SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5384-64685

UNIVERZÁLNE, PLATFORMOVO NEZÁVISLÉ KONZOLOVÉ ROZHRANIE

DIPLOMOVÁ PRÁCA

2018 **Bc. Juraj Vraniak**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5384-64685

UNIVERZÁLNE, PLATFORMOVO NEZÁVISLÉ KONZOLOVÉ ROZHRANIE

DIPLOMOVÁ PRÁCA

Študijný program:	Aplikovaná informatika
Číslo študijného odboru:	2511
Názov študijného odboru:	9.2.9 Aplikovaná informatika
Školiace pracovisko:	Ústav informatiky a matematiky
Vedúci záverečnej práce:	RnDr. Igor Kossaczký, CSc.
Konzultant:	Rndr. Peter Praženica, Ing. Gabriel Szabó

Bratislava 2018 **Bc. Juraj Vraniak**

Základné údaje

Typ práce:	Diplomová práca
Názov témy:	**Akupunktúrne body – hľadanie, meranie a zobrazovanie**
Stav prihlásenia:	schválené
Stav témy:	schválené (prof. Dr. Ing. Miloš Oravec - Garant študijného programu)
Vedúci práce:	doc. Ing. Marek Kukučka, PhD.
Fakulta:	Fakulta elektrotechniky a informatiky
Garantujúce pracovisko:	Ústav informatiky a matematiky - FEI
Max. počet študentov:	1
Akademický rok:	2016/2017
Navrhol:	doc. Ing. Marek Kukučka, PhD.
Abstrakt:	Akupunktúra patrí medzi najstaršie liečebné praktiky sveta a je to jedna z kľúčových častí tradičnej čínskej medicín Keďže použitie liečebných metód odvodených od akupunktúry je stále viac rozšírené, z medicínskeho pohľadu nanajvýš aktuálne venovať sa základnému výskumu v tejto oblasti a pokúsiť sa objasniť základné fyziologick a biofyzikálne mechanizmy stojace za preukázanými klinickými efektami. Z prehľadu publikovaných elektrickýc vlastností akupunktúrnych bodov a dráh vyplýva potreba dôsledného overenia hypotézy elektrickej rozoznateľnos akupunktúrnych štruktúr. Očakáva sa, že môžu mať nižšiu impedanciu a vyššiu kapacitu oproti okolitým kontrolný bodom na pokožke. Výstupom mapovania pokožky budú 2D a 3D napäťové/impedančné mapy z povrchu tela. S tým prístupom bude možné nielen lokalizovať prípadný akupunktúrny bod, ale aj študovať jeho ohraničenie, povrchov elektrickú štruktúru či jeho veľkosť. Súčasťou výskumu je aj realizovanie meraní závislosti impedancie od frekvenc v akustickom pásme frekvencií 100 Hz – 20 kHz a vplyvu rôznych parametrov na rozoznávanie pozície, tva a štruktúry akupunktúrnych bodov.

Obmedzenie k téme

Na prihlásenie riešiteľa na tému je potrebné splnenie jedného z nasledujúcich obmedzení

Obmedzenie na študijný program

Tabuľka zobrazuje obmedzenie na študijný program, odbor, špecializáciu, ktorý musí mať študent zapísaný, aby sa mohol na danú tému

Program	Zameranie	Špecializácia
I-API aplikovaná informatika	I-API-MSUS Modelovanie a simulácia udalostných systémov	-- *nezadané* --
I-API aplikovaná informatika	I-API-ITVR IT v riadení a rozhodovaní	-- *nezadané* --

Obmedzenie na predmety

Tabuľka zobrazuje obmedzenia na predmet, ktorý musí mať študent odštudovaný, aby sa mohol na danú tému prihlásiť.

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Bc. Juraj Vraniak
Diplomová práca:	Univerzálne, platformovo nezávislé konzolové rozhranie
Vedúci záverečnej práce:	RnDr. Igor Kossaczký, CSc.
Konzultant:	Rndr. Peter Praženica, Ing. Gabriel Szabó
Miesto a rok predloženia práce:	Bratislava 2018

Práca sa venuje problematike programovacích jazykov, ich syntaxea a dačo snáď vymyslím.
Kľúčové slová: programovacé jazyk, analýza prekladu, prekladač, plugin, architektúra,

návrhové vzory

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Bc. Juraj Vraniak
Master's thesis:	Universal, platform independent console interface
Supervisor:	RnDr. Igor Kossaczký, CSc.
Consultant:	Rndr. Peter Praženica, Ing. Gabriel Szabó
Place and year of submission:	Bratislava 2018

The aim of the thesis is dedicated to programing languages, their syntax and will figure something out later.

Keywords: programing language, translation analysis, translator, plugin, architecture, dessign patterns

Vyhlásenie autora

Podpísaný Bc. Juraj Vraniak čestne vyhlasujem, že som diplomovú prácu Univerzálne, platformovo nezávislé konzolové rozhranie vypracoval na základe poznatkov získaných počas štúdia a informácií z dostupnej literatúry uvedenej v práci.

Vedúcim mojej diplomovej práce bol RnDr. Igor Kossaczký, CSc.

Bratislava, dňa 10.5.2018

.....

podpis autora

Poďakovanie

Touto cestou by som sa chcel poďakovať vedúcim práce RnDr. Igorovi Kossaczkému, CSc, Rndr. Peterovi Praženicovi, Ing. Gabrielovi Szabóovi za cenné rady, odbornú pomoc, trpezlivosť a konzultácie pri vytvorení diplomovej práce.

Obsah

Zoznam obrázkov a tabuliek

Zoznam skratiek

API Aplikačné rozhranie

Zoznam algoritmov

Úvod

V dnešnej dobe rôznorodosť operačných systémov a absencia jednotnej platformy na vytváranie skrípt vo väčšine prípadov vyžadujú ich duplikovanie alebo viacnásobnú implementáciu. Čiastočným riešením tohto problému je použitie skriptovacieho jazyka s podporou cieľových platform. Zásadným problémom skriptovacích jazykov pri riešení tohto problému je absencia syntaktických a funkčných konštrukcií, ktoré sú už overené a široko používané, ako napríklad pajpa, izolovanie príkazov alebo presmerovanie štandardného a chybového vstupu a výstupu. Cieľom práce je analyzovať populárne konzolové rozhrania (napr. Bourne Shell, Power Shell, C-Shell) a skriptovacie jazyky (napr. Python, Groovy, Lua), porovnať ich syntax, funkcionality a limity. Následne navrhnúť nové konzolové rozhranie, ktoré bude spájať funkcionality identifikované ako výhody počas analýzy so zameraním na administrátorské úlohy. Pri implementácií je tiež kľúčovým faktorom identifikácia nových funkcionalít, ktoré by mohli uľahčiť vývoj robustných skrípt. Rozhranie musí umožňovať interaktívny aj skriptovaný módus. Očakáva sa možnosť integrácie rozhrania do iných systémov rôznej veľkosti a komplexity, od malých utilít a rutín až po enterprise aplikácie a ľahká rozšíriteľnosť rozhrania o nové príkazy a funkcionality.

1 Analýza stavu

1.1 Operačné systémy

Informatika a informačné technológie je pomerne mladá vedná disciplína. Jej začiatky je možné datovať od druhej polky dvadsiateho storočia, čo momentalne predstavuje necelých sedemdesiat rokov. Za tento čas informatika zaznamenala enormný rast či vo vývoji hardwéru alebo softwéru. Operačný systém je základná časť akéhokoľvek počítaťového sytému, je to kus softvéru, ktorý umožnuje počítačom fungovať. Oblasť operačných systémov v poslednej dobe prechádza rapídnymi zmenami nakoľko počítače sa stali súčasťou každodenného života od malých zariadení napríklad v automobiloch až po najsofisitkovanejšie servery nadnárodných spoločností. Aj keď v dnešnej dobe poznáme mnohé operačné systémi, my sa zameriame na Windows, Mac OS, Unix a Linux.[3]

1.1.1 Windows

Microsoft Windows uviedol svoje prvé operačné systémy v roku 1985 ako nadstavbu MS DOS. Jeho popularita rýchlo rástla až vyvrcholila dominantným postavením na trhu v osobných počítačoch. V roku 1993 začal vydávať špecializované operačné systémy pre servery, ktoré prinášali novú funkcionalitu pre počítače používané ako servery. Pre účely automatizácie sa na Windows serveroch používajú hlavne powershell scripty, písane v rovnomennom jazyku powershell.

Predinštalovaný software Málo predinštalovaného softvéru jedine powershell na widows serveri je defaultne, ostatné programátorské tooly je potrebné stiahnúť dodatočne.

1.1.2 MacOs

Mac tiež ponúka serverovú verziu svojho operačného systému pod názvom OS X Server, ktorý začal písať svoju históriu v roku 2001, avšak neteší sa takej obľube ako Windows, unix alebo linux server. Skriptovacím jazykom pre OS X server nieje špecifický jazyk, je možné vybrať si z Pythonu, JavaSriptu, Perl, AppleScriptu, Swiftu alebo napríklad Ruby. Každý jazyk prináša určité plusy, ale zároven mínusy čo je však najpodstatnejšie nie je tam štandardizovaný skriptovací jazyk.

Predinštalovaný software Predinštalovaný software pre developerov na Mac OS je python, apple script, Ruby, bash, objective-c. Donedávna bola štandardom aj Java.

1.1.3 Unix

Patrí medzi prvé operačné systémy pre servery, ktorých vývoj začal v roku 1970 v priebehu rokov vzniklo nespočetné množstvo nových verzií Unixu. Unixové servery sa tešil veľkej obľube hlavne v minulosti momentálne sú na ústupe hlavne kvoli vyšším nákladom na ich zaobstaranie a prevázku. Pre účely unixu sa vytvoril Unix shell, oľúbený scriptovací jazyk, ktorý sa v rôznych obmenách teší veľkej obľube medzi administrátormi a automatizačnými programátormi.

Predinštalovaný software Na vačšine unixových systémoch je predinštalovaný bash a open JDK.

1.1.4 Linux

Prvé vydanie Linux bolo 17. septembra 1991, bol rozšírený na najviac platforiem a momentálne sa pýši tým, že je jediný používany operačný systém na TOP 500 superpočítačoch(mainframoch) Skriptovací jazyk Unix shell resp. jeho najrozšírenejšia forma Bash.

Predinštalovaný software Predinštalovaný software vo väčšine distribúciách linuxu sú bash, Open Jdk - Java, niektoré distribúcie ponúkajú Python. RedHat začína s podporou .net frameworku.

1.2 Programovacie jazyky

S príchodom osobných počítačov no najmä serverov, sa programátori zaujímali o automatizáciu procesov, ktoré na danom stoji bolo spočiatku potrebné spúšťať manuálne. Nakoľko tieto úlohy neboli na toľko komplexné ako samotné programy, ktoré spúštali bolo vhodné na tieto úlohy využiť/vytvoriť skriptovacie jazyky. V nasledujúcej časti si priblížime zopár programovacích jazykov, ktoré sa v dnešnej dobe bežne používajú na tvorbu automatizovaných skriptov.

1.2.1 Shell

Je skriptovacím jazykom pre unixové distribúcie. Počas rokov prešiel roznymi zmenami a rozšíreniami. Verzie shellu su: sh, csh, ksh,tcsh, bash. Bash sa momentálne teši najväčšej obľube no zsh je verzia shellu, ktorá má najviac rôznych rozšírení funkcionality ako aj veľa priaznivcov medzi developermi. V nasledujúcich častiach všeobecne zhodnotíme jednotlivé výhody resp. nevýhody tohoto skriptovacieho jazyka.

Výhody

- automatizácia často opakujúcich sa úloh
- dokáže zbiehať zloťité zloťené príkazy ako jednoriadkový príkaz - tzv. reťazenie príkazov
- ľahký na používanie
- výborné manuálové stránky
- ak hovoríme o Unix shelli je portabilný naprieč platformami linuxu-unixu
- jednoduché plánovanie automatických úloh

Nevýhody

- asi najväčšou nevýhodou je ze natívne nefunguje pod windowsom, existuju iba rozne emulátory a nástroje tretích strán, ktoré sprostredkujú jeho funkcionalitu.
- pomalé vykonávanie príkazov pri porovnaní s inými programovacími jazykmi
- nový proces pre skoro každý spustený príkaz
- zložitejší na pamatanie si rôznych prepínačov, ktoré dané príkazy podporujú
- nejednotnosť prepínočov(hoc to by asi ani nešlo)
- neprenosný medzi platformami
- shell nepridáva vlastné príkazy, používa iba tie, ktoré sú dostupné na konkrétnom počítači

Popis a zhodnotenie jazyka Unix Shell je obľúbeným scriptovacím jazykom, vhodným na automatizovanie každodenných operácií. Je jedným z najpoužívanejších skriptovacích jazykou vôbec, nakoľko všetky linuxove, unixové servery využívajú práve tento jazyk ako svoj primárny. V nasledujucich častiach budem popisovať Bash, ktorý je najrozšírenejšia verzia Unix shellu. Medzi jeho silné stránky patrí jednoduchá manipulácia s crontable, pomocou ktorej vie admin jednoducho planovať beh procesov. Ďalšou zaujímavou vymoženosťou jazyka je pajpa. Pajpa je klasickým príkladom vnútro-procesorovej komunikácie : odovzdáva štandardný výstup "stdout"procesu na štandardný vstup "stdinïného procesu, viď príklad.

```
zarrelli:~\$ ls -lah | wc -l
35
```

V uvedenom príklade sme vylistovali obsah adresára v ktorom sa práve nachádzame a výstupom z programu sme naplnili štandardný vstup aplikácie "wc"ktorá spočíta, koľko riadkov sa nachádza vo vstupe, ktorý jej bol dodaný. Príkaz za znakom pajpy "beží v subshelli, čo znamená, že nebude schopný zmodifikované hodnoty v rodičovskom procese. Zlyhanie príkazu v pajpe vedie k takzvanej "zlomenej pajpe", v tomto prípade exekúcia príkazov skončí. [1]

Taktiež niektoré často používané príkazy majú pozmenený spôsob zápisu. Ako príklad si uvedieme príkaz for, pri ktorom bash používa nsledovnú syntax:

Algoritmus 1 Bash ukážka rôznych volaní for cyklu. [1]

```
// prvý spôsob zápisu podobná vylepšenej verzii z predchádzajúceho príkladu
for placeholder in list_of_items
do
action_1 \$placeholder
action_2 \$placeholder
action_n \$placeholder
done
//kolekcia vo fore môze byť reprezentovaná vymenovaním prvkov
//priamo za "in" časťou
for i in 1 2 3 4 5
do
echo "\$i"
done
// c-like prístup
for ((i=20;i > 0;i--))
{
        if (( i % 2 == 0 ))
        then
        echo "\$i is divisible by 2"fi
}
exit 0
```

Ako je vidieť z príkladu for používa podobnú syntax ako ostatné jazyky, a ešte ju rozširuje, druhý spôsob môže uľahčiť prácu napríklad pri prototypovaní skriptu. Vyššie spomenuté použitia niesú jediné, shell poskytuje možnosť vložiť parametre pre nasledovnú iteráciu priamo z konzoly ako v nasledovnom príklade.

Algoritmus 2 Bash ukážka volania skriptu s for cylom priamo z konzoly . [1]

```
//telo skriptu
#!/bin/bash
i=0
for cities
do
echo "City $((i++)) is: $cities"
done
exit 0

//následné volanie z konzoly
./for-pair-input.sh
Belfast Redwood Milan Paris
City 0 is: Belfast
City 1 is: Redwood
City 2 is: Milan
City 3 is: Paris
```

Avšak syntax jazyka sa učí taždie nakoľko používa rôzne prepínače, ktoré novému používateľovi nemusia byť sprvu jasné. V tabuľke uvádzame príklad prepínačov pre if, ktorý pre podmienkovú časť používa hranaté zátvorky namiesto okrúhlych na aké sme zvyknutý z väčšiny programovacích jazykov. Za zmienku stojí tiež, že napríklad Unix shell nepoužíva žiadne zátvorky v podmienkovej časti príkazu, na ukončenie podmienkovej časti sa používa bodkočiarka, čo spôsobuje problémi pri prenositeľnosti. If ponúka aj ďalšie prepínače no zhodnotili sme, že pre ilustráciu budú postačovať aj príklady uveené v tabuľke. Najvačsia nevýhoda je, že ani shell script nieje jazyk, ktorý by bol multiplatformový a teda ak by sme mali prostredie, kde servery bežia na rôznych operačných systémoch, potrebujeme poznať ďalší jazyk, ktorým docielime rovnaké alebo aspoň podobné výsledky.

Reťazcové porovnanie	Popis
Str1 = Str2	Vráti true ak sa porovnávané reťazce rovnajú.
Str1 != Str2	Vráti true ak porovnávané reťazce nie sú rovnaké.
-n Str1	R Vráti true ak reťazec nie je null resp. o dĺžke 0.
-z Str1	Returns true ak reťazec je null resp. o dĺžke 0.

Numerické porovnanie	Popis
expr1 -eq expr2	Vráti true ak sú porovnávané výrazy rovné.
expr1 -ne expr2	Vráti true if ak nie sú porovnávané výrazy rovné.
expr1 -gt expr2	Vráti true ak je hodnota premmenej expr1 väčšia než hodnota premennej expr2.
expr1 -ge expr2	Vráti true ak je hodnota premmenej expr1 väčšia alebo rovná hodnote premennej expr2.
expr1 -lt expr2	Vráti true ak je hodnota premmenej expr1 menšia než hodnota premennej expr2.
expr1 -le expr2	Vráti true ak je hodnota premmenej expr1 menšia alebo rovná hodnote premennej expr2.
! expr1	Operátor "!"zneguje hodnotu premennej expr1.

Tabuľka 1: Ukážka prepínačov v podmienkovom výraze if [4]

1.2.2 Powershel/Classic command line

Command line je zakladnym skriptovacim jazykom pre windows distribucie, ktorý poskytuje malé API pre svojich používateľov. Aj kôli tomu Miscrosoft prišiel s novým jazykom Powershell. Powershell je kombináciou príkazového riadku, funkcionálneho programovania a objektovo orentovaného programovania. Je založený na .NET frameworku, ktorý mu dáva istú mieru flexibility, ktorá v

TODO: dpoisat dake sprostosti

Jeho vyhody a nevyhody si popiseme v nasledujucich castiach.

Výhody

- bohaté api
- výborne riešeny run-time
- flexibilný
- veľmi jednoduché prepnúť z .NET frameworku

- dokáže pridávať funkcionalitu používaním tired a funkcií z .net knižníc

Nevýhody

- bohaté api - nejednoznačné, kedy čo použiť
- niektoré výhody jazyka sú až nevhodne skryté pred používateľmi
- staršie verzie serverov nie sú Powershell-om podporované ako novšie
- dokumentácia je horšia ako v prípade Shell scriptu

Popis a zhodnotenie jazyka Powershell je obľúbený medzi programátormi a administrátormi, ktorý pracujú pod operačným systémom Windowsu. Do nedávna kým Pewrshell bežal na .NET frameworku ho nebolo možné používať mimo operačných systémov Windows, avšak s príchodom frameworku .NET Core sa situácia zmenila. Tento frawework je momentálne open source, jeho zdrojové kódy boli zverejnené a je doň možné prispievať. Okrem iného podporuje rovnaké alebo aspoň podobné štruktúry ako Shell script a poskytuje v niektorých prípadoch rovnaké príkazy ako napríklad : mv, cp, rm, ls. Jednen zo zásadných rozdielov medzi Shellom a Powrshellom je ten, že kým v Shelli sú pre vstup aj výstup používané textové reťazce, ktoré je potrebné rozparsovať a interpretovať v Powershelli je všetko presúvané ako objekt. Po preštudovaní si materiálov jazyka z Mastering Windows PowerShell Scripting - Second Edition [5], ide o najzásadnejší rozdiel, nakoľko ostatné veci boli zrejme navrhované v spolupráci s používateľmi Shell scriptu.

Pre demonštráciu rozdielov pri odovzdávaní si parametrov medzi príkazmi prikladám nasledovný príklad.

Algoritmus 3 Ukážka použitia pipe v powershell. [2]

```
function changeName(\$myObject)
{
        if (\$myObject.GetType() -eq [MyType])
        {
                //vypíš obsah premennej
                \$myObject.Name
                //zmeň reťazec pre atribút name
                \$myObject.Name = "NewName"
        }
        return \$myObject
}

// Vytvorenie objektu s argumentom OriginalName a následné použitie funkcie
//PS> \$myObject = New-Object MyType -arg "OriginalName"
//PS> \$myObject = changeName \$myNewObject
//OriginalName
//PS> \$myObject.Name
//NewName
// Ukážka s využitím pipe
//PS> \$myObject = New-Object MyType -arg "OriginalName" | changeName
//OriginalName
//PS> \$myObject.Name
//NewName
```

1.2.3 Python

Do analýzy sme sa rozhodli pripojiť aj Python. Python sme si nevybrali náhodne, nakoľko je jedným z najpopulárnejších programovacích jazykov súčastnosti. Je viacúčelový, patrí mezi vyššie programovacie jazyky, objektovo orientovaný, interaktívny, interpretovaný a extrémne používateľsky prijateľný.

Výhody

- Je ľahko čitateľný, tým pádom ľahšie pochopiteľný
- Syntax orientovaná na produktivitu

- Multiplatformový - po inštalácii interpretera
- Množstvo rôznych knižníc
- Open source

Nevýhody

- Rýchlosť
- Slabšia dokumentácia
- Nevhodný pre úlohy pracujúce s vyšším množstvom pamäte
- Nevhodný pre viac-procesorovú prácu
- Nevhodný pre vývoj na mobilných zariadeniach
- Limitácie pri prístupe k databázam

Popis a zhodnotenie jazyka Ako sme spomínali Python je jeden z najobľúbenejších jazykov súčastnosti, veľkú časť tejto komunity tvoria vedci, ktorý nemajú rozsiahle programátorské znalosti. Práve jednoduchosť, čitateľnosť a pochopiteľnosť jazyka sa nemalou mierou podieľajú na tomto fakte. Taktiež tu nieje nutné manažovať pamäť a iné netriviálne záležitosti ničších programovacích jazykov. Jazyk je síce objektovo orientovaný no jednoducho sa v ňom píšu aj skripty. Jazyk poskytuje štruktúry ako pipa, dokážeme v ňom jednoducho pracovať s procesmi, vytvárať triedy, inštancie, jednoducho prototypovať a odsymulovať rôzne problémy. Veľkou výhodou tohoto jazyka je, že je open source s veľkou komunitou, ktorá rada testuje nové vydania, nahlasujú problímy a tým pádom sa jazyk rýchlejšie a kvalitnejšie vyvíja. Na Pythone vznikli zaujímave webové framworky ako napríklad Django. Každá strana má dve mince a ani Python nie je stopercentný. Tým, že je to interpretovaný jazyk neprekypuje rýchlosťou. Veľa ľudí sa zaoberá rýchlosťou jazykov, zisťujú efektívnosť pri rôznych úkonoch ako napríklad cykly, volania funkcií, aritmetika, prístup k pamäti, vytváranie objektov. V nasledujúcich tabuľkách je vidno rozdiel v rýchlosti jednotlivých testov.

Aj keď sme spomenuli viaceré nedostatky, asi najväčším je rýchlosť. Nvm čo ďalej.

Jazyk	**Force field benchmark**	**Array reverse benchmark**	**Rolling average benchmark**
C++ (-O2)	1.892	4.367	0.005
Java 7	2.469	3.776	0.463
C# (normal)	10.712	14.071	0.621
JavaScript	16.159	13.162	1.312
Python 2	717.2	1485	71.550
Python 3	880.7	1466	81.143

Tabuľka 2: Porovnanie rýchlostí rôznych jazykov[6]

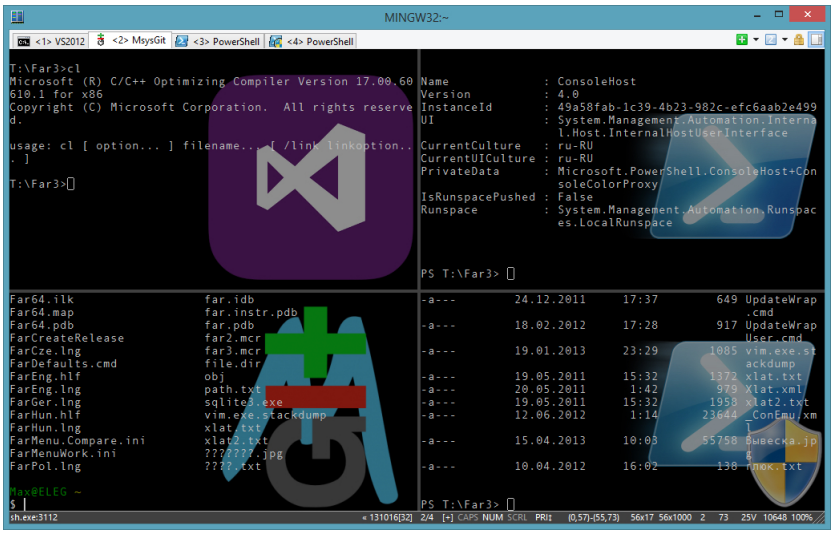
1.3 Existujúce riešenia

Existuje množstvo emulátorov a nástrojov tretích strán, ktoré sprostredkúvajú funkcionality shell scriptu do Windowsu, nikeotré z nich si predstavíme.

1.3.1 ConEmu

ConEmu je konzolový emulátor, ktorý poskytuje jednoduché GUI do ktorého je možné vložiť viacero konzol. Dokáže spúšťať jednoduché GUI aplikácie ako napríklad Putty, Cygwin. Poskytuje množstvo nastavení ako nastavenie kurzora, priehľadnosti, písma a pod. Podporuje Windows 2000 a neskoršie verzie. Neposkytuje verziu pre ine operačné systémy.

Skúsenosti ConEmu je podarený emulátor, ktorý je schopný vykonávať akýkoľvek skript. Používaním sme neprišli na závažné nedostatky, ktoré by neboli popísane v ich issues logu na github. Ale ako každý software je aj ConEmu náchylný na chyby. Podla ich issues logu sa do ich oficiálnych releasov dostávajú rôzne problémy, ktoré neboli problémom v predchadzajúcich verziách. V tomto prípade je na zvážení každého či problémy, ktoré sa môžu dostávať do jednotlivých verzií emulátora stoja za jeho použitie, resp. či jeho kladné stránky sú natoľko dobré aby prevýšili zápory.

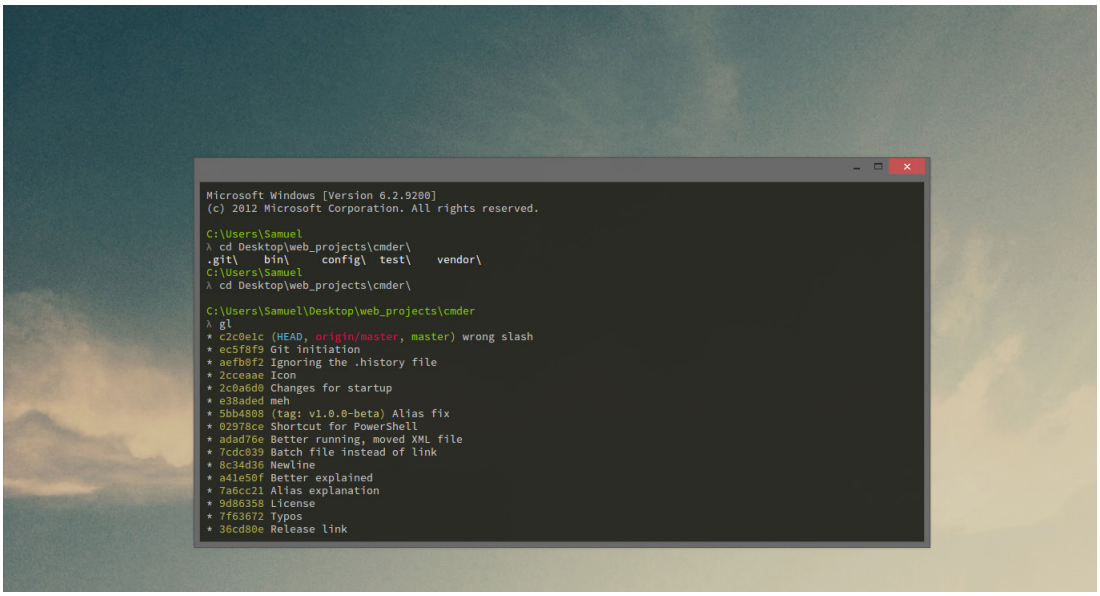


Obrázok 1: Ukážka ConEmu emulátora

1.3.2 cmder

Cmder je ďalším príkladom emulátora shell terminálu. Vychádza z troch projektov ConEmu, Clink a Git pre Windows - voliteľná súčasť. ConEmu sme si predstavili v predchádzajúcej časti s jeho kladymi a zápormi. Clink, konkrétne clink-completions je v projekte využívaný na zvýšenie komfortu pri písaní skriptov, nepridáva ďalšiu shellovú funkcionalitu.

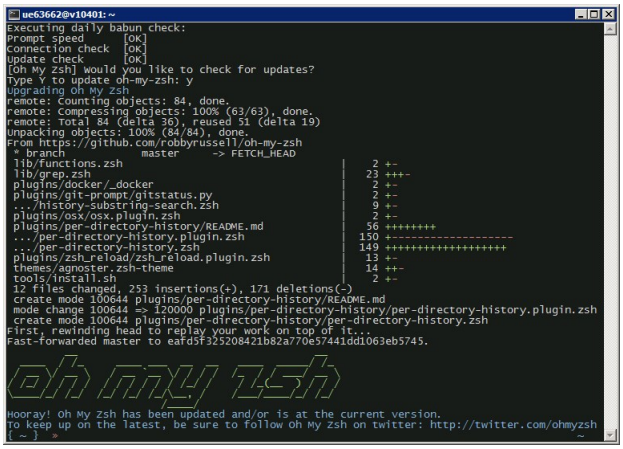
Skúsenosti ConEmu je príjemný nástroj, dokáže zjednodušiť človeku prácu obzvlášť ak je zvyknutý na programovanie v shell scripte. Nakoľko cmder používa ConEmu ako emulátor shell terminálu a je úzko určený pre Windows platformu nemožno hovoriť o multiplatformovom riešení.



Obrázok 2: Ukážka cmder emulátora

1.3.3 Babun

Babun je ďalším z množstva emulátorov pre Windows, ktorý je nadstavbou cygwinu. Vo svojom jadre používa zshell a bash, ktoré sme popísali ako populárne medzi komunitou. Prináľa vlasné gui, ktoré dokáže zafarbovať text podla zdrojoveho jazyka, čo zvyšuje prehľadnosť. Je tam git, svn, puython, perl. Tiež má integrované sťahovanie nových packagov, ktoré ponúka cygwin pomocou kľúčového slova pact. Prenositeľnosť skriptov z unixových strojov je zabezpečená tým, že používa bash a zsh, avšak je to emulátor čisto pre Windows distribúcie.



Obrázok 3: Ukážka babun emulátora

1.3.4 MobaXterm

Poskytuje množstvo funkcionality avšak je zatažený licenciou v hodnote 50 eur.

Neplatená verzia

- Plná podpora SSH a X serveru
- Vzdialená plocha (RDP, VNC, Xdmcp)
- Vzdialený terminál (SSH, telnet, rlogin, Mosh)
- X11-Forwarding
- Automatický SFTP prehliadač
- Podpora pluginovt
- Možnosť inšalačnej alebo prenositeľnej verzie
- Plná dokumentácia
- Maximálne 12 spojení
- Maximálne 2 SSH tunely
- Maximálne 4 maká
- Maximálne 360 sekúnd pre Tftp, Nfs a Cron

Platená verzia

- Všetky vymoženosti z neplatenej verzie Home Edition +
- Možnosť upraviť svoju uvítaciu správu a logo
- Modifikovať profilový skript
- Odstrániť nechcené hry, šetriče obrazovky alebo nástroje
- Nelimitovaný počet spojení
- Nelimitovaný počet tunelov a makier
- Nelimitovaný čas behu pre sieťové daemony
- Podpora centrálneho hesla

- Profesionálny support
- Doživotné právo používania

1.4 Zhodnotenie analyzovaných technológií

2 Preklad jazykov

Pri programovacích jazykoch nás zaujímajú ich vyjadrovacie schopnosti ako aj vlastnosti z hľadiska ich rozpoznania. Tieto vlasnosti sa týkajú programovania a prekladu, pričom obe je potrebné zohľadniť pri tvorbe jazyka. V dnešnej dobe sa používajú na programovanie hlavne takzvané vyššie programovacie jazyky, môžeme ich označiť ako zdrojové jazyky. Na to aby vykonávali čo používateľ naprogramoval je potrebné aby boli pretransformované do jazyka daného stroja. Spomínanú transformáciu zabezpečuje prekladač, prekladačom máme na mysli program, ktorý číta zdrojový jazyk a transformuje ho do cieľového jazyka, ktorému rozumie stroj.[1]

2.1 Kompilátor proces prekladu

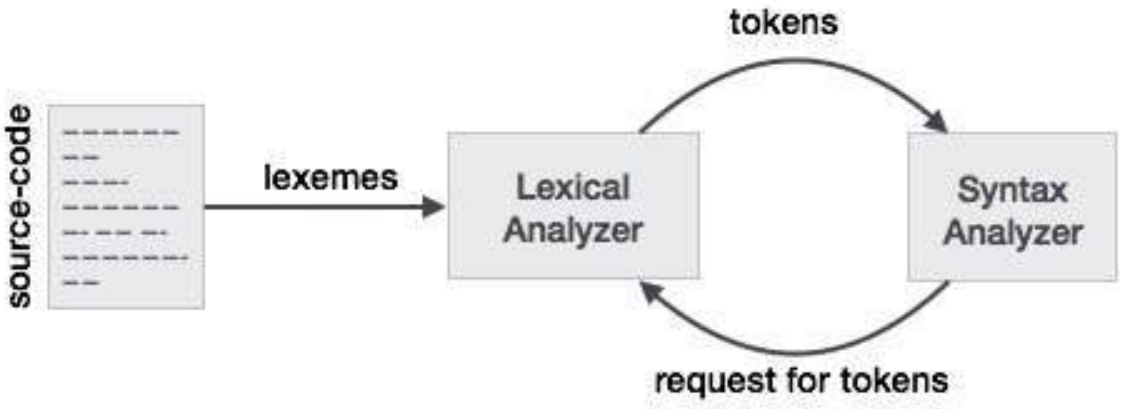
Aby bol preklad možný, musí byť zdrojový kód programu napísaný podľa určitých pravidiel, ktoré vyplývajú z jazyka. Proces prekladu je možné rodeliť na 4 hlavné časti.

- lexikálna analýza
- syntakticka analýza
- spracovanie sémantiky
- generovanie cieľového jazyka

Podrobnejšie si stručne popíšeme všetky štyri časti, ktoré majú pre nás z hľadiska prekladu najväčší zmysel.

2.1.1 Lexikálna analýza

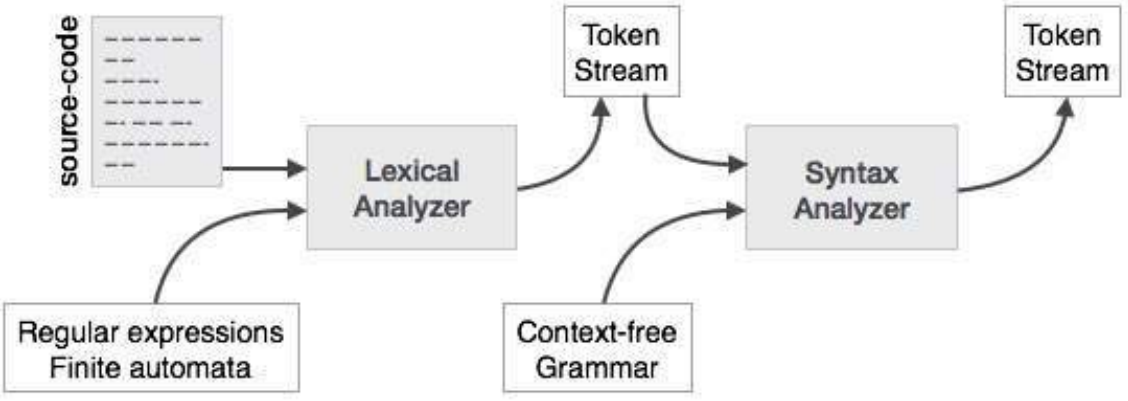
Lexikálna analýza je prvou fázou kompilátora. Dopredu napísaný zdojový kód je postupne spracovávaný preprocesorom, ktorý vytvára takzvané lexémy.
Lexémou nazývame postupnosť alfanumerických znakov. Tieto postupnosti znakov sú následne vkladané do lexikálneho analyzátora, ktorý ma za úlohu vytvoriť zo vstupných lexém tokeny slúžiace ako vstup pre syntaktický analyzátor.
Tokeny sa vytvárajú na základe preddefinovaných pravidiel, ktoré sa v programovacích jazykoch definujú ako pattern. V prípade, že lexikálny analyátor nieje schopný nájsť pattern pred danú lexému musí vyhlásiť chybu počas tokenizácie.
Výstupom z lexikálnej analýzy sú takzvané tokeny, ktoré tvoria vyššie jednotky jazyka ako kľúčové slová jazyka, konštanty, identifikátory, operátory a iné.



Obrázok 4: Ukážka práce lexikálneho analyzátora

2.1.2 Syntaktická analýza

Ďalšou fázou je syntaktická analýza. Úlohou Syntaktického analyzátora je kontrola správnosti vytvorených tokenov s uchovaním niektorých získaných informácií o štruktúre skúmanej syntaktickej jednotky. Syntaktická analýza sa radí medzi bezkontextové gramatiky. Po skoncení syntaktickej analýzy prichádza na rad sémantická analýza.



Obrázok 5: Ukážka práce syntaktickeho analyzátora

2.1.3 Limitácia syntaktickej analýzy

Syntaktický analyzátor ziska vstup z tokenu, ktorý vytvorí lexikálny analyzátor. Lexikálne analyzátory sú zodpovedné za validitu tokenu.Syntaktické analyzátory majú nasledovné limitácie.

- nedokážu zistiť validitu tokenu
- nedokážu zistiť či je token používaný pred tým ako je deklarovaný
- nedokážu zistiť či je token používaný pred tým ako je inicializovaný
- nedokážu zistiť validitu operácie, ktorú token vykonáva

2.1.4 Semantická analýza

Sémantická analýza má za úlohu interpretovať symboly, typy, ich vzťahy.Sémantická analýza rohoduje či má syntax programy význam alebo nie. Ako príklad zisťovania významu môžeme uviesť jednoduchú inicializáciu premennej.

```
int integerVariable = 6

int secondIntegerVariable = "six"
```

Oba príklady by mali prejsť cez lexikálnu a syntaktickú analýzu. Je až na sémantickej analýze aby rozhodla o správnosti zápisu programu a v prípade nesprávneho zápisu informovala o chybe. Hlavné úlohy sémantickej analýzy sú":

- zisťovanie dosahu definovaných tokenov takzvaný scoping
- kontrola typov
- deklaracia premenných
- definícia premenných
- viacnásobná deklarácia premenných v jedno scope

2.1.5 Generovanie cieľového jazyka

Generovanie cieľového jazyka môžeme považovať za poslednú fázu kompilátora. V tejto fáze sa preklápa jazyk z vyššieho jazyka do strojového jazyka, ktorý úspešne prešiel cez analyzačné časti .

2.2 Iterpreter

popis interpretera, hladam cosi schopne.

2.3 Abeceda a vyhradené slová jazyka

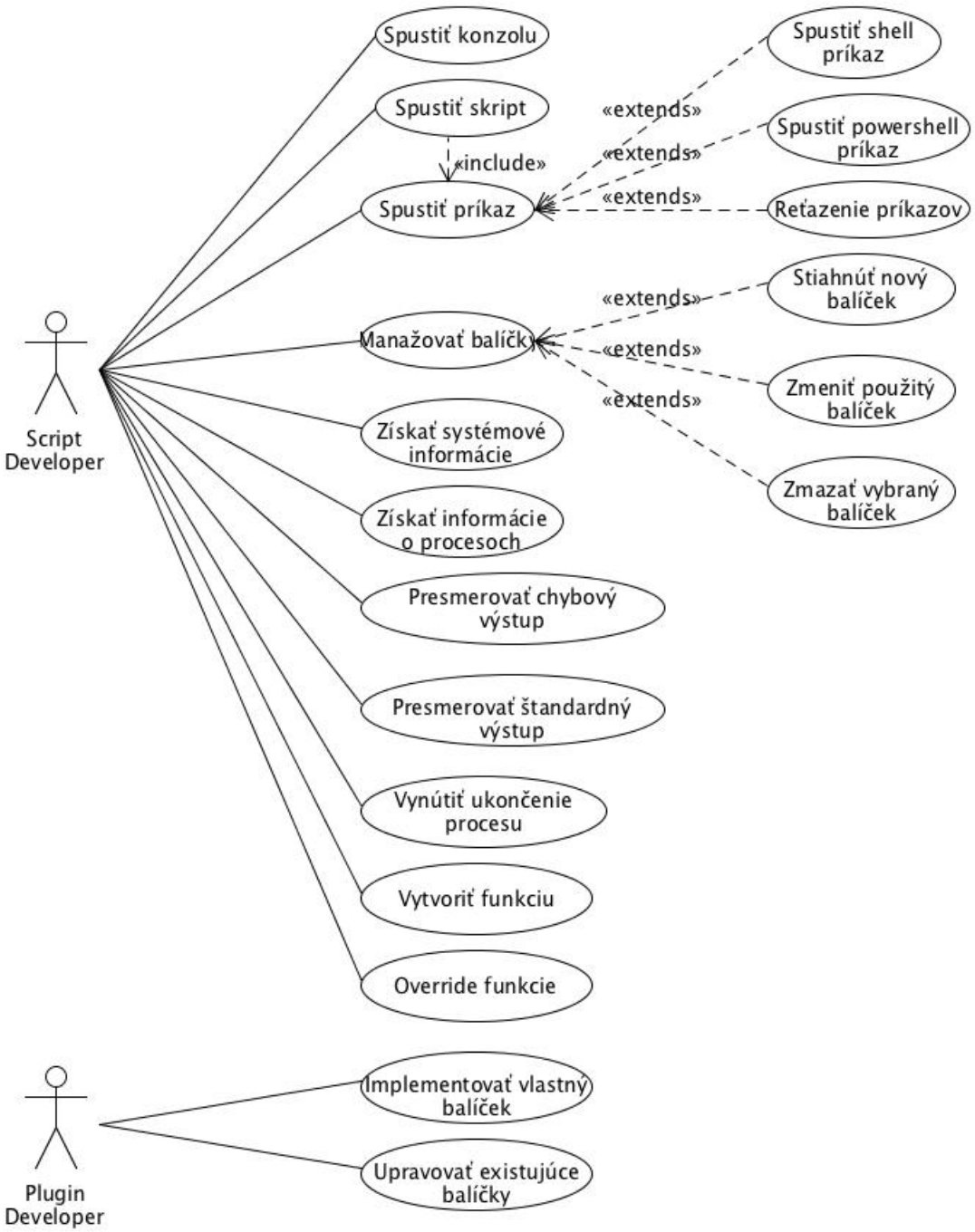
abeceda jazyka, popis ake pismena-slova rozpoznava, ake su vyhradene slova jazyka a bla bla Doplit daco o co sa bude dat opret v navrhu a arch.

2.4 Procedúry a algoritmy

procedúra - konečná postupnosť inštrukcií, ktorá sa dá vykonať mechanicky. Doplit daco o co sa bude dat opret v navrhu a arch.

3 Návrh riešenia

3.1 Prípady použitia



Obrázok 6: Prípady použitia pre navrhovanú aplikáciu

3.2 Popis use casov

V tejto časti sa venueme popisu jednotlivých use casov. Use case diagram spolu s popisom sú základnými prvkami na ktorých je možné špecifikovať novo vznikajúci software. Je dôležité aby najpodstatnejšie časti systému boli špecifikované na začiatku, aby sa pri navrhovaní aplikácie mohli prijať rozhodnutia, ktorými bude možné zaručiť, že výsledné riešenie bude to najlepšie možné, vyhovujúce špecifikácii. Ako je zjavné aj z priloženého diagramu prípadov použitia, pre aplikáciu sme identifikovali dvoch hráčov : Vývojár skriptov a Vývojár balíčkov. Títo hráči majú jednu spoločnú črtu - pre obe platí, že hráč je vývojár. Avšak je rozdiel medzi vývojárom skriptu a vývojom nových súčastí systému, čo je zjavne vidieť aj z popisu konkrétnych prípadov použitia.

3.2.1 Vývojár skriptov

Rola sa zameriava hlavne na používanie hotovej aplikácie, prácu s balíčkami, vytváranie skriptov, efektívne využívanie dostupného Aplikačné rozhranie (API).

Spustiť konzolu

Use case	Spustiť konzolu
Podmienky	Používateľ musí disponovať stiahnutou aplikáciou.
Vstup	Nie je potrebný žiany vstup od používateľa.
Popis	Konzolové rozhranie sa spustí.
Výstp	Konzola zobrazí základné údaje o konfigurácii.
Chyba	Konzola sa nespustí, musí však poskytnúť informáciu o chybe ktorá pri štarte nastala.

Tabuľka 3: Use case : Spustiť konzolu

Spustiť príkaz

Use case	Spustiť príkaz
Podmienky	Shell aplikácia musí byť spustená
Vstup	Textový reťazec obsahujúci príkaz a jeho argumenty.
Popis	Používateľ zadá valídny príkaz, následne získa výstup pre zadaný príkaz.
Výstp	Textový reťazec, ktorý v závislosti od programu variuje v dĺžke a obsahu.
Chyba	V prípade zlyhania je používateľ informovaný o probléme, ktorý nastal.

Tabuľka 4: Use case : Spustiť príkaz

Spustiť skript

Use case	Spustiť skript
Podmienky	Shell aplikácia musí byť spustená a skript správne napísaný.
Vstup	Vstupom je skript, ktorý v hlavičke definuje balíčky ktoré bude používať. Za nimi môže nasledovať čokoľvek od definície premenných, funkcií. V tele skriptu musí byť zadefinovaná metóda main(String args).
Popis	Vykonajú sa všetky príkazy tak ako sú napísané v zdrojovom súbore.
Výstp	Výstup je textový reťazec, závislý na logike skriptu.
Chyba	V prípade chyby pri sťahovaní závislostí, exekúcie príkazov, alebo iných komplikácií za behu program zapisuje na stderr chybové hlášky spolu zo základným popisom problému, tracom.

Tabuľka 5: Use case : Spustiť skript

Spustiť shell príkaz

Use case	Spustiť shell príkaz
Podmienky	Shell aplikácia musí byť spustená a skript správne napísaný. Taktiež musí byť v operačnom systéme ktorý podporuje shell.
Vstup	Textový reťazec obsahujúci príkaz a jeho argumenty.
Popis	Používateľ zadá valídny príkaz, následne získa výstup pre zadaný príkaz.
Výstp	Textový reťazec, ktorý v závislosti od programu variuje v dĺžke a obsahu.
Chyba	V prípade zlyhania je používateľovi vratený chybový kód.

Tabuľka 6: Use case : Spustiť shell príkaz

Spustiť powershell príkaz

Use case	Spustiť powershell príkaz
Podmienky	Shell aplikácia musí byť spustená a skript správne napísaný. Systém musí mať nainštalovaný powershell.
Vstup	Textový reťazec obsahujúci príkaz a jeho argumenty.
Popis	Používateľ zadá valídny príkaz, následne získa výstup pre zadaný príkaz.
Výstp	Textový reťazec, ktorý v závislosti od programu variuje v dĺžke a obsahu.
Chyba	V prípade zlyhania je používateľovi vratený chybový výstup z powershellu.

Tabuľka 7: Use case : Spustiť powershell príkaz

Reťazie príkazov

Use case	Reťazie príkazov
Podmienky	Shell aplikácia musí byť spustená. Vstup musí byť zadaný v správnom formáte.
Vstup	Textový reťazec obsahujúci sekvenciu príkazov, ich argumenty spojené znakom pajpy |.
Popis	Systém rozozná, že ide o zreťazený príkaz a následne začne vykonávať príkazy v poradí v akom boli zadané. Jednotlivé príkazy odovzdajú svoje výstupy svojmu nasledovníkovy po úspešnom ukončení. Príkazy sa vykonávajú až kým nepríde na posledný príkaz v sekvencii, alebo ako počas behu príde pri niektorom z príkazov ku chybe. O chybe je používateľ oboznámený a chyba je zapísaná na štandardný chybový výstup.
Výstp	Textový reťazec, ktorý v závislosti od programu variuje v dĺžke a obsahu, výstup bude vygenerovaný posledným príkazom sekvencie.
Chyba	O chybe je používateľ oboznámený a chyba je zapísaná na štandardný chybový výstup.

Tabuľka 8: Use case : Reťazie príkazov

Manažovať balíčky

Use case	Manažovať balíčky
Podmienky	Shell aplikácia musí byť spustená.
Vstup	Textový reťazec obsahujúci príkaz "pkgälebo aky si vyberem a jeho argumenty.
Popis	Používateľ bude schopný nahrať, zmazať, nahradiť vybraný balíček.
Výstp	Textový reťazec, ktorý v závislosti od programu variuje v dĺžke a obsahu - možno bude informovať o šťahovacom procese.
Chyba	O chybe je používateľ oboznámený a chyba je zapísaná na štandardný chybový výstup v podobe stack tracu napr. .

Tabuľka 9: Use case : Manažovať balíčky

Stiahnúť nový balíček

Use case	Stiahnúť nový balíček
Podmienky	Shell aplikácia musí byť spustená. Príkaz na stiahnutie balíčka musí byť správne zadaný.
Vstup	Textový reťazec obsahujúci príkaz "pkg download <názov balička>"
Popis	Program sa ako prvé pozrie do adresára balíčkov či už daný balíček nebol stiahnutý, ak nie stiahne nový balíček a načíta ho medzi aktívne balíčky. V opačnom prípade medzi aktívne balíčky načíta používateľom zvolený balíček.
Výstp	Textový reťazec informujúci o úspešnosti sťahovania.
Chyba	Vypíše chybu do stderr v prípade, že daný balíček na servery neexistuje, používateľ nemá internetové pripojenie.

Tabuľka 10: Use case : Stiahnúť nový balíček

Zmeniť použitý balíček

Use case	Zmeniť použitý balíček
Podmienky	Shell aplikácia musí byť spustená. Príkaz na zmenenie používaného balíčka musí byť správne zadaný.
Vstup	Textový reťazec obsahujúci príkaz "pkg change <názov nahradzovaného balička> <názov nahradzujúceho balička>
Popis	Program zmení používaný balíček z aktuálne používaného na balíček vybratý používateľom. Táto voľba je aplikovateľná iba pre spravovanie verzií existujúcich balíčkov. V prípade, že nahradzujúci balíček nieje dostupný lokálne, používateľ bude vyzvaný stiahnúť daný balíček.
Výstp	Textový reťazec informujúci o úspešnosti výmeny, alebo informujúci o potrebe stiahnutia balíčka.
Chyba	V prípade ak príde počas zmeny balíčkov k chybe, bude zapísana na stderr.

Tabuľka 11: Use case : Zmeniť použitý balíček

Zmazať vybraný balíček

Use case	Zmazať vybraný balíček
Vstup	Textový reťazec obsahujúci príkaz "pkg delete <názov balička>.
Podmienky	Shell aplikácia musí byť spustená. Príkaz na zmazanie vybraného balíčka musí byť správne zadaný.
Popis	Program zmaže používateľom vybraný balíček z aktívnych balíčkov a následne ho fyzicky zmaže z disku.
Výstp	Textový reťazec informujúci o úspešnosti zmazania zadaného balíka
Chyba	V prípade nesprávneho odstránenia balíčka z aktívnych alebo pri následnom mazaní zo súborového systému bude informácia o chybe presmerovaná na stderr.

Tabuľka 12: Use case : Zmazať vybraný balíček

Získať systémové informácie

Use case	Získať systémové informácie
Vstup	Vstupom je textový reťazec "sysinfo".
Podmienky	Shell aplikácia musí byť spustená. Používateľ vloží valídny príkaz na vyžiadanie systémových informácií.
Popis	Program vypíše na štandardný výstup informácie o systémových informáciách ako napríklad využitie procesoru, využitie pamäte RAM, využitie oddielu swap, teplotu zariadení a podobné.
Výstp	Výstupom je textový reťazec, formátovaný do riadkov. Každému riadku prislúcha jedna informácia napr. CPU, ďalší riadok RAM atď.. V prípade viac jadrového procesora sa vypísu informácie o každom z jadier.

Chyba	V prípade, že používateľ nemá právo na získanie informácií program vypíse dôvod priamo na stdout. Tak isto tam vypíše aj akékoľvek chyby ku ktorým môže prísť počas behu.

Tabuľka 13: Use case : Získať systémové informácie

Získať informácie o procesoch

Use case	Získať informácie o procesoch
Vstup	Vstupom je textový reťazec "processes".
Podmienky	Shell aplikácia musí byť spustená. Používateľ vloží valídny príkaz na vyžiadanie informácií o procesoch.
Popis	Program vypíše na štandardný výstup informácie o bežiacich procesoch, používateľoch, ktorý tieto procesy spúšťajú, koľko procesoru, pamäte RAM používajú.
Výstp	Výstupom je prehľadný výpis v podobe tabuľky, kde každý riadok zodpovedá jednému procesu. Nad jednotlivými hodnotami je hlavný riadok, ktorý popisuje o akú hodnotu ide.
Chyba	V prípade, že nieje možné získať informácie o procesoch je táto skutočnosť zobrazená na stdout a popis chyby sa presmeruje na stderr.

Tabuľka 14: Use case : Získať informácie o procesoch

Presmerovať chybový výstup

Use case	Presmerovať chybový výstup
Vstup	Pre presmerovanie na chybový výstup je potrebné dodržať syntax command stderr> file
Podmienky	Shell aplikácia musí byť spustená. Používateľ vloží valídny príkaz na presmerovanie chybového výstupu.
Popis	Program presmeruje chybový výstup kam mu používateľ v príkaze prikáže.
Výstp	Výstup programu predstavuje textový reťazec s popisom chyby, ktorá nastala.
Chyba	Ak by došlo k chybe zaloguje sa do logu aplikácie.

Tabuľka 15: Use case : Presmerovať chybový výstup

Presmerovať štandardný výstup

Use case	Presmerovať štandardný výstup
Podmienky	Shell aplikácia musí byť spustená. Používateľ vloží valídny príkaz na presmerovanie štandardného výstupu.
Vstup	Pre presmerovanie na štandardný výstup je potrebné dodržať syntax command stderr> file
Popis	Program presmeruje štandardný výstup kam mu používateľ v príkaze prikáže.
Výstp	Výstup programu predstavuje textový reťazec s výstupom zo skriptu alebo príkazu.
Chyba	Ak by došlo k chybe zaloguje sa do logu aplikácie.

Tabuľka 16: Use case : Presmerovať štandardný výstup

Vynútiť ukončenie procesu

Use case	Vynútiť ukončenie procesu
Vstup	Pre vynútenie ukončenia procesu je potrebné použitie kľúčového slova kill <pid>.
Podmienky	Shell aplikácia musí byť spustená. Používateľ vloží valídny príkaz na ukončenie procesu.
Popis	Program presmeruje štandardný výstup kam mu používateľ v príkaze prikáže.
Výstp	Výstup programu je textový reťazec informujúci o úspešnosti ukončenia procesu.
Chyba	Ak pri ukončovaní procesu príde k chybe, informácie sa presunú na stderr.

Tabuľka 17: Use case : Vynútiť ukončenie procesu

Vytvoriť funkciu

Use case	Vytvoriť funkciu
Podmienky	Používateľ musí mať prístup k akémukoľvek textovému editoru.
Vstup	Funkcia musí byť správne zadefinovaná. Synax pre definovanie funkcie : function <návratový typ> <názov funkcie>(parametre funkcie)telo funkcie.
Popis	Používateľ napíše funkciu, ktorá bude prečítaná programom a vykonaná.
Výstp	Funkcia vracia premennú s definovanou návratovou hodnotou.
Chyba	V prípade, že nastane chyba pri exekúcii funkcie program skončí a program zapíše informácie o chybe na stderr.

Tabuľka 18: Use case : Vytvoriť funkciu

Override funkcie

Use case	Override funkcie
Podmienky	Používateľ musí mať prístup k akémukoľvek textovému editoru.
Vstup	Nad funciou je potrebné zapísať @Override alebo @Override(číslo) čo prekladaču povie, že má používať práve túto verziu funkcie, v druhom prípade číslo značí prioritu pri prepisovaní. Najnižšia priorita je 1, metóda s najvžšším číslom teda s najvyššou prioritou bude použitá v skripte.
Popis	Používateľ napíše funkciu, ktorá bude prečítaná programom a vykonaná. Navyše bude nahradzovať funkciu s rovnakým názvom.
Výstp	Premenná, ktorá je uvedená v definícii funkcie.
Chyba	V prípade zle zadefinovanej syntaxe je problém zapísaný na stderr a vykonávanie skriptu je ukončené.

Tabuľka 19: Use case : Override funkcie

3.2.2 Vývojár balíčkov

Ako je z názvu role zjavné, tento hráč bude mať na starosti hlavne vývoj aplikácie, starať sa o jej funkcionalitim v zmysle rozširovania API, ktoré môže vývojár skriptov používať pre efektívnejšiu prácu.

Implementovať vlastný balíček

Use case	Implementovať vlastný balíček
Podmienky	Používateľ musí mať nainštalovanú Java SDK vo verzii 8, mať prístup k textovému editoru.
Vstup	Balíček obsahujúci všetky potrebné rozhrania, ktoré musí vývojár balíčka implementovať.
Popis	Používateľ implementuje novú funkcionalitu v jave, následne všetky zdrojové súbory skompiluje a pridá do jar súboru určeného na ukladanie nových balíčkov.
Výstp	Balíček, ktorý je možné nahrať do aplikácie a používať ako jeden z príkazov.
Chyba	Chyba môže nastať pri vytváraní balíčka, kedy ho o chybe informuje prekladač jazyka v ktorom je balíček implementovaný. V prípade neúspešného načítania je používateľ informovaný priamo v konzole na stout.

Tabuľka 20: Use case : Implementovať vlastný balíček

Upravovať existujúce balíčky

Use case	Upravovať existujúce balíčky
Podmienky	Používateľ musí mať nainštalovanú Java SDK vo verzii 8, mať prístup k textovému editoru.
Vstup	Zdrojové súbory už existujúceho balíčka.
Popis	Používateľ upraví implementáciu alebo pridá novú funkcionalitu v jave, následne všetky zdrojové súbory skompiluje a pridá do jar súboru určeného na ukladanie nových balíčkov
Výstp	Po úprave je balíček možné nahrať do aplikácie a používať ako jeden z príkazov.
Chyba	

Tabuľka 21: Use case : Upravovať existujúce balíčky

3.3 Prvotný nástrel

3.4 Oddelenie štruktúry

4 Architektúra aplikácie

Ako sme ukázali existuje veľké množstvo skiptovacích jazykov ci, ktoré dokážu efektívne automatizovať dennodennú prácu avšak majú jeden spoločný nedostatok - nie sú multiplatformové. Táto vlastnosť môže byť pre niekoho nepodstatná, no pri veľkých projektoch kde sa míňa množstvo prostriedkov na automatizáciu to až tak zanedbateľný fakt nie je. Stačí si len predstaviť koľko času zabere tvorba automatizovaných skriptov pre jednu platformu a pripočítať rovnaké množstvo času pre každú ďalšiu. Niekto by mohol namietať, že pre ďalšie platformy to toľko času nezabere nakoľko logika skriptov je uz definovaná. Tu treba brať ohľad na to, že nie každý jazyk poskytuje programátorovi rovnaké API a teda treba rátať s možnosťou, že niekde bude potrebné doimplementovať veci chýbajúce v jazyku. Preto vyššie spomenuté dôvody sme sa rozhodli pre vytvorenie nového jazyka, ktorý by bol jednoducho rozšíriteľný, manažovateľný, ľahko pisateľný a platformovo nezávislý. [**morf**]

4.1 Java

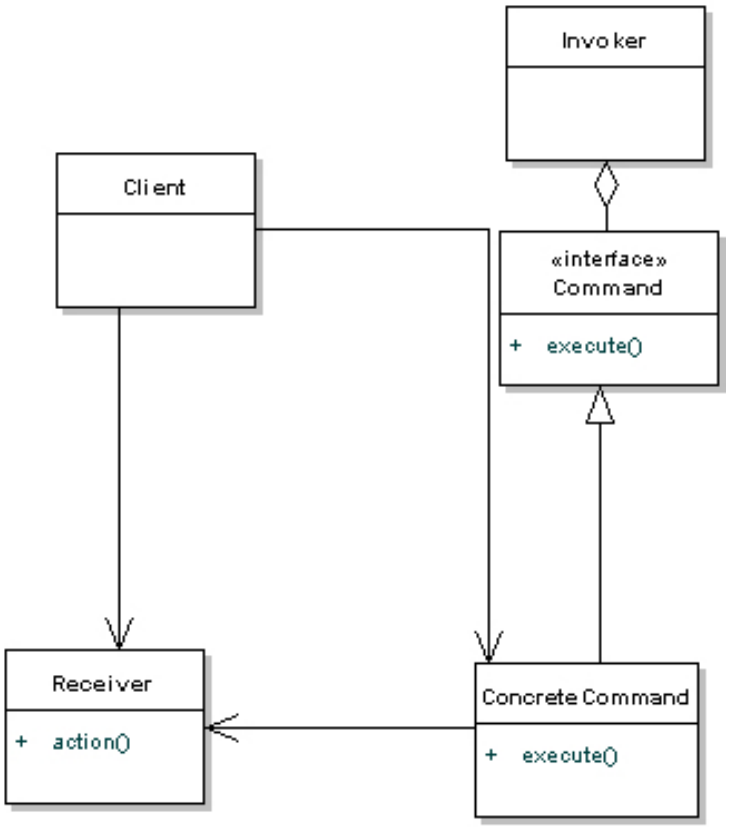
Je vyvíjaný spoločnosťou Oracle. Jeho syntax vychádza z jazykov C a C++. Zdrojové programy sa nekompilujú do strojového kódu, ale do medzistupňa, tzv. „byte-code“, ktorý nie je závislý od konkrétnej platformy. Táto vlastnosť Javy nám veľmi vyhovuje pre dosiahnutie cieľa platformovej nezávislosti. Ďalším veľmi podstatným faktom je, že v Jave programuje veľké množstvo developerov, tým pádom majú open source projekty veľkú šancu, že si ich komunita devoloperov osvojí a prispeje k ich postupnému zlepšovaniu.

4.2 Pouzite navrhove vzory

Aby sme zaručili rozšíriteľnosť, manažovateľnosť a ďalšie zásady dobrého softwéru bolo potrebné zvoliť vhodnú arhitektúru, ktorú popisujú použité návrhové vzory.

4.2.1 Command - príkaz

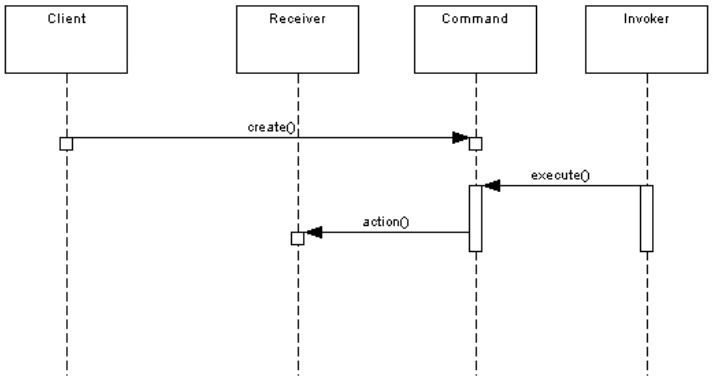
Command pattern je známy behaviorálny návrhový vzor, používa sa najmä na menežovanie algoritmov, vzťahov a zodpovednosti medzi objektami. Cieľom vzoru je zapúzdriť požiadavku(request) ako objekt tým pádom parametrizovať klienta s rôznymi požiadavkami a zabezpečiť operáciu spať.
Command vzor deklaruje rozhranie pre všetky budúce commandy a zároveň execute() metódu, ktorú s vypýta Receiver commandu aby splnil požadovanú operáciu. Receiver je objekt, ktorý vie ako požadovanú operáciu splniť. Invoker pozná command a pomocou implementovanej execute() metódy dokáže vyvolať požadovanú operáciu. Klient potrebuje



Obrázok 7: Class diagram Command návrhového vzoru

implemenotvaž ConcreteCommand a nastavit Receiver pre command. ConcreteCommand definuje spojenie medzi action a receiver. Keď Invoker zavolá execute() metódu na ConcreteCommand spustí tým jednu alebo viac akcií, ktoré budú bežať pomocou Receivera.

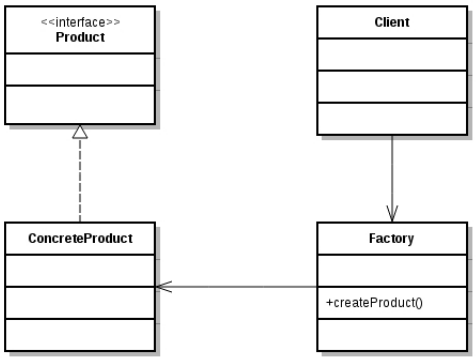
Pre lepšie pochopenie je proces zobrazený aj na sekvenčnom diagrame.



Obrázok 8: Sekvenčný diagram Command návrhového vzoru

4.2.2 Factory - továreň

Factory návrhový vzor patrí do sekcie vytváracích vzorov, pomocou tohoto vzoru budeme schopný vytvárať objekty bez toho aby sme prezradili logiku ich vytvárania klientovi. Diagram návrhového vzoru je mozné vidieť na nasledujúcom obrázku.



Obrázok 9: Class diagram Factory návrhového vzoru

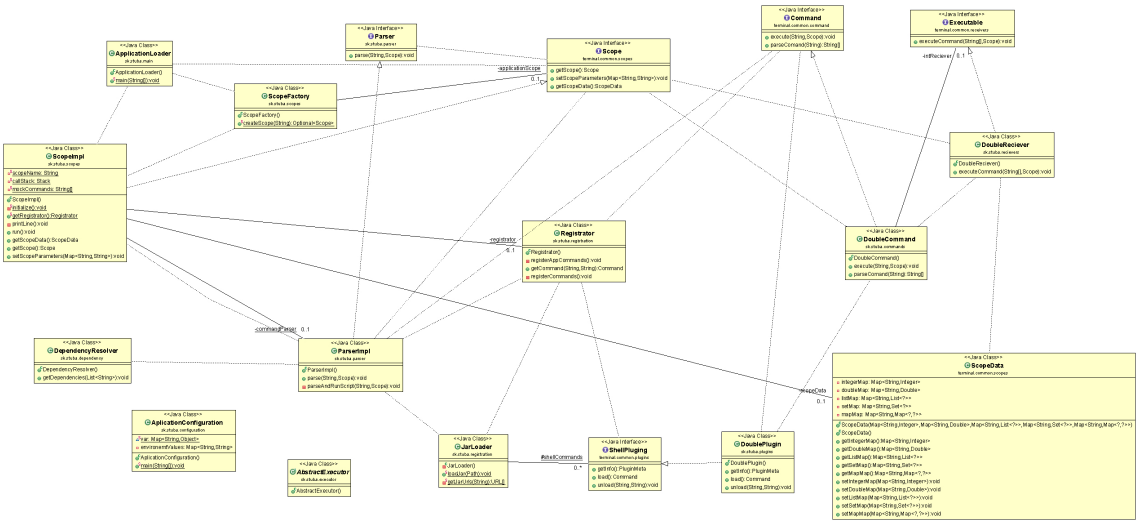
4.2.3 Interpreter

mozno pouzijem

4.3 Komponenty aplikácie

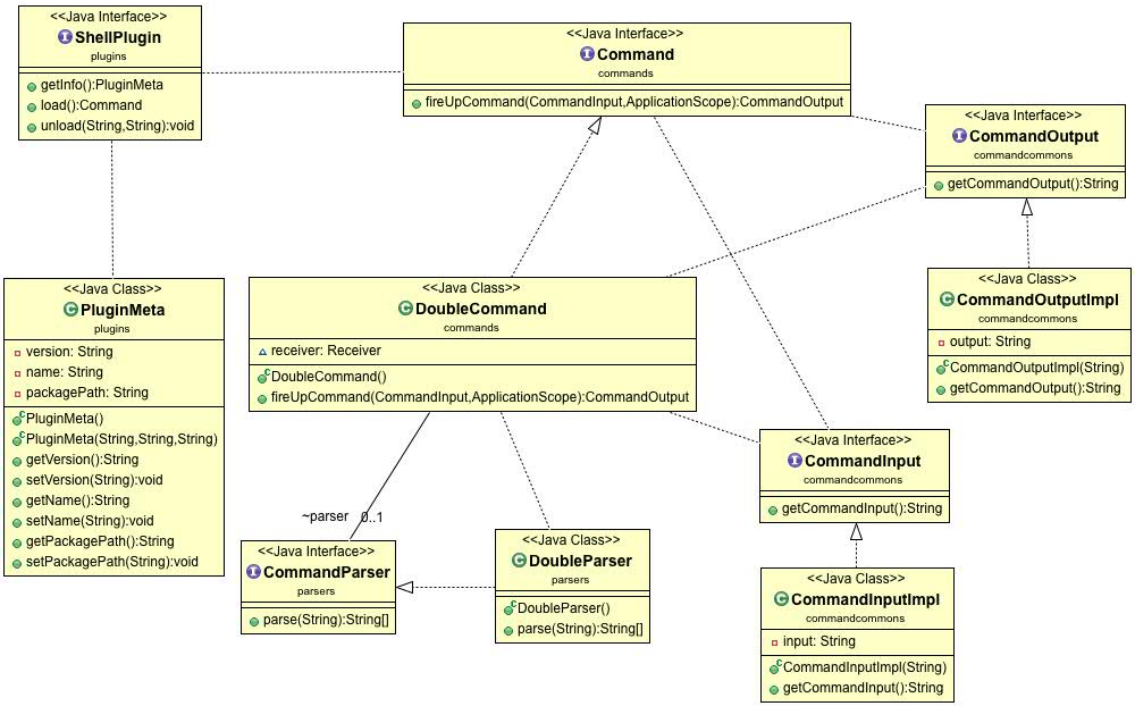
Ako prvé bolo treba zistiť z akých komponentov sa bude aplikácia skladať. Bolo treba zamyslieť sa čo a ako to chceme dosiahnuť. v prvom návrhu sme identifikovali nasledovné komponenty. Rozhodli sme sa, že vytvoríme plugin systém kvôli tomu, čo najdem a napíšem do analyzy.

- Parser - vstupov aj vystupov
- Loader jar súborov
- Sťahovač dependencií - jarka ktoré momentálne produkt neobsahuje napr. cusotm riešenia
- Scoping



Obrázok 10: Pvrvé funkčné riešenie

Z nasledovného class diagramu nebolo na prvý pohľad zreteľne viditeľné aké komponenty v programe existujú preto bolo potrebné zamyslieť sa ako by sa dali tieto časti rozumne rodeliť. Z prvotného návrhu sme vytiahli plugin. Pre implementáciu pluginu sme sa rozhodli použiť architektúru command patternu. Class diagram implementácie je viditeľný na nasledovnom obrázku.
Popisat scoping Vytvaranie commandov



Obrázok 11: Class diagram pluginu

5 Zhodnotenie výsledkov

Zatiaľ sa toho nespravilo hodne ale verím, že sa to tu cele zaplní.

Záver

Cieľom práce bolo zanalyzovať populárne konzolové rozhrania rovnako aj skriptovacie jazyky, ktoré sú často využívané pri administrácii počítačových systémov. Taktiež bolo treba nájsť jednotlivé výhody ako aj nedostatky jednotlivých rišení, zhodnotiť ich a nájsť medzi nimi rozumný prienik, ktorý bolo treba dostať do použiteľnej podoby. Kládli sme dôraz hlavne na to aby naše riešenie bolo čím najlepšie upraviteľné aby mohlo vyhovieť požiadavkam rôznych používateľov.

Zoznam použitej literatúry

1. ZARRELLI, Giorgio. *Mastering Bash*. 1. vyd. Birmingham : Packt Publishing, 2017, 2004. ISBN: 9781784396879.
2. CIACCIO, Robert S. *PowerShell vs. the Unix Shell*. 18-12-2010. Dostupné tiež z: `https://superuser.com/questions/223300/powershell-vs-the-unix-shell`.
3. ABRAHAM SILBERSCHATZ Peter B. Galvin, Greg Gagne. *Operating System Concepts - Ninth Edition*. 9. vyd. Wiley, 2012, 2012. ISBN: 978-1118063330.
4. KOLUGURI, Naveen. *If / Else Statements (Shell Scripting) - Code Wiki*. 11-11-2017. Dostupné tiež z: `http://codewiki.wikidot.com/shell-script:if-else`.
5. BRENTON J.W. BLAWAT, Chris Dent. *Mastering Windows PowerShell Scripting - Second Edition*. 2. vyd. Birmingham : Packt Publishing, 2017, 2004. ISBN: 9781787126305.
6. NICHOL, Alex. *unixpickle/Benchmarks: Some language performance comparisons.* 12-04-2017. Dostupné tiež z: `https://github.com/unixpickle/Benchmarks`.

Prílohy

A CD s aplikáciou

B Návod na spustenie a používanie aplikácie

Ako spustit a pouzivat app.