalejší než zpracování, jak je prováděno v této verzi. Pro nápravu této vlastnosti bychom museli vytvořit službu, která by dynamicky plnila databázi na pozadí při změně souboru. Tím by se usnadnily dotazy a mohlo by se dosáhnout velké rychlosti zpracování dotazů. Dále by to vedlo k rozšíření možností dotazů. Nebyli bychom omezeni jednoduchými výrazy, které používáme nyní, ale mohli bychom použít celý aparát `SQL` dotazů. Umožnilo by to dotazy s klauzulemi `GROUP BY`, které by nám značně ulehčily situaci u dotazů, které bychom chtěli provádět např. spočítat velikost všech přenesených dat pro všechny uživatele v systému.
Toto rozšíření je značně problematické a zavrhli jsme jeho použití v nynější implementaci, kvůli mnoha problémům, které bychom museli překonat. Pak nejen vyjmenované v předchozím textu, ale i dalších. Velkým problémem by bylo vytváření tabulek, poněvadž řádky mohou obsahovat různé sémantické jednotky, tak by stejně musel při mnoha dotazech vytvářet další pomocné tabulky. Nešlo by uložit jeden soubor do jedné tabulky se kterou by následně pracovaly všechny dotazy nad daným souborem.

Úprava by skýtala mnoho pozitivních rozšíření, které bychom si přáli, ovšem přináší mnoho problémů s prací v databázi, které by nešly jednoduchou cestou obejít. Pro každý dotaz by patrně musela existovat samostatná tabulka, a program běžící na pozadí starající se o její plnění za běhu.

6.2 Grafická konstrukce výrazů

Uživatel by měl mít možnost zasahovat do tvorby uživatelsky přívětivých regulárních výrazů a upravovat jejich složení a vytvářet uživatelské typy podle vlastních požadavků.
O tomto rozšíření jsme rovněž uvažovali v nynější verzi. Aby uživatel měl možnost vytvářet si vlastní uživatelsky definované regulární výrazy. Vše je založeno na tom, že by uživatel mohl složit ze základních jednotek vlastní výraz. Mohl by je označit a prohlásit o nich, že se jedná o uživatelský typ. Pro tvorbu je vytvořen aparát, který by to mohl umožňovat, První z nich je rozdělení na dále nedělitelné shluky písmen, číslic a oddělovacích znaků. Uživatel by se proto nezabýval jednotlivými znaky, ale až vyššími celky. Dále je navrženo označování použitých a nepoužitých jednotek v třídě uživatelského regulárního výrazu, což by nám spojení umožnilo.
Existuje několik věcí proti rozšíření. Nejjednodušší by bylo, aby spojování a rozpojování bylo možné pouze u sekvencí, to znamená bez opakování. Tím by nemohlo dojít ke kolizím. Problém by mohl nastat v případě, že by uživatel mohl rozpojit sekvenci v které probíhá opakování (např. Cesta v souborovém systému může obsahovat libovolný shluk znaků oddělených znakem lomítka). Uživatel by mohl způsobit to, že nedojde ke shodě na žádném řádku, respektive pouze na daném řádku. Takže se naskýtá otázka, jak tuto práci udělat natolik automaticky a zároveň umožnit uživateli shlukovat a rozdělovat výrazy. Řešením by mohlo být nepracovat na nejnižší úrovni, ale umožnit uživateli pracovat až nad sémantickými celky, které by mohl spojovat do sekvencí po sobě jsoucích výrazů pod jedním jménem, tím by se nezměnila sémantika výsledného regulárního výrazu a uživatel by mohl vytvářet větší sémantické celky.

Po zvážení možností jsme proto danou funkčnost neimplementovali do nynějšího verze programu, ale nechali cestu k danému rozšíření otevřenou. Rozšíření uživateli poskytne shlukování více jednotek do celku, podle uvážení uživatele.

6.3 Lokalizace souborů

Při zpracování textu v souborech nastal někdy problém s lokalizací znaků. Hlavní problém nastával při převodu data z textové reprezentace. Problém byl vyřešen nastavením systémové proměnné `LC_TIME = "C"`. Rozšířením by bylo zbavit se nastavení v programu a umožnit buď nabídnout možnost uživateli na změnu kódování, či se pokusit o automatickou detekci. Systémové záznamy používají implicitně kódování nastavené na `C`, proto jsme si mohli dovolit ho nastavit napevno. Rozšířením by bylo umožnit autodetekci jazyka, např. podle názvů dnů v týdnu. V době práce na diplomovém projektu byla objevena chyba jazyka Python, která způsobovala, že lokalizace mohla být programově změněna pouze jednou. Při druhé změně došlo k navrácení na systémovou hodnotu, což způsobovalo chyby zpracování datumových položek. Proto byl problém vyřešen nastavením na hodnotu při spuštění.

Rozšíření o lokalizaci by byla vhodná. Ovšem zatím není zcela možná kvůli problémům jazyka python. Lokalizace by se mohla týkat i zpracování textu, zatím není nijak řešena a vše závisí na nastavení v systému.

Kapitola 7

Závěr

Námi navržený systém by měl být schopen analyzovat různé formy záznamů. Po analýze stávajících systémů jsme došli k názoru, že neexistují nástroje pracující na úrovni hledání obecnějších statistik systému a rozsáhlejší analýze záznamů. Proto stávající systémy rozšíříme o tuto možnost a dáme systému možnost pracovat na úrovni obecné analýzy, která již není konkrétně svázána s IDS systémy. Přínosem této práce je navržení uživatelsky přívětivých regulárních výrazů, které umožní jednodušší dotazy nad záznamy a vytvoření systému, který tyto výrazy bude využívat a poskytne funkce na zpracování. V rámci semestrálního projektu byly vyřešeny první dva body zadání. Zkoumal se současný stav, IDS systémy a analýza záznamů. Dále byl vytvořen návrh systému dle bodu 2. Při diplomovém projektu byly dořešeny zbylé části zadání, implementace navrženého modelu, testování aplikace a navržené rozšíření systému.

Při studiu problematiky byly nalezeny různé způsoby, jak k analýze přistupovat. Každý z těchto systémů měl své výhody i nevýhody. Většinou v reálných systémech byly použity jednodušší varianty zkoumání souborů. Systémy reálně nasazené obsahují totiž velké množství informací a složitější systémy pak zbytečně zatěžují server. Námi navržený systém je dostatečně obecný na libovolnou analýzu, která bude prováděna. Naimplementovali jsme systém pro práci s uživatelsky přívětivými regulárními výrazy. Systém umožňuje zvolení řádku v souboru a práci se sémantickými jednotkami. U těch umožní nastavit podmínky, agregační funkce a důležitosti. Vytvořený systém jsme testovali na funkčnost, zda-li neobsahuje chyby. Vytvořili jsme řádkového klienta pro dávkové zpracování systémových záznamů, který je použitelný ve spojení s programem spouštějící programy v zadaný čas. Tím nám umožní opakované dotazy, které můžeme využít ke statistickým účelům a vytvářet na základě výsledků grafy.

Možná rozšíření aplikace jsou naznačeny v předchozí kapitole a můžeme je shrnout do zlepšení práce na uživatelských dotazech. Které bychom měli rozšířit, tak aby tvořila ucelenější množinu, či směrem možnosti uživatelské definice sémantických jednotek. Diskutovali jsme v předchozí kapitole problémy, které s tím vyvstávaly a jaká by byla možná východiska z problémů.
Při práci na diplomovém projektu jsme se ponořili do sémantických významů regulárních výrazů a poučili se o problémech které s tím pramení, včetně problémů s jednoznačností výrazů. Testovali jsme rychlost zpracování regulárních výrazů, při které jsme zjistili, že pro rychlost je velmi význačné ukládání výrazů, čím více se ukládá podřetězců regulárních výrazů. dochází k rapidnímu zpomalování, kterému nelze zabránit ani ukládáním do nepojmenovaných polí. Dále jsme se poučili, že i tak vyspělý jazyk jako je Python může

obsahovat chyby, na které jsme narazili, konkrétněji problém s nastavováním locale. Vyzkoušeli jsme si práci na grafickém uživatelském rozhraní knihovny GTK, přes knihovnu pro python PyGTK. Poznali jsme výhody a úskalí dynamického nahrávání vzhledu za pomocí knihovny glade, patřící projektu GTK, a úskalí událostmi ovládané aplikace.

Práce má velký potenciál být užitečnou široké veřejnosti, a to právě díky jednoduššímu zkoumání souborů se záznamy, které si takto bude moci zanalyzovat i neznalý člověk a v případě potřeby na výsledky provedené analýzy reagovat.

Literatura

[1] WWW stránky, Wikipedia HIDS. http://en.wikipedia.org/wiki/Host-based_intrusion_detection_system (aktuálnost ke dni 28. 12. 2007).

[2] WWW stránky, Wikipedia IDS. http://en.wikipedia.org/wiki/Intrusion_detection_system (aktuálnost ke dni 28. 12. 2007).

[3] WWW stránky, Projekt Analog. http://www.analog.cx/ (aktuálnost ke dni 31. 12. 2007), 2007.

[4] WWW stránky, Projekt AWstat. http://awstats.sourceforge.net/ (aktuálnost ke dni 31. 12. 2007), 2007.

[5] Bace, R.; Mell, P.: *NIST Special Publication on Intrusion Detection Systems.* National Institute of Standards and Technology, 2001, http://csrc.nist.gov/publications/nistpubs/800-31/sp800-31.pdf (aktuálnost ke dni 29. 12. 2007).

[6] Cid, D. B.: *Analysis using OSSEC.* 2007, ausCERT.

[7] Haring, D.: Analýza systémových logů: Logcheck. 2. prosince 2000, http://www.linux.cz/noviny/2001-02/clanek03.html (aktuálnost ke dni 29. 12. 2007).

[8] Häring, D.: *Kontrola integrity systému - Tripwire.* 3. března 2001, http://www.linux.cz/noviny/2001-04/clanek04.html.

[9] Noris, I.: Nástroje pre sledovanie systému. 6. dubna 2007, http://deja-vix.sk/sysadmin/tools.html (aktuálnost ke dni 30. 12. 2007).

[10] Petr, H.: Bezpečnost informačních systémů. Slidy k předmětu, 2007.

[11] Sundaram, A.: An introduction to intrusion detection. *Crossroads*, ročník 2, č. 4, 1996: s. 3–7, doi:http://doi.acm.org/10.1145/332159.332161.

[12] Tripwire Inc.: *Tripwire Enterprise Overview.* 2007.

[13] Wang, X.; Yu, H.: How to Break MD5 and Other Hash Functions. In EUROCRYPT, 2005.

Dodatek A

Obsah CD disku

Součástí práce jsme vytvořili program. Program je implementován v jazyce python za použití knihovny GTK pro zpracování grafického rozhraní. Na CD disku který je součástí práce se nachází zdrojové texty programu, a zdrojové texty diplomové práce, včetně vytvořeného balíku vhodného na instalaci.

A.1 Diplomová práce – text

Zdrojové texty diplomové práce včetně vysázeného dokumentu se nachází v adresáři `/dp-texts` na CD disku. Adresář obsahuje tyto důležité soubory:

dp.pdf – Vysázený text diplomové práce v elektronické podobě ve formátu PDF.

dp.tex – Zdrojový soubor diplomové práce pro sázecí systém LaTeX.

testy.tex – Zdrojový soubor pro sázecí systém LaTeXobsahující testy aplikace, který je vložen do dp.tex.

dp.bib – Zdrojový soubor bibliografických údajů k diplomové práci.

Makefile – Soubor pro program `make` starající se o sestavení textů diplomové práce.

A.2 Diplomová práce – program

Adresář `/prog` na CD disku obsahuje soubory potřebné ke spuštění aplikace. Poněvadž je aplikace psána v jazyce Python, tak jsou soubory ke spuštění zároveň i zdrojovými texty aplikace.
Seznam souborů:

README – Soubor popisující zkráceně aplikaci.

INSTALL – Popis instalace programu do systému.

LICENSE – Licence programu pod kterou jsou zdrojové kódy programu šířeny.

ufre_gui.py – Hlavní soubor pro spouštění grafického uživatelského rozhraní.

ufre_cli.py – Klient pro příkazovou řádku.

File.py – Práce se souborem.

Row.py – Zpracování řádku z třídy file.

Word.py – Rodičovská třída pro shluky.

AlphaWord.py – Shluk znaků.

BlankWord.py – Shluk bílých znaků.

DigitWord.py – Shluk číslicových znaků.

DotWord.py – Shluk teček.

SlashWord.py – Shluk znaků lomítka.

QuoteWord.py – Shluk znaků uvozovek.

InterWord.py – Shluk zbylých mezislovních oddělovačů.

InterWords.py – Obalení shluku oddělovačů.

Parser.py – Parsování řádku na shluky.

REsemantic.py – Rodičovská třída sémantických jednotek.

Phrase.py – Sémantická jednotka v uvozovkách.

Dat.py – Sémantická jednotka reprezentující datum.

IP.py – Sémantická jednotka reprezentující IP adresu.

Size.py – Sémantická jednotka udávající velikost.

URL.py – Sémantická jednotka pro URL.

Path.py – Sémantická jednotka pro adresářovou cestu.

Value.py – Sémantická jednotka pro obecné číslo.

Zbytek.py – Sémantická jednotka zbytku, který není dál specifikován.

UserFriendlyRE.py – Uživatelsky přívětivý regulární výraz a jeho zpracování.

Query.py – Třída podmínek.

Podminka.py – Jedna podmínka přidávaná do třídy Query.

Analyser.py – Analyzátor textu pomocí regulárních výrazů.

gladeiface/inface.glade – Soubor popisu grafického uživatelského rozhraní za použití GTK/Glade.

Dodatek B

Uživatelská příručka

Program na analýzu systémových záznamů je navržen na ulehčení práce systémových administrátorům a uživatelům kteří mají zájem, zkoumat záznamy spuštěných služeb a chtějí se vyhnout psaní složitějších regulárních výrazů, či používání jiných nástrojů na analýzu textu. Poskytuje jednoduché grafické rozhraní pro přidávání omezení na analyzované řádky. umožňuje interaktivní analýzu a dávkové zpracování klientem pro příkazový interpret. Vytváří sémantické (významové) celky, tak aby oddělil uživatele od samostatných shluků písmen, číslic a dalších znaků.
Program je inspirován existujícími IDS systémy a jejich schopnostmi analyzovat systémové záznamy pomocí regulárních výrazů.

Možnosti:

- Interaktivní práce v grafickém rozhraní.
- Dávkové zpracování řádkových klientem.
- Automatické vytváření sémantických jednotek.
- Přidávání omezení na analyzovaný řádek.
- Odebírání omezení.
- Práce s agregačními funkcemi (součty, průměry, minima, maxima, počty) u vhodných sémantických jednotek.
- Poskytuje výstup všech vyhovujících Rádku a výstup agregačních funkcí.
- Ukládání dotazů do souborů pro další zpracování.

Grafické prostředí umožní zkonstruovat regulární výraz vyhovující vybranému řádku. Který může být přímo v grafickém prostředí analyzován, či může být uložen a využit později. Aplikace v této verzi neposkytuje ještě plné možnosti analýzy, které by byli vhodné a budou rozšířeny v dalších verzích aplikace.

B.1 Návod na instalaci

Instalace nevyžaduje žádná omezení na uživatelská práva. Program může nainstalovat libovolný uživatel mající práva spouštět programy.

Před instalací je potřeba zkontrolovat zda se na systému nachází potřebné moduly pro jazyk python.
Požadavky:

- Python – interpret jazyka verze 2.4.4 a vyšší
- PyGTK – knihovna pro práci s grafických subsystémem GTK a knihovnou Glade pro dynamické nahrávání vzhledu aplikace. Minimální verze je 2.6, nižší verze nelze použít pro neobsahující prvky grafického rozhraní.

B.1.1 Instalace pro všechny uživatele systému

1. Rozbalte archív `prog.tar.gz` do libovolného adresáře:

   ```
   $ tar xzvf prog.tar.gz
   $ OUTPUT_PATH ="/opt/ufre"
   $ mv prog $OUTPUT_PATH
   ```

   Tím jsme vytvořili adresář s programem `/opt/ufre`

2. Vytvořte symbolické linky na spouštěcí soubory do standardních cest systému:

   ```
   $ ln -sf $OUTPUT_PATH/ufre_gui.py /usr/bin/ufre_gui
   $ ln -sf $OUTPUT_PATH/ufre_cli.py /usr/bin/ufre_cli
   ```

Po předchozích krocích by měl být program okamžitě připravený ke spuštění vykonáním příkazů: `ufre_gui, ufre_cli`

B.1.2 Instalace pro uživatele

1. Rozbalte archív `prog.tar.gz` do libovolného adresáře:

   ```
   $ tar xzvf prog.tar.gz
   $ OUTPUT_PATH ="~/ufre"
   $ mv prog $OUTPUT_PATH
   ```

   Tím jsme vytvořili adresář s programem `/home/adresář_uživatele/ufre`

2. Vytvoříme alias pro daný příkazový interpret, v případě příkazového interpretu `bash`, upravíme startovací soubor `.bashrc` v domovském adresáři a přidáme následující řádky:

   ```
   $ alias ufre_gui="~/ufre/ufre_gui.py"
   $ alias ufre_cli="~/ufre/ufre_cli.py"
   ```

Program bude spouštěn stejnými příkazy jako v předchozí verzi: `ufre_gui, ufre_cli`, ale musí být znovu načten příkazový interpret, jinak se změna startovacího skriptu neprojeví.

B.2 Příklady použití

Uvedeme si 2 příklady, pro grafické prostředí a pro řádkového klienta

B.2.1 Grafický klient

Pokusíme se ukázat, jak nastavit program tak aby ukázal unikátní IP adresy v souboru `vsftpd.log`. Vybereme záznam s řádkem který chceme použít. Ktomu se nejvíce hodí řádek obsahující řetězec CONNECT udávající pokus o připojení:

0	Mon Aug 28 23:18:04 2006 [pid 15347] CONNECT: Client "192.168.50.23"

Nejdůležitější je nastavit hodnotu *Unique* u IP adresy. U žádné další jednotky nejsou nastaveny žádné omezující podmínky.

Date	name2	none	Mon Aug 28 23:18:04 2006
Other	name7	none	
Other	name4	none	[
Other	name8	none	pid
Value	name3	none	15347
Other	name5	none	]
Other	name9	none	CONNECT
Other	name6	none	:
Other	name10	none	Client
Phrase	name1	unique	"192.168.50.23"

Výstup programu, při stisknutí tlačítka pro analýzu.

```
"192.168.50.23"
"169.254.68.6"
"169.254.68.10"
"192.168.1.83"
"192.168.50.201"
"192.168.50.50"
"192.168.50.1"
"89.29.26.68"
"195.113.168.254"
"89.96.28.220"
"10.26.24.6"
"192.168.2.132"
"212.96.178.82"
```

Výsledek byl očekávaný v této podobě, a zobrazuje unikátní IP adresy ze kterých se připojovalo na daný FTP server.

B.2.2 Řádkový klient

Pomocí grafického rozhraní vytvoříme dotaz, který uložíme do určeného souboru, např. `quest.inp`. Spustíme program s následujícímu parametry, vsftpd.log je systémový záznam FTP serveru, a quest.inp soubor s uloženým dotazem.

```
$ ufre_cli -i vsftpd.log -r quest.inp
"192.168.50.23"
"169.254.68.6"
"169.254.68.10"
"192.168.1.83"
"192.168.50.201"
"192.168.50.50"
"192.168.50.1"
"89.29.26.68"
"195.113.168.254"
"89.96.28.220"
"10.26.24.6"
"192.168.2.132"
"212.96.178.82"
```

Tím jsme zobrazili unikátní IP adresy z kterých bylo k serveru přistupováno.// Dále můžeme ještě spustit výstup nastavených agregačních funkcí:

```
$ ufre_cli -i vsftpd.log -r quest.inp -a
count(name1) => 13
```

Takže jsme zjistili, že vyhovujících řádků je 13, k FTP serveru bylo přistupováno z 13 unikátních IP adres.

B.3 Parametry programu

B.3.1 Řádkový klient

Klient pro příkazový řádek přijímá několik parametrů, které ovlivňují její funkce. Základní syntaxe je následující

```
ufre_cli <-i input_file> <-r regular_expression_file> [-a] [-h]
```

Seznam argumentů:

-i --input=input_file Jméno vstupního souboru určeného pro analýzu programem. Soubor systémového záznamu.

-r --regexpfile=regular_expression_file Jméno souboru s regulárním výrazem, soubor je kontrolován zda-li se jedná o správný formát regulárního výrazu.

-a Přepíná výstup z výpisu vyhovujících řádku na výstup agregačních funkcí.

-h Zobrazí nápovědu programu

B.3.2 Grafické rozhraní

Grafické rozhraní neposkytuje zpracování příkazové řádky a všechny parametry a volby jsou zadávány přes grafické rozhraní. Grafické rozhraní je určeno pro interaktivní analýzu. Hlavní okno analyzátoru nabízí změnu jména a důležitosti. Jméno má vliv na výstup agregačních funkcí, kde se jméno zobrazuje. Důležitost nám dává možnost upravovat podmínky jak se řádek bude zobrazovat.

Změna důležitosti:

Variable – Obecná hodnota bez jakých koliv omezení, a bude ve výsledku zobrazena.

Constant – Hodnota která je omezena na hodnotu danou tímto výrazem. žádná jiná se ve výsledku nevyskytne

None – Daný text je vyloučen ze zobrazení ve výsledku, nejsou brány v potaz omezení a agregační funkce.

Hidden – Podobné jako Variable s tím rozdílem, že sémantická jednotka nebude zobrazena ve výsledku.

Unique – Unikátní hodnota, ve výsledku se zobrazí pouze první nalezená která vyhovuje všem ostatním podmínkám.

Přidávání podmínek a agregačních funkcí:
Pro každý zvolený řádek můžeme přidávat omezení na zvolený řádek. Omezení určují jestli daný řádek bude zvolen do výsledku, či nikoliv. Omezení pro danou sémantickou jednotku jsou vytvářeny v disjunktivní normální formě. Máme několik typů podmínek, které jsou závislé na tom o jaký se jedná typ.

Contain – Testuje ve výrazu obsah podřetězce. Výsledek je nezávislý na typu a u všech se provádí pouze textové porovnání, bez jakýchkoliv úprav.

Equal – Testuje se shodnost na daný výraz, výraz je převáděn do standardní formy pro daný typ, např. datum je převáděno do ISO formátu data.

>= – Testuje zda-li je daný výraz větší nebo roven podmínce, Testování je omezeno pouze na Číselné typy a datum.

<= – Testuje zda-li je daný výraz menší nebo roven podmínce, Testování je omezeno pouze na Číselné typy a datum.

Agregační funkce:
V programu existuje několik agregačních funkcí, základní je count, vypisující vyhovujících řádků pro daný výraz. Ostatní funkce jsou určeny pouze pro číselné typu.

Count – Vypíše počet vyhovujících řádku

Sum – Součet všech vyhovujících hodnot

Avg – Průměrná hodnota

Min – Minimum z vyhovujících hodnot

Max – Maximum z vyhovujících hodnot

Dodatek C

Případy aplikací v praxi

Uvedeme několik příkladů jak aplikaci použít. Vzorové příklady je možné reprodukovat na stejných log souborech. se stejnými parametry tak jak jsou nastaveny.

C.1 Unikátní IP adresy

Cílem testu je vypsat pouze unikátní IP adresy ze souboru vsftpd.log FTP serveru `vsftp`. Pro tuto činnost musíme zvolit řádek s řetězcem CONNECT značící začátek připojení na server FTP.

Zvolený řádek v souboru:

0	Mon Aug 28 23:18:04 2006 [pid 15347] CONNECT: Client "192.168.50.23"

Nastavení důležitostí u jednotek: nejdůležitější je nastavit hodnotu *Unique* u IP adresy. V tomto souboru není sémantická jednotka určena jako IP adresa z toho důvodu, že IP adresa je uzavřena v uvozovkách, proto je učeno jako sémantický typ *Phrase*. U žádné jednotky nejsou nastaveny žádné omezující podmínky.

Date	name2	none	Mon Aug 28 23:18:04 2006
Other	name7	none	
Other	name4	none	[
Other	name8	none	pid
Value	name3	none	15347
Other	name5	none	]
Other	name9	none	CONNECT
Other	name6	none	:
Other	name10	none	Client
Phrase	name1	unique	"192.168.50.23"

Výstup programu, při stisknutí tlačítka pro analýzu:

```
"192.168.50.23"
"169.254.68.6"
"169.254.68.10"
"192.168.1.83"
"192.168.50.201"
"192.168.50.50"
"192.168.50.1"
"89.29.26.68"
"195.113.168.254"
"89.96.28.220"
"10.26.24.6"
"192.168.2.132"
"212.96.178.82"
```

Výsledek byl očekávaný v této podobě, a zobrazuje unikátní IP adresy z kterých se připojovalo na daný FTP server.

C.2 Špatné přihlášení uživatele na FTP server

Úkol který jsme si stanovili je nalézt uživatele, kteří se pokusili neúspěšně přihlásit na FTP server. Výstupem bude seznam unikátních uživatelů a jejich IP adres. Z výsledku pak můžeme usuzovat na události proč k těmto událostem nastalo.

Zvolený řádek v souboru: Jako vstupní byl zvolen řádek který indikuje úspěšné či neúspěšné přihlášení. (LOGIN OK, LOGIN FAIL)

```
1    Mon Aug 28 23:18:16 2006 [pid 15346] [scottyh] OK LOGIN: Client "192.168.50.23"
```

Celkový přehled nastavení všech jednotek:

Type	Name	Importance	Expression	Podminky	Agregace
Date	name2	none	Mon Aug 28 23:18:16 2006		
Other	name9	none			
Other	name4	none	[		
Other	name10	none	pid		
Value	name3	none	15346		
Other	name5	none	]		
Other	name11	none			
Other	name6	none	[		
Other	name12	unique	scottyh		count
Other	name7	none	]		
Other	name13	variable	OK LOGIN	{contain(FAIL) }	
Other	name8	variable	:		
Other	name14	none	Client		
Phrase	name1	variable	"192.168.50.23"		

Nastavení omezení a agregace u jednotky: Nastavili jsme jedinou podmínku aby daná sémantická jednotka obsahovala podřetězec FAIL, který udává nesprávné přihlášení.

Query no.	Type	Value
3	contain	FAIL

Výstup řádků vyhovující podmínce:

```
Output
jura FAIL LOGIN:"169.254.68.6"
x FAIL LOGIN:"192.168.50.1"
zycvbnm FAIL LOGIN:"192.168.50.1"
ondra FAIL LOGIN:"195.113.168.254"
```

Výstup agregační funkce navracející počet řádků:

```
Agregation
>> total count: 31795
count(name12) => 4
```

Ve výsledku vidíme, dle vlastního nastavení dokonce zaznamenané jedno heslo zadané omylem jako uživatelské jméno (zxcvbnm). Další byli neúspěšně zadaná hesla. Dle výstupu se můžeme zachovat, jak postupovat dále, můžeme např. zablokovat dané IP adresy, tak aby se nemohlo opakovat neúspěšné přihlášení z určených IP adres.

C.3 Počet přenesených byte pro konkrétního uživatele

Zadáním testu je zjistit počet přenesených byte konkrétního uživatele v souboru FTP serveru. Jako uživatel byl zvolme `scottyh`. Nastavili jsme i další agregační funkci na zjištění průměrné přenosové rychlosti. **Zvolený řádek v souboru:** Byl zvolen řádek se záznamem obsahující řetězec DOWNLOAD

```
6307    Thu Oct 12 21:10:06 2006 [pid 15456] [scottyh] OK DOWNLOAD: Client "192.168.50.201", "/a/texts/factbook/docs/app-f-s.ht
```

Celkový přehled nastavení všech jednotek: Nastavili jsme jméno na konstantní hodnotu, aby se počítala velikost pouze u zadaného uživatele. Nastavili jsme agregační funkce u velikostí souborů na součet, a u přenosové rychlosti na průměrnou rychlost.

Type	Name	Importance	Expression	Podminky	Agregac
Other	name16	none	pid		
Value	name6	none	15456		
Other	name8	none	]		
Other	name17	none			
Other	name9	none	[		
Other	name18	constant	scottyh		
Other	name10	none	]		
Other	name19	none	OK DOWNLOAD		
Other	name11	none	:		
Other	name20	none	Client		
Phrase	name1	none	"192.168.50.201"		
Other	name12	none	,		
Other	name21	none			
Phrase	name2	variable	"/a/texts/factbook/docs/app-f-s.html"		
Other	name13	none	,		
Other	name22	none			
Size	name4	variable	74925 bytes		sum
Other	name14	none	,		
Other	name23	none			
Size	name5	variable	9073.53Kbyte/sec		avg

Výstup agregační funkce navracející počet řádků:

```
Agregation
>> total count: 31795
avg(name5) => 4335210.80727
sum(name4) => 21737666153.0
```

Celková velikost přenesených byte uživatele se pohybuje okolo $22GB$ a průměrná rychlost stahování souborů $4,3MB/s$.

C.4 Počet stahovaných souborů prohlížečem Mozilla-Firefox

Pro různé výzkumy je potřeba zjistit počet různých druhů prohlížečů které přistupovali na daný webový server. Pro testování byl použit systémový záznam serveru apache `access.log` a hledali jsme záznamy obsahující podřetězec Firefox.

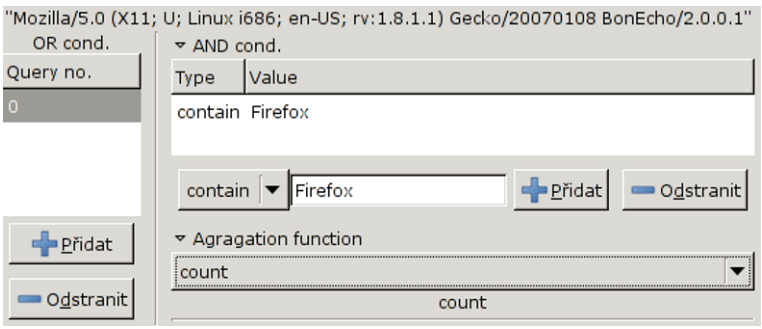
Zvolený řádek v souboru:

```
3    127.0.0.1 - - [12/Feb/2007:21:44:31 +0100] "GET /fotky HTTP/1.1" 301 306 "-" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.1) Gecko/
```

Celkový přehled nastavení všech jednotek: Není důležité které jednotky jsou nastaveny pro zobrazení, poněvadž nás ve výsledku zajímá pouze počet řádků které vyhovují. Ovšem pro ověření správnosti a kompletní přehled je dobré si zobrazit IP adresy a identifikační řetězce.

Type	Name	Importance	Expression
IP adress	name5	variable	127.0.0.1
Other	name12	variable	
Other	name8	none	-
Other	name13	variable	
Other	name9	none	-
Other	name14	none	
Other	name10	none	[
Date	name4	none	12/Feb/2007:21:44:31 +0100
Other	name11	none	]
Other	name15	none	
Phrase	name1	none	"GET /fotky HTTP/1.1"
Other	name16	none	
Value	name6	none	301
Other	name17	none	
Value	name7	none	306
Other	name18	none	
Phrase	name2	none	"-"
Other	name19	none	
Phrase	name3	variable	"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.1) Gecko/20070

Nastavení podmínky:



Výstup agregační funkce navracející počet řádků:

```
Agregation
>> total count: 6618
count(name3) => 891
```

Zjistili jsme počet přístupů na webový server z webového prohlížeče Mozilla/Firefox, výsledek je 891 z 6618 záznamů. Změnou podmínek může být testován libovolný prohlížeč. Vytvořením testů pro jednotlivé prohlížeče, může být vytvořen statistický systém pro porovnání vývoje prohlížečů na trhu.