Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Luka Baša

Razvoj domensko-specifičnega jezika za Slack bote

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: viš. pred. dr. Igor Rožanc

Ljubljana, 2019

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Razvoj domensko-specifičnega jezika za Slack bote

Tematika naloge:

Domensko-specifični jezik (DSL) je programski jezik, ki je prilagojen določeni domeni in kot tak omogoča poznavalcem učinkovitejše pisanje domenskih rešitev. V diplomski nalogi prikažite razvoj DSL-ja, ki je namenjen izdelavi spletnih (ro)botov s programskim orodjem Slack. V delu najprej predstavite značilnosti DSL-jev, problemsko domeno razvoja botov ter orodje Slack. Osrednji del naloge naj predstavlja razvoj notranjega DSL-ja z uporabo programskega jezika Ruby. Uporabnost razvitega DSL-ja potrdite z izdelavo preprostega spletnega bota in prikazom njegovega delovanja. Nalogo zaključite z analizo uporabnosti razvitega jezika.

Za vso podporo in motivacijo se zahvaljujem družini, prijateljem in svojemu dekletu. Hvala tudi mentorju, viš. pred. dr. Igorju Rožancu, za pomoč in usmerjanje.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
DSL	Domain-Specific Language	domensko-specifični jezik
GPL	General-Purpose Language	splošno namenski jezik
API	Application Programming Interface	aplikacijski programski vmesnik
IDE	Integrated Development Environment	razvojno okolje
SQL	Structured Query Language	strukturirani povpraševalni jezik
HTML	Hyper Text Markup Language	jezik za označevanje nadbesedila
CSS	Cascading Style Sheetst	kaskadne stilske podloge
XML	eXtensible Markup Language	razširljivi označevalni jezik

Povzetek

Naslov: Razvoj domensko-specifičnega jezika za Slack bote

Avtor: Luka Baša

Cilj diplomskega dela je razvoj domensko-specifičnega jezika (DSL-ja), kateri bo uporabnikom omogočal hitro, enostavno in zanesljivo izdelavo Slack bota. DSL-ji so programski jeziki namenjeni specifični domeni. V našem primeru DSL razvijamo za manjšo domeno, katera pokriva področje izdelave Slack botov.

Ključna razloga za izdelavo omenjenega DSL-ja sta časovna potratnost izdelave Slack bota in zahtevnost programiranja. Analiza problema je pokazala, da dejanska rešitev v obliki DSL-ja še ne obstaja oziroma ni javno dostopna. Zato smo se odločili za razvoj notranjega DSL-ja, ki temelji na programskem jeziku Ruby. Da bi bila arhitektura našega DSL-ja čim preglednejša, smo izvedbo rezdelili na dva dela. DSL razčlenjevalnik in DSL odjemalec. Prvi skrbi za obdelavo DSL skripte, drugi pa za komunikacijo s programskim orodjem Slack. Skupaj omogočata enostavno izdelavo Slack botov s različnimi funkcionalnostimi. Prikaz in uporabo teh lahko vidimo na primeru Slack bota Meppo, izdelanega v sklopu diplomske naloge. Končna analiza DSL-ja je pokazala, da so bili cilji izpolnjeni, saj DSL občutno poenostavi in pohitri izdelavo Slack botov.

Ključne besede: domensko-specifični jezik, DSL, Ruby, Slack, bot.

Abstract

Title: Development of Domain-Specific Language for the Slack bots

Author: Luka Baša

The goal of the thesis is to present the development of Domain-Specific Language (DSL), which allows users to quickly, easily and reliably produce Slack bots. DSLs are programming languages for a specific domain. In our case, a DSL is developed for a smaller domain that covers the creation of the Slack bots.

The key reasons for making such a DSL are the time-consuming production of the Slack bots and complex programming skills required. The analysis of the problem has shown an actual solution in the form of DSL is missing or at least not publicly available. We decided to develop an internal DSL based on the programming language Ruby. To design the architecture of our DSL as simple as possible, we have divided it into two parts: DSL parser and DSL client. The first one deals with the processing of DSL scripts while the other communicates with the Slack software. Together they make it easy to create Slack bots with different functionality. We can see and use these examples on the Slack bot Meppo, which was created as part of the thesis. The final DSL analysis proved that the goals were successfully fulfilled. DSL considerably simplifies and speeds up the production of Slack bots.

Keywords: Domain-Specific Language, DSL, Ruby, Slack, bot.

Poglavje 1

Uvod

Dandanes si podjetja uspešno poslovanje zagotavljajo z učinkovito uporabo različnih orodij. Eno izmed njih je programsko orodje Slack. Gre za napredno komunikacijsko orodje, ki je priljubljeno predvsem v računalniških in informacijskih podjetjih, njegovo uporabo pa zaznamo tudi na marsikaterem drugem področju [13]. Slack je javnost pritegnil predvsem zaradi svojih razširitev, med katere spadajo tudi boti (angl. internet bots - bots). To so aplikacije, ki z orodjem Slack komunicirajo v realnem času, s tem pa uporabniku omogočajo različne funkcionalnosti [15].

Izdelava slack bota je dokaj preprosto opravilo, a le za uporabnika, ki dobro pozna določen programski jezik. Da bi to pomankljivost odpravili, smo se odločili za izdelavo DSL-ja, ki bo uporabniku omogočal hitrejšo in enostavnejšo izdelavo bota, hkrati pa ta zaradi uporabe DSL-ja ne bo več potreboval programerskega predznanja.

Naš cilj je tako ustvariti DSL s čim enostavnejšo in berljivejšo sintakso, ki omogoča učinkovito definicijo botov. Ta mora obenem uporabniku še vedno omogočati dovolj svobode pri izdelavi Slack bota.

Diplomsko delo je v grobem razdeljeno na tri dele. Najprej bomo na kratko predstavili tehnologije, jezike in orodja, ki so potrebna za lažje razumevanje osrednjega dela diplome (DSL, Ruby, Slack). Sledi podroben opis razvoja DSL-ja od ideje do končne rešitve. Da bi ta del ustrezneje pojasnili,

predstavimo izdelavo našega lastnega bota Meppo. Za konec predstavimo še pridobljene rezultate, izpolnitev ciljev, statistične podatke, primerjave in mnenja uporabnikov ter jih podrobno analiziramo.

Poglavje 2

Domensko-specifični programski jeziki

2.1 Domensko-specifični programski jeziki

Meje med domensko-specifičnimi programskimi jeziki (angl. Domain-Specific Language - DSL), programskimi knjižnicami in ogrodji velikokrat niso povsem jasne in jih je težko določiti [21]. Sama definicija DSL-jev pa je precej preprosta, vendar nikoli dokončno dorečena. Tako lahko DSL definiramo kot računsko omejen, programski jezik za reševanje problemov v specifični domeni [5].

Definicija je sestavljena iz štirih ključnih elementov: [5]

- **Računsko omejen:** DSL vsebuje samo funkcionalnosti, ki so nujno potrebne za opravljanje naloge v specifični domeni. Ni ga priporočljivo uporabljati v splošne namene.
- **Programski:** DSL uporablja človek za podajanje ukazov računalniku. Struktura DSL-ja je sestavljena tako, da jo človek z lahkoto razume, računalnik pa brez težav izvaja.
- **Jezik:** DSL je programski jezik. Kot tak mora imeti lastnosti jezika, ki se posamezne stavke oziroma ukaze združuje v smiselno celoto.

- **Domensko speifičen:** DSL je resnično uporaben samo, če služi točno določeni specifični domeni.

2.2 Vrste DSL-jev

V grobem DSL-je delimo na dve pomembnejši veji: **notranje** (angl. internal) in **zunanje** (angl. external) DSL-je [21].

2.2.1 Notranji DSL-ji

Notranji DSL temelji na splošno-namenskem programskem jeziku (angl. general-purpose language - GPL). To pomeni, da je izvajalna skripta DSL-ja veljavna programska koda v GPL-ju, na katerem DSL temelji. Vendar notranji DSL pokriva le majhen oziroma potreben del GPL-ja in njegovih funkcionalnosti. Kot rezultat moramo imeti občutek, da gre za programski jezik, ki je narejen po meri in ne spominja na programski jezik gostitelja [5]. Za izdelavo notranjega DSL-ja lahko uporabimo katerikoli GPL, vendar najpogosteje uporabimo tiste, ki omogočajo metaprogramiranje (poglavje 3.2) [21]. Eden izmed najbolj priljubljenih GPL-jev za izdelavo notranjih DSL-jev je Ruby [11].

2.2.2 Zunanji DSL-ji

Zunanji DSL je za razliko od notranjega DSL-ja ločen od glavnega programskega jezika (GPL-ja), ki ga uporablja aplikacija. To pomeni, da ima DSL praviloma svojo sintakso in ni odvisen od gostiteljskega programskega jezika. Za delovanje potrebujemo rezčlenjevalnik (angl. parser), ki poskrbi, da se DSL koda rezčleni in pretvori v gostiteljski programski jezik [5].

2.3 Prednosti in slabosti DSL-jev

Glavne prednosti DSL-jev so: [5, 4, 21]

- **Učinkovitost:** Delo s pomočjo DSL-ja postane lažje in učinkovitejše. Ne potrebujemo več nepotrebnega časa za pisanje GPL kode, ampak vse rešimo z nekaj vrsticami DSL kode. S tem se poveča učinkovitost dela in zmanjša časovna zahtevnost.

- **Kakovost:** DSL pripomore k sami kakovosti programa. Program vsebuje manj napak, programskih hroščev, ima ustrezno arhitekturo in kot tak omogoča enostavnejše vzdrževanje.

- **Validacija in verifikacija:** Ker so DSL-ji semantično bogatejši od GPL-jev, je veliko lažje implementirati analize. Prav tako lahko uporabniku podrobneje opišemo napake, saj so te v večini primerov povezane z domeno.

- **Vključenost domenskih poznavalcev:** S pomočjo DSL-ja lahko domenski poznavalci (angl. domain experts) sami ali s pomočjo razvijalca v paru enostavno razvjajo svoje DSL skripte tudi brez programerskega predznanja. Če delo poteka v paru, koristne informacije o domeni pridobijo tudi razvijalci.

- **Specifična orodja:** Zunanji DSL mora biti podprt z uporabniškimi orodji. Primer takega orodja je razvojno okolje (angl. Integrated development environment - IDE). S pomočjo sodobnega IDE-ja lahko močno izboljšamo uporabniško izkušnjo in učinkovitost.

- **Hitrost učenja:** Domenski poznavalci lahko v učenju DSL-ja napredujejo zelo hitro. Zaradi domeni prilagojene sintakse in semantike je domenskemu poznavalcu veliko lažje razumeti DSL kot celoten GPL.

- **Izoliranost:** DSL-ji v večini primerov omogočajo neodvisnost od ciljne platforme, zato jo lahko brez težav menjujemo.

Seveda imajo DSL-ji tudi nekaj slabosti. Najzaznavnejše so: [5, 4, 21]

- **Dolgotrajno in drago:** Izdelava samega DSL-ja je lahko časovno zelo potratna, zato se podjetja nerada odločajo zanjo. Razvoj DSL-ja

nas tako lahko veliko stane, še posebej, če ga razvijamo za majhno in neznano specifično domeno.

- **Potrebne izkušnje:** DSL lahko dobro in kvalitetno razvijejo le izkušeni razvijalci, ki dovolj vedo o domeni.

- **Razvoj in vzdrževanje:** Tako kot GPL je tudi DSL potrebno nenehno vzdrževati in nadgrajevati. Če to fazo zanemarimo DSL zastara in sčasoma postane neuporaben.

- **Razvoj novih DSL-jev:** Čeprav DSL za neko domeno že obstaja, se velikokrat zgodi, da podjetja tega ne vedo. Razvijajo novo različico, ki je zelo podobna obstoječi.

- **Učenje DSL-ja:** Učenje DSL-ja je veliko hitrejše od učenja GPL-ja, a še vedno zahteva svoj čas. Podjetje, ki je naročnik nekega DSL-ja mora v stroške vključiti tudi izobraževanje domenskih poznavalcev, ki bodo ta DSL uporabljali.

	GPL	**DSL**
Domena	velika in zapletena	majhna in dobro definirana
Velikost jezika	velika	majhna
Turingova popolnost	vedno	zelo redko
Uporabniške abstrakcije	sofisticirane	omejene
Izvajanje	preko GPL	izvirno
Življenska doba	leta do desetletja	meseci do leta
Razvito od	guru ali odbor	nekaj inžinirjev in domenskih strokovnjakov
Uporabniška skupnost	velika, anonimna in razširjena	majhna, dostopna in lokalna
Razvoj	počasen, pogosto standariziran	hiter
Zastarelost kode	skoraj nemogoča	mogoča

Tabela 2.1: Primerjava med DSL-ji in GPL-ji [21].

Zanimive so tudi primerjave med DSL-ji z GPL-ji (tabela 2.1), iz katerih lahko razberemo nekatere prednosti in slabosti med slednjimi. Opazimo, da GPL-ji pokrivajo veliko večjo in zapletenejšo domeno, so večji jezik in

vedno Turingovo popoln. Živijo lahko desetletja, zastarelost kode pa skoraj ni mogoča. Na drugi strani DSL pokriva majhno in dobro definirano domeno, njegov razvoj je hiter, saj gre za majhen jezik, ki je pogosto razvit s strani nekaj inžinirjev in domenskih poznavalcev. Zastarelost kode je mogoča, življenska doba jezika pa se giblje od nekaj mesecov do let.

2.4 Primeri DSL-jev

Poznamo veliko različnih DSL-jev. Medtem, ko so nekateri namenjeni manjšim zapirtim domenam oziroma domenskim poznavalcem, se druge uporablja globalno in zelo pogosto. Med najbolj znane vsekakor sodijo: [20]

- **SQL** ali strukturirani povpraševalni jezik za delo s podatkovnimi bazami (angl. Structured Query Language) [18]. Primer vidimo v kodi 2.1.

```
SELECT a.title, a.description, a.author, i.url, i.width
FROM Articles a
WHERE a.premium_only = 1 AND a.author != "Tomi Jerovina"
INNER JOIN Images i ON Articles.article_id = Images.article_id
ORDER BY a.post_datetime DESC;
```

Koda 2.1: Primer SQL kode.

- **HTML** ali jezik za označevanje nadbesedila (angl. Hyper Text Markup Language) [7]. Izgled kode vidimo v kodi 2.2.

```
<html>
  <head>
    <title>Test website for amazing DSL</title>
    <link rel="stylesheet" type="text/css" href="style.css">
  </head>
  <body>
    <div id="main_container">
      <h1>DSLs rules!</h1>
      <p>We are changing the world with Technology - Bill Gates</p>
    <div>
    <div id="bottom">
      <span class="small">
        Welcome to our <i>website</i>!
      </span>
    </div>
  </body>
</html>
```

Koda 2.2: Primer HTML kode.

- **CSS** ali kaskadne stilske podloge (angl. Cascading Style Sheets) [3]. Primer je viden v kodi 2.3.

```
* {
    background-image: linear-gradient(black, white, black);
    font-size: 17px;
    color: blue;
}

.main_container{
    height: 400px;
    background-image: repeating-linear-gradient(blue, yellow, blue);
    padding: 0 0 20px 0;
    margin-top: 10px;
}

#mid_select:hover{
    color: #323232;
    font-weight: bold;
}
```

Koda 2.3: Primer CSS kode.

- **XML** ali razširljivi označevalni jezik (angl. eXtensible Markup Language) [22]. Kako izgleda vidimo v kodi 2.4.

```
<?xml version="1.0" encoding="UTF-8"?>
<DSL>
    <internal>
        <name>Orange DSL</name>
        <price>$2000</price>
        <description>
            DSL for factory Orange.
        </description>
    </internal>
    <external>
        <name>Black DSL</name>
        <price>$15000</price>
        <description>
            DSL for factory Black.
        </description>
    </external>
</DSL>
```

Koda 2.4: Primer XML kode.

- **Latex** sistem za pripravo dokumentov [8]. Primer vidimo v kodi 2.5.

```
\documentclass{article}
\title{Test document for amazing DSL}
\author{Luka Basa}
\date{January 2019}
\begin{document}
    \maketitle
    DSLs rules!
\end{document}
```

Koda 2.5: Primer Latex kode.

Poglavje 3

Ruby in Slack

3.1 Programski jezik Ruby

Ruby je sistemsko neodvisni, objektno usmerjen splošni programski jezik) [9]. Razvil ga je Yukihiro "Matz" Matsumoto leta 1993, leta 1995 pa ga je javno objavil. Prepoznamo ga po logotipu rdečega dragulja (slika 3.1). Ena izmed njegovih glavnih značilnosti je izjemno "čista" in enostavna sintaksa, ki uporabniku omogoča izkoristiti neverjetno moč GPL-ja. Zanimiva lastnost Ruby-ja je objektna usmerjenost, saj čisto vse sestavljajo objekti (tudi številke, polja, nize,..). To pomeni, da lahko čisto vsem stvarem nastavljamo njihove lastnosti (angl. properties) in ukrepe (ang. actions) [9, 1].

Slika 3.1: Logotip programskega jezika Ruby.

Rubz je eden izmed najučinkovitejših jezikov. Združuje veliko dobrih funkcionalnosti iz drugih GPL-jev, kot so [19]:

- dinamično dodeljevanje tipov,
- izjeme (angl. exceptions),

- kodni bloki (angl. code-blocks),
- enostavno uporabo objektno usmerjenih knjižnic,
- udobno in že znano sintakso, ki izhaja iz drugih GPL-jev (C++, Eiffel, Perl in Python),
- poljubno natančna cela števila in njihovo pretvarjanje,
- lahke in nepotratne niti (angl. threads),
- vgrajene regularne izraze (angl. regular expressions).

3.2 Ruby in DSL

Ruby je izvrsten programski jezik za pisanje notranjih DSL-jev, saj omogoča metaprogramiranje (angl. metaprogramming) [11]. Metaprogramiranje je tehnika programiranja, ki omogoča programu, da med izvajanjem gradi (piše) nov program. To na grobo pomeni, da lahko program programira sam sebe ali druge programe [11]. Dobro metaprogramiranje v jeziku Ruby je mogoče predvsem zaradi njegove sintakse-blokov, ki podpirajo novo kontrolno strukturo, dinamike ter možnosti, da lahko spreminjamo vse, kar se spremeniti da. S pomočjo metaprogramiranja v Ruby-ju lahko enostavno razvijemo svoj DSL [6].

3.3 Programsko orodje Slack

Slack je programsko orodje za sodelovanje, ki povezuje neko organizacijo oziroma skupino tako, da lahko uspešno opravlja svoje delo. Glavna naloga Slack-a je povezati med seboj vse ljudi in "stvari", ki so pomembne. Gre za neke vrste komunikacijsko orodje, ki nam omogoča na tisoče različnih razširitev (angl. plugins). Nekatere najbolj znane so: Google dokumenti, koledar, opominki, alarmi, aplikacija za vremensko napoved, aplikacija za poročanje programskih napak ter še mnoge druge. Vredno je tudi omeniti,

da Slack lahko poganjamo na vseh večjih operacijskih sistemih, mobilnih napravah z nameščenim Android-om ali iOS-om, ter na vseh novejših brskalnikih [13, 16].

Slika 3.2: Slack logotip.

3.4 Slack API

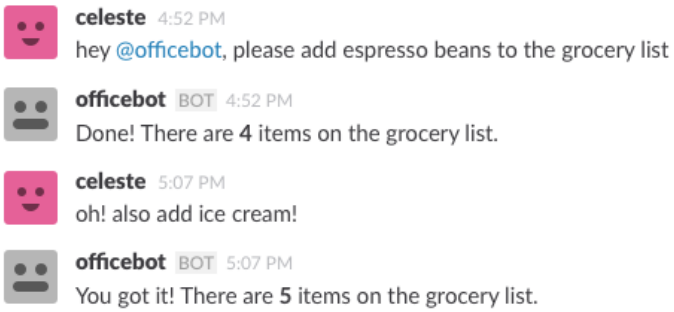
Kot smo že omenili so glavna prednost orodja Slack njegove razširitve oziroma zunanje aplikacije. Da te lahko komunicirajo z orodjem, je Slack ekipa razvila svoj API (angl. Application Programming Interface), kateri vse skupaj močno poenostavi. To pomeni, da lahko naš program s pomočjo ustrezne avtentikacije preko klicev na Slack API dostopa do naših podatkov v Slack-u. Za vse prepoznavnejše programske jezike obstajajo tudi razni API odjemalci (angl. API Clients), kot so "slack-ruby-client" (Ruby), "SlackAPI" (C#) in "slack-client" (Java). Odjemalec programiranje še bolj poenostavi. Skrbi za vse potrebno, predvsem pa za to, da komunikacija z API-jom poteka brez težav. Programerju se tako s tem ni potrebno ukvarjati in se lahko osredotoči le reševanje problema [14].

3.5 Slack boti

Slack bot (angl. Slack (ro)bot) je aplikacija, ki preko Slack API-ja komunicira s Slackom v realnem času. To pomeni, da nenehno posluša in spremlja uporabnike in dogodke, ter se nanje ustrezno odziva [15]. Način odziva je seveda stvar implementacije, odgovori pa lahko s sporočilom, ukazom ali nekim

dejanjem. Slack boti so lahko karkoli in lahko počnejo najrazličnejše stvari. Lahko uporabljajo celo umetno inteligenco ali podatkovno rudarjenje.

Bot (angl. internet bot) je potemtakem avtomatiziran program, ki se izvaja preko omrežja. Nekateri se izvajajo samodejno, drugi pa se odzivajo z določenim ukazom na določen vhod (angl. input). Poznamo več vrst botov. Najpogostejši so spletni pajki (angl. web crawlers), komunikacijski boti (angl. chat bots) in zlonamerni boti (angl. malicious bots) [2]. Slack bot spada v kategorijo komunikacijskih botov. Na sliki 3.3 vidimo primer komunikacijskega Slack bota, ki uporabniku omogoča izdelavo nakupovalnega seznama.



Slika 3.3: Primer Slack bota iz uradne Slack API dokumentacije [15].

Poglavje 4

Razvoj notranjega DSL-ja za Slack bota

4.1 Problemi in cilji

Za razvoj notranjega DSL-ja, ki bo pripomogel k izdelavi Slack botov, smo se odločili zaradi sledečih razlogov:

- **Časovna potratnost:** izdelava Slack bota je lahko zelo časovno potratna. Vsakič znova moramo poskrbeti za ustrezno komunikacijo med botom in Slackom, spisati logiko za ukaze, konfigurirati bota in podobno.
- **Nepreglednost:** koda Slack bota je ponavadi zapletena, dolga in težko pregledna. Programiranje bota zato postane zoprno in časovno zamudno opravilo.
- **Potrebno znanje:** uporabnik lahko ustvari svojega Slack bota le, če ima predhodno programersko znanje vsaj enega GPL-ja. Uporabnik brez ustreznega znanja težko ustvari Slack bota.
- **Zapleteno vzdrževanje:** po zaključeni izdelavi hitro pozabimo, kako vse skupaj sploh deluje. Če želimo karkoli spremeniti, potrebujemo ve-

liko časa in truda. Kodo je potrebno pregledati, posodobiti in testirati, kar je zamudno.

Iz teh razlogov smo si zastavili naslednje jasne cilje:

- ustvariti želimo DSL z **enostavno** in **berljivo** sintakso,
- s pomočjo katerega bo izdelovanje Slack botov **hitro** in **preprosto**,
- predvsem pa želimo ustvariti DSL, ki bo uporabnikom omogočal izdelavo Slack bota **brez programerskega predznanja**.

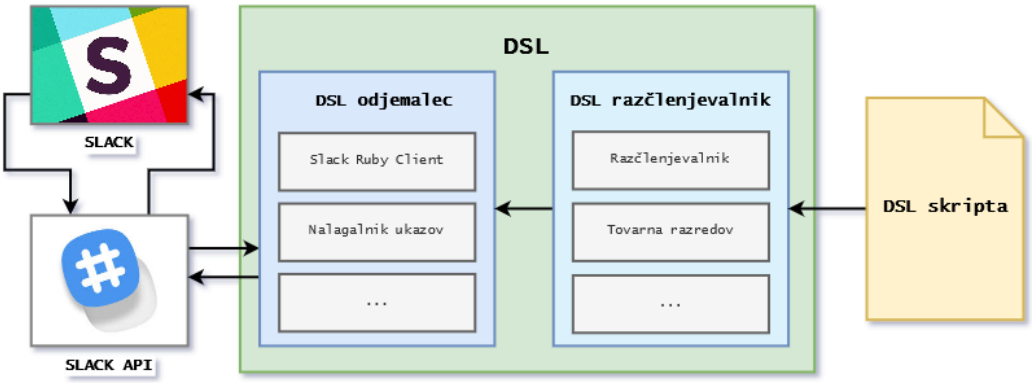
4.2 DSL za Slack bote

Potek izdelave DSL-jev podrobno in sistematično opiše dr. Marjan Mernik v članku [10]. Kljub dobro predstavljenim smernicam smo se odločili, da jih bomo zaobšli, saj bi bil takšen razvoj preobsežen za to diplomsko delo. Usmerili smo se v izdelavo notranjega DSL-ja in sicer v programskem jeziku Ruby. Cilji je bil ustvariti DSL s čim enostavnejšo in jasno berljivo sintakso, medtem pa uporabniku še vedno omogočati dovolj svobode pri izdelavi Slack bota. Zaradi te potrebe je bil DSL narejen tako, da poleg že vgrajenih funkcionalnosti omogoča programerjem dodajanje funkcionalnosti po meri (angl. custom features), katere lahko kasneje v DSL skripti enostavno uporabimo brez spreminjanja zasnove samega DSL-ja.

4.3 Ogrodje DSL-ja

DSL za izdelavo Slack botov se v grobem deli na dva dela: DSL razčlenjevalnik in DSL odjemalec. Prvi vsebuje vso potrebno logiko za branje, razčlenjevanje in obdelavo DSL skripte. Deli se na manjše dele, kot so razčlenjevalnik (skrbi za razčlenitev skripte), tovarna razredov (ustvari potrebne razrede) in pregledovalnik napak (preverja in sporoča napake v DSL kodi).

Drugi del je DSL odjemalec, ki skrbi za komunikacijo s Slack-om preko Slack API-ja, pošiljanje ustreznih odzivov glede na DSL skripto ter za izpisovanje sporočil in napak v konzolo. Sestavljen je iz manjših delov kot so: odprtokodni API odjemalec "Slack Ruby Client" [17], nalagalnik ukazov (ukaze, ki nam jih vrne DSL razčlenjevalnik, pripnemo na Slack Ruby Client) in izpisovalnik (izpisuje napake in sporočila v konzolo). Celotno strukturo vidimo na sliki 4.1.



Slika 4.1: Okvirna arhitektura DSL-ja.

4.3.1 DSL razčlenjevalnik

DSL razčlenjevalnik je glavni del našega DSL-ja. Njegova naloga je prejeto DSL kodo (ki je hkrati popolnoma veljavna ruby koda) prebrati, razčleniti na posamezne dele, se nanje pravilno odzvati in vrniti rezultate v obliki podatkov oziroma ukazov, ki jih bo DSL odjemalec kasneje razumel. Ena izmed glavnih metod pri izdelavi DSL-jev v Ruby-ju je `instance_eval`. Metoda izvede podano kodo (v našem primeru blok kode, koda 4.1) v kontekstu sprejemnika (objekta, v našem primeru instanca razreda) [12].

```
1 class DSL
2   def process(&block)
3     instance_eval(&block)
4     { config: @config, commands: @commands }
5   end
6
7   def config(&block)
```

```
8       @config = Config.new.init_config(&block)
9     end
10
11    def command(name = nil, &block)
12      @cmd_class = Commands.new if @cmd_class.nil?
13      @commands = @cmd_class.init_command(name, &block)
14    end
15  end
```

Koda 4.1: Primer uporabe metode `instance_eval`.

Da bi lažje razumeli celotno delovanje, si poglejmo delovanje DSL razčlenjevalnika na krajšem primeru. Podano imamo krajšo DSL skripto (koda 4.2), v kateri je definirana konfiguracija Slack bota z enim ukazom (angl. command). Ta našemu Slack botu ukazuje, da za vsako besedilo, ki je enako nizu `DSL` odgovori z nizom `Domain-Specific Language`.

```
1  require_relative 'slack_bot'
2
3  SlackBot.define do
4    config do
5      token 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
6    end
7
8    command(:dsl) do
9      on 'DSL'
10     output 'Domain-Specific Language'
11   end
12 end
13
14 SlackBot.run
```

Koda 4.2: Kratka DSL skripta.

DSL skripta najprej pokliče razred `SlackBot` in njegovo metodo `define`, v katero pošlje blok kode (`&block`). Metoda nato ustvari novo instanco razreda `DSL` (kar predstavlja DSL razčlenjevalnik), na njej pokliče metodo `process` in ji posreduje prejeti blok kode (koda 4.3).

```
1  class SlackBot
2    def self.define(&block)
3      @core = DSL.new.process(&block)
4
5      pp(BREAK_LINE, 'BOT CORE', BREAK_LINE, @core, BREAK_LINE) if @core[:config][:debug]
       == 'deep'
6
7      @bot = Bot.new(@core)
8    end
9
10   def self.run
11     @bot.start
12   end
13 end
```

Koda 4.3: Razred SlackBot.

Metoda `process` razreda `DSL` blok kode sprejme in ga celotnega požene z Ruby metodo `instance_eval`. To pomeni, da se v našem primeru najprej požene metoda `config`, nato pa metoda `command` (koda 4.4). Metoda `config` kreira novi razred `Config`, na katerem kliče metodo `init_config`, ki zopet ponovi postopek. Blok kode razčleni, obdela in vrne nastavitev za našega Slack bota, katero bo kasneje potreboval DSL odjemalec. Podobno deluje tudi metoda `command`. Najprej ustvari razred `Commands`, če ta še ne obstaja, nato pa temu razredu pripne oziroma doda ukaz preko metode `init_command`. Globje v temu razredu se vse skupaj še močneje razčleni na različne vrste razredov in metod, katerih sedaj ne bomo podrobnejše opisovali. Slednjič DSL razred vrne podatkovno strukturo, ki vsebuje vse potrebno (nastaviteve in ukaze), da lahko nastavimo in poženemo DSL odjemalca.

```
1  class DSL
2    def process(&block)
3      instance_eval(&block)
4      { config: @config, commands: @commands }
5    end
6
7    def config(&block)
8      @config = Config.new.init_config(&block)
9    end
10
11   def command(name = nil, &block)
12     @cmd_class = Commands.new if @cmd_class.nil?
13     @commands = @cmd_class.init_command(name, &block)
14   end
15 end
```

Koda 4.4: Razred DSL.

4.3.2 DSL odjemalec

Da lahko poženemo DSL odjemalec, potrebujemo nastavitve in ukaze, ki nam jih priskrbi DSL razčlenjevalnik. Ko imamo vse potrebno, tvorimo instanco razreda `Bot`, kateremu podamo ustrezne podatke (konfiguracijo in ukaze, koda 4.3). V instanci nato nastavimo Slack bota, mu pripnemo naše ukaze ter se povežemo na Slack API. V kolikor ni prišlo do nobene napake ali opozorila, Slack bota še poženemo. S tem našemu Slack botu omogočimo komunikacijo z programskim orodjem Slack v realnem času. Vso komunikacijo med Slack botom in orodjem Slack lahko seveda spremljamo

tudi v konzoli in sicer tako, da v DSL skripti omogočimo razhroščevanje.

4.4 Funkcionalnosti DSL-ja

Trenutni DSL nam omogoča kar nekaj funkcionalnosti, še veliko pa jih je mogoče implementirati dodatno (`custom_features`). Naštejmo nekaj najbolj uporabljenih:

- **Odziv na točno določeno besedilo:** Slack bot se lahko odzove na točno določeno besedilo sporočila uporabnika.
- **Odziv na točno določen niz v besedilu:** Slack bot se odzove, če se v besedilu sporočila uporabnika pojavi točno določeni niz.
- **Odziv na omembo:** Slack bot se odzove, če ga uporabnik omeni v besedilu sporočila.
- **Kombinacija zgornjih odzivov:** možno je uporabiti tudi kombinacijo odziva na omembo z drugima dvema odzivoma. Primer: uporabnik je omenil Slack bota v besedilu sporočila, hkrati pa se v besedilu nahaja tudi točno določen niz.
- **Odzovi se z nizom:** Slack bot se lahko odzove z točno določenim nizom.
- **Odzovi se z funkcionalnostjo:** Slack bot se lahko odzove z implementirano funkcionalnostjo. Trenutno bot vsebuje dve, prikaz trenutnega vremena in vrednosti kriptovalut.
- **Odzovi se s po meri narejeno funkcionalnostjo:** Slack bot se odzove z funkcionalnostjo po meri, ki jo programer napiše v posebni za to namenjeni ruby skripti `custom_features.rb` kot metodo razreda `CustomFeatures`.

- **Konzola in napake:** DSL omogoča tudi konzolni prikaz komunikacije med orodjem Slack in Slack botom. Prav tako je poskrbljeno za javljanje različnih napak pri pisanju DSL skripte.

Poglavje 5

Razvoj Slack bota s pomočjo DSL-ja

5.1 Slack bot Meppo

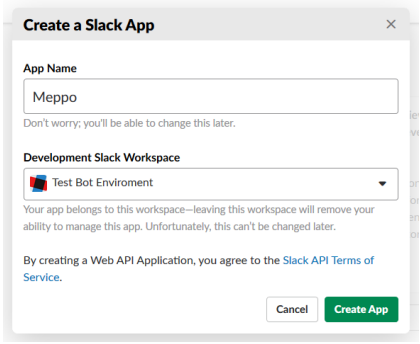
V namen predstavitve delovanja DSL-ja bomo izdelali preprostega Slack bota. Poimenujmo ga Meppo. Da bi prikazali čim več funkcionalnosti, ki jih DSL omogoča, si ga zamislimo nekako takole:

- Meppo uporabnika pozdravi, če ga ta v klepetu omeni.
- Meppo zna uporabniku sporočiti vrednost kriptovalut.
- Meppo zna uporabniku sporočiti trenutno vremensko stanje v določenem kraju.
- Meppo ob nizu `sport` uporabniku ponudi bližnjico do spletnih novic `24ur.com` rubrike šport.
- Meppo ve, da imamo radi glasbo in ob omembi niza `yt` uporabniku predlaga stran `youtube.com`.

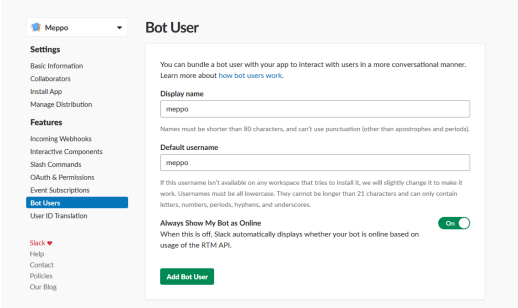
5.2 Od zasnove do celote

Izdelava Slack bota je s pomočjo DSL-ja zelo enostavna. Za demonstracijo bomo izbrali Slack bota Meppo. Da pa lahko začnemo s samim pisanjem DSL skripte, je potrebno prej urediti še nekaj stvari.

- **1. korak:** V orodju Slack moramo ustvariti novo aplikacijo (slika 5.1) in nanjo pripeti bot uporabnika (angl. bot user) (slika 5.2). Aplikacijo moramo nato prijaviti v našem Slack delovnem okolju (slika 5.3), saj bomo potrebovali ključ bot uporabnika (slika 5.4).

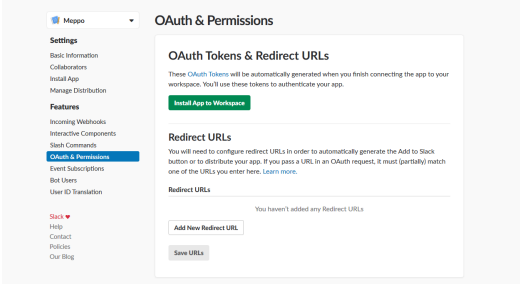


Slika 5.1: Kreiranje nove aplikacije v Slack-u.

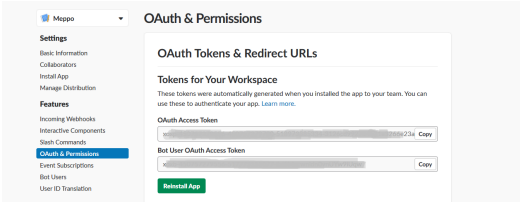


Slika 5.2: Dodajanje bot uporabnika v Aplikacijo.

- **2. korak:** Ko imamo ključ, lahko začnemo z razvojom Slack bota. Najprej ga definiramo in nastavimo konfiguracijo (koda 5.1).



Slika 5.3: Avtentikacija aplikacije v Slack delovnem okolju.



Slika 5.4: Generirani ključi po avtentikaciji.

```
1 SlackBot.define do
2   config do
3     token 'xoxb-450757-xxxxxxx-0vnUYw9iUqw'
4     debug 'deep'
5   end
```

Koda 5.1: Začetna nastavitev Slack bota.

- **3. korak:** Po konfiguraciji sledijo ukazi, katere želimo, da jih naš Slack bot izvršuje. Kot prvega bomo implementirali ukaz `:hello`, kateri bo po uporabnikovi omembi bota Meppota, pozdravil uporabnika (koda 5.2).

```
1   command(:hello) do
2     on '@'
3     output 'Hello :), how are you today?'
4   end
```

Koda 5.2: Ukaz `:hello`.

- **4. korak:** Implementirajmo bližnjico do športnih novic. Ko bo uporabnik vnesel besedilo sport, naj mu Meppo vrne povezavo do ustreznih

novic (koda 5.3).

```
1  command(:sport_news) do
2    on 'sport'
3    output 'https://www.24ur.com/sport'
4  end
```

Koda 5.3: Ukaz `:sport_news`.

- **5. korak:** Sedaj je na vrsti ukaz za povezavo do glasbe. Ukaz bo dokaj podoben ukazu `:sport_news`, z manjšo razliko. Želimo, da se naš Slack bot Meppo odzove na niz "yt"ne glede na to, kje se ta pojavi (koda 5.4).

```
1  command(:youtube) do
2    search_on 'yt'
3    output 'https://www.youtube.com/'
4  end
```

Koda 5.4: Ukaz `:youtube`.

- **6. korak:** Za implementacijo nam preostaneta še dve funkcionalnosti in sicer prikazovanje trenutnega vremena ter vrednosti kriptovalut. Tukaj je sintaksa malo drugačna (koda 5.5).

```
1  command(:weather) do
2    search_on '@ weather in *'
3    run_feature(:weather) do
4      key '9b3d88xxxxxxxxxxxxxbe30e'
5    end
6  end
7
8  command(:crypto) do
9    search_on '@ status of *'
10   run_feature(:crypto)
11 end
```

Koda 5.5: Ukazi z funkcionalnostjo.

- **7. korak:** Na koncu Slack bota še poženemo. Z vrstico: `SlackBot.run`

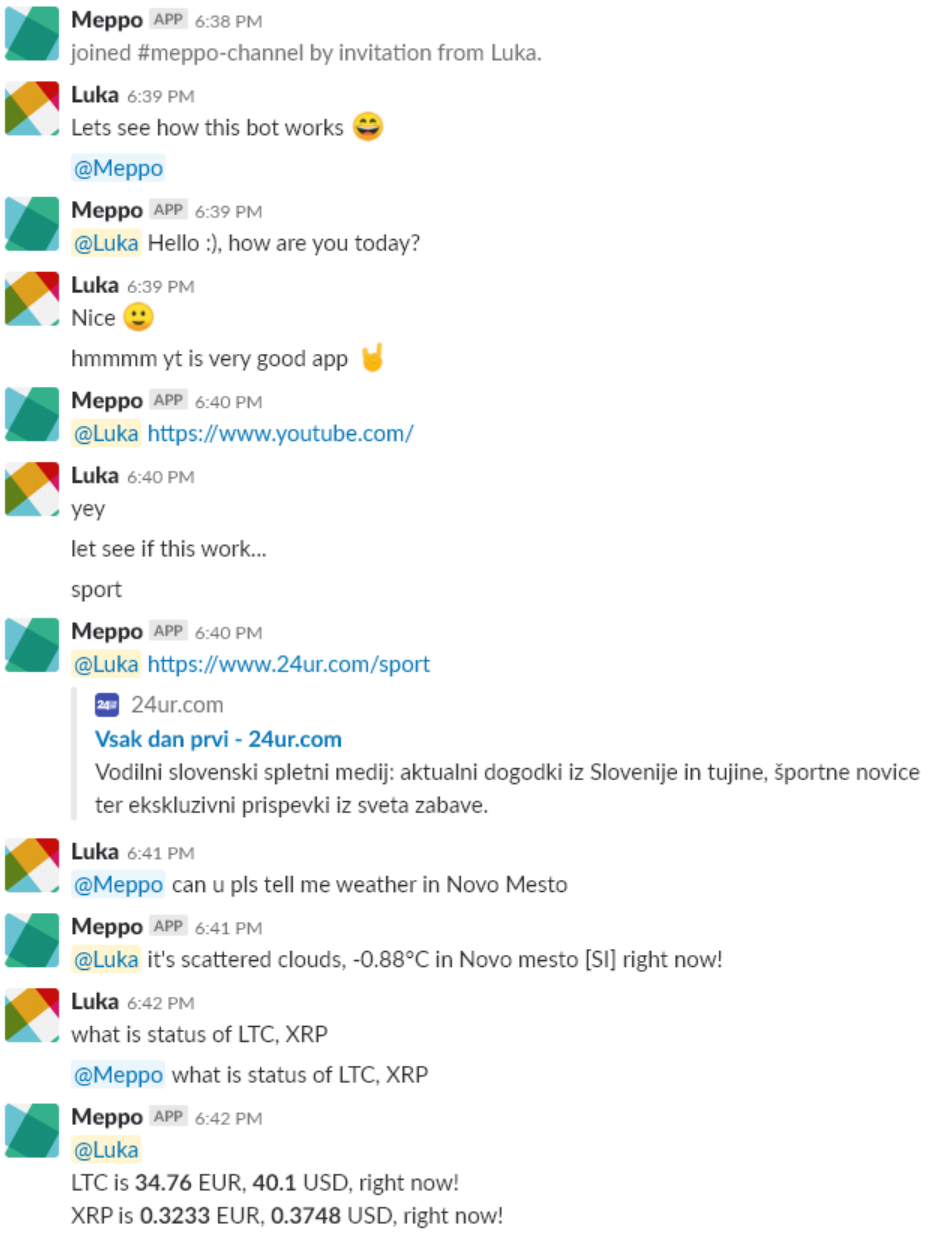
S tem je naš bot Meppo zaključen. Celotno DSL kodo si lahko ogledamo na kodi 5.6.

```
1 SlackBot.define do
2   config do
3     token 'xoxb-450757-xxxxxxx-0vnUYw9iUqw'
4     debug 'deep'
5   end
6
```

```
7   command(:hello) do
8     on '@'
9     output 'Hello :), how are you today?'
10  end
11
12  command(:sport_news) do
13    on 'sport'
14    output 'https://www.24ur.com/sport'
15  end
16
17  command(:youtube) do
18    search_on 'yt'
19    output 'https://www.youtube.com/'
20  end
21
22  command(:weather) do
23    search_on '@ weather in *'
24    run_feature(:weather) do
25      key '9b3d88xxxxxxxxxxxxxbe30e'
26    end
27  end
28
29  command(:crypto) do
30    search_on '@ status of *'
31    run_feature(:crypto)
32  end
33 end
34
35 SlackBot.run
```

Koda 5.6: Celotna DSL koda.

Delovanje bota Meppo prikazuje slika 5.5. Slack bot deluje kot je bilo pričakovano. Prepozna le ukaze, ki smo mu jih definirali, ter se nanje ustrezno odzove.



Slika 5.5: Delovanje bota Meppo.

Poglavje 6

Analiza

6.1 Rezultati

DSL se je pokazal kot odlična rešitev, čeprav smo za njegov razvoj porabili veliko časa. Zmanjšal je tako število potrebnih vrstic kot tudi časovno zahtevnost za izdelavo Slack bota. Če za primer vzamemo Slack bota Meppo (podpoglavje 5.1) in v tabeli 6.1 zanj prikažemo primerjavo med izdelavo bota v DSL-ji in programskem jeziku Ruby vidimo, da DSL močno zmanjša število vrstic in čas razvoja. Za razvoj v DSL-ju je potrebno približno kar desetkrat manj vrstic in časa. Prav tako vidmo, da je izdelava DSL-ja časovno potratno opravilo. Razvoj DSL-ja, pa se bo vsekakor izplačal na dolgi rok, saj bomo za izdelavo večjega števila Slack botov prihranili ogromno časa. Če predpostavimo, da v povprečju izdelujemo enako zahtevne Slack bote kot je Meppo, na izdelavo Slack bota prihranimo 225 minut. To pomeni, da bo razvoj DSL-ja in učenje metaprogramiranja časovno pokrito po izdelavi destih Slack botov (račun 6.1).

$$(30ur + 5ur)/225minut = 9.33 \tag{6.1}$$

	DSL	**Ruby**
Dolžina kode (št. vrstic)	37	300+
Učenje jezika Ruby	15 ur + 5 ur (metaprogramiranje)	15 ur
Učenje Slack API-ja	5 ur	5 ur
Razvoj jezika (DSL-ja)	30 ur	0
Razvoj Slack bota	15 minut	240 minut

Tabela 6.1: Primerjava med izdelavo bota Meppo v DSL in Ruby.

6.2 Testiranje uporabnikov

Za testiranje učinkovitosti DSL-ja v praksi smo izbrali različne osebe. Oseba 1 je izkušeni programer. Oseba 2 je študent pedagoške fakultete smeri računalništvo in matematika. Oseba 3 je študentka zdravstvene fakultete. Vse osebe smo obravnavali posamezno. Za uvod smo jim najprej postavili tri vprašanja:

- **Vprašanje 1:** Kako dobro poznaš programsko orodje Slack?
- **Vprašanje 2:** Približno oceni, koliko časa bi porabil(a) za razvoj Slack bota (chat bot) v svojem najljubšem programskem jeziku?
- **Vprašanje 3:** Če bi obstajal lažji način za izdelavo Slack bota, ki ne bi zahteval programerskega znanja, ali bi ga poskusil(a)?

Odgovore lahko vidimo v tabeli 6.2.

	Oseba 1 (programer)	**Oseba 2 (š. računalništva)**	**Oseba 3 (š. zdravstva)**
Vprašanje 1	sem redni uporabnik	slišal sem že zanj	žal, ne poznam
Vprašanje 2	do 2 uri	verjetno več kot en dan	res,nevem
Vprašanje 3	seveda	da	morda

Tabela 6.2: Odgovori oseb na vprašanja 1,2 in 3.

Osebam smo nato na kratko predstavili naše diplomsko delo, naše cilje ter končni izdelek DSL za izdelavo Slack botov. Tega smo demonstrirali še

podrobneje in jim dovolili, da ga sami preizkusijo. Testiranje na posameznika je bilo zaradi lažje izvedbe časovno omejeno in je potekalo 30 minut. Po končanem preizkusu smo jim postavili dodatna vprašanja:

- **Vprašanje 4:** Kako bi ocenil(a) težavnost izdelave Slack bota preko DSL-ja?
- **Vprašanje 5:** Kako bi ocenil(a) časovno zahtevnost izdelave Slack bota preko DSL-ja?
- **Vprašanje 6:** Ali bi DSL priporočil prijatelju, ki želi izdelati svoj Slack bot?

Odgovore lahko vidimo v tabeli 6.3.

	Oseba 1 (programer)	**Oseba 2 (š. računalništva)**	**Oseba 3 (š. zdravstva)**
Vprašanje 4	dokaj enostavno	izgleda kar enostavno	nekaj srednjega
Vprašanje 5	veliko hitreje	kar hitro	ko se enkrat naučiš dokaj hitro
Vprašanje 6	mislim, da ja	bi	nevem, morda

Tabela 6.3: Odgovori oseb na vprašanja 4,5 in 6.

Odziv je pozitiven. Testne osebe so bile presenečene nad enostavnostjo in uporabnostjo. Dobili smo tudi nekaj povratnih informacij. Oseba 1 bi želela DSL z večjim naborom funkcionalnosti, oseba 3 pa boljšo dokumentacijo o uporabi DSL-ja za lažje razumevanje.

Poglavje 7

Zaključek

7.1 Zaključek

DSL za izdelavo Slack botov je izpolnil vse zadane cilje. Z njegovo uporabo nam je uspelo močno zmanjšati čas razvoja, kar je bil eden naših glavnih ciljev. Prav tako smo znižali zahtevnost programske kode, saj uporabniki sedaj ne potrebujejo več poznavanja GPL-ja, da bi ustvarili svoj Slack bot.

DSL koda je lažje berljiva, učljiva, krajša in enostavnejša v primerjavi z GPL kodo. Uporabniki lažje sledijo kodi in v njej opazijo napake. DSL skripta bo vedno pregledna in lepo strukturirana, saj je uporabnik prisiljen uporabljati določeno strukturo.

Uporabniki se lahko preko DSL-ja spoznajo z domeno in programskim orodjem Slack, kar je še en pokazatelj, da je DSL uspešno opravil svojo nalogo. Učenje DSL-ja poteka dokaj hitro, saj gre za enostavnejši in manjši DSL.

Upoštevati moramo tudi čas razvoja DSL-ja, ki vsekakor ni majhen, a še vedno sprejemenljiv. Vsekakor se razvoj DSL-ja splača, če ga bomo v prihodnje uporabljali za izdelavo večjega števila Slack botov. Prav tako menimo, da je (v primeru Slack botov) razvoj notranjega DSL čisto dovolj. Razvoj zunanjega DSL-ja, ki je veliko zahtevnejši, ni potreben.

7.2 Vzdrževanje in izboljšave

DSL potrebuje vzdrževanje. V našem primeru, je potrebno vzdrževati in biti pozoren na kar nekaj stvari. Ena izmed ključnih je komunikacija s Slack API-jem. Podjetja velikokrat posodobijo svoje API-je na novo različico, kar pomeni, da starejša koda ni več kompatibilna. Posledično tudi naš DSL ne deluje več in ga moramo posodobiti. Ker gre za notranji DSL, je pomembno tudi vzdrževanje in posodabljanje kode glede na Ruby različico.

DSL za izdelavo Slack botov ima še veliko prostora za nadgradnjo in izboljšave. Nekatere smo izpustili namenoma, saj so za to diplomsko delo preobsežne. Primer take je vključitev umetne inteligence, kot so prilagojeni odzivi Slack bota glede na uporabnikov profil (uporabnikovo analizo) ali pa novih funkcionalnosti, kot so koledar, zapiski, igre in tako dalje.

Literatura

[1] About ruby. Dosegljivo: `https://www.ruby-lang.org/en/about/`. [Dostopano: 26. 12. 2019].

[2] Bot. Dosegljivo: `https://techterms.com/definition/bot/`. [Dostopano: 21. 1. 2019].

[3] Css. Dosegljivo: `https://www.w3.org/Style/CSS/`. [Dostopano: 27. 1. 2019].

[4] Domain-specific language. Dosegljivo: `https://en.wikipedia.org/wiki/Domain-specific_language`. [Dostopano: 29. 11. 2018].

[5] Martin Fowler. *Domain-Specific Languages*. Addison-Wesley Professional, 2010.

[6] Vandenburg G. *Metaprogramming Ruby*. O'Reilly, 2005.

[7] Html. Dosegljivo: `https://www.w3.org/html/`. [Dostopano: 27. 1. 2019].

[8] Latex. Dosegljivo: `https://www.latex-project.org/`. [Dostopano: 20. 1. 2019].

[9] Suresh Mahadevan. *Making use of Ruby*. Wiley, 2002.

[10] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37:316–, 12 2005.

[11] Paolo Perrotta. *Metaprogramming Ruby: program like the Ruby pros.* Pragmatic Bookshelf, 2010.

[12] Ruby instance_eval. Dosegljivo: `https://apidock.com/ruby/Object/instance_eval`. [Dostopano: 6. 1. 2019].

[13] Slack. Dosegljivo: `https://slack.com/features`. [Dostopano: 26. 12. 2019].

[14] Slack api. Dosegljivo: `https://api.slack.com/`. [Dostopano: 4. 1. 2019].

[15] Slack bots. Dosegljivo: `https://api.slack.com/bot-users`. [Dostopano: 4. 1. 2019].

[16] Slack help center. Dosegljivo: `https://get.slack.help/hc/en-us`. [Dostopano: 26. 12. 2019].

[17] Slack ruby client. Dosegljivo: `https://github.com/slack-ruby/slack-ruby-client`. [Dostopano: 4. 1. 2019].

[18] Sql. Dosegljivo: `https://www.iso.org/standard/63555.html`. [Dostopano: 27. 1. 2019].

[19] Syngress. *The Ruby Developer's Guide.* Syngress, 2002.

[20] The complete guide to domain specific languages. Dosegljivo: `https://tomassetti.me/domain-specific-languages/`. [Dostopano: 1. 1. 2019].

[21] Markus Voelter. *DSL Engineering: Designing, Implementing and Using Domain-Specific Languages.* dslbook.org, 2013.

[22] Xml. Dosegljivo: `https://www.w3.org/XML/`. [Dostopano: 27. 1. 2019].