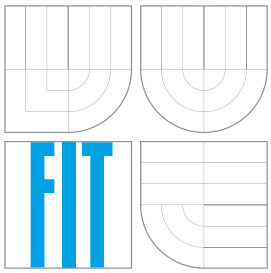
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

AUTOMATICKÉ KONFIGUROVÁNÍ SLUŽEB OPERAČNÍHO SYSTÉMU

AN AUTOMATIC CONFIGURATION OF SERVICES OF OPERATING SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. PETER SCHIFFER
AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEŠ SMRČKA, Ph.D.
SUPERVISOR

BRNO 2013

Abstrakt

Tato diplomová práce se zabývá konfigurací operačních systémů, jejich možnostmi a způsoby nastavení. Představuje rozdíly v konfiguraci operačních systémů podle jejich zaměření a pokročilé možnosti konfigurace operačních systémů pomocí aplikací třetích stran. Praktická část diplomové práce se zabývá návrhem nového počítačového jazyka zaměřeného na popis konfigurace operačního systému a jeho služeb. Tento popis konfigurace sloužící na automatickou konfiguraci systémových služeb se překládá na sekvenci konfiguračních příkazů. Výhoda jazyka spočívá v jeho dobré čitelnosti pro člověka, ale podobnost s přirozeným jazykem přináší určitou úroveň nejednoznačnosti. Navrhovaná metoda automatického generování příkazů řeší nejednoznačnost vyhledáváním co nejméně destruktivního řešení v podobě kombinaci konfiguračních příkazů.

Abstract

This Master thesis describes the configuration of operating systems and their capabilities. It introduces differences between configuration of different operating systems according to their specialization, and it introduces advanced configuration of operating systems with third-party applications. The practical part of the thesis is a design of a new computer language aimed at describing a configuration of an operating system and its services. Such a description is used to automatically configure system services by translating it to a sequence of configuration commands. An advantage of the language is its readability by a human, but its similarity with natural languages introduces a certain level of ambiguity. The proposed method of automatic generation of commands deals with the ambiguity by searching and selecting as least as possible destructive commands.

Klíčová slova

konfigurace operačního systému, správa několika operačních systémů, počítačový jazyk, přirozený jazyk, nejednoznačnost

Keywords

configuration of operating system, management of multiple operating systems, computer language, natural language, ambiguity

Citace

Peter Schiffer: Automatické konfigurování služeb operačního systému, diplomová práce, Brno, FIT VUT v Brně, 2013

Automatické konfigurování služeb operačního systému

Prohlášení

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

. . . . . .
Peter Schiffer
20. května 2013

Poděkování

Rád by som poďakoval svojmu vedúcemu práce, doktorovi Aleši Smrčkovi, za vedenie práce a užitočné rady, ktoré mi poskytol.

Obsah

Kapitola 1

Úvod

S konfiguráciou operačného systému sa už stretol takmer každý jeho užívateľ. Na niektorých operačných systémoch je prívetivejšia než na iných, avšak vždy je jej úloha nenahraditeľná. Podoba a spôsob konfigurácie závisí od druhu operačného systému a od jeho zamerania. Iný prístup ku konfigurácii nájdeme v operačnom systéme určenom pre osobný počítač a iný v operačnom systéme určenom pre servery.

Moja diplomová práca je zameraná na automatickú konfiguráciu operačného systému. Jej cieľom je vytvorenie nového jednoduchého jazyka, ktorý bude podobný prirodzenému jazyku. Táto výhodná vlastnosť pre užívateľov nesie so sebou jednu nevýhodu a to čiastočnú nejednoznačnosť niektorých príkazov. Ich význam bude závisieť od ostatných príkazov, a teda od celkového kontextu.

V prvej časti mojej práce priblížim aktuálny stav konfigurácie rôznych operačných systémov, ich možností, výhody a nevýhody. Taktiež skúsim priblížiť pokročilé možnosti konfigurácie operačných systémov pomocou aplikácií tretích strán, poprípade správu veľkého počtu serverov. V druhej časti sa zameriam na popis, návrh a implementáciu samotného jazyka. Ukážem, z akých častí sa program v danom jazyku skladá, aká gramatika bola pre jazyk zvolená a aké výsledky boli pri použití jazyka dosiahnuté.

Prvý bod zadania je splnený v kapitole 2. Druhý a tretí bod zadania je splnený v kapitole 3, zaoberajúcou sa návrhom jazyka. Kapitola 5 sa zaoberá implementáciou prekladača jazyka a nachádza sa v nej splnený štvrtý bod zadania. Posledný, piaty bod zadania je splnený v prílohe F.

Kapitola 2

Súčasný stav

Táto kapitola sa zaoberá problematikou konfigurácie rôznych operačných systémov, zhrňuje ich výhody a nevýhody. V druhej časti sa kapitola zaoberá pokročilou konfiguráciou operačných systémov a aplikáciami na správu veľkého počtu serverov.

2.1 Konfigurácia operačných systémov

Vytvoriť dobrú konfiguráciu operačného systému nikdy nie je jednoduché. Vždy je treba zhodnotiť veľké množstvo faktorov. Takisto je dôležité zjednotiť ovládanie rôznych odlišných služieb, systémových programov a často aj programov tretích strán. V tejto podkapitole sa budem zaoberať konfiguráciou dvoch najrozšírenejších operačných systémov, priblížim ich možnosti, vlastnosti, ich výhody a nevýhody.

2.1.1 Linuxové operačné systémy

Linuxové operačné systémy sú najrozšírenejšie vo svojej serverovej podobe, teda bez grafického užívateľského rozhrania. V ovládaní cez príkazový riadok spočíva ich najväčšia výhoda. Takmer celý operačný systém je možné nakonfigurovať v textovom editore, skriptovací jazyk *Bash* dostupný v termináli má široké možnosti a veľa dostupných programov pre príkazový riadok na spracovanie textu ho robia priam ideálnym operačným systémom na servery s obmedzeným prístupom.

Samozrejme aj pre Linuxové operačné systémy existuje grafické užívateľské rozhranie. Medzi dve najvýznamnejšie grafické nadstavby pre Linux patria *GNOME* a *KDE*, no existuje aj veľa iných, viac či menej populárnych. V tejto podkapitole sa avšak budem zaoberať hlavne konfiguráciou pomocou príkazového riadku, kde ako som už spomenul, spočíva najväčšia výhoda týchto operačných systémov.

Všetko je súbor

Jedným z hlavných princípov Linuxového operačného systému je: všetko je súbor [1]. Či už máme na mysli štandardný vstup či výstup, alebo blokové zariadenie (napríklad disk), vždy sa jedná o súbor. Preto môžme bezpečné predpokladať, že formát konfiguračných dát bude tiež súbor, a konkrétne to bude súbor textový. Takáto reprezentácia prináša určité výhody. Na vytváranie a úpravu konfiguračných súborov postačujú základné nástroje pre prácu s textom, ako napríklad rôzne textové editory, prúdové editory (*SED*), poprípade regulárne výrazy. Textový formát je tiež dobre dostupný a zrozumiteľný človeku [10]. Na

druhú stranu, textová reprezentácia konfiguračných dát má aj svoje nevýhody. Formát a štruktúra týchto súborov je presne špecifikovaná a musí byť dodržaná, inak hrozí, že konfiguračný súbor nebude použiteľný. Toto nie je problém ak so súborom pracuje výhradne aplikácia, ktorá ho používa. Problém nastáva vo chvíli keď súbor upravuje človek. Človek môže pri editovaní súboru spraviť chybu, preklep, nedodržať formát, použiť nepodporovanú hodnotu. Textové súbory neobsahujú žiadnu formu kontroly obsiahnutých dát, a človek sa o chybe dozvie až neskôr keď požadovanú aplikáciu spustí, či reštartuje službu. V lepšom prípade sa aplikácia korektne spustí a zobrazí varovnú správu o chybe v konfiguračnom súbore, v horšom prípade sa aplikácia nespustí vôbec. Niektoré aplikácie obsahujú nástroje na kontrolu platnosti textových konfiguračných súborov pred ich použitím (napríklad *Apache* web server), avšak to je skôr výnimka než pravidlo. Ďalšia, možno diskutabilná nevýhoda tohto formátu ukladania konfiguračných dát je neefektívne spracovávanie na strane počítača, pričom existuje niekoľko efektívnejších spôsobov ukladania dát v počítači.

Adresár /etc

Všetky systémové konfiguračné súbory v Linuxových operačných systémoch sa nachádzajú v adresári */etc* [13]. Väčšina súborov je čitateľná pre všetkých užívateľov v systéme, výnimku tvoria súbory obsahujúce citlivé informácie, súkromné kľúče a podobne. Upravovať tieto konfiguračné súbory avšak môže spravidla iba užívateľ *root*.

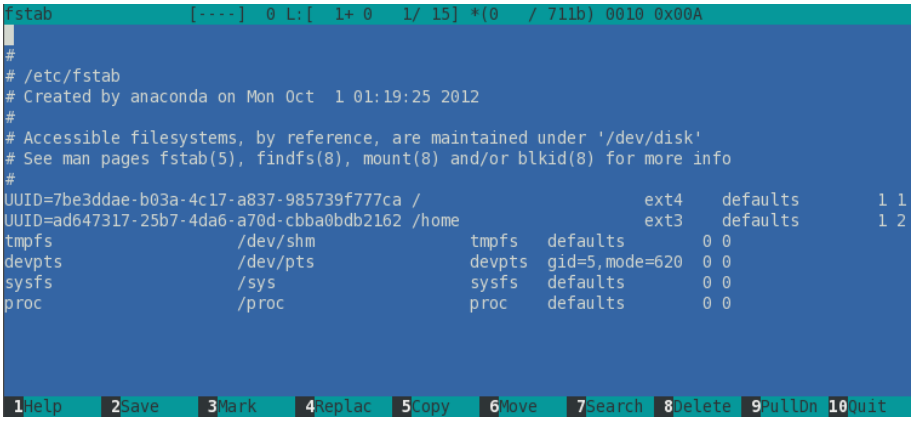
Štruktúra adresára */etc* nie je presne definovaná [16]. Obsahuje rôzne súbory a aj adresáre, štruktúru si určujú aplikácie, ktoré svoje konfiguračné súbory do adresára */etc* umiestňujú. Ak aplikácia obsahuje jeden konfiguračný súbor, väčšinou je umiestnený priamo v adresári. Aplikácie s viacerými konfiguračnými súbormi obsahujú v */etc* adresári svoj vlastný pod-adresár. Ten môže obsahovať ďalšie pod-adresáre alebo súbory.

Väčšina konfiguračných súborov musí byť správne pomenovaná, aplikácie súbory vyhľadávajú podľa názvu, avšak niektoré aplikácie načítavajú všetky súbory z určitého adresára, pričom na názve jednotlivých súborov nezáleží - vďaka tomuto je možné zložitú konfiguráciu rozložiť do viacerých súborov a tým zvýšiť prehľadnosť a udržateľnosť. Súbory v */etc* adresári môžu mať koncovku *.conf* a adresáre, pre väčšiu prehľadnosť, môžu mať koncovku *.d*.

Užívateľské konfiguračné súbory

Systémové konfiguračné súbory môžu postačovať na konfiguráciu servera alebo počítača s jediným užívateľom. Konfigurácia sa značne komplikuje ak má jeden systém využívať viacero užívateľov. Každý z týchto užívateľov má rôzne požiadavky a tým pádom vyžaduje aj rozdielnu konfiguráciu operačného systému či aplikácií. V Linuxovom operačnom systéme sa tento problém rieši užívateľskými konfiguračnými súbormi. Tieto súbory sú zväčša uložené v domovskom adresári užívateľa ako skryté súbory (názov súboru začína znakom *'.'*). Štruktúra je podobná adresáru */etc*.

Užívateľské konfiguračné súbory majú prednosť pred systémovými, teda najprv sa použije konfigurácia z užívateľských súborov a zo systémových súborov sa použije zvyšok. V praxi sa to často robí tak, že aplikácia načíta konfiguráciu zo systémových súborov a potom z užívateľských, pričom spoločné hodnoty systémových a užívateľských dát prepíše tými užívateľskými.



Obrázek 2.1: Prostredie textového editoru programu *Midnight Commander*.

Štruktúra konfiguračných súborov

Konfiguračné súbory môžu mať rôznu štruktúru. V drvivej väčšine konfiguračných súborov je možné používať komentáre, riadky začínajúce väčšinou znakmi *'#'* alebo *';'*. Komentáre môžu obsahovať ľubovoľné vety, slúžia pre ľudí na sprehľadnenie súboru, aplikácie tieto riadky ignorujú.

Najjednoduchšie konfiguračné súbory obsahujú iba hodnoty, spravidla jednu hodnotu na riadku. Takúto štruktúru má napríklad konfiguračný súbor */etc/hosts*. Zložitejšie konfiguračné súbory obsahujú premenné a hodnoty. Najzložitejšie konfiguračné súbory tiež obsahujú premenné spolu s hodnotami, avšak tieto premenné môžu byť ďalej rozdelené do rôznych sekcií, môžu obsahovať rôzne zoznamy ako hodnoty, poprípade konfiguračný súbor môže obsahovať odkaz na iný konfiguračný súbor. Ako príklad zložitejšieho konfiguračného súboru môže slúžiť konfiguračný súbor webového servera *Apache*.

Úprava konfiguračných súborov pomocou textových editorov

Ako som už spomenul v úvode kapitoly, výhoda konfiguračných súborov v textovej podobe je, že je možné tieto súbory jednoducho editovať v textových editoroch. Používané textové editory v termináli môžu byť napríklad jednoduché na používanie (*nano*, *joe*), zložitejšie na používanie ale s oveľa väčšími možnosťami (*vim*, *emacs*), poprípade textový editor, ktorý je súčasťou správcu súborov *Midnight Commander* (obrázok 2.1).

Táto forma editovania konfiguračných súborov je vhodná na jednoduchú, manuálnu úpravu súborov, avšak nie je možné takto upravovať súbory automaticky (napríklad ako súčasť inštalácie), a taktiež nie je možné takto upravovať súbory hromadne.

Úprava konfiguračných súborov pomocou prúdových editorov

Prúdové editory spracovávajú *"prúd"* textu. Program dostane na vstupe text, nejak ho spracuje a na výstup pošle upravený text. Tieto editory využívajú hlavne regulárne jazyky a teda aj typické operácie s textom sú hlavne vyhľadávanie a nahradzovanie textu. Typickým prúdovým editorom je program *sed*, ale je možné sem zaradiť aj program *awk*.

Výhoda týchto programov spočíva v jednoduchom upravovaní viacerých súborov súčasne (teda ak požadujeme vykonať tú istú operáciu na viacerých súboroch) a v možnosti automatizácie úkonov (prúdové editory je možné použiť v skriptoch).

```
# yum -y install httpd
```

Obrázek 2.2: Príklad inštalácie webového servera *Apache* pomocou nástroja *yum* vo *Fedore*.

Využitie skriptovacích jazykov pri úprave konfiguračných súborov

Širšie možnosti úpravy konfiguračných súborov ponúkajú skriptovacie jazyky. Výhodu majú tie, ktoré boli vyvinuté na úpravu textu (ako napríklad *perl* [15]), poprípade *Bash*, ktorý je dostupný priamo v termináli. *Perl* využíva hlavne regulárne jazyky, ale jeho možnosti sú väčšie ako možnosti programu *sed*. *Bash* obsahuje rôzne jednoúčelové programy a príkazy na spracovávanie súborov a textu. Pomocou týchto príkazov a programov je možné vytvárať skripty, poprípade programy, ktoré vykonávajú zložitejšie operácie s konfiguračnými súbormi. Tieto skripty môžu byť plne automatické, alebo môžu využívať vstup od užívateľa.

Skriptovacie jazyky ponúkajú pravdepodobne najširšie možnosti konfigurácie systému, avšak sú aj najzložitejšie. Užívateľ sa musí tieto jazyky a programy naučiť používať aby ich mohol efektívne využívať. Taktiež tieto nástroje sú príliš všeobecné, boli vytvorené na spracovávanie textu a súborov, a konfiguračné súbory patria iba do jednej podmnožiny zo všetkých dostupných možností.

Využitie verzovacích nástrojov

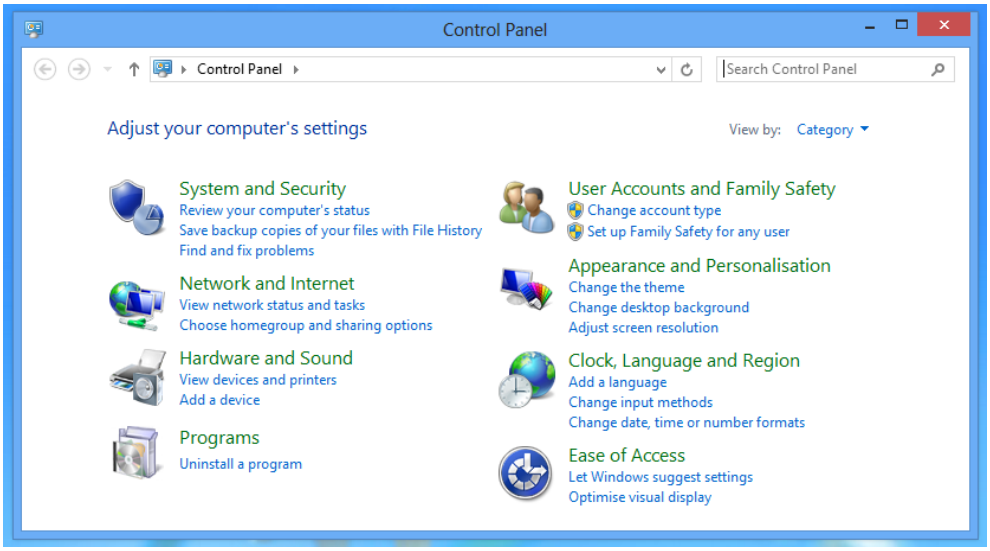
V kapitole o nástrojoch na úpravu konfiguračných súborov by som mal spomenúť aj verzovacie nástroje. Programy ako *Subversion* alebo *Git* môžu byť veľmi užitočné pri úprave, či skôr obnove pokazených alebo nefunkčných konfiguračných súborov [8]. Pomocou týchto nástrojov je možné udržiavať históriu a súčasne aj zálohu editovaných konfiguračných súborov. To, že záloha dát je nevyhnuteľná je dnes už zrejmé, avšak aj návrat k poslednej funkčnej verzii konfiguračného súboru môže byť veľmi neoceniteľný, hlavne v prípade ak je nutné s konfiguráciou značne experimentovať. Ako príklad použitia môže poslúžiť umiestnenie celého adresára */etc* do repozitára jedného zo spomenutých nástrojov.

Iné spôsoby konfigurácie

Pomocou textových súborov avšak nie je možné konfigurovať úplne všetko. Na určité úkony je nutné využiť aj rôzne programy a utility dostupné v systéme. Medzi takéto úkony napríklad patrí inštalácia balíkov, správa služieb operačného systému, správa užívateľov a skupín a podobné. Podporné programy sa väčšinou spúšťajú z príkazovej riadky spolu s parametrami podľa potreby, no často existujú aj grafické nadstavby týchto programov dostupné v grafickom užívateľskom prostredí. *Fedora* využíva na správu balíkov program *yum* [6], *Debian* používa *aptitude* [3]. Na správu služieb sa v Linuxových operačných systémoch využíval hlavne program *service* spolu so skriptami dostupnými v adresári `/etc/init.d`, no postupne ich začína nahradzovať komplexný správca systému a služieb *systemd* [4], ktorý je avšak s predchádzajúcom spôsobom konfigurácie kompatibilný. Správa užívateľov a skupín taktiež využíva viacero rôznych programov, v závislosti od distribúcie. Na príklade 2.2 je zobrazená inštalácia balíka pomocou nástroja *yum*.

2.1.2 Operačné systémy rodiny Windows

Ako som už vyššie spomenul, jednou z hlavných výhod Linuxových operačných systémov je ich možnosť konfigurácie cez terminál. To ich robí ideálnym operačným systémom na rôzne servery a počítače s obmedzeným prístupom. Na druhej strane avšak tým trpí užívateľská



Obrázek 2.3: Ovládací panel systému Windows.

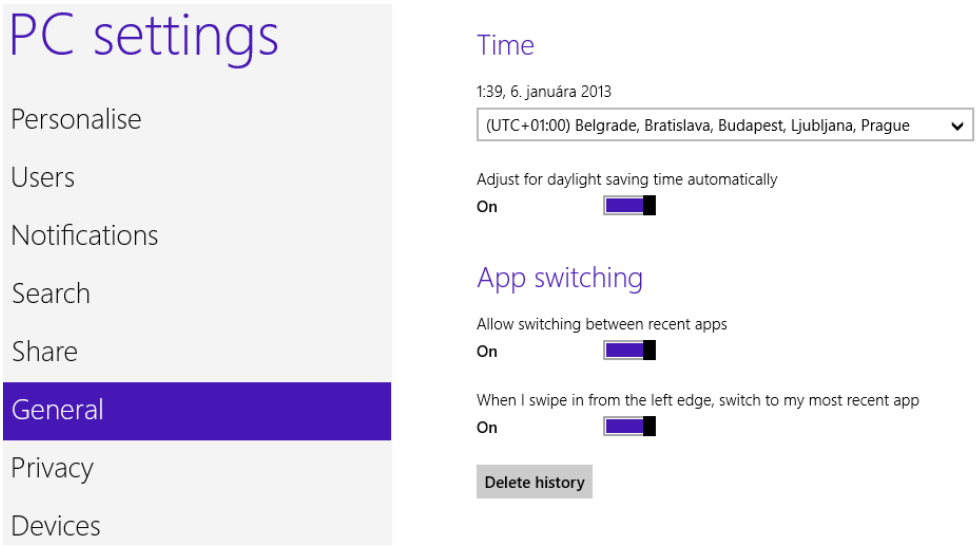
prívetivosť. Užívateľ bez konkrétnych znalostí môže nakonfigurovať len veľmi málo, inak sa musí naučiť službu konfigurovať z dokumentácie, príkladov a podobne. Na opačnej strane tohto problému je rodina operačných systémov Windows.

Operačné systémy Windows boli od začiatku zamerané svojim grafickým užívateľským rozhraním na užívateľskú prívetivosť. Konfigurácia týchto operačných systémov je možná pomocou grafických nástrojov v ovládacom paneli (obrázok 2.3), ktorý združuje rôzne služby, rozdelené do kategórií. Všetky možnosti konfigurácie sú spracované graficky, vstupy sú ošetrené tak, aby užívateľ mohol zadať iba platné hodnoty a informácie o jednotlivých voľbách sú dostupné okamžite. Možno iba rozdelenie jednotlivých možností do kategórií je trochu neprehľadné a nelogické, no pri takomto veľkom počte volieb je to pochopiteľné.

Od verzie Windows 8 obsahuje tento operačný systém grafickú užívateľskú nadstavbu *Metro*. Súčasťou tejto nadstavby je ďalšia vrstva konfigurácie celého systému (obrázok 2.4). Pravdepodobne ide o snahu sprehľadniť a zjednodušiť konfiguráciu systému začínajúcim užívateľom Windows.

Operačný systém Windows ukladá svoje konfiguračné dáta v registroch [14]. Registre Windows je databáza kľúčov a hodnôt [9]. Nachádzajú sa tam rôzne nastavenia, od nízkoúrovňových až po užívateľské. Na disku je táto databáza uložená v binárnom formáte, čo umožňuje efektívnejšiu prácu pri načítavaní a ukladaní dát voči textovej reprezentácii dát. Ďalšia výhoda spočíva v tom, že hodnoty v databázi sú silne typované a teda nie je možné omylom zadať reťazec tam kde sa očakáva číslo. Databáza taktiež prirodzene poskytuje integritu uložených dát pomocou atomických operácii a transakcií. Nevýhoda avšak môže nastať keď registre začnú obsahovať príliš veľa fragmentovaných údajov a základné operácie ako vkladanie, upravovanie a mazanie hodnôt sa spomalia. Vtedy sa môže začať spomaľovať celý operačný systém.

Registre Windows poskytujú verejné API, pomocou ktorého môžu upravovať hodnoty aplikácie a rôzne skripty [7]. Vzdialený prístup do registrov je možný pomocou služby vzdialených registrov. Na lokálnu úpravu hodnôt slúži program *regedit* (obrázok 2.5), ktorý zobrazuje kľúče a hodnoty v stromovej štruktúre. Pre užívateľa avšak nie je tento nástroj veľmi prívetivý, databáza sa môže zdať neprehľadná, súvisiace kľúče nemusia byť na prvý



Obrázek 2.4: Konfigurácia systému v prostredí *Metro*.

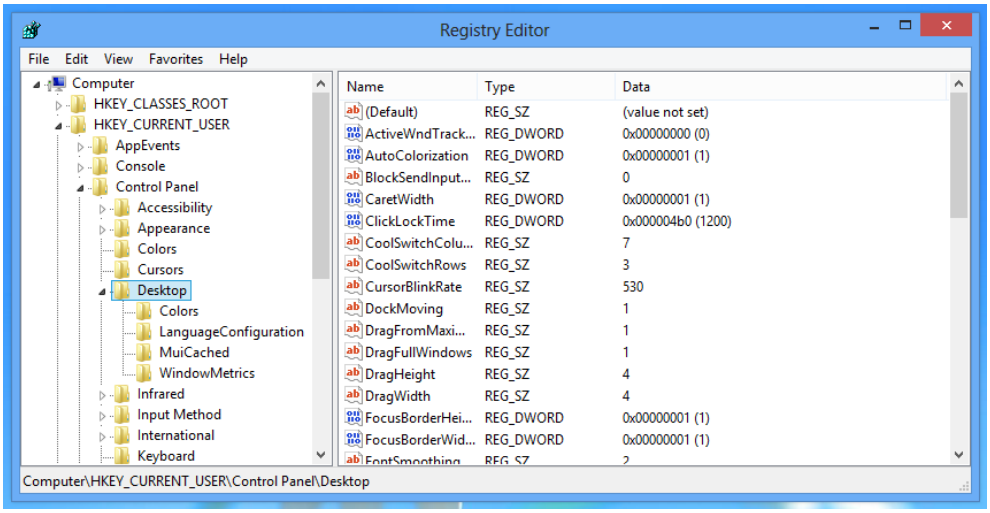
pohľad zrejmé.

Aplikácie tretích strán môžu taktiež využívať službu registrov, no zvyčajne ukladajú konfiguračné súbory do svojho pod-adresára v užívateľskom adresári. Typický formát konfiguračných súborov pre Windows aplikácie je *INI* súbor, avšak nie je to pravidlo. Napríklad *.NET* alebo *Java* aplikácie využívajú na ukladanie konfiguračných dát väčšinou *XML* súbory.

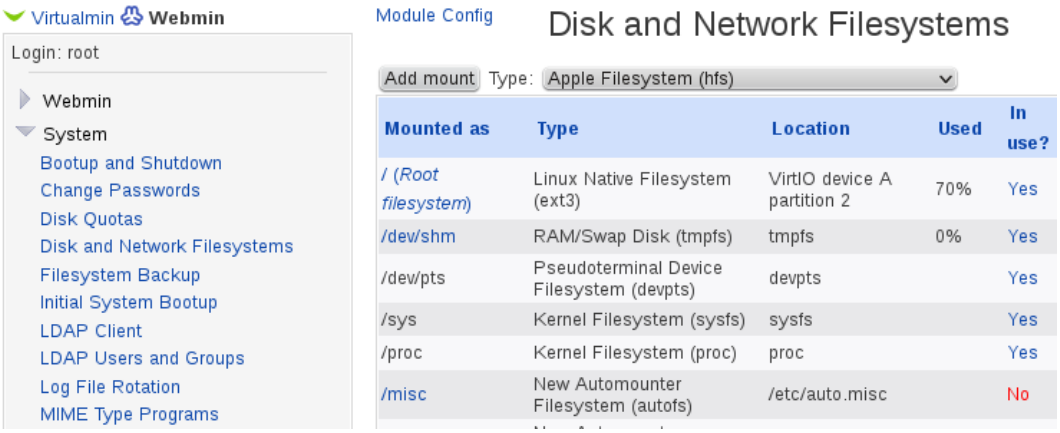
2.2 Pokročilé možnosti konfigurácie operačných systémov

Možnosti konfigurácie operačného systému boli vyvinuté tak, aby čo najefektívnejšie spĺňali svoju úlohu. Toto je v poriadku ak má užívateľ na starosti jeden, dva, možno zopár počítačov v domácnosti, malej kancelárii alebo ak sa stará o veľmi malý počet serverov. Problém nastáva, keď množstvo spravovaných počítačov rastie, čo je v dnešnej dobe bežný prípad v strednej a veľkej firme alebo v serverovni s niekoľko stovkami až tisíckami serverov. Na spravovanie takéhoto množstva počítačov už nástroje poskytované samotnými operačnými systémami nepostačujú a je nutné zaviesť konfiguračný manažment.

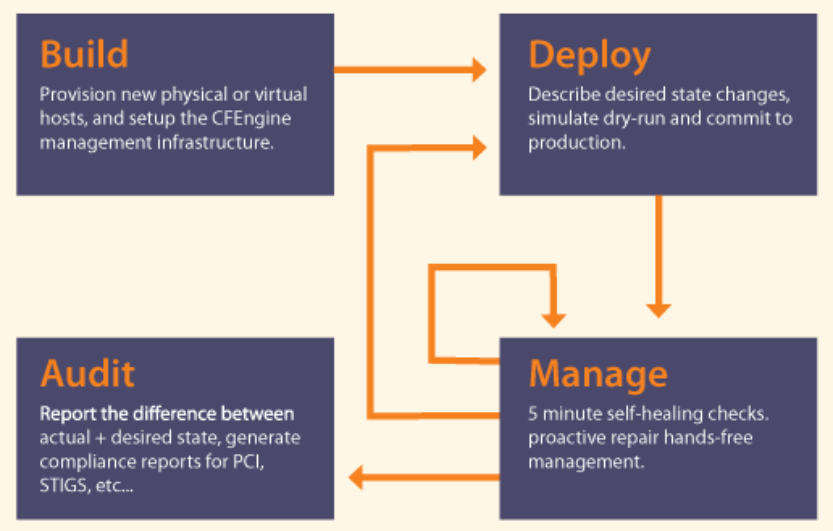
Konfiguračný manažment môže mať niekoľko podôb. Od sady jednoduchých skriptov cez upravené inštalačné obrazy až po sofistikovaný informačný systém. Vždy ho však sprevádza nejaký proces, už to nie je iba software sám o sebe. Pokročilý software na správu operačného systému a jeho aplikácií by sme mohli rozdeliť na pomyselné dve skupiny. Jednu skupinu by tvoril software, ktorý spravuje iba jeden počítač. Tento software spravidla abstrahuje konfiguráciu operačného systému a ponúka rovnaké možnosti konfigurácie pre viacero druhov operačných systémov a aplikácií. Prvá skupina nachádza svoje uplatnenie hlavne pri správe vzdialených serverov, napríklad pre webhosting a podobne. Týmto spôsobom sa nespravuje veľký počet serverov, jeho výhoda spočíva hlavne v abstrakcii a ucelenosti konfigurácie. Ako príklad môžem uviesť konfiguračný software *Webmin* [5], pomocou ktorého je možné spravovať operačný systém pomocou webového prehliadača. *Webmin* je určený pre Linuxové operačné systémy, abstrahuje jednotlivé rozdiely v konfigurácii a vytvára jednotné prostredie na konfiguráciu všetkých častí systému (obrázok 2.6). Je modulárny a rozšíriteľný tak, aby mohol jednoducho podporovať konfiguráciu nových služieb a aplikácií.



Obrázek 2.5: Program *regedit*.



Obrázek 2.6: Ukážka užívateľského rozhrania programu *Webmin*.



Obrázek 2.7: Základné využitie systému *CFEngine*. Zdroj [2].

```
bundle agent app_db_mysql {
vars:
  redhat::
    ‘‘match_package’’ slist => {‘‘mysql’’, ‘‘mysql-server’’};
  debian::
    ‘‘match_package’’ slist => {‘‘mysql-server’’};
processes:
  ‘‘mysqld’’ -> ‘‘start_mysql’’,
    restart_class => ‘‘start_mysql’’,
          comment => ‘‘Make sure the back-end MySQL is running’’;
commands:
  start_mysql::
    ‘‘/etc/init.d/mysql restart’’,
       handle => ‘‘start_mysql’’,
      comment => ‘‘Start the mysqld service’’;
    ‘‘/usr/bin/mysqldump --all-databases >/var/cfe/dumps/mysql.xml’’
      comment => ‘‘Dump all MySQL databases to xml’’,
      contain => in_shell,
       action => if_elapsed(‘‘8460’’);
}
```

Obrázek 2.8: Príklad ukážky nasadenia `mysql` servera v jazyku *CFEngine*. Zdroj [2].

Druhá skupina by obsahovala pokročilejší software na konfiguráciu operačného systému. Štruktúra tohto software už spravidla obsahuje klientskú a serverovú časť, kde klienti sa inštalujú na spravované počítače, ktoré následne komunikujú so serverovou častou. Táto je iba jedna a ponúka možnosti na správu niekoľko desiatok až stoviek počítačov. Klienti na jednotlivých spravovaných počítačoch zbierajú potrebné dáta a aktuálne nastavenia, ktoré odosielajú na server. Na serveri si môže užívateľ prehľadne zozbierané dáta zobraziť a vyhodnotiť. Požadovanú konfiguráciu server rozposiela klientom, ktorý majú na starosti správne nastavenie staníc. Ako zástupcu tejto skupiny by som chcel predstaviť *CFEngine* [2], veľmi populárny systém na konfiguračný manažment. Jeho vývoj začal v roku 1993 na univerzite v Oslo. Využíva spomenuté paradigma klient-server, je vytvorený tak, aby dokázal spravovať niekoľko tisíc klientských staníc. Jeho možnosti sú široké, najvýznamnejšie prípady užitia sú zobrazené na obrázku 2.7. Ako väčšina podobných riešení, aj *CFEngine* obsahuje vlastný jazyk, ktorý slúži na konfiguráciu klientov.

Kapitola 3

Návrh jazyka pre popis konfigurácie operačného systému

V tejto kapitole sa zameriam na vlastnú tvorbu, novo-vytvorený jazyk navrhnutý na konfiguráciu Linuxového operačného systému. V úvode kapitoly je popísaný jazyk z užívateľského hľadiska a dve základné časti, z ktorých sa bude skladať program v tomto jazyku. Následne je popísaná jeho gramatika a základný návrh implementácie, podmienky a akcie nachádzajúce sa v jazyku, cieľový operačný systém a použitá knižnica na prácu s konfiguračnými súbormi.

3.1 Popis jazyka

Novo-vytvorený jazyk musí spĺňať niekoľko pravidiel. Musí byť veľmi jednoduchý pre užívateľov a aspoň čiastočne by sa mal podobať prirodzenému jazyku. Tieto vlastnosti by mali zabezpečiť krátku krivku učenia jazyka pre jeho užívateľov, čo je pravdepodobne najdôležitejšia vlastnosť ak má byť nový spôsob konfigurácie úspešný. Dnes existuje veľa rôznych jazykov a každý užívateľ sa najprv dobre rozhodne či sa mu nový jazyk oplatí učiť sa pred tým než ho začne používať. Prínos nového jazyka a jeho užívateľská prívetivosť, alebo inak povedané zložitosť či jednoduchosť zohrávajú istotne jednu z najdôležitejších úloh pri voľbe novej technológie.

Program v tomto jazyku je rozdelený na dve časti. Prvá časť sa nazýva `Preconditions`, druhá `Postconditions`. Obe časti sa skladajú z rovnakých príkazov. Jeden riadok v programe predstavuje jednu podmienku, pričom vždy sa jedná o podmienky. Na základe toho, kde sa tieto podmienky nachádzajú, nadobúdajú rôzny význam. Väčšina podmienok bude obsahovať kladnú a negatívnu verziu, takže užívateľ môže jednoducho nastaviť aby niečo neplatilo. Obe verzie podmienky budú vyzerať prirodzene, teda zápor bude vyjadrený kľúčovým slovom `not`, či už vo forme `is not` alebo `does not`.

V prvej časti programu sa vyhodnocujú aktuálne nastavenia systému, prebiehajú testy. Ak (a iba ak) sú všetky podmienky na aktuálnom systéme splnené, pokračuje sa druhou časťou programu. Účel tejto sekcie spočíva v tom, aby si užívateľ mohol definovať za akých podmienok bude chcieť požadovanú úpravu konfigurácie vykonať. Prvá časť programu bude voliteľná.

V druhej časti, `Postconditions`, sa systém nastavuje. Prebiehajú také akcie, aby všetky zadané podmienky v tejto časti boli po skončení programu pravdivé. Podmienky pre obe časti programu sú rovnaké, jedna podmienka môže byť použitá v prvej či druhej časti. Na

základe umiestnenia sa vytvára jej význam. To je veľká výhoda pre užívateľov, pretože môžu používať tie isté príkazy, ktorých je menej.

Z doteraz napísaného vyplýva, že každá podmienkou bude (vnútorne) obsahovať dve akcie. Jedna akcia bude overovať platnosť danej podmienky a druhá akcia bude systém upravovať tak, aby následne zvolená podmienka platila. Druhá akcia bude často rozdelená na ďalšie dve akcie, podľa toho, či podmienka je kladná alebo negatívna.

Ako som už spomenul, novo-vytvorený jazyk bol navrhnutý tak, aby bol čo najjednoduchší a čo najviac podobný prirodzenému jazyku. Dôsledok tohto prístupu je ten, že niektoré podmienky samy o sebe môžu byť celkom nejednoznačné, čím sa dostávame k jednej dôležitej vlastnosti tohto jazyka. Akcie vykonávané v časti `Postconditions` nemusia zodpovedať presne podmienkam tam uvedeným, pretože nemusia byť dostatočné. Pri vykonávaní akcií, či už kladných alebo záporných, často sa budú musieť zobrať do úvahy aj podmienky uvedené v časti `Preconditions`. To znamená, že pre vykonanie niektorých akcií bude dôležitý celkový kontext programu. Ako jednoduchý príklad uvediem presunutie súboru. V prvej časti programu podmienim existenciu určitého súboru. V druhej časti programu budem testovať cestu označeného súboru v inom adresári. Avšak na vykonanie výslednej akcie potrebujem vedieť informácie z oboch podmienok na to, aby som mohol súbor bezpečne skopírovať do nového adresára. Ak do druhej časti pridám podmienku, že súbor sa v starom adresári už nenachádza, tak bude súbor presunutý. V tomto prípade je výsledná akcia vykonaná na základe troch podmienok.

3.2 Gramatika

Keďže jazyk neobsahuje žiadne vnorené výrazy, bol navrhnutý ako pravý regulárny jazyk so všetkými pravidlami v tvare $A \rightarrow xB$ alebo $A \rightarrow x$ kde A a B sú neterminály a x je reťazec terminálov. Kompletnú gramatiku je možné nájsť v prílohe E.

Pre implementáciu jazyka boli zvažované dva prístupy. Prvý tradičný prístup bol vytvoriť *LL(1)* gramatiku na rozklad jazyka. Výhoda tohto prístupu by spočívala v rýchlosti a jednoduchosti rozkladu, avšak kľúčová nevýhoda, kvôli ktorej bol tento prístup zamietnutý bola zložitejšia úprava už implementovaného jazyka. Pri vytváraní *LL(1)* gramatiky musí byť jazyk kompletný a jednoznačný, inak môžu nastať problémy pri implementácii, poprípade neskôr keď je nutné už implementovaný jazyk upraviť. Preto bol zvolený druhý prístup, keď sa pri spracovávaní každej vety jazyka bude porovnávať pozícia kľúčových slov jazyka. Tento prístup bude algoritmicky o niečo zložitejší než rozklad jazyka pomocou *LL(1)* gramatiky, avšak úpravy už implementovaného jazyka budú oveľa jednoduchšie. Druhý prístup bude taktiež v prípade potreby jednoducho rozšíriteľný o nové kľúčové slová a prvky jazyka.

3.3 Základný návrh implementácie jazyka

Prekladač jazyka som sa rozhodol vytvoriť v jazyku *Python 2.x*. Cieľový jazyk prekladu bude *Bash*. Ako základný podporovaný operačný systém som zvolil Linux *Fedora* vo verzii 17. Výsledný skript nebude závislý na konfigurácii systému, na ktorom bol vytvorený, z tohto hľadiska bude univerzálny. Ak by sa jazyk v budúcnosti podarilo rozšíriť o ďalšie podporované operačné systémy, cieľový systém bude možné zvoliť pri preklade.

Linuxová distribúcia *Fedora* 17 bola zvolená ako základný podporovaný operačný systém, no výsledné skripty by mali fungovať na tejto Linuxovej distribúcie od verzie 15. Hlavné dva predpoklady, na ktorých budú výsledné skripty závisieť sú použitie *rpm* balíčkovacieho

systému a správcu služieb a systému *systemd*. Ďalší predpoklad sa vzťahuje k použitiu príkazu *sudo*. Vo *Fedore*, v základnej konfigurácii, stačí pridať užívateľa do skupiny *wheel* aby mohol spúšťať príkaz *sudo*.

Prvá časť programu (`Preconditions`) sa preloží do podmienok v *Bashi* vždy tak, že pri každom spustení výsledného programu sa vždy vykonajú všetky testy. Ak niektoré z podmienok nebudú splnené, vo výstupe budú uvedené všetky nesplnené podmienky, so zrozumiteľným popisom pre užívateľa.

Druhá časť programu (`Postconditions`) bude náročnejšia. Výsledok tejto časti programu budú také akcie, ktoré keď sa vykonajú, všetky podmienky v tejto časti programu budú pravdivé. Dôležité je si uvedomiť, že niektoré podmienky závisia priamo na ďalších podmienkach, či už v druhej alebo prvej časti programu. Druhý dôležitý aspekt, ktorý treba zobrať do úvahy je, že k výsledku niektorých akcií vedie rôzna cesta. Na nájdenie ideálneho riešenia tohto problému, ak samozrejme bude existovať, bude nutné implementovať algoritmus využívajúci prehľadávanie daného stavového priestoru.

Na prácu s konfiguračnými súbormi použijem software *Augeas* [11]. Tento nástroj abstrahuje formát konfiguračných súborov, ktoré následne reprezentuje ako strom. Dáta v strome je možné čítať aj zapisovať, načítavanie a ukladanie stromu je skryté. Na popis konfiguračných súborov využíva `lenses` [12], čo sú súbory napísane v špecifickom jazyku. Každý konfiguračný súbor má spravidla svoju `lens`. Vďaka tomu je *Augeas* jednoducho rozšíriteľný o podporu nových súborov.

3.4 Podmienky a akcie

Na základe kapitoly 2 je možné povedať, že konfigurácia Linuxového operačného systému sa dá rozdeliť do troch základných kategórií. Keďže základom konfigurácie v tomto druhu operačných systémov je súbor, prvá kategória sa zaoberá prácou so súbormi a adresármi, ich vytváraním, presúvaním, mazaním. V druhej kategórii sa nachádza práca s obsahom textových súborov. Posledná kategória zahŕňa prácu s rôznymi konfiguračnými nástrojmi a utilitami.

Na základe týchto troch kategórií je nutné vytvoriť podmienky a akcie tak, aby užívateľ pri práci s adresármi, súbormi a ich obsahom nebol obmedzovaný, a súčasne, aby táto práca bola prirodzená a blízka bežnému jazyku. Z tretej kategórie je nutné vybrať podporu pre také nástroje a programy, ktoré sú často využívané a dôležité pre všeobecnú konfiguráciu operačného systému. Úplný zoznam podmienok je možné nájsť v prílohe B a zoznam akcií sa nachádza v prílohe C.

3.4.1 Práca so súbormi a adresármi

Pri práci so súbormi a adresármi musí byť možné testovať ich polohu, existenciu a či sa na danej ceste nachádza súbor alebo adresár. Pri konfigurácii je taktiež môžné stretnúť so symbolickými odkazmi, takže testovanie či súbor je odkaz môže byť často užitočné. Okrem polohy sú v Linuxovom operačnom systéme dôležité aj prístupové práva k súborom, ich vlastníci a skupiny do ktorých patria. Preto, v jazyku musia byť dostupné aj podmienky, pomocou ktorých bude možné testovať súbory na ich prístupové práva, vlastníka a skupinu. Na možnosti ktoré systém práv v Linuxe poskytuje, je práca s ním jednoduchá, no na prvý pohľad nie intuitívna. Aby boli všetky možnosti práce so systémom práv zachované, bude v jazyku podmienka na testovanie práv v osmičkovom formáte. Aby jazyk podporoval aj

zjednodušenú prácu s správami, budú do jazyka pridané tri ďalšie podmienky, ktoré budú testovať súbor na práva čítania, zápisu a spúšťania samostatne.

3.4.2 Práca so obsahom súborov

V prípade konfiguračných súborov na základe ich líšiacej sa zložitosti je možné rozdeliť prácu s textovými súbormi na dva spôsoby. Jedným spôsobom je možné pracovať s celým obsahom súboru, bez bližšej kontroly nad reťazcami v súbore obsiahnutými. Kvôli tomuto prípadu budú do jazyka pridané podmienky testujúce celý obsah súboru alebo jeho časť. Druhým spôsobom sa pracuje s premennými v konfiguračných súboroch. Tento spôsob vyžaduje úplnú kontrolu nad obsahom súboru, takže podmienky ktoré sa budú zaoberať touto časťou konfigurácie budú využívať v pozadí nástroj *Augeas*.

3.4.3 Práca s konfiguračnými nástrojmi

Zo spôsobov konfigurácie, ktoré vyžadujú na modifikáciu systému rôzne nástroje, bude jazyk podporovať nasledujúce operácie. Inštaláciu a odstraňovanie balíčkov, spravovanie služby *cron*, spravovanie bežiacich služieb v systéme a správu užívateľov a skupín. Tieto operácie boli zvolené tak, aby umožňovali konfigurovanie najzákladnejších súčastí operačného systému. Na operácie ktoré nie sú priamo podporované v jazyku bude možné použiť príkaz `execute`, ktorý iba jednoducho predá svoj parameter *Bashu*.

3.4.4 Akcie

Akcie v jazyku musia umožňovať vykonávať zmeny v systéme na základe vstupných podmienok obsiahnutých v programe. Tieto akcie avšak nebudú zodpovedať podmienkam 1:1, pretože niektoré podmienky neobsahujú dostatok informácií na to, aby sa z nich dala vytvoriť kompletná akcia. Akcií teda bude menej ako samotných podmienok, ale kategórie konfigurácie budú zastúpené rovnako.

Kapitola 4

Popis syntaxe a použitia vytvoreného jazyka

Táto kapitola sa zaoberá popisom syntaxe novo-vytvoreného jazyka a podmienok ktoré sú v jazyku dostupné. Kompletný zoznam všetkých podmienok je možné nájsť v prílohe D.

4.1 Dátové typy a premenné

Jazyk obsahuje tri dátové typy, ktoré môžu byť uložené v premenných, a to reťazec, súbor a adresár. Každá z týchto hodnôt môže byť uvedená v dvojitých alebo jednoduchých úvodzovkách, nutnosť je to iba v prípade, ak hodnota obsahuje medzeru.

```
string my_name ‘‘John Doe’’
file f /home/doe/Documents/cv.pdf
directory d ‘‘/home/doe/my dir’’
```

Obrázek 4.1: Príklad definície premenných.

Hodnotu reťazcovej premennej je možné definovať zo súboru.

```
string file_c is in /home/doe/my_file
```

Obrázek 4.2: Príklad načítania obsahu súbora do premennej.

Pri definovaní premennej sa nevykonáva žiadna typová kontrola. Je to hlavne z toho dôvodu, že súbor alebo adresár nemusí pred spustením skriptu existovať, prípadne súbor môže byť definovaný relatívnou cestou, ktorá neobsahuje žiadne lomítka a teda typová kontrola by bola neefektívna.

Rozsah platnosti premenných je definovaný nasledovne. Premenné definované v časti `Preconditions` platia v tejto časti a aj v časti `Postconditions`, ak tam nie je premenná predefinovaná. Premenné definované v časti `Postconditions` platia iba v tejto časti. Na poradí príkazov v jazyku nezáleží, takže z hľadiska použitia premenných užívateľom je dôležité iba ich umiestnenie v správnej časti programu.

Premenné definované v časti `Preconditions` nemajú žiadny skutočný dopad na výsledný skript, teda žiadne podmienky na základe týchto premenných sa nevygenerujú, slúžia iba na uľahčenie práce užívateľovi. Na druhú stranu, premenné definované v druhej časti, `Postconditions`, výsledný skript ovplyvniť môžu. Ak sa definuje premenná typu sú-

bor alebo adresár v druhej časti, a ten súbor alebo adresár neexistuje, môže sa, v závislosti na ostatných podmienkach, vytvoriť.

Reťazce načítané zo súboru je možné v časti `Preconditions` porovnávať. Konkrétne, či sa dva reťazce zhodujú alebo nie, alebo či reťazec obsahuje respektíve neobsahuje podreťazec. Toto je niekoľko z mála podmienok, ktoré nie sú podporované aj v časti `Postconditions`.

```
my_var contains ‘‘enabled’’
```

Obrázek 4.3: Príklad kontroly podreťazca v premennej.

4.2 Práca so súbormi a adresármi

Jazyk síce obsahuje dva rôzne dátové typy pre súbory a adresáre, ale pri spracovávaní umiestnenia a práv medzi týmito dvomi typmi nerozlišuje. Užívateľ samozrejme môže explicitne definovať či chce pracovať so súborom alebo adresárom, ale nie je to podmienka. V prípade že je nutné vedieť o aký typ sa jedná a nie je to jasne definované, prekladač sa to pokúsi odhadnúť. Ak sa mu to nepodarí, predpokladá že to je súbor a užívateľa na to upozorní varovnou hláškou.

Pomocou jazyka je možné súbory a adresáre vytvárať, kopírovať, presúvať a mazať. Je možné testovať či zadaná cesta existuje a čo sa na jej konci nachádza. Je možné vytvárať symbolické odkazy, či mazať obsah súboru alebo adresára.

```
preconditions:
    file f abc
    f is not empty
postconditions:
    f is placed in dir1
```

Obrázek 4.4: Príklad práce so súborom.

Prístupové práva je možné definovať priamo ako trojicu čísiel v osmičkovej sústave, alebo pomocou troch podmienok (spolu s ich negáciami) prirodzeným spôsobom. Zmena vlastníka a skupiny súboru je tak isto možná.

```
preconditions:
    file f abc
postconditions:
    owner of f is john_doe
    f is readable
    f is not writable
```

Obrázek 4.5: Príklad zmeny prístupových práv.

4.3 Práca s obsahom súborov

S obsahom súborov je možné pracovať dvomi spôsobmi. Prvým spôsobom sa pracuje s celým obsahom súboru, druhý spôsob umožňuje presnejšie pracovanie s konkrétnymi časťami obsahu súboru vďaka nástroju *Augeas*. Pri prvom spôsobe je možné zmeniť celý obsah súboru, alebo iba pridať reťazec na koniec súboru, či ľubovoľný reťazec zo súboru odstrániť.

V príklade 4.6 sa na koniec súboru `/etc/hosts.deny` pridá reťazec a zmení sa obsah súboru `/home/doe/my_new_file`.

```
preconditions:
    file f /etc/hosts.deny
postconditions:
    f content contains ‘‘ALL: example.org’’
    file m /home/doe/my_new_file
    m content is ‘‘this is my newly created file’’
```

Obrázek 4.6: Príklad prvého spôsobu práce s obsahom súboru.

K tomuto príkladu je ešte nutno dodať, že ak súbor `/etc/hosts.deny` daný reťazec už obsahuje, tak ďalší sa už nepridá. A ak súbor `/home/doe/my_new_file` neexistuje, vytvorí sa, pretože premenná `m` bola definovaná v časti `Postconditions`. Ak by bola definovaná v časti `Preconditions` a súbor by neexistoval, výsledný skript by po spustení zlyhal.

Augeas reprezentuje obsah súboru ako strom. Každá premenná alebo časť súboru je vyjadrený cestou. Jednoduchšie konfiguračné súbory s premennými si vystačia s názvom súboru a premennej, zložitejšie vyžadujú aj sekciu.

```
preconditions:
    file ssh_conf /etc/ssh/sshd_config
    file hosts_allow /etc/hosts.allow
postconditions:
    ListenAddress is ‘‘10.0.1.50’’ in ssh_conf
    Port is ‘‘2222’’ in ssh_conf
    hosts_allow content contains ‘‘sshd: 12.34.56.78’’
    sshd is enabled
    sshd is running
```

Obrázek 4.7: Príklad zjednodušenej ukážky konfigurácie `SSH` démona.

Nastavenie portu z vyššie uvedeného príkladu sa preloží na príkaz:

```
augtool set /files/etc/ssh/sshd_config/Port ‘‘2222’’
```

Podmienky obsahujúce sekciu umožňujú užívateľovi oveľa väčšiu voľnosť pri manipulácii s konfiguračnými súbormi. Daňou za túto možnosť je avšak možno nie tak jednoduchá a prehľadná syntax ako vo zvyšku jazyka a taktiež nutnosť poznať konfiguračný súbor a jeho reprezentáciu v nástroji augeas.

```
name is ‘‘John Doe’’ in /etc/passwd in johndoe
```

Obrázek 4.8: Príklad ukážky konfigurácie súboru `/etc/passwd`.

4.4 Konfigurácia vybraných súčastí operačného systému

Jazyk okrem doteraz menovaných všeobecných podmienok obsahuje aj špecifické podmienky na konfiguráciu vybraných súčastí operačného systému. Tieto súčasti boli vybrané podľa dôležitosti a úlohy v operačnom systéme ako aj podľa spôsobu použitia, teda či je možné

tieto súčasti konfigurovať pomocou súborov, alebo je nutné použiť externé programy. Uprednostnené boli súčasti pracujúce s externými programami, medzi ktorými je možné nájsť napríklad správu služieb, balíkov, užívateľov, či nastavenie cronu alebo predvolených príkazov.

4.4.1 Správa balíkov, služieb a užívateľov

Správa balíkov obsahuje iba jedinú podmienku (a jej negovanú verziu), ktorá rozhoduje o tom, či je balík nainštalovaný alebo nie. Správa služieb obsahuje dva druhy podmienok, jedna rozhoduje o tom, či služba beží a druhá o tom, či sa služba má spustiť automaticky pri štarte systému. Keď sa služba povolí, neznamená to že sa aj automaticky spustí, takže na novo-nainštalovanú službu, ktorú chceme mať povolenú pri štarte systému a aj okamžite spustenú, musíme použiť obe podmienky súčasne.

```
preconditions:
    httpd package is not installed
postconditions:
    httpd package is installed
    httpd is enabled
    httpd is running
```

Obrázek 4.9: Príklad nainštalovania a spustenia `Apache` servera.

Užívatelia a skupiny majú k dispozícii štyri druhy podmienok. Pomocou dvoch je možné užívateľov a skupiny vytvárať a mazať. Ďalšia podmienka slúži na pridávanie a odstraňovanie užívateľov zo skupín. Posledná podmienka z tejto skupiny slúži na spravovanie administrátorov. Pod administrátorom sa rozumie užívateľ, ktorý môže spúšťať príkaz `sudo`, vo *Fedore* typicky užívateľ patriaci do skupiny `wheel`.

```
preconditions:
    email_users is group
postconditions:
    john is user
    john belongs to email_users
    erik is admin
```

Obrázek 4.10: Príklad práce s užívateľmi.

4.4.2 Predvolené príkazy, cron a príkaz execute

Podmienka zaoberajúca sa predvolenými príkazmi využíva na svoju realizáciu Linuxový program `alternatives`. Cron je možné nastavovať dvoma spôsobmi. Prvý zjednodušený spôsob pracuje iba so súbormi, pričom je možné nastaviť pravidelné spúšťanie daného súboru iba v štyroch intervaloch, a to na každú hodinu, deň, týždeň či mesiac. Výsledkom tohto spôsobu je vytvorenie symbolického odkazu na súbor do adekvátneho adresára cronu v `/etc`. Druhý spôsob pridáva záznam do súboru `/etc/crontab`, a teda dovoľuje pracovať ako so súbormi, tak aj so samostatnými príkazmi. Čas opakovania je možné nastaviť v bežnej syntaxi cronu.

Podmienka `execute` tvorí jedinú výnimku svojho druhu v jazyku, pretože je to vlastne príkaz, ktorého celý reťazcový parameter sa predáva na spracovanie Bashu. Tento príkaz bol pridaný do jazyka preto, aby bolo možné jednoducho spúšťať príkazy, ktoré jazyk priamo nepodporuje.

```
postconditions:
    /usr/java/latest/bin/java is default command for java
    /home/user/bin/backup is executed daily
    ‘‘ntpdate pool.ntp.org’’ is executed every ‘‘0 0 * * *’’
```

Obrázek 4.11: Príklad predvoleného príkazu a cronu.

```
execute ‘‘mount -a’’
```

Obrázek 4.12: Príklad použitia `execute` podmienky.

4.5 Použitie prekladača

Prekladač jazyka ponúka dva základné spôsoby prekladu. Keďže je napísaný v jazyku *Python*, funkciu zodpovednú za preklad je možné použiť v ľubovoľnom skripte tohto jazyka ako modul. Funkcia má názov `compile_source` a obsahuje dva vstupné parametre. Prvý, názov súboru so vstupným programom a druhý, voliteľný, názov výstupného súboru. Ak druhý parameter nie je zadaný, použije sa predvolený názov súboru, a to `a.sh`.

Prekladač je samozrejme možné spustiť ako samostatný skript, ktorý má názov `compile.py` a vyžaduje jeden povinný parameter, názov vstupného súboru. Pomocou parametra `-o, --output` je možné definovať názov výstupného súboru, ktorý má bez použitia tohto parametru už spomenuté predvolené meno `a.sh`.

Kapitola 5

Implementácia prekladača

V tejto kapitole budem popisovať samotnú implementáciu prekladača. Ako som už spomenul v návrhu, prekladač je napísaný v programovacom jazyku *Python*. Jadro prekladača pozostáva z troch kľúčových častí, syntaktického analyzátora, hľadania výsledného kódu a generátora výsledného kódu.

5.1 Syntaktický analyzátor

Pri navrhovaní jazyka boli zvažované dva spôsoby implementácie syntaktického analyzátora (parsera). Jedna, vytvoriť *LL(1)* gramatiku jazyka a implementovať bežný syntaktický analyzátor tejto gramatiky. Tento prístup avšak obsahuje zásadnú nevýhodu, kvôli ktorej bol zamietnutý, a to takú, že takto implementovaný parser vyžaduje aby gramatika jazyka bola úplná a konečná. Pri návrhu takéhoto druhu jazyka je veľmi ťažké, ak nie nemožné, vytvoriť správnu a úplnú gramatiku už pri návrhu. Existuje relatívne vysoké riziko, že pri implementácii a testovaní jazyka sa nájdu určité časti gramatiky, ktoré bude nutné upraviť. V tomto prípade by dodatočná úprava jazyka bola oveľa zložitejšia a zabrala by viacej času. Na druhej strane, navrhnutý jazyk je pomerne jednoduchý zo syntaktického hľadiska. Každá podmienka je samostatná, jazyk neobsahuje žiadne zátvorky, zanorovanie, či zložitejšie konštrukcie. S ohľadom na túto skutočnosť boli všetky podmienky navrhnuté tak, aby použité kľúčové slová a ich pozícia v rámci podmienky bola jedinečná. S týmto predpokladom už bolo celkom jasné, že implementácia vlastného parsera bude z hľadiska neskorších úprav oveľa výhodnejšia, aj keď samotná implementácia bude pravdepodobne náročnejšia, než implementácia parsera *LL(1)* gramatiky.

Základný princíp výsledného parsera je pomerne jednoduchý. Jedná sa o lineárne prehľadávanie tabuľky všetkých možných podmienok. Tento spôsob bol uprednostnený pred konečným automatom hlavne z toho dôvodu, že jazyk je syntakticky jednoduchý a jednotlivé podmienky často nemajú spoločný žiadny vzor. Takže základom parsera sa stala tabuľka obsahujúca zoznam všetkých podmienok, kde každá podmienka obsahuje zoznam kľúčových slov na konkrétnej pozícii, zástupný symbol pre každú užívateľsky zadanú hodnotu a celkový počet symbolov, ako jednoduchú optimalizáciu pre zrýchlenie prehľadávania tabuľky.

```
‘‘STR_EQUAL’’: [3, ‘‘NAK’’, ‘‘IS’’, ‘‘NAK’’],
‘‘STR_NOT_EQUAL’’: [4, ‘‘NAK’’, ‘‘IS’’, ‘‘NOT’’, ‘‘NAK’’],
```

Obrázek 5.1: Príklad ukážky dvoch záznamov z tabuľky podmienok. “NAK” je zástupný symbol.

```
{'PRECOND': [
    {'EXP_ID1': ['data1', 'data2']},
    {'EXP_ID2': ['data3']}],
 'POSTCOND': [
    {'EXP_ID3': ['data4', 'data5']}]}
```

Obrázek 5.2: Príklad štruktúry reprezentujúcej vstupný súbor.

Samotné prehľadávanie tabuľky a hľadanie správneho výrazu je celkom priamočiare. Každý riadok vo vstupnom súbore, ak to nie je názov sekcie, komentár alebo prázdny riadok sa rozdelí na symboly a porovnajú sa kľúčové slová s každou podmienkou v tabuľke, ktorá obsahuje taký istý počet symbolov ako hľadaný výraz. Ak sa taká podmienka v tabuľke nájde, identifikátor výrazu sa spolu s extrahovanými parametrami uloží do štruktúry reprezentujúcej vstupný súbor. Zoznam všetkých aktuálne implementovaných podmienok je možné nájsť v prílohe B.

Práca s premennými v tomto jazyku je oproti bežným jazykom zjednodušená. Premennú je možné definovať, ale nie je možné zmeniť jej hodnotu. Vďaka tomu je možné ihneď po vytvorení štruktúry reprezentujúcej vstupný súbor nahradiť väčšinu výskytov premenných ich hodnotami. Jedinú výnimku tvoria premenné, ktorých hodnota sa načítava zo súboru. Kvôli týmto premenným sa do výsledného skriptu pridáva funkcia, ktorá nahradzuje premenné hodnotami až tam, keďže hodnota pri preklade nie je známa. Ďalšia zložitejšia vec, ktorú bolo treba vyriešiť bola viditeľnosť premenných. Tá je definovaná tak, že premenné definované v časti `Preconditions` platia aj v časti `Postconditions` ak tam nie sú predefinované. Premenné definované v časti `Postconditions` platia iba v tejto časti. Ďalej treba spomenúť, že aj po nahradení premenných hodnotami vo všetkých podmienkach, je nutné si pôvodné definície premenných uložiť v štruktúre na ich pôvodnom mieste. Môžu mať totiž váhu neskôr pri hľadaní riešenia.

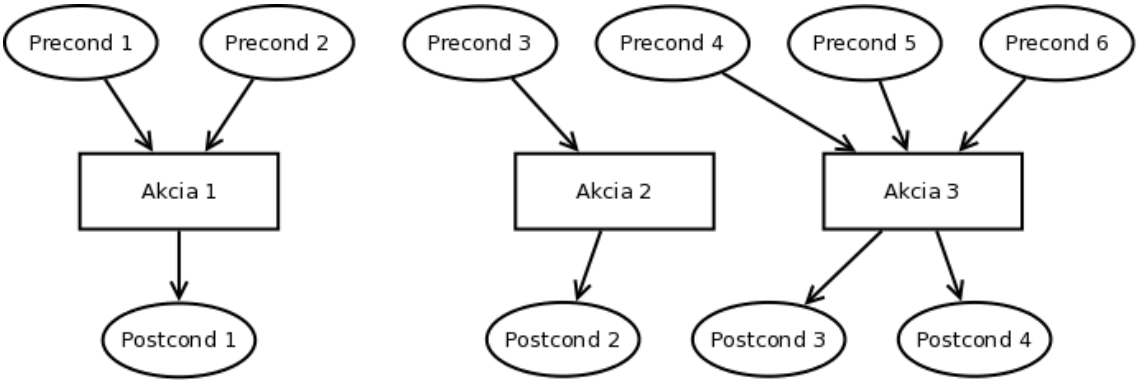
5.2 Hľadanie riešenia

Ako už bolo niekoľkokrát spomenuté, vstupný program jazyka je rozdelený na dve časti. Parsovanie vstupného súboru prebehlo rovnako pre obe časti, ale riešenie sa hľadá iba pre podmienky v časti druhej.

Hľadanie riešenia programu spočíva v prehľadávaní kombinácií dostupných akcií tak, aby pre všetky vybrané akcie v danej kombinácii bol dostupný platný vstup, a aby výsledok tejto kombinácie akcií modifikoval cieľový systém spôsobom, aby všetky dodané podmienky v druhej časti programu boli splnené.

Zoznam všetkých dostupných akcií v jazyku je možné nájsť v prílohe C. Na prvý pohľad je zrejmé, že nie všetky akcie spolu navzájom súvisia. Preto bolo možné tieto akcie rozdeliť do podskupín tak, aby jedna podskupina obsahovala iba akcie, ktoré spolu aspoň čiastočne súvisia. Tieto skupiny nazývame kontexty a momentálne ich v jazyku existuje 10, pričom každá podmienka patrí práve do jedného kontextu. Po spracovaní vstupného súboru a rozdelení dostupných podmienok do kontextov sa kontexty delia ďalej, do lokálnych kontextov. Lokálny kontext obsahuje podmienky z jedného kontextu a súčasne iba také podmienky, ktoré určitým spôsobom spolu zdieľajú dáta. Rozdelením podmienok do niekoľkých skupín sa zmenšila časová a pamäťová náročnosť hľadania riešenia, avšak pribudli určité obmedzenia, s ktorými sa je nutné vysporiadať.

Okrem už spomenutého kontextu, každá akcia obsahuje zoznam vstupných a výstupných



Obrázek 5.3: Ukážka mapovania podmienok na akcie na dvoch príkladoch.

podmienok, s ktorými dokáže pracovať. Takýchto zoznamov môže byť viac, keďže niektoré akcie spĺňajú viacej podmienok súčasne. Napríklad akcia presunutia súboru alebo adresára. Jej výsledkom nie je iba to, že súbor sa nachádza na inom mieste, ale aj to, že súbor na pôvodnom mieste už neexistuje. Medzi ďalšie informácie, ktoré obsahujú niektoré akcie, patria zoznamy zakázaných podmienok. Ak lokálny kontext obsahuje niektorú zo zakázaných podmienok práve spracovávanej akcie, tak táto akcia sa zamietne a ako riešenie sa ďalej neuvažuje. Ako príklad môžem uviesť akciu vytvorenia súboru. Ak vstupný program obsahuje podmienku, že daný súbor nie je prázdny v časti `Preconditions`, akcia vytvorenia súboru nedáva zmysel a ihneď sa môže pokračovať ďalšou akciou.

Akcia ďalej obsahuje samostatné dáta s ktorými pracuje a štruktúru reprezentujúcu požiadavky a výsledok akcie. Požiadavky a výsledok majú rovnaký formát a porovnávajú sa medzi sebou počas prehľadávania akcií pri hľadaní riešenia. Od vstupných a výstupných podmienok sa odlišujú v tom, že nereprezentujú to čo podmienka môže vykonať, ale čo vyžaduje a vyprodukuje. Požiadavky a výsledok sú vždy buď kladné alebo záporné. Napríklad akcia vytvorenia súboru má požiadavku, aby súbor ktorý sa bude vytvárať ešte neexistoval. Naopak, jej výsledok je, že vytváraný súbor existuje. Zložitejší príklad je možné nájsť pri práci s obsahom súborov, keď sa kontroluje celková zmena obsahu voči zmene čiastočnej.

Požiadavky a výsledok sa používajú dvoma spôsobmi. Prvý je v rámci lokálneho kontextu, keď sa kontroluje splniteľnosť každého kontextu samostatne. Druhý spôsob je na konci hľadania riešenia, keď sa kontroluje celková integrita a splniteľnosť výsledku.

Ďalší problém, ktorý vznikol rozdelením akcií do kontextov a lokálnych kontextov sú chýbajúce závislosti medzi jednotlivým kontextami. Na vyriešenie tohto problému bolo nutné pridať do každého kontextu možné závislosti. Napríklad, kontext zaoberajúci sa obsahom súboru vyžaduje aby sa vykonal až po kontexte ktorý sa zaoberá polohou súboru. Tieto závislosti sa nachádzajú na vyššej úrovni ako požiadavky a výsledok akcií, a vo výsledku sa dopĺňajú.

Takže príprava pred hľadaním riešenia v podobe správnej kombinácie akcií vyzerá asi nasledovne. Výsledok syntaktického analyzátora, (štruktúra zobrazená v príklade 5.2), sa najprv rozdelí podľa náležitosti podmienok do kontextov na niekoľko menších štruktúr. V týchto štruktúrach sa následne zanalyzujú spracovávané dáta. Najprv sa zozbierajú dáta v deklaráciách premenných a potom zvyšok, ktorý sa nachádza v ostatných podmienkach. Na základe takto zozbieraných dát sa vytvoria lokálne kontexty podľa toho, ako dáta spolu súvisia. Po rozdelení dát sa do lokálnych kontextov pridajú aj samotné podmienky pracujúce s týmito dátami. Takto pripravené lokálne kontexty sa už iba zoradia podľa závislostí a

```
Node (value 2) action: None
    Node (value 0) action: MOVE_FDIR
    Node (value 1) action: COPY_FDIR
        Node (value 0) action: REMOVE_FDIR
```

Obrázek 5.4: Príklad zjednodušenej reprezentácie stromu možných riešení pri presúvaní súboru.

odstránia sa kontexty, ktoré neobsahujú žiadne podmienky pochádzajúce z druhej časti programu.

Riešenie sa hľadá pre každý lokálny kontext samostatne. Reprezentované je stromovou štruktúrou, kde každý uzol tohto stromu obsahuje práve jednu akciu, odkaz na rodiča, zoznam odkazov na deti a svoju hodnotu. Hľadanie konečného riešenia pozostáva z niekoľkých fáz. Najprv sa hľadá také riešenie, ktoré obsahuje iba jednu samostatnú akciu. Keď sa prehľadajú všetky akcie v tomto kroku, pokračuje sa na ďalší, bez ohľadu na to či sa nejaké odpovedajúce akcie našli alebo nie. V nasledujúcom kroku sa hľadá také riešenie, ktoré pozostáva z viacerých akcií tak, že každá akcia poskytuje iba časť riešenia a všetky nájdené akcie poskytujú úplné riešenie. Po nájdení takejto kombinácie akcií, sa jednoducho ohodnotia uzly výsledného stromu tak, aby sa preferovala najkratšia cesta k výsledku. Po ohodnotení sa prekontroluje integrita nájdenej cesty, teda či nájdené riešenie vôbec je reálne a pokračuje sa generovaním výsledného skriptu.

Môže sa stať, že aj po ohodnotení stromu zostanú dve kandidátne akcie rovnocenne ohodnotené. Preto existuje ešte jeden spôsob ako vytvoriť výsledné riešenie. Generátor podporuje generovanie výsledného skriptu pre takzvané kombinované akcie, čo znamená, že je možné vytvoriť šablónu pre určitú kombináciu akcií, ktorá sa použije pri výslednom generovaní. Momentálne v jazyku existujú iba dve takéto šablóny, ktoré sa použijú ak nie je možné rozhodnúť či sa má súbor vytvoriť alebo kopírovať a či sa má obsah súboru zmazať, alebo vytvoriť nový. Samozrejme je opäť možné takéto akcie rozšíriť o ľubovoľný počet.

5.3 Generátor cieľového kódu a výsledný skript

Výsledný skript sa generuje dvoma spôsobmi. Najprv sa vygenerujú podmienky pre prvú časť vstupného programu. Táto časť je relatívne priamočiara, pre každú podmienku v časti `Preconditions` sa vygeneruje jedna testovacia podmienka v skripte. Ak testovaná podmienka nie je platná, poznačí sa tento stav a po kontrole všetkých ostatných podmienok z prvej časti programu sa skript ukončí.

Výsledný kód pre akcie je o niečo zložitejší. Každá podmienka má svoju šablónu uloženú v súbore `actions.dat`. Pri generovaní tejto časti skriptu sa prechádza vygenerovaný strom akcií po ohodnotenej ceste a dáta z týchto akcií sa dopĺňajú do pripravených šablón. Všetky šablóny jazyka sú navrhnuté tak, aby brali do úvahy nejednoznačnosť jazyka. To znamená, že pred tým než sa konkrétna akcia vykoná, prebehne určitý počet súvisiacich testov, ktoré zisťujú či je vôbec možné a bezpečné danú akciu vykonať, poprípade či už výsledok danej akcie neexistuje. Ak výsledok už existuje, skript o tom informuje užívateľa a pokračuje ďalšou akciou. Ak niektorý test zistí, že danú akciu nie je možné vykonať, skript to oznámi užívateľovi chybovou hláškou, ukončí sa a ďalej nepokračuje. Ak všetky testy prejdú úspešne, akcia sa vykoná. Avšak momentálne sú všetky šablóny pre akcie vytvorené tak, aby neboli priamo deštruktívne. Takže výsledkom akcie zmazať súbor bude iba premenovanie tohto súboru, konkrétne pridaním prípony `~`.

Kapitola 6

Zhodnotenie výsledku a ďalší vývoj

Zhodnotiť takýto druh projektu je náročné a takmer nemožné bez hodnotenia podľa rôznych kritérií. V tejto kapitole sa zameriam na dosiahnuté výsledky, pokúsim sa predložiť ukážky programov, ktoré fungujú výborne a také, pre ktoré program už nedokáže nájsť riešenie. Skúsim zhodnotiť jednoduchosť jednotlivých pravidiel z pohľadu použitia užívateľmi na základe jednoduchého prieskumu, použiteľnosť a bezpečnosť výsledných skriptov v skutočnom prostredí, a možnosti akými by sa mohol vývoj tohto jazyka ďalej uberať.

6.1 Dosiahnuté výsledky

Dosiahnuté výsledky je možné považovať za dobré. Vstupné programy je možné rozdeliť do štyroch kategórií podľa nájdeného výsledku. Programy, pre ktoré je možné riešenie nájsť a prekladač ho nájde, programy, pre ktoré sa riešenie nájde, avšak nemusí byť také aké by si užívateľ želal, programy, ktoré riešenie mať môžu a nemusia pretože sú príliš vágne zadané a programy ktoré riešenie nemajú, napríklad preto, že niektoré podmienky si odporujú, alebo podobne. V pokročilej fáze implementácie, keď začínalo byť najdôležitejšie testovanie sa ukázalo, že viac než samotný algoritmus hľadania riešenia, ktorý je relatívne všeobecný, je dôležité nastavenie samotných akcií. Ladenie, aké vstupy ktorá akcia vyžaduje a aké môže poskytnúť výstupy vyžadovalo značné množstvo testovania. Vzhľadom k počtu všetkých možností, ktoré sa vo vstupnom programe môžu vyskytnúť, ani v tomto stave nie je možné povedať, že výsledok je finálny. Samozrejme, chyby sa v softvéri vyskytnúť môžu hocikedy, ale teraz mám skorej na mysli interpretovanie toho, čo užívateľ zamýšľal a toho čo zapísal. Vo výsledku významnú úlohu zohrávajú aj výsledné šablóny akcií, pretože pri niektorých kombináciách je takmer nemožné rozhodnúť o výsledku bez znalosti aktuálnej konfigurácie. Pripomínam, že prekladač bol navrhnutý tak, aby výsledné skripty boli nezávislé na systéme na ktorom bol skript vytvorený, takže pri preklade sa aktuálna konfigurácia systému vôbec neuvažuje.

Zjednodušene povedané, jazyk sa zaoberá tromi skupinami konfigurácie. Prvá skupina sa zaoberá prácou so súbormi a adresármi, ich vytváraním, presúvaním a mazaním. Do druhej skupiny patrí práca s obsahom textových súborov. Posledná skupina má na starosti vybrané súčasti operačného systému. V ďalšej časti tejto podkapitoly sa budem venovať jednotlivým týmto skupinám samostatne.

6.1.1 Práca so súbormi a adresármi

Práca so súbormi v jazyku má pravdepodobne najväčšiu rozmanitosť a preto v nej môžu vznikať najmenej jednoznačné časti programu. Podmienky pracujúce so súbormi sa podarilo navrhnúť tak, aby boli blízke prirodzenému jazyku a aby sa s nimi pracovalo intuitívne.

Najjednoduchšie vytvorenie prázdneho súboru sa zapíše v jazyku jedinou podmienkou (príklad 6.1). Skopírovanie súboru do novo-vytvoreného adresára je možné vidieť na príklade 6.2. Príklad 6.3 zobrazuje presunutie súboru do adresára, avšak pri lepšom preskúmaní tohto príkladu je zrejmé, že by sa dal vysvetliť aspoň dvomi spôsobmi. Druhý spôsob by nemusel mať riešenie, keďže je možné na dve podmienky v druhej časti programu pozerať ako na protichodné. Podmienky a akcie v jazyku sú avšak nastavené tak, aby nájdené riešenie bolo vždy čo najmenej deštruktívne.

```
preconditions:
postconditions:
    file f abc
```

Obrázek 6.1: Príklad najjednoduchšieho vytvorenia súboru.

```
preconditions:
    file f abc
postconditions:
    directory d ddd
    f is placed in d
```

Obrázek 6.2: Príklad skopírovania súboru do adresára.

```
preconditions:
    file f abc
postconditions:
    f is placed in ddd
    f does not exist
```

Obrázek 6.3: Príklad presunutia súboru do adresára.

Ďalší príklad, kedy sa jazyk správa prirodzene je viacnásobné opakovanie akcie, aj ak sa opakuje iba jedna podmienka z celej sady. Výsledkom príkladu 6.4 budú dve akcie, a to kopírovanie a presunutie súboru.

Jednoduchý príklad nesplniteľného programu môže byť príklad 6.5. Podľa zadaných podmienok by riešenie pozostávalo z vytvorenia nového adresára, avšak konkrétne v tomto prípade hľadanie zlyhá preto, lebo sa nepodarí rozhodnúť či `abc` je súbor alebo adresár.

6.1.2 Práca s obsahom súborov

Väčšina podmienok pracujúcich sú súbormi je závislá na programe *Augeas*. Podmienky zaoberajúce sa s celým obsahom súboru, alebo iba s jeho časťou, ak nezáleží na tom kde v súbore sa táto časť nachádza, *Augeas* nevyužívajú a preto tieto podmienky nie sú nijak obmedzené. Naopak, *Augeas* určité obmedzenia má. Jedným z nich je to, že každý súbor ktorý chceme editovať pomocou tohto nástroja, musí ním byť priamo podporovaný. Na druhej strane, väčšina dôležitých systémových súborov podporovaná je a pridanie podpory pre nový súbor je veľmi jednoduché. Taktiež, takýto prístup je nutný ak sa vyžaduje presná a

```
preconditions:
    file f abc
postconditions:
    f is placed in dir1
    f is placed in dir2
    f does not exist
```

Obrázek 6.4: Príklad presunutia súboru do dvoch adresárov.

```
preconditions:
    file f abc
postconditions:
    f is directory
```

Obrázek 6.5: Príklad bez riešenia.

jednotná práca so zložitými konfiguračnými súbormi. Z tohto vyplýva, že toto obmedzenie nie je možné brať ako jasnú nevýhodu. Druhé obmedzenie, ktoré *Augeas* prináša súvisí s užívateľským rozhraním. Keďže každý súbor upravovaný týmto nástrojom je reprezentovaný ako strom, každá upravovaná časť súboru je reprezentovaná cestou v tomto strome. Na toto obmedzenie už je možné pozerať ako na čiastočnú nevýhodu, pretože aj keď na jednej strane sa tento problém netýka všetkých súborov, a väčšinou je tento formát zrejmý pri nahliadnutí do upravovaného konfiguračného súboru, reprezentácia niektorých súborov nie je veľmi intuitívna. Na príklade 6.6 je zobrazená reprezentácia časti súboru `/etc/nsswitch.conf`. Ďalšia nevýhoda ktorá sa môže vyskytnúť pri niektorých súboroch je, že ak chceme upraviť premennú v súbore, ktorá tam ešte neexistuje, je nutné ju najprv vytvoriť. Dôsledok je taký, že modifikácia takéhoto súboru sa neuloží. Teoreticky je možné túto nevýhodu vyriešiť upravením výsledného kódu tak, že pred tým než sa vykoná úprava takejto premennej, tak sa skontroluje či požadovaná premenná sa v súbore nachádza a ak nie, najprv sa tam vytvorí.

```
/files/etc/nsswitch.conf/database[5] = ‘‘bootparams’’
/files/etc/nsswitch.conf/database[5]/service[1] = ‘‘nisplus’’
/files/etc/nsswitch.conf/database[5]/reaction
/files/etc/nsswitch.conf/database[5]/reaction/status = ‘‘NOTFOUND’’
/files/etc/nsswitch.conf/database[5]/reaction/status/action = ‘‘return’’
/files/etc/nsswitch.conf/database[5]/service[2] = ‘‘files’’
```

Obrázek 6.6: Príklad reprezentácie časti súboru `/etc/nsswitch.conf` v programe *augeas*.

Na príklade 6.7 je zobrazená jednoduchá práca s obsahom súboru. Výsledkom bude nahradenie obsahu súboru reťazcom `“hello world”`. Druhá podmienka nebude mať žiadny praktický efekt, keďže slovo `“hello”` sa v súbore nachádzať určite bude.

Hľadanie riešenia pre príklad 6.8 skončí bez výsledku, pretože hoci je tento príklad podobný s príkladom 6.7, podmienky v časti `Postconditions` si navzájom odporujú.

Aj keď *Augeas* má svoje nevýhody, väčšinu konfiguračných súborov je možné editovať podobne ako v príklade 6.9, teda relatívne prirodzene a intuitívne.

6.1.3 Práca s vybranými časťami operačného systému

Podmienky patriace do tejto časti existujú v jazyku hlavne preto, aby užívateľovi uľahčili prácu s najvýznamnejšími časťami operačného systému. Vzhľadom k tomu, že tieto pod-

```
preconditions:
    file f abc
postconditions:
    f content is ‘‘hello world’’
    f content contains ‘‘hello’’
```

Obrázek 6.7: Príklad jednoduchej práce s obsahom súboru.

```
preconditions:
    file f abc
postconditions:
    f content is ‘‘hello world’’
    f content contains ‘‘calendar’’
```

Obrázek 6.8: Príklad jednoduchej práce s obsahom súboru bez riešenia.

mienky viac menej iba obaľujú určité príkazy, sú samy o sebe celkom jednoduché a dobre zrozumiteľné. Jediná výhrada by mohla byť k správe služieb. Na takejto úrovni abstrakcie by možno bolo lepšie, ak by správa služieb obsahovala iba jednu podmienku, kde by boli obe akcie spojené. Takže by bolo možné službu iba spustiť a povoliť, alebo iba zastaviť a zakázať.

6.2 Prieskum použiteľnosti

Jednoduchosť a užívateľskú prívetivosť jazyka je možné overiť iba prieskumom medzi skutočnými užívateľmi. Pripravil som zadania dvoch príkladov, ktoré som spolu s prehľadom jazyka poskytol piatim dobrovoľníkom, ktorí mali stredné až pokročilé znalosti základnej práce s počítačom. Niektorí z nich boli programátori, ale nikto z nich nemal veľmi pokročilé znalosti zo skriptovacieho jazyka *Bash*. Ich úlohou bolo vytvoriť dva programy, ktoré spĺňali požadované zadanie, a ku každému príkladu vyplniť krátky dotazník zameraný na prínos a prívetivosť jazyka. Dotazník sa nachádza v prílohe G.

6.2.1 Prvý testovací príklad

V prvom príklade mal užívateľ za úlohu vytvoriť skript na zálohu *MySQL* databáze, ktorý musí bežať pravidelne každý deň. Vstupné podmienky boli iba dostupný a bežiaci *MySQL* server. Príklad 6.10 zobrazuje vzorové riešenie. Názov balíka, služby a obsah skriptu bol užívateľom dostupný.

6.2.2 Druhý testovací príklad

V druhom príklade mal užívateľ za úlohu nainštalovať balíky `php` a `phpmyadmin` na server ak tam bežia služby *mysqld* a *httpd*, zvýšiť hodnotu premennej `memory_limit` v súbore `/etc/php.ini`, vytvoriť konfiguračný súbor editora *vim*, vytvoriť dvoch nových užívateľov a obom nakopírovať predtým vytvorený konfiguračný súbor `.vimrc`. Príklad 6.11 opäť zobrazuje jeden z rôznych možných vzorových riešení. Užívatelia mali dostupné názvy balíkov a obsah konfiguračného súboru `.vimrc`.

```
preconditions:
    file f /etc/php.ini
postconditions:
    memory_limit is 256MB in f in PHP
```

Obrázek 6.9: Príklad konfigurácie *PHP*.

```
preconditions:
    mysql-server package is installed
    mysqld is running
    file backup /var/backup/mysql-script
postconditions:
    backup exists
    backup is executable
    backup content is ‘‘mysqldump --all-databases >/var/backup/mysql.xml’’
    backup is executed daily
```

Obrázek 6.10: Príklad vzorového riešenia prvého príkladu.

6.2.3 Výsledok prieskumu

Ukázalo sa, že najslabší článok počas testovania bol poskytnutý prehľad jazyka. Užívatelia dostali k dispozícii iba stručný prehľad jazyka dostupný v prílohe D, avšak to nie je dobrý zdroj učenia sa jazyka. Na lepšie učenie jazyka slúži kapitola 4. Kvôli tomuto nedostatku mali užívatelia čiastočne problém s prvým príkladom, keď nevedeli jazyk správne uchopiť bez funkčného príkladu a aspoň čiastočného popisu niektorých podmienok, práce s premennými a podobne. Avšak aj s týmto znevýhodnením traja dokázali vytvoriť prvý program správne. Druhý príklad išiel všetkým užívateľom oveľa jednoduchšie a hoci je dlhší, väčšina ho mala dokončený skorej ako prvý.

Kladné výsledky prieskumu je možné zhrnúť takto. Osvedčila sa strmá krivka učenia jazyka, užívatelia už po prvom príklade dobre porozumeli princípu jazyka a spôsobu jeho použitia. Užívatelia taktiež dokázali oceniť jednu sadu podmienok použitú v oboch častiach programu, podľa spätnej väzby je to intuitívne a prirodzené. Menej skúsení užívatelia ocenili zjednodušenú a jednotnú prácu s balíčkami, službami a cronom v jazyku. Iba jeden užívateľ by si zvolil vyriešiť zadané príklady v *Bashi*, zvyšní užívatelia by si zvolil prezentovaný jazyk.

Na druhú stranu, niektorí užívatelia mali problémy s premennými a ich použitím v jazyku, no za toto negatívum je možné čiastočne označiť nevhodné materiály na učenie. Druhým najvýznamnejším problémom sa vyskytlo rozdelenie práce so súbormi a obsahom súborov. Pri vytváraní súborov iba jeden užívateľ použil podmienku na existenciu súboru, ostatní použili definíciu premennej v druhej časti programu, a iba nastavili obsah súboru. Prekladač je schopný nájsť takéto riešenie, avšak bez definície premennej by výsledný skript po spustený skončil s chybou o neexistujúcom súbore. Riešením tohto nedostatku by mohlo byť lepšie previazanie akcií stojacimi za správou súborov a ich obsahov, avšak v takom stave ako je jazyk aktuálne navrhnutý by táto úprava nebola triviálna.

6.3 Návrhy na ďalší vývoj

Medzi hlavné vylepšenia do ďalších verzií by som zaradil vylepšenie rozšíriteľnosti podmienok a akcií. V aktuálnej verzii sú síce podmienky a akcie relatívne jednoducho rozšíriteľné,

```
preconditions:
    mysqld is running
    httpd is running
    file vimrc ~/.vimrc
postconditions:
    php package is installed
    phpmyadmin package is installed
    memory_limit is 256M in /etc/php.ini
    vimrc content contains ‘‘syntax on\nset number’’
    user1 is user
    user2 is user
    vimrc is placed in /home/user1
    vimrc is placed in /home/user2
```

Obrázek 6.11: Príklad vzorového riešenia druhého príkladu.

avšak ich dáta sa nachádzajú priamo v zdrojových súboroch prekladača. Vylepšenie by bolo, ak by sa tieto dáta nachádzali v samostatných dátových súboroch. Takto by sa správa a rozšíriteľnosť značne vylepšila.

Ďalšie technické vylepšenie sa týka vygenerovaného skriptu. Momentálne sú všetky akcie v skripte nastavené tak, aby boli čo najmenej deštruktívne, teda pri mazaní sa v skutočnosti súbor iba premenuje a podobne. Akcie, ktorých výsledok je deštruktívny, by mohli obsahovať dve varianty, a teda skutočnú verziu a jej bezpečnú variantu. Výsledný skript by potom mohol podporovať prepínač, ktorý by medzi týmito dvoma variantami mohol prepínať, pričom základná verzia by bola bezpečná, a ak by užívateľ chcel vykonať deštruktívne zmeny v systéme naozaj, použil by novo-podporovaný prepínač.

Kapitola 7

Záver

Podarilo sa mi zanalyzovať rôzne spôsoby konfigurácie Linuxových operačných systémov a nástrojov, pomocou ktorých je možné túto konfiguráciu zjednodušiť. Získané poznatky je možné nájsť v kapitole 2, spolu s popisom konfigurácie operačných systémov rodiny Windows. Na základe týchto poznatkov som následne navrhol nový jazyk na popis konfigurácie operačného systému.

Jazyk je určený pre užívateľov, ktorí si chcú uľahčiť konfiguráciu operačného systému, avšak nechcú nasadzovať komplexné riešenia do svojej počítačovej infraštruktúry. Výsledok prekladu jazyka je skript v skriptovacom jazyku *Bash*, ktorý je prenositeľný v rámci svojho druhu operačného systému, s minimálnymi závislosťami na externých knižniciach. Vďaka tomuto prístupu je možné pomocou tohto jazyka nakonfigurovať i práve nainštalovaný systém podľa ľubovolných potrieb.

V kapitole 3 sa nachádza popis jazyka a základný návrh implementácie. Program jazyka sa skladá z dvoch častí, `Preconditions` a `Postconditions`. Obe časti obsahujú sadu podmienok, ktoré sa v prvej časti vyhodnocujú a podľa ktorých sa v druhej časti programu systém konfiguruje. Ako jediná závislosť výsledného skriptu je program *Augeas*, ktorý slúži na abstrakciu formátu konfiguračných súborov. Vďaka nemu môžu výsledné skripty pristupovať ku všetkým konfiguračným súborom rovnako, čo prispieva k ich prehľadnosti a zrozumiteľnosti.

Podrobný popis jazyka a jeho použitia sa nachádza v kapitole 4. V kapitole 5 je možné nájsť všetky implementačné detaily a popis algoritmu použitého k nájdeniu konečného riešenia vstupných programov. Zhrnutie výsledkov a výsledný prieskum sa nachádza v kapitole 6.

Počas tejto práce sa mi podarilo vytvoriť nový funkčný jazyk určený na užívateľsky prívetivú konfiguráciu operačného systému. Myslím si, že výsledok sa podaril, čo dokazuje aj výsledok finálneho prieskumu použiteľnosti. Ďalšie smerovanie vo vývoji jazyka by sa malo zamerať na rozšírenie funkcionality vygenerovaného skriptu, zlepšenie šablón výsledných akcií a pridanie podpory pre ďalšie Linuxové operačné systémy.

Prvý bod zadania je splnený v kapitole 2, druhý a tretí bod zadania je možné nájsť v kapitole 3. Štvrtý bod zadania je splnený v kapitole 5. Príloha F obsahuje demonštračný príklad použitia jazyka na konfigurácii poštového serveru, a teda splnený piaty bod zadania.

Literatura

[1] Understanding Unix Concepts. 1996, [Online], [cit. 2012-11-15].
URL http://www.cs.bgu.ac.il/~arik/usail/concepts/toc.html

[2] CFEngine. 2013, [Online], [cit. 2012-12-15].
URL http://cfengine.com/

[3] Debian – Details of package aptitude. 2013, [Online], [cit. 2013-03-20].
URL http://packages.debian.org/experimental/aptitude

[4] freedesktop.org - Software/systemd. 2013, [Online], [cit. 2013-03-20].
URL http://www.freedesktop.org/wiki/Software/systemd

[5] Webmin. 2013, [Online], [cit. 2012-12-15].
URL http://www.webmin.com/

[6] Yum Package Manager. 2013, [Online], [cit. 2013-03-20].
URL http://yum.baseurl.org/

[7] Esposito, D.: Windows 2000 Registry: Latest Features and APIs Provide the Power to Customize and Extend Your Apps. 2000, [Online], [cit. 2012-12-12].
URL http://msdn.microsoft.com/en-us/magazine/bb985037.aspx

[8] Lueninghoener, C.: Getting Started with Configuration Management. 2011, [Online], [cit. 2012-12-15].
URL https://www.usenix.org/system/files/login/articles/105457-Lueninghoener.pdf

[9] Microsoft: Registry (Windows). 2012, [Online], [cit. 2012-12-12].
URL http://msdn.microsoft.com/en-us/library/ms724871.aspx

[10] Raymond, E. S.: The Art of Unix Programming. 2003, [Online], [cit. 2012-11-20].
URL http://www.faqs.org/docs/artu/

[11] Red Hat: Augeas. 2013, [Online], [cit. 2012-12-20].
URL http://augeas.net/

[12] Red Hat: Augeas - Stock lenses. 2013, [Online], [cit. 2012-12-20].
URL http://augeas.net/stock_lenses.html

[13] Russell, R.; Quinlan, D.; Yeoh, C.: Filesystem Hierarchy Standard. 2004, [Online], [cit. 2012-11-22].
URL http://www.pathname.com/fhs/pub/fhs-2.3.html

[14] Russinovich, M. E.; Solomon, D. A.: *Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server 2003, Windows XP, and Windows 2000*. Microsoft Press, 2005, ISBN 978-0-7356-1917-3.

[15] Sheppard, D.: Beginner's Introduction to Perl. 2000, [Online], [cit. 2012-12-10]. URL http://www.perl.com/pub/2000/10/begperl1.html

[16] Subodh, S.: Understanding Linux configuration files. 2001, [Online], [cit. 2012-11-30]. URL http://www.ibm.com/developerworks/linux/library/l-config/index.html

Dodatek A

Obsah CD

CD obsahuje správu v formáte pdf, zdrojové súbory k vytvoreniu tejto správy, zdrojové súbory prekladača a všetky príklady z tejto správy v samostatných súboroch a k nim vygenerované výsledné skripty.

Dodatek B

Sada implementovaných testov

Reťazce

Reťazce sú rovnaké. / Reťazce nie sú rovnaké.
Reťazec obsahuje podreťazec. / Reťazec neobsahuje podreťazec.

Súbory

Súbor (alebo adresár) existuje. / Súbor (alebo adresár) neexistuje.
Súbor (alebo adresár) je súbor. / Súbor (alebo adresár) nie je súbor.
Súbor (alebo adresár) je adresár. / Súbor (alebo adresár) nie je adresár.
Súbor (alebo adresár) je prázdny. / Súbor (alebo adresár) nie je prázdny.
Súbor (alebo adresár) je symbolický odkaz na iný súbor (alebo adresár). / Súbor (alebo adresár) nie je symbolický odkaz na iný súbor (alebo adresár).
Vlastníkom súboru (alebo adresára) je. / Vlastníkom súboru (alebo adresára) nie je.
Skupina súboru (alebo adresára) je. / Skupina súboru (alebo adresára) nie je.
Súbor (alebo adresár) má práva.
Súbor (alebo adresár) je možné čítať. / Súbor (alebo adresár) nie je možné čítať.
Do súboru (alebo adresára) je možné zapisovať. / Do súboru (alebo adresára) nie je možné zapisovať.
Súbor (alebo adresár) je možné spustiť. / Súbor (alebo adresár) nie je možné spustiť.
Súbor sa nachádza v adresári. / Súbor sa nenachádza v adresári.

Obsah súborov

Obsah súboru je reťazec. / Obsah súboru nie je reťazec.
Súbor obsahuje reťazec. / Súbor neobsahuje reťazec.
Súbor obsahuje reťazec v sekcii. / Súbor neobsahuje reťazec v sekcii.

Premenné v súboroch

Premenná v súbore je reťazec. / Premenná v súbore nie je reťazec.
Premenná v súbore a v sekcii je reťazec. / Premenná v súbore a v sekcii nie je reťazec.
Premenná v súbore obsahuje reťazec. / Premenná v súbore neobsahuje reťazec.
Premenná v súbore a v sekcii obsahuje reťazec. / Premenná v súbore a v sekcii neobsahuje reťazec.

Premenná sa nachádza v súbore. / Premenná sa nenachádza v súbore.
Premenná sa nachádza v súbore a v sekcii. / Premenná sa nenachádza v súbore a v sekcii.

Predvolené príkazy

Predvolený príkaz pre program je. / Predvolený príkaz pre program nie je.

Cron

Program je spustený každú hodinu.
Program je spustený každý deň.
Program je spustený každý týždeň.
Program je spustený každý mesiac.
Príkaz (alebo program) je spustený vždy v čase.

Balíčky

Balíček je nainštalovaný. / Balíček nie je nainštalovaný.

Služby

Služba beží. / Služba nebeží.
Služba je povolená. / Služba je zakázaná.

Užívatelia a skupiny

Užívateľ existuje. / Užívateľ neexistuje.
Skupina existuje. / Skupina neexistuje.
Užívateľ je v skupine. / Užívateľ nie je v skupine.
Užívateľ je administrátor. / Užívateľ nie je administrátor.

Dodatek C

Sada implementovaných akcií

Súbory

Vytvor súbor (alebo adresár).
Skopíruj súbor (alebo adresár).
Presuň súbor (alebo adresár).
Vymaž súbor (alebo adresár).
Vymaž obsah súboru (alebo adresára).
Vytvor symbolický odkaz na súbor (alebo adresár).
Nastav vlastníka súboru (alebo adresára).
Nastav skupinu súboru (alebo adresára).
Nastav práva súboru (alebo adresára).
Povoľ čítanie súboru (alebo adresára). / Zakáž čítanie súboru (alebo adresára).
Povoľ zápis do súboru (alebo adresára). / Zakáž zápis do súboru (alebo adresára).
Povoľ spustenie súboru (alebo adresára). / Zakáž spustenie súboru (alebo adresára).

Obsah súborov

Nastav obsah súboru na reťazec. / Vymaž obsah súboru ak je zhodný s reťazcom.
Pridaj do súboru. / Odstráň zo súboru.
Pridaj do súboru do sekcie. / Odstráň zo súboru zo sekcie.

Premenné v súboroch

Nastav premennú v súbore na reťazec. / Vymaž obsah premennej v súbore ak sa zhoduje s reťazcom.
Pridaj do premennej v súbore reťazec. / Odstráň z premennej v súbore reťazec.
Odstráň premennú zo súboru.

Spustenie príkazu v bashi

Spusti príkaz.

Predvolené príkazy

Nastav predvolený príkaz pre program. / Zruš predvolený príkaz pre program.

Cron

Nastav spustenie program na každú hodinu.
Nastav spustenie programu na každý deň.
Nastav spustenie programu na každý týždeň.
Nastav spustenie programu na každý mesiac.
Nastav spustenie príkazu (alebo programu) na stanovený čas.

Balíčky

Nainštaluj balíček. / Vymaž balíček.

Služby

Spusti službu. / Zastav službu.
Povoľ službu. / Zakáž službu.

Užívatelia a skupiny

Vytvor užívateľa. / Vymaž užívateľa.
Vytvor skupinu. / Vymaž skupinu.
Pridaj užívateľa do skupiny. / Odstráň užívateľa zo skupiny.
Sprav užívateľa administrátorom. / Odober užívateľovi práva administrátora.

Dodatek D

Prehľad jazyka

V tejto prílohe sa nachádza úplný prehľad jazyka. Anglické slová písané malým písmom sú kľúčové slová jazyka. Anglické slová písané veľkým písmom sa nahrádzajú užívateľským vstupom, ktorý môže byť definovaný hodnotou alebo premennou. Ak užívateľský vstup obsahuje medzery, musí byť uvedený v úvodzovkách. Časti uvedené v hranatých zátvorkách sú voliteľné.

Dátové typy

- string
- file
- directory

Definícia premenných

- DATA_TYPE VAR_NAME VALUE
  - Príklady:
    - string my_str “this is string value”
    - file f /etc/fstab
- string VAR_NAME is in FILE [in SECTION1[/SECTION2[/...]]]

Operácie s reťazcami, súbormi a adresármi

- Práca s reťazcami
  - STRING1 is [not] STRING2
  - STRING1 [does not] contain[s] STRING2
- Práca so súbormi a adresármi
  - FILE/DIR [does not] exist[s]
  - FILE/DIR is [not] file
  - FILE/DIR is [not] directory
  - FILE/DIR is [not] empty

  - FILE/DIR1 is [not] symbolic link to FILE/DIR2
  - owner of FILE/DIR is [not] STRING
  - group of FILE/DIR is [not] STRING
  - FILE/DIR is [not] readable
  - FILE/DIR is [not] writable
  - FILE/DIR is [not] executable
  - FILE is [not] placed in DIR
- Práca s obsahom súborov
  - FILE content is [not] STRING
  - FILE content [in SECTION1[/SECTION2[/...]]] [does not] contain[s] STRING
- Práca s premennými v súboroch
  - VAR is [not] STRING in FILE [in SECTION1[/SECTION2[/...]]]
  - VAR [does not] contain[s] STRING in FILE [in SECTION1[/SECTION2[/...]]]
  - VAR is [not] in FILE [in SECTION1[/SECTION2[/...]]]

Konfigurácia súčastí operačného systému

- Predvolené príkazy
  - FILE is [not] default command for STRING
- Cron
  - FILE is executed hourly|daily|weekly|monthly
  - FILE/STRING1 is executed every STRING2
- Správa balíkov
  - STRING package is [not] installed
- Správa služieb
  - STRING is [not] running
  - STRING is [not] enabled
- Správa užívateľov
  - STRING is [not] user
  - STRING is [not] group
  - STRING1 [does not] belong[s] to STRING2
  - STRING is [not] admin
- Príkaz execute
  - execute STRING

Dodatek E

Pravá regulárna gramatika jazyka

$G = (N, \Sigma, P, S)$

$N = \{S, A1, A2, A3, T, B1, B2, C1, C2, C3, C4, C5, C6, C7, C8, C9, D1, D2, E1, E2, F1, G1, G2, H1, I1, I2, J1, J2, J3, J4\}$

$\Sigma = \{string, file, directory, is, not, content, contain, contains, does, not, exist, exists, empty, symbolic, link, owner, group, of, has, permissions, readable, writable, executable, in, execute, default, command, for, executed, hourly, daily, weekly, monthly, every, package, installed, running, enabled, user, group, belong, belongs, admin, to, \_variable\text{-}name, \_string, \_file\text{-}path, \_dir\text{-}path, \_string\text{-}val, \_file\text{-}dir\text{-}val, \_file\text{-}val, \_dir\text{-}val, \_section\}$

P :
$S \rightarrow string\ \_variable\text{-}name\ \_string$
$S \rightarrow file\ A1$
$A1 \rightarrow \_variable\text{-}name\ \_file\text{-}path$
$S \rightarrow directory\ A2$
$A2 \rightarrow \_variable\text{-}name\ \_dir\text{-}path$
$S \rightarrow string\ \_variable\text{-}name\ A3$
$A3 \rightarrow is\ in\ \_file\text{-}path$
$A3 \rightarrow is\ in\ \_file\text{-}path\ in\ \_section$
$T \rightarrow \_string\text{-}val$
$S \rightarrow \_string\text{-}val\ B1$
$B1 \rightarrow is\ T$
$B1 \rightarrow is\ not\ T$
$S \rightarrow \_string\text{-}val\ B2$
$B2 \rightarrow contains\ T$
$B2 \rightarrow does\ not\ contain\ T$
$S \rightarrow \_file\text{-}dir\text{-}val\ C1$
$C1 \rightarrow exists$
$C1 \rightarrow does\ not\ exist$
$S \rightarrow \_file\text{-}dir\text{-}val\ C2$
$C2 \rightarrow is\ file$
$C2 \rightarrow is\ not\ file$
$S \rightarrow \_file\text{-}dir\text{-}val\ C3$
$C3 \rightarrow is\ directory$
$C3 \rightarrow is\ not\ directory$

$S \rightarrow \_file\text{-}dir\text{-}val\ C4$
$C4 \rightarrow is\ empty$
$C4 \rightarrow is\ not\ empty$
$S \rightarrow \_file\text{-}dir\text{-}val\ C5$
$C5 \rightarrow is\ symbolic\ link\ C9$
$C5 \rightarrow is\ not\ symbolic\ link\ C9$
$C9 \rightarrow to\ \_file\text{-}dir\text{-}val$
$S \rightarrow owner\ of\ \_file\text{-}dir\text{-}val\ B1$
$S \rightarrow group\ of\ \_file\text{-}dir\text{-}val\ B1$
$S \rightarrow \_file\text{-}dir\text{-}val\ has\ \_string\text{-}val\ permissions$
$S \rightarrow \_file\text{-}dir\text{-}val\ C6$
$C6 \rightarrow is\ readable$
$C6 \rightarrow is\ not\ readable$
$S \rightarrow \_file\text{-}dir\text{-}val\ C7$
$C7 \rightarrow is\ writable$
$C7 \rightarrow is\ not\ writable$
$S \rightarrow \_file\text{-}dir\text{-}val\ C8$
$C8 \rightarrow is\ executable$
$C8 \rightarrow is\ not\ executable$
$S \rightarrow \_file\text{-}val\ content\ D1$
$D1 \rightarrow B1$
$D1 \rightarrow in\ \_section\ B1$
$D1 \rightarrow B2$
$D1 \rightarrow in\ \_section\ B2$
$S \rightarrow \_file\text{-}val\ D2$
$D2 \rightarrow is\ in\ \_dir\text{-}val$
$D2 \rightarrow is\ not\ in\ \_dir\text{-}val$
$S \rightarrow \_string\text{-}val\ is\ E1$
$S \rightarrow \_string\text{-}val\ is\ not\ E1$
$E1 \rightarrow \_string\text{-}val\ in\ E2$
$E2 \rightarrow \_file\text{-}val$
$E2 \rightarrow \_file\text{-}val\ in\ \_section$
$S \rightarrow \_string\text{-}val\ contains\ E1$
$S \rightarrow does\ not\ contain\ E1$
$S \rightarrow \_string\text{-}val\ is\ in\ E2$
$S \rightarrow \_string\text{-}val\ is\ not\ in\ E2$
$S \rightarrow execute\ \_string\text{-}val$
$S \rightarrow \_file\text{-}val\ is\ F1$
$S \rightarrow \_file\text{-}val\ is\ not\ F1$
$F1 \rightarrow default\ command\ for\ \_string\text{-}val$
$S \rightarrow \_file\text{-}val\ is\ executed\ G1$
$G1 \rightarrow hourly$
$G1 \rightarrow daily$
$G1 \rightarrow weekly$
$G1 \rightarrow monthly$
$S \rightarrow \_file\text{-}val\ G2$
$S \rightarrow \_string\text{-}val\ G2$
$G2 \rightarrow is\ executed\ every\ \_string\text{-}val$

$S \rightarrow$ _*string-val package* $H1$
$H1 \rightarrow$ *is installed*
$H1 \rightarrow$ *is not installed*
$S \rightarrow$ _*string-val* $I1$
$S \rightarrow$ _*string-val* $I2$
$I1 \rightarrow$ *is running*
$I1 \rightarrow$ *is not running*
$I2 \rightarrow$ *is enabled*
$I2 \rightarrow$ *is not enabled*
$S \rightarrow$ _*string-val* $J1$
$S \rightarrow$ _*string-val* $J2$
$S \rightarrow$ _*string-val* $J3$
$S \rightarrow$ _*string-val* $J4$
$J1 \rightarrow$ *is user*
$J1 \rightarrow$ *is not user*
$J2 \rightarrow$ *is group*
$J2 \rightarrow$ *is not group*
$J3 \rightarrow$ *belongs to* _*string-val*
$J3 \rightarrow$ *does not belong to* _*string-val*
$J4 \rightarrow$ *is admin*
$J4 \rightarrow$ *is not admin*

Dodatek F

Príklad inštalácie poštového servera

```
preconditions:
    postfix package is installed
    file main_cf /etc/postfix/main.cf
    file head_checks /etc/postfix/header_checks
    file postfix_bin /usr/sbin/sendmail.postfix
    main_cf exists
    head_checks exists
    postfix_bin exists
    main_cf is writable
    head_checks is writable
    postfix_bin is executable
postconditions:
    myhostname is mail.server.com in main_cf
    mydomain is server.com in main_cf
    myorigin is $mydomain in main_cf
    inet_interfaces is all in main_cf
    mydestination contains ‘‘, $mydomain’’ in main_cf
    home_mailbox is Maildir/ in main_cf
    header_checks is regexp:/etc/postfix/header_checks in main_cf
    head_checks content contains ‘‘/^From:.*<#.*@.*>/ REJECT’’
    sendmail is not running
    sendmail is not enabled
    postfix_bin is default command for mta
    postfix is enabled
    postfix is running
```

Dodatek G

Dotazník prieskumu použiteľnosti jazyka

G.1 Prvý príklad

Vytvorte program v zadanom jazyku, ktorý za predpokladu že sa na cieľovom systéme nachádza a beží *MySQL* server, vytvorí nový skript, ktorý bude každý deň túto databázu zálohovať. *MySQL* balíček sa vo *Fedore* nazýva `mysql-server`, služba `mysqld`. Ako príkaz na zálohovanie databázy použite "`mysqldump -all-databases >/var/backup/mysql.xml`"

```
preconditions:
postconditions:
```

Podarilo sa vám dokončiť tento príklad?
Na stupnici od 1 do 5, kde 1 je najjednoduchšie a 5 najťažšie, ako by ste ohodnotili tento príklad z pohľadu náročnosti na jeho vytvorenie?
Čo by bolo pre vás ľahšie: vytvoriť skript v tomto jazyku, alebo obdobný skript v skriptovacom jazyku *Bash*?

G.2 Druhý príklad

Vytvorte program v zadanom jazyku, ktorý za predpokladu že sa na cieľovom systéme nachádzajú bežiace služby *MySQL* a *Apache*, nainštaluje balíky *PHP* a *phpMyAdmin*, zvýši hodnotu premennej `memory_limit` v súbore `/etc/php.ini` na `256M`, vytvorí konfiguračný súbor editora *vim* `.vimrc` obsahujúci reťazec "`syntax on\nset number`", vytvorí dvoch nových užívateľov v systéme a vytvorený konfiguračný súbor im nakopíruje do domovských adresárov. Balíčky vo *Fedore* majú názvy `mysql-server`, `httpd`, `php` a `phpmyadmin`, služby majú názvy `mysqld` a `httpd`.

```
preconditions:
postconditions:
```

Podarilo sa vám dokončiť tento príklad?
Na stupnici od 1 do 5, kde 1 je najjednoduchšie a 5 najťažšie, ako by ste ohodnotili tento príklad z pohľadu náročnosti na jeho vytvorenie?
Čo by bolo pre vás ľahšie: vytvoriť skript v tomto jazyku, alebo obdobný skript v skriptovacom jazyku *Bash*?