TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N2612 – Elektrotechnika a informatika

Studijní obor: 1802T007 – Informační technologie

Automatický generátor textu

Automatic text generator

Diplomová práce

Autor: **Bc. Jan Kadlec**

Vedoucí práce: **Mgr. Jiří Vraný, Ph.D**

V Liberci 16. 8. 2012

(Originální zadání)

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/200 o právu autorském, zejména § 60 (školní dílo). Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé DP a prohlašuji, že **souhlasím** s případným užitím své diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít svou diplomovou práci či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce.

Datum:

Podpis:

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu diplomové práce, Mgr. Jiřímu Vranému, Ph.D. za jeho čas, ve kterém mi poskytl cenné rady a připomínky, kterými se zasloužil o vznik tohoto díla. Dále musím poděkovat MgA. Janě Bernartové, pro níž byla tato práce součástí vlastního rozsáhlého projektu a která také pomohla se vším, s čím bylo potřeba. Poslední dík patří týmu Readability, který poskytl přístup k jejich webové platformě umožňující parsování textových dat z webu.

Abstrakt

Cílem této diplomové práce bylo vytvořit webovou aplikaci, která bude automaticky publikovat texty v anglickém jazyce, skládající se z náhodně generovaných vět. Aplikace se měla tvářit jako webový blog, na kterém nebude na první pohled patrné, že je generován automaticky, bez zásahu člověka. K dosažení cíle bylo nutné využít především metod lingvistické analýzy, ale i dalších prostředků z oblasti informatiky. Hlavní částí této práce je analýza textů na téma *Generativní umění*. Z těchto textů a informací získaných z jejich jazykového rozboru je následně sestavována gramatika, jenž slouží pro automatické generování náhodně sestavených vět. Tato gramatika se musí neustále vyvíjet, proto webová aplikace obsahuje také poloautomatický kolektor dat, který prochází internet, sbírá texty a po kontrole je analyzuje, archivuje a zařazuje do gramatiky.

Po technické stránce bylo pro tuto aplikaci využito programovacího jazyka Python a knihovny NLTK (Natural Language Tool Kit), která slouží právě k operacím s lidským jazykem – klasifikaci, rozbor a sémantickou analýzu. Ke sloučení všech částí práce do webového rozhraní bylo využito frameworku Django.

Klíčová slova: Automatický generátor textu, Python, Django framework, Lingvistická analýza, bezkontextová gramatika

Abstract

The goal of this thesis was to create a web application, that should be able to publish automatically generated articles. Application should look like a webblog written by a human writer. To achieve this goal, it was necessary to use linguistic analysis methods and also a processing from the information technologies field. The main part of this thesis is parsing articles about the „*Generative art*“. A context-free grammar is generated from these text data and from the gathered informations. This grammar is used for automatic generation of random articles. Grammar should be updated continously, therefor a semi-automatic text crawler is also part of this job.

Python programming language and NLTK library (Natural language toolkit) were used to create the application. All the parts of this program were integrated into a web application, using the Django framework.

Keywords: Automatic text generator, Python, Django framework, Lingvistic analysis, non-context grammar

Obsah

Seznam obrázků

Seznam tabulek a grafů

1. Úvod

Podstatou této diplomové práce bylo naprogramovat automatický generátor anglického textu na téma Generativní umění, který má vytvářet „lidské“ texty skládající se z náhodně sestavených vět. Generátory textu se k různým účelům využívají již dlouhou dobu. Existují různé druhy - od těch nejjednodušších, které produkují naprosto nesmyslné, tzv. „dummy“ texty, až po ty sofistikovanější. Nejpokročilejší generátory pracují s větnou stavbou a dalšími pravidly přirozeného jazyka. Takové jsou schopny vytvářet téměř dokonalé, gramaticky správné texty. Při bližším prozkoumání lze zjistit, že většina takto vyprodukovaného textu jsou věty, které docela nedávají smysl, ovšem na první pohled se tváří, jakoby byly napsány člověkem. Což je zčásti pravda, protože vycházejí z ručně psané gramatiky. Lidský jazyk je totiž velmi rozmanitý, neustále se vyvíjí, a to nejen co se týče slovní zásoby, ale i stavbou vět. Angličtina v tomto ohledu patří mezi „jednodušší“ jazyky, narozdíl od češtiny, ale i přesto žádný z lidských jazyků není možné popsat konečnou množinou pravidel.

Cílem této práce bylo vytvoření pokročilého generátoru, který umí vyprodukovat texty na vysoké úrovni, ale na rozdíl od již existujících generátorů si svoji zásobu jazyka sám automaticky rozvíjí. Všechny generované texty se týkají pojmu Generativní umění, který je popsán hned v první kapitole této práce. Toto téma bylo zvoleno, protože výsledek diplomové práce je součástí dlouhodobého projektu MgA. Jany Bernartové, zabývající se touto problematikou.

Stejně rozmanité jako lidský jazyk jsou i přístupy k jeho analýze, proto nebylo vždy jednoduché vybrat ten nejvhodnější postup. Většinou však rozhodovaly omezení počítače při zpracování lidského jazyka a vždy byl zvolen ten nejlepší kompromis mezi mírou náhodnosti vět oproti míře chybovosti v gramatice.

K dosažení cíle bylo nutné ošetřit sběr textů z internetu a vytvořit algoritmy, které text analyzují, rozebírají a připravují jej na část poslední – zpětné sestavování fragmentů do náhodných vět, potažmo odstavců textu. Hlavní výhodou této automatizace je postupné zvyšování rozmanitosti textů, ačkoliv kvalita generovaných článků není tak vysoká jako v případě ručně psaných gramatik. S tím se ovšem od začátku počítalo.

Výsledkem této práce je webová aplikace, která se pro běžného návštěvníka tváří jako webový blog obsahující právě generované články. V pozadí stojí administrátorské rozhraní, které obsahuje archiv textů využitých pro vytvoření gramatiky a nástroje umožňující správu této aplikace. Archiv textů, spolu s odkazy na zdroje, bude časem využit pro další studijní účely.

K vytvoření logiky aplikace bylo využito programovacího jazyka Python, doplněného o volně dostupné knihovny. Tou nejzásadnější součástí byl určitě soubor knihoven a programů NLTK (Natural Language Toolkit), který slouží k analýze přirozeného jazyka. Lidský faktor je v procesu generování využit jen při vyhodnocování vhodnosti stažených textů a ke spouštění automatických procesů sběru dat a generování gramatiky / textů.

Krátký úvod do problematiky generativního umění zprostředkovává kapitola s číslem 2. Ve třetí kapitole lze najít teoretickou rešerši na téma způsobů generování textu a zpracování přirozeného jazyka. Poskytuje také detailnější pohled na knihovnu NLTK a představuje použité nástroje. Čtvrtá kapitola se věnuje návrhu aplikace. V kapitole číslo 5 je potom podrobně popsáno praktické řešení, včetně způsobu využití nalezených nástrojů.

2. Generativní umění

„Generativní umění označuje umění, které bylo zčásti nebo kompletně vytvořeno za pomoci autonomního (samostatně pracujícího) systému. Autonomní systém ve smyslu generativního umění je systém takový, který dokáže pracovat bez přispění člověka a umí nezávisle rozhodovat o součástech uměleckého díla, které by jinak vyžadovaly rozhodnutí učiněná samotným umělcem. Běžně je termín Generativní umění používán k označení počítačem generovaného uměleckého díla, které je stanoveno algoritmicky.“ [1]

Z definice, kterou lze nalézt na anglické Wikipedii lze tedy zjistit, že hlavní skupinou spadající pod termín Generativní umění jsou počítačovými algoritmy vytvořená díla, ať už grafická, zvuková, nebo textová. Ovšem generativní umění může být vytvářeno i za využití chemických, biologických, mechanických či robotických systémů. Dále může být vytvářeno tzv. chytrými materiály[2], což jsou takové látky, které mají schopnost rozpoznat vybranou změnu vnějších podmínek a výrazně na ni reagovat nějakým způsobem. Na Wikipedii lze nalézt i zmínku o manuální randomizaci, matematice, datovém mapování, symetrii nebo tzv. teselaci (vytváření dvojrozměrných pláten za využití opakování geometrického tvaru bez překryvů a mezer).

V podstatě tedy jakýkoliv systém, který nabízí možnost náhodnosti či interakce s okolím nebo jiným systémem, lze využít k tvorbě generativního umění, které tím pádem pokrývá širokou paletu vědeckých i nevědeckých oborů. Nakonec i výsledek této práce, tedy automatický generátor náhodných textů, lze považovat za jistou formu generativního umění.

3. Práce s textovými daty

Existuje několik používaných způsobů generování náhodného textu, od těch nejjednodušších, až po složitější. Bez výchozích dat lze generovat text například na základě sestavování náhodných znaků, ale ve většině případů vychází generování z předem získaných textových dat. Druh potřebných dat i způsob jejich rozboru je závislý na algoritmu použitém pro generování, proto je nejprve důležité vědět, jaké jsou možnosti vytváření náhodného textu.

3.1 Způsoby generování náhodného textu

3.1.1 Nejjednodušší metody generování textu

Nejjednodušší metodou generování náhodného textu, která se nabízí, je pouhý výběr slov nebo vět ze slovníku bez zkoumání dalších vzájemných vztahů mezi nimi. Výběr celých vět může být problematický z pohledu autorských práv, navíc v jednotkách vět už se nejedná o náhodný text. U výběru slov lze dosáhnout realističnosti dodržováním rozdělení distribuce slov podle délky, ovšem takový text nedává naprosto žádný smysl a proto není pro účel této práce zajímavý.

3.1.2 Lorem Ipsum

Nejznámějším generátorem textu je pravděpodobně „Lorem Ipsum“ generátor [3], který je hojně využíván v odvětví tiskařství či webdesignu k simulaci opravdového textu. Tento „dummy“ text není přímo náhodným textem, ale pochází z díla klasické latinské literatury, starého skoro 2000 let. Existuje mnoho variací generátorů tohoto textu – některé využívají pouze vystřižené pasáže, některé přidávají i náhodná slova. Důvodem využití tohoto náhodného textu v odvětví tiskařství či webdesignu je ten, že má více méně běžné rozdělení distribuce slov – především kombinace jejich délky. Odstavec takového textu působí na první pohled jako běžně čitelná angličtina (nebo jiný jazyk), která zároveň neodvádí pozornost od designu svým obsahem (pokud čtenář neovládá latinu).

3.1.3 Markovovy řetězce

Pokročilejší metodou je generování textu založené na tzv. Markovových řetězcích[4]. Markovův řetězec označuje stochastický proces, který má Markovovskou vlastnost. Ta říká, že v každém stavu procesu je pravděpodobnost navštívení dalších stavů nezávislá na dříve navštívených stavech[5]. Tento typ generátoru je také znám jako N-gramový generátor. Z jazykového korpusu vytvoří dvojice (trojice až n-tice) slov podle toho, jak za sebou v textu následují. „N“ v názvu udává, na jaké n-tici slov je model postaven. Pokud je na následující větu aplikován trigramový model (založený na trojicích slov, tedy na dvojici aktuálních a jednom následujícím):

```
"The quick brown fox jumps over the brown fox who is slow jumps over the brown fox who is dead."
```

Je vytvořen jazykový korpus, ve kterém se na levé straně objevují všechny po sobě jdoucí dvojice slov, a na pravé straně jednotky slov, které mohou po dané dvojici následovat.

```
{('The', 'quick'): ['brown'],
 ('brown', 'fox'): ['jumps', 'who', 'who'],
 ('fox', 'jumps'): ['over'],
 ('fox', 'who'): ['is', 'is'],
 ('is', 'slow'): ['jumps'],
 ('jumps', 'over'): ['the', 'the'],
 ('over', 'the'): ['brown', 'brown'],
 ('quick', 'brown'): ['fox'],
 ('slow', 'jumps'): ['over'],
 ('the', 'brown'): ['fox', 'fox'],
 ('who', 'is'): ['slow', 'dead.']}
```

Pokud by generování textu začalo dvojicí slov „brown fox“, následovalo by jedno ze dvou slov „jumps“ nebo „who“. Protože slovo „who“ je v možnostech uvedeno dvakrát, je větší pravděpodobnost, že bude zvoleno. Dvojice slov je uvažována jako aktuální stav, možnosti na pravé straně jako stav následující. Výběr následujícího stavu je tedy podle definice závislý pouze stavu aktuálním. Čím větší je jazykový korpus, tím více je možností a tím větší je variabilita generovaného textu.

Tento typ generátoru je schopný vyprodukovat dobře vypadající věty, a pokud jsou v textu slova ponechána i s diakritikou, vytváří dojem reálného textu. Často se ale stává, že je vygenerována věta nereálně dlouhá, nebo obsahující opakující se výrazy.

3.1.4 Bezkontextové gramatiky

Nejvíce sofistikovaný způsob generování náhodného textu je založen na bezkontextové gramatice, což je jeden z druhů formální gramatiky. Přesnou definici bezkontextové gramatiky lze najít například na webu Univerzity v Rochesteru [6]:

V lingvistice a informatice označuje pojem bezkontextová gramatika formální gramatiku, ve které mají všechna přepisovací pravidla tvar $A \rightarrow \beta$*, kde A je neterminál a* β *řetězec terminálů a/nebo neterminálů.*

V přirozeném jazyce se dá bezkontextová gramatika uvést na příkladu, kde je startovacím neterminálním symbolem celá věta - souvětí. Ta je přepsána na další neterminály, věty jednoduché (věta hlavní a vedlejší). Ty jsou dále přepsány na větné fráze (každá věta jednoduchá například na frázi jmennou a slovesnou). Fráze se skládají z dalších neterminálů – slovních druhů. Jmenná fráze například z několika přídavných jmen a jména podstatného. A konečně, slovní druhy jsou přepsány na terminály – konkrétní slova.

SCIgen

Jedním z příkladů generátoru postaveného na bezkontextové gramatice je SCIgen [7], což je generátor pseudo-vědeckých textů, které kromě textu obsahují i grafy, číselné hodnoty a citace. Autoři získali pro tento generátor publicitu, když jím vygenerovaný dokument pod názvem *Rooter: A Methodology for the Typical Unification of Access Points and Redundancy* poslali jako výsledek vědecké práce na konferenci „*World Multiconference on Systemics, Cybernetics and Informatic“* v roce 2005. Falešný vědecký dokument byl přijat a jeho autoři byli pozváni k přednášce na konferenci. Ihned poté byl tento hoax autory odhalen a jejich pozvánka na konferenci byla stornována. Ukázku dokumentu vytvořeného tímto generátorem lze najít v příloze A.

Kaant Generator

Podobným typem generátoru je Kaant Generator[8], pojmenovaný podle své hlavní gramatiky založené na generování pseudo-filozofických textů ve stylu pruského filozofa Immanuela Kaanta[9]. Tento generátor je popsán ve výukové knize Dive into Python v kapitole věnované parsování XML souborů. Pro lepší pochopení principu, na kterém funguje, obsahuje i velice jednoduchou gramatiku generující číslo v dvojkové soustavě:

```
<grammar>
<ref id="bit">
  <p>0</p>
  <p>1</p>
</ref>
<ref id="byte">
  <p>
     <xref id="bit"/><xref id="bit"/>
     <xref id="bit"/><xref id="bit"/>
     <xref id="bit"/><xref id="bit"/>
     <xref id="bit"/><xref id="bit"/>
  </p>
</ref>
</grammar>
```

Parser pracující s tímto typem gramatiky funguje na principu vnořených referencí. Elementy `<xref />` fungují jako odkaz na elementy `<ref>`, které obsahují buď další reference (tedy neterminály), nebo přímo hodnoty, ze kterých ve výsledku sestává vygenerovaný text (terminály). Takových referencí může být v gramatice libovolné množství, nesmí se ale stát, že by se při nějaké posloupnosti přepisování program zacyklil. Ukázka textu vygenerovaného tímto generátorem je v příloze B.

3.2 Sběr textových dat

K získávání dat z internetu lze využít několik volně dostupných modulů a knihoven napsaných v programovacím jazyce Python. Obecným postupem je procházení internetu dle určitého klíče, vyhledávání cenných dat a jejich dolování z webu pro další použití. Konkrétně u sběru textových dat je nutné provést následující kroky:

1) Prohledávání webových stránek

2) Parsování URL adres z webu pro další postup ve vyhledávání

3) Parsování dalších cenných textových dat a jejich zpracování

3.2.1 Nástroje procházení webu a analýzy HTML souborů

Aby bylo možné získat obsah webové stránky, je nutné nejprve načíst její obsah, který je v drtivé většině případů ve formátu HTML. Tento obsah je dále nutné zpracovat a najít v něm to potřebné – v tomto případě odkazy na další webové stránky.

Urllib2

Urllib2 je modul definující funkce a třídy, které pomáhají při otevírání URL adres (většinou přes protokol HTTP). Tento modul nabízí velmi jednoduché rozhraní – jeho nejpoužívanější součástí je metoda `urlopen`. Obsahuje i komplexnější rozhraní pro zpracování na webu běžných situací – jako základní ověřování (autentikace), soubory cookie, proxy servery atd. Urllib2 je následovníkem knihovny Urllib a zjednodušuje načtení obsahu URL adresy pomocí třídy `request`. Více o tomto modulu lze najít na webových stránkách dokumentace jazyka Python [10].

HTMLParser

Modul HTMLParser je základním analyzátorem textových dat ve formátu HTML a XHTML obsaženým v instalaci Python. Lze s ním procházet strom HTML souboru a do jisté míry je schopný zpracovat i chybné soubory.

BeatifulSoup

BeautifulSoup [11] je jeden z dostupných modulů v jazyce Python, navržený pro dolování dat z HTML a XML souborů. Existují i rychlejší a optimalizovanější parsery v jazyce Python, ovšem BeatifulSoup má velikou přednost v tom, že je velmi robustní a umí si poradit i s vadnými soubory, obsahujícími například překlepy ve značkách elementů, křížící se elementy a podobně. BeatifulSoup vrací HTML/XML strom, který se v případě vadného zdroje podobá původnímu dokumentu co nejvíce. Umožňuje tedy vydolovat hodnotná data i z velmi špatně napsané webové stránky.

LXML

Knihovna LXML[12] slouží ke zpracování dat ve formátu XML a HTML a práce s ní je v případě složitějších operací jednodušší než s předchozí knihovnou BeatifulSoup. Pro zpracování HTML souborů obsahuje stejnojmennou třídu, která musí být nejprve naimportována. Poté jsou HTML data ve formátu textového řetězce zpracována metodou `fromstring()`, která vytvoří objekt typu `lxml`.

3.2.2 Získávání textových dat z webu

Hlavním důvodem sběru URL adres je nacházení těch, ze kterých lze vytěžit textová data pro generátor gramatiky. Textových dat lze nalézt spoustu, problémem je, jak cenná data odlišit od těch nepotřebných. Webové stránky jsou plné dnes plné reklam, interaktivních prvků a dalšího „ruchu“, který je třeba odfiltrovat. Proto bylo nutné vytvořit nebo nalézt nástroj, umožňující nalezení právě těch dat, kterých je potřeba.

Arc90 Readability

Projekt Readability začal jako jednoduchý Javascriptový nástroj pro zjednodušení čtení na webu. Byl vytvořen v roce 2009 newyorskou společností Arc90[13], která se zabývá designem a tvorbou technologií. Tento nástroj byl z Javascriptu portován i pro další programovací jazyky. Volně dostupnou verzi[14] pro jazyk Python portoval v například Nirmal J. Patel[15], výzkumník v oblasti mobilních technologií.

Readability.com

Nástroj Readability postupně z jednoduché knihovny vyrostl v multi-platformní online službu[16], která je využívána širokou veřejností po celém světě. Nyní funguje jako online webová platforma, sloužící především běžným uživatelům internetu. Jejím principem je, že z běžné webové stránky vytáhne pouze tělo hlavního článku a předloží jej webovému prohlížeči „očištěné“ od všeho ostatního. Využití této služby má pravděpodobně slibnou budoucnost především ve světě mobilního internetu, kde se čtenář na malém displeji chytrého telefonu může na stránce plné reklam a dalšího ruchu lehce ztrácet. Uživatel má možnost si požadované články uložit na serveru Readability pod vlastní URL a kdykoliv si je v klidu přečíst.

V základním provedení „pro čtenáře“ funguje jako modul do internetového prohlížeče, který přidá na lištu ovládací prvky, kterými si lze článek rovnou přečíst v očištěné formě, nebo uložit na později. Readability načte požadovanou URL, vyčistí text a uloží jej na vlastním serveru pod novým odkazem. Kromě běžného využití v prohlížeči tým Readability nabízí i podporu pro vývojáře.

JSON formát

Platforma Readability vrací data ve formátu JSON[17](JavaScript Notation Object), což je JavaScriptový objektový zápis - datový formát nezávislý na počítačové platformě určený pro přenos dat. Ta mohou být organizována v polích, nebo agregována v objektech. JSON je zcela obecný formát, navíc čitelný pro člověka a může sloužit pro přenos dat v libovolném programovacím jazyce.

3.3 Způsob rozboru textových dat

Textová data lze analyzovat různými způsoby, jejichž volba závisí na dalším využití těchto dat. Rozbor textu pro využití N-gramového generátoru byl již popsán v kapitole 3.1.3 – Markovovy řetězce. Dalším možným způsobem rozboru dat je zkoumání větné struktury na základě slovních druhů a větných členů.

3.3.1 Part Of Speech Tagging

Slovním druhům se v angličtině většinou říká *parts of speech*, nebo také *word classe*s. **Part of speech tagging**[18] je potom proces, při kterém je věta rozebírána na jednotlivá slova, kterým je zároveň přiřazen tzv. tag (neboli značka), která určuje jejich slovní druh. Slovní druhy v angličtině se dělí na dvě větší skupiny. Tou první jsou Open word classes, neboli otevřené slovní druhy, jejichž zásoba se neustále rozšiřuje o nová slova, která vznikají v rámci jazyka, nebo jsou přebírána z jazyků cizích. Naproti tomu jsou closed word classes, tedy uzavřené druhy, které většinou plní spíše gramatickou funkci. Jejich slovní zásoba je pevně daná a nová slova mezi ně nepřibývají. Příkladem open classes jsou třeba podstatná jména, mezi closed classes spadají například zájmena.

Open Classes

- Noun – podstatné jméno (dog, house, love, speech, art…)
- Lexical Verb – významové sloveso (go, stand, run, love…)
- Adjective – přídavné jméno (blue, big, beautiful…)
- Adverb – příslovce (quickly, lovely, rarely…)

Closed Classes

- Determiner – determinátor (Tato kategorie v češtině neexistuje. Mezi determinátory patří v angličtině členy, přivlastňovací a ukazovací zájmena, číslovky a další slova. Ve zkratce lze říct, že se jedná o slovo, které stojí před podstatným jménem a ovlivňuje jeho význam. Příklady – a, the, his, that, another, first…)
- Auxiliary Verb – pomocné sloveso (do, can, must…)
- Preposition – předložka (on, in, since, in front of…)
- Conjunction – spojka (and, but, if…)
- Interjenction – zájmena (Tato skupina bývá někdy zařazována mezi Open Classes, protože její zásoba není daná, na rozdíl od ostatních slovních druhů ale tato kategorie je jen zřídka zaznamenána ve slovnících a pro tuto práci v podstatě bezvýznamná).
-

3.3.2 Větné členy – „Chunks“

Větné členy jsou nejmenšími jednotkami větné struktury. Jsou to řetězce přilehlých slov ve větě, která jsou spojena jednoznačně určenou závislostí. Slova se spojují ve fráze, které se stávají větnými členy – těmi se může stát každé plnovýznamové slovo, které je součástí věty a má tak svoji větnou funkci.[19] Nejjednodušším příkladem v češtině je spojení přídavného a podstatného jména – *Rozvodněná řeka*, *strakatý pes* atd. Větné členy, nebo v angličtině větné fráze, se v počítačové lingvistice někdy nazývají také jako *chunks*.

3.3.3 Vyšší větné celky

Větnými celky, které stojí nad frázemi, jsou tzv. *clauses*, což jsou v podstatě věty jednoduché, ze kterých se skládají složitější věty. Anglická definice clause je poměrně jednoduchá[20]:

„‚Clause’ je skupina slov, která obsahuje podmět a přísudek. Může být samostatnou větou nezávislou, nebo větnou konstrukcí, která je součástí jiné věty (věta závislá)“

Clause, neboli věta jednoduchá, může být tvořena jen základní skladební dvojicí podmět a přísudek - např. „Jane reads. (Jana čte.)“ Dále může obsahovat předmět – „Jane reads a book. (Jana čte knihu.)“. Podmět může být také nevyjádřený („Čte knihu“.), což se ovšem v angličtině stává zřídka.

Problematika vyšší větné stavby je již složitější než určování slovních druhů a větných frází, více o ní se lze dočíst například na webu o anglické gramatice `grammar.about.com`.

3.4 Natural Language Toolkit

Natural Language Toolkit[21] (NLTK), je soubor knihoven a programů pro analýzu přirozeného jazyka v programovacím jazyce Python. Obsahuje také jazykové korpusy, ukázková data, grafické demonstrace a velice obsáhlou dokumentaci. Součástí této dokumentace je i kniha s názvem Natural Language Processing with Python[22] (Analýza přirozeného jazyka v Pythonu), která je napsána tak, že se z ní lze učit u samotnému programovacímu jazyku Python bez předchozích zkušeností.

NLTK obsahuje silné nástroje pro rozbor přirozeného jazyka, jako například modul pro rozdělení textu na věty, určování slovních druhů atd.

3.4.1 Dělení textu na věty

K rozdělení delšího textu na věty slouží NLTK balíček s názvem `tokenize` [23], který je určen k rozdělování textu podle různých kritérií. Dělení textu na věty není triviální problém – nelze se spoléhat pouze na tečky (např. kvůli zkratkám uprostřed věty) nebo velká písmena (vlastní jména, případně věta může začínat malým písmenem). Balíček obsahuje také modul `Punkt Sentence Tokenizer`, který je určen přímo pro dělení textu na jednotlivé věty. Funguje na principu vytváření modelu pro zkratky, kololokace (ustálená slovní spojení) a slova, kterými obvykle začínají věty. Běžně musí být natrénován na rozsáhlém korpusu čistého textu v daném jazyce, ve kterém má být použit, naštěstí v knihovně NLTK je pro angličtinu k dispozici již předtrénovaný.

3.4.2 Určení slovních druhů s NLTK

Pro určení kategorie jistých slov je nutné zvážit i jejich kontext a umístění ve větě. Pokud by bylo potřeba zařadit například slovo FAST, možné odpovědi by byly čtyři: toto slovo může vystupovat jako podstatné jméno (půst), sloveso (postit se), přídavné jméno (rychlý), či příslovce (rychle).

Jak již bylo řečeno v kapitole 3.3.1, Part of speech tagging je proces identifikace jednotlivých slovních druhů v textu. Značkovač, nebo také „tagger“, je potom program, který je schopný určit slovní druhy ve větě. NLTK poskytuje nezbytné nástroje pro značkování („tagování“) slovních druhů. Těmi jsou různé třídy

značkovačů, které hledají slovní druhy v textu. Některé z nich pracují na základě regulárních výrazů - těm k fungování postačí zadání určitých podmínek. Jiné je třeba předem natrénovat. Mezi ně patří značkovače učící se na tréninkové sadě dat.

Určení slovních druhů bylo zásadní částí této práce, proto následuje popis všech značkovačů knihovny NLTK, které byly v práci použity. Popis jejich implementace a testování lze najít v praktické části práce, kapitole 5.3.1.

Default Tagger

Výchozí a nejjednodušší tagger přiřazuje stejný tag všem slovům. Tento značkovač sám o sobě nesděluje žádné cenné informace, lze ho využít především jako základ pro další značkování nebo jiné jazykové analýzy.

Regexp tagger

Regexp tagger přiřazuje slovní druhy na základě regulárních výrazů. Je vhodné ho použít na základě morfologické struktury slov a je třeba pro něj vytvořit vzory, podle kterých bude přiřazovat jednotlivé tagy.

```
patterns =
[(r'^-?[0-9]+(.[0-9]+)?$', 'CD'),   # číslovky
(r'.*able$', 'JJ'),                 # adjectives
(r'.*ness$', 'NN'),      # podstatná jména vytvořená
                              ze jmen přídavných
(r'.*ly$', 'RB'),        # příslovce
(r'.*ing$', 'VBG'),      # slovesa průběhová
(r'.*ed$', 'VBD'),       # slovesa minulého času
(r'^[A-Z].*s$', 'NNPS'), # podstatná jména vlastní
                           (množné číslo)
(r'.*s$', 'NNS'),        # podstatná jména
                           (množné číslo)
(r'^[A-Z].*$', 'NNP'),   # podstatná jména vlastní
                           (jednotné číslo)
(r'.*', 'NN')]           # podstatná jména jednotná
                           (výchozí hodnota)
```

Tagy jsou přiřazovány od začátku seznamu do konce, jako výsledek je použit první vyhovující regulární výraz, na který tagger narazí. Proto je poslední výraz v seznamu v podstatě implicitní hodnotou, která je přiřazena, pokud žádná jiná nevyhovuje. V tomto seznamu je vidět, že slova jsou vyhodnocována hlavně podle koncovky slova a velkého počátečního písmena).

Unigram tagger

Unigram tagger funguje tak, že pro každé slovo najde nejčastěji se vyskytující značku v tréninkovém jazykovém korpusu. Jak již bylo řečeno, pro přesné určení slovního druhu nestačí jen reference ve slovníku, ale i kontext slova ve větě, proto je tento typ taggeru vhodné použít pouze jako zálohu (záložní taggery budou zmíněny později v praktické části práce).

Bigram tagger

Bigram tagger pracuje stejně jako předchozí Unigram Tagger s jazykovým korpusem, s tou výjimkou, že hledá nejvíce pravděpodobný slovní druh pro dané slovo, s ohledem na slovní druh slova předcházejícího. Tento tagger je v podstatě nejjednodušší z tzv. N-Gramových taggerů. Je nazýván bigram, protože využívá dvě informace – aktuální slovo a předešlý tag. Tento tagger je již schopný zachytit jistý kontext slova ve větě. Jako příklad lze uvést dvě věty:

```
He is fast. (Je rychlý – FAST jako přídavné jméno)
He rides fast. (Jede rychle – FAST jako příslovce)
```

Pokud Bigram Tagger narazí na slovo FAST, bere v úvahu i slovní druh slova před ním. V obou případech předchází sloveso, v první variantě ovšem sloveso pomocné (z uzavřené třídy slov), v druhé variantě sloveso významové (z otevřené třídy slov). Z analýzy tréninkového korpusu tedy zjistí, že za pomocným slovesem vystupuje slovo FAST většinou (a v tomto případě nejspíš vždy) jako přídavné jméno a správně jej určí.

N-gram tagger

N-Gramový značkovač je zobecněním předchozího bigramového. N-Gram tagger hledá nejpravděpodobnější slovní druh pro slovo s ohledem na slovní druh N mínus 1 slov předcházejících. Čím vyšší je N, tím větší je potřeba tréninkový korpus,

v opačném případě se zvažované sekvence neobjevují dost často, aby z nich bylo možné vytěžit spolehlivou statistiku. V běžné jazykové analýze se většinou používá maximálně trigramový tagger, kde N je rovno 3.

Affix tagger

Tento tagger pracuje podobně jako unigram tagger, s tím rozdílem, že zvažuje pouze podřetězce fixní délky hledaného slova a hledá nejpravděpodobnější slovní druh pro tyto podřetězce. Může být použit jak pro prefixy (předpony), nebo suffixy (přípony). Affix taggeru se přiřadí parametr pro délku hledaného podřetězce a parametr minimální délky kořene slova (_min_stem_length_). Pro slova, která jsou delší než součet minimální délky kořene slova a délky podřetězce je použit záložní tagger.

Brill tagger

Brillův značkovač funguje jinak než předchozí zmíněné značkovače. Brill tagger využívá počáteční značkovač (kterým může být jeden z předchozích, nebo jejich libovolná kombinace), a přidělené tagy upravuje na základě transformačních pravidel. Tyto pravidla jsou naučena na jazykovém korpusu pomocí třídy `FastBrillTaggerTrainer` (rychlý „trenér“ Brillova značkovače). Tento trénink je založen na jednom nebo více vzorových transformačních pravidlech, která jsou zadána značkovači jako tzv. kandidátská. Následně jsou aplikována na tréninkovém korpusu a volí se ta, která vedou ke zlepšení přesnosti odhadu slovních druhů.

Trénovací fáze Brillova značkovače probíhá následovně:

- Iterativně se počítá chybové skóre každého kandidátského transformačního pravidla (rozdíl mezi počtem chyb před a po aplikování transformačního pravidla)
- Je zvoleno nejlepší pravidlo cyklu
- Pravidlo je přidáno do sady zvolených pravidel a aplikováno na text
- Toto se opakuje, dokud žádné z kandidátských pravidel nedosáhne skóre vyššího než je předem zvolená hranice
- Správně zvolená hranice je důležitá – nízká může vést k nepřiměřeně dlouhému času trénování s pouze malým vylepšením přesnosti

Tato transformační pravidla jsou nazývána kontextuální, a jsou ve formátu

`tag1 -> tag2  POKUD (`*`podmínka`*`)`

Tedy tag1 je přepsán na tag2 při splnění určité podmínky. Podmínkou může být například to, že předchozí/následující tag označuje slovní druh X, nebo předchozí/následující slovo je slovo W…).

3.4.3 Hledání slovních frází s NLTK

Obecný přístup k vytváření větných frází za pomoci NLTK spočívá v definování pravidel nebo regulárních výrazů, sestávajících pouze ze značek slovních druhů, se kterými jsou následně porovnávána vstupní data – tedy věty s označenými slovními druhy.

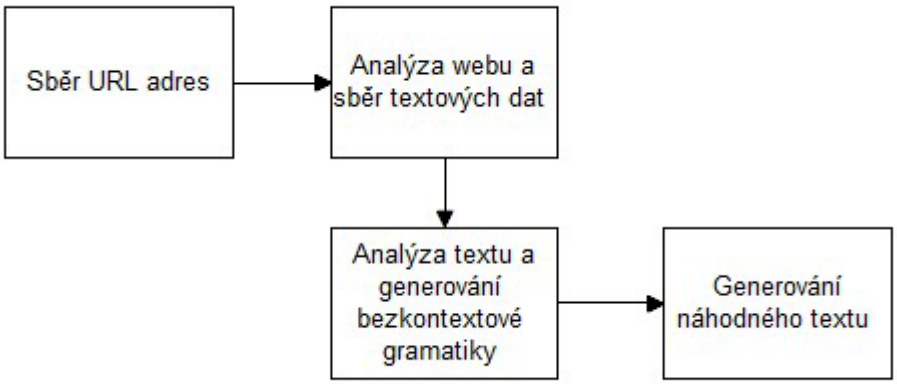
Manuální vytvoření takových pravidel nebo výrazů by byla práce velmi náročná, zdlouhavá a náchylná k chybám, protože jen těžko lze postihnout veškeré možné vazby mezi jednotlivými slovními druhy. Alternativním přístupem je trénink n-gramového chunkeru nad jazykovým korpusem, stejně jako byl natrénován i značkovač slovních druhů, tedy přístup založený na pravděpodobnosti. Přičemž v tomto případě, místo trénování značkovače na dvojicích `(slovo, tag)`, kde `tag` je značka slovního druhu, se používá trénování na sekvenci `(tag, iob)`, kde `iob` je značka větné fráze, definovaná v jazykovém korpusu Conll2000.

Protože hledání slovních frází funguje na stejném principu jako hledání slovních druhů, není potřeba se o něm v této části více rozepisovat. Jediným rozdílem je nutnost oddělit slova od tagů a pro procesu vyhledání slovních frází je znovu spojit, což je popsáno v kapitole číslo 5.3.3.

4. Návrh aplikace

Jako způsob generování byl zvolen přístup využívající bezkontextové gramatiky. K tomu bylo potřeba získávat textová data k analýze, ze kterých bylo možné gramatiku sestavit a průběžně rozšiřovat. K nalezení textových dat zase bylo zapotřebí procházet internet, tím pádem bylo nutné ošetřit i sběr URL adres navštěvovaných za účelem dolování textu. Za základní kameny aplikace byly tedy zvoleny následující 4 procesy:

1) Sběr URL adres
2) Analýza webových stránek a dolování textových dat
3) Rozbor textových dat a vytváření gramatiky
4) Generování náhodného textu z gramatiky



Obrázek 1: Návrh základních prvků aplikace

Celý systém zpracování a generování textu měl být co nejvíce autonomní. Vzhledem k tomu, že vyhodnocení vhodnosti URL odkazů (potažmo textových dat) by bylo příliš složité, musel do procesu vstoupit i lidský faktor. K tomu bylo zapotřebí opatřit aplikaci také administrátorským rozhraním, které umožňuje vykonávat nezbytné úkony:

1) Kontrolu a schvalování URL adres
2) Kontrolu a schvalování textových dat

Pokud by byla člověkem zodpovědným za kontrolu dat objevena zajímavá adresa či textová data jinou cestou, než procesem integrovaným ve webové aplikaci (náhodným či cíleným objevením při procházení internetu), měla by aplikace také umožnit tuto adresu či text zařadit manuálně do databáze.

Dále bylo nutné ošetřit úložiště dat. Pro ukládání URL adres, stažených a vygenerovaných článků bylo nejvhodnější použít klasický způsob uložení v databázi. Gramatika potom byla ukládána do XML souboru (o tom více v kapitole 5.4.1).

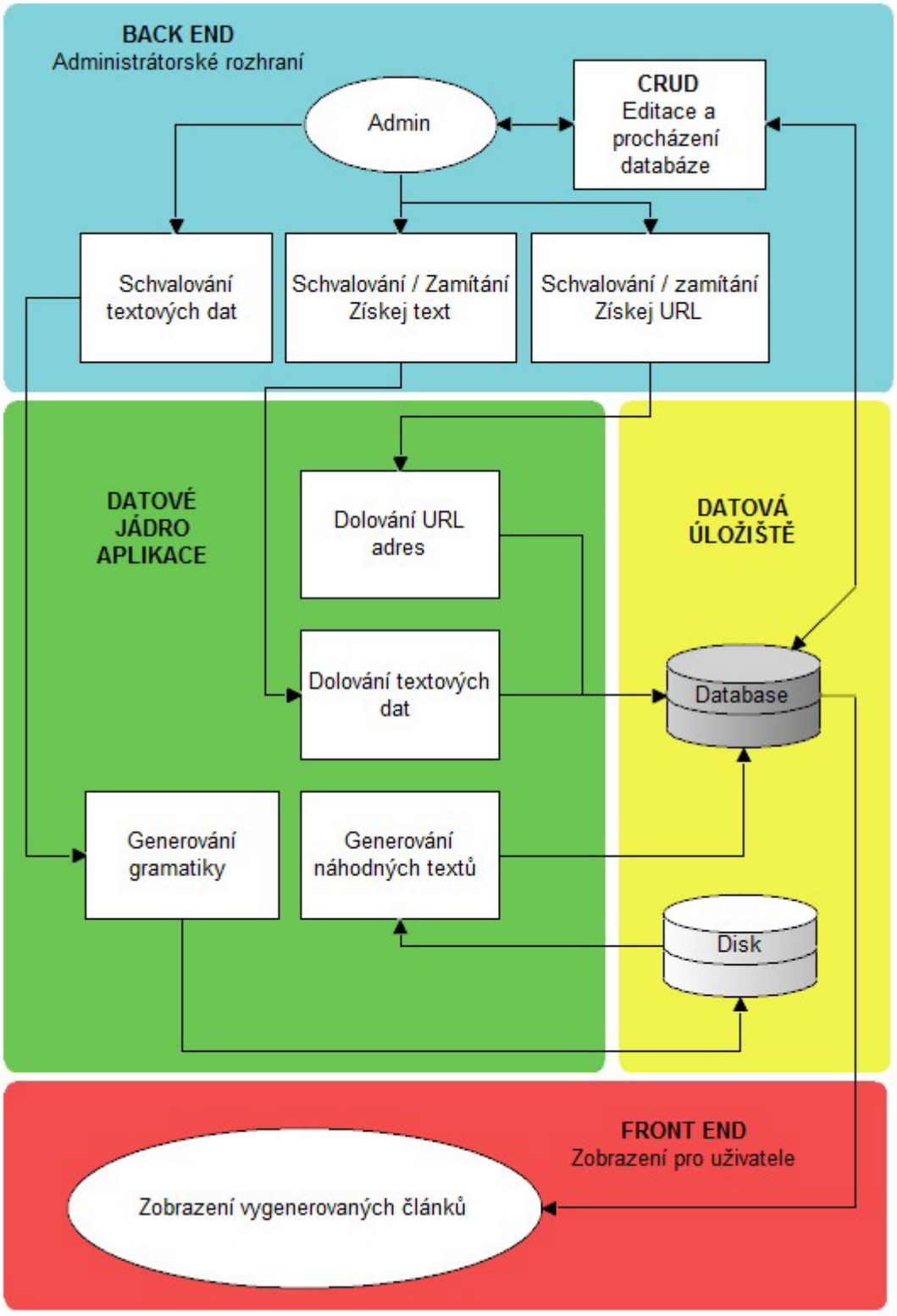
Záznamy v databázi bylo také nutné nějakým způsobem filtrovat pro zobrazení v administraci – jednak podle data, ale také podle toho, jestli už byly se záznamem prováděny nějaké operace.

Jako poslední část aplikace musela být implementována tzv. frontend část, tedy zobrazení pro veřejnost. Web se navenek měl tvářit jako jednoduchý webový blog. Frontend byl tedy navržen jako obyčejná stránka zobrazující vygenerované články. Aby byl docílen potřebný efekt, bylo zobrazení pro veřejnost opatřeno i jednoduchým grafickým designem.

Dle doporučení při zadání byla aplikace postavena na webovém frameworku Django[24], což je framework napsaný v jazyce Python, usnadňující vývoj webových aplikací. Striktně využívá tzv. MVC architektury[25] - Model – View – Controller, která rozděluje datový model aplikace, uživatelské rozhraní a řídící logiku do tří nezávislých komponent.

Webová aplikace byla navržena tak, aby se skládala ze čtyř hlavních částí:

1) **Backend** – nebo administrátorské rozhraní, ve kterém jsou prováděny úkoly vyžadující zákrok člověka. (Úpravy záznamů v databázi a spouštění procedur – dolování adres, textových dat a generování)
2) **Datová logika aplikace** – která obsahuje datové modely jednotlivých procedur, tedy algoritmy pro dolování dat, rozbor textových dat, generování gramatiky a generování náhodného textu.
3) **Datová úložiště** – kterými jsou databáze MySQL a XML soubor obsahující bezkontextovou gramatiku
4) **Frontend** – obsah, který se zobrazí běžnému návštěvníkovi. V tomto případě velmi jednoduchý – jedna stránka zobrazující generované články.



Obrázek 2: Detailní schéma aplikace

4.1 Datové modely

Jako první bylo nutné navrhnout podobu datových modelů, které jsou popisem dat a obsahují pole a chování ukládaných dat. Pro tuto práci bylo zapotřebí vytvořit tři modely:

1) URL adresy
2) Stažená textová data (články)
3) Vygenerovaná textová data

Částečně mimo se nacházejí data bezkontextové gramatiky, která jsou uložena v XML souboru. V popisu modelů je pouze operace, která provádí zápis do tohoto souboru. Důvodem toho byla možnost využití již hotového, volně dostupného Kaant generátoru, který je popsán v kapitole 3.1.4 – Bezkontextové gramatiky.

URL Adresy (Links)		
Název pole	**Datový typ**	**Popis**
URL	URLField	URL adresa.
Created	DateTimeField	Datum přidání URL adresy (především z důvodu filtrování)
State	CharField	Udává stav záznamu, zda-li se jedná o nově přidanou adresu, adresu již prohledanou, nebo adresu zamítnutou.
Crawled	CharField	Udává, zda-li byla adresa již prohledána stran textových dat.

Tabulka 1: Datová struktura záznamů URL adres

Stažené články (Posts)		
Název pole	**Datový typ**	**Popis**
Title	CharField	Titulek článku.
Body	TextField	Tělo článku.
URL	URLField	URL adresa, na které byl článek nalezen.
Created	DateTimeField	Datum a čas přidání článku do databáze.
Image	URLField	URL adresa úvodního obrázku (je-li k dispozici).
Author	CharField	Jméno autora (je-li k dispozici).
State	CharField	Stav článku – nově přidaný článek není rovnou zařazen do gramatiky, ale čeká se na jeho kontrolu administrátorem.

Tabulka 2: Datová struktura záznamů nalezených článků

Generované články (GenPosts)		
Název pole	**Datový typ**	**Popis**
GenTitle	CharField	Titulek generovaného článku.
GenBody	TextField	Tělo generovaného článku.
GenCreated	DateTimeField	Datum a čas přidání generovaného článku do databáze.

Tabulka 3: Datová struktura záznamů vygenerovaných článků

- Časové údaje (pole datového typu DateTimeField) jsou automaticky vyplňovány funkcí datetime.now() – při zápisu do databáze jsou tedy označeny aktuálním datem a časem

- Pro údaje o stavu záznamu (State, Crawled) by bylo z hlediska optimalizace vhodnější použít číselný datový typ Integer, nebo Boolean pro ty, které mají pouze dva stavy. Problém byl ale v tom, že každý údaj je vybírán ze seznamu možností, které mají hodnotu a popisek, sloužící pro snažší orientaci administrátora:

```
LINK_STATES = (
  ('N', 'New'),
  ('S', 'Searched'),
  ('R', 'Refused')
)
```

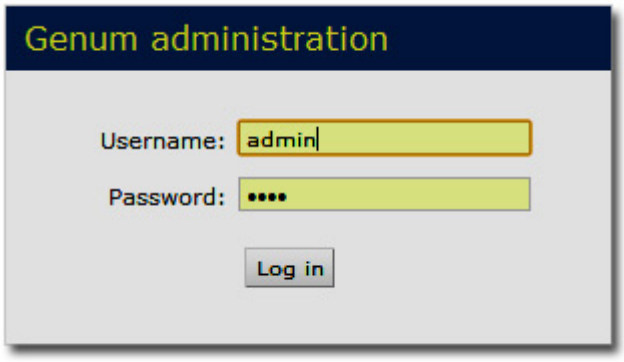
- Tento popisek je určen ke zobrazení v uživatelském rozhraní. Z neznámého důvodu se popis nezobrazuje, je-li jako hodnota možnosti použit datový typ Integer nebo Boolean. Proto byl použit datový typ Char.

- Záznamy URL adres obsahují stav označující zamítnutí – důvodem toho je, že při hledání nových URL jsou nalezené porovnávány s databází, jestli už se v ní nenacházejí. Pokud by byly zamítnuté odkazy smazány, stávalo by se, že by se při jejich opětovním nalezení znovu uložily do databáze

- Jméno autora nebo URL adresa úvodního obrázku jsou vyplněny pouze tehdy, jsou-li v parsovaném článku nalezeny (vice v kapitole věnující se platformě Readability).

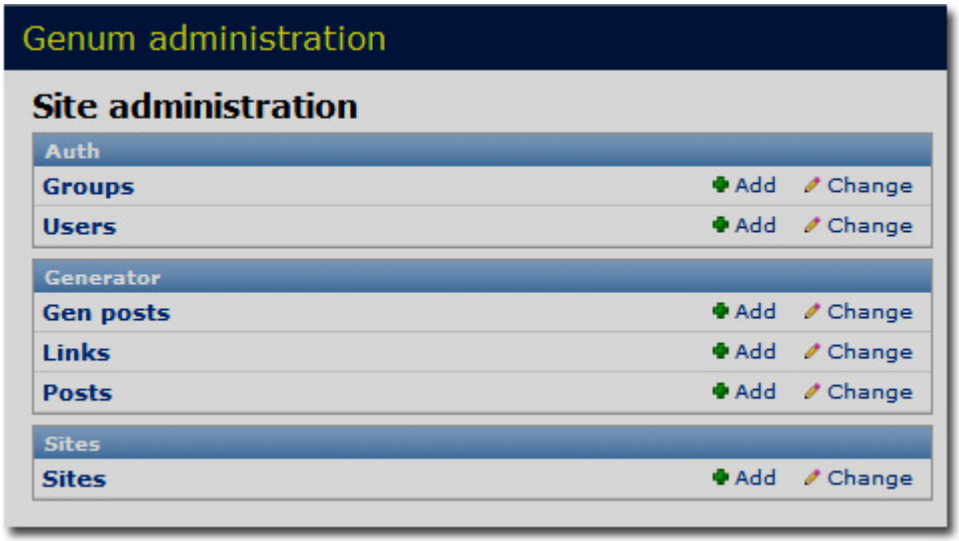
4.2 Administrátorské rozhraní

Framework Django obsahuje integrované administrační prostředí, které je generováno dynamicky podle datového modelu. Bez jakéhokoliv programování umožňuje CRUD operace (Create – Read – Update – Delete), neboli vytváření, čtení, aktualizaci a mazání záznamů v databázi. Po zprovoznění administrace a synchronizaci databáze je možné se do administrace přihlásit pod údaji tzv. superuživatele (havního administrátora), který je vytvořen při první synchronizaci databáze.



Obrázek 3: Přihlašovací stránka administrace

Administrace umožňuje kromě operací se záznamy i vytváření uživatelů dalších s různými právy a je plně upravitelná dle potřeb projektu. Po zaregistrování datových modelů vypadá hlavní stránka administrace takto:



Obrázek 4: Hlavní stránka administračního rozhraní

Menu administrace se skládá ze tří částí:

- **Auth** – administrace uživatelů a uživatelských skupin
- **Generator** – aplikace Generator a její datové modely, tedy záznamy URL adres, stažené a vygenerované články
- **Sites** – administrace stránek; pro tuto práci není důležitá

V tomto stavu zajišťuje administrátorská aplikace zobrazení dat a možnost vykonávání CRUD operací. K dalšímu přizpůsobení administračního rozhraní bylo nutné vytvořit pro každý model administrátorskou třídu, která obsahuje příslušné metody jak pro zobrazování dat, tak pro vykonávání akcí s daty.

4.2.1 Metody pro zobrazování seznamu dat

Výpis všech modelů v administraci je řazen dle data přidání, podle nějž lze záznamy také filtrovat. Řazení a filtrování je vestavěnou součástí administrace a k jeho zprovoznění stačí jeden řádek kódu pro každý model. Nezbytné metody pro zobrazení jednotlivých modelů jsou následující:

Links:

- Filtrování dle data přidání, dle stavu (new, searched, refused) a podle toho, jestli byla URL použita pro získání textu (crawled, not crawled yet)
- V seznamu zobrazení URL adresy, data přidání a stavů; kliknutí na záznam otevírá adresu URL v novém okně prohlížeče

Posts:

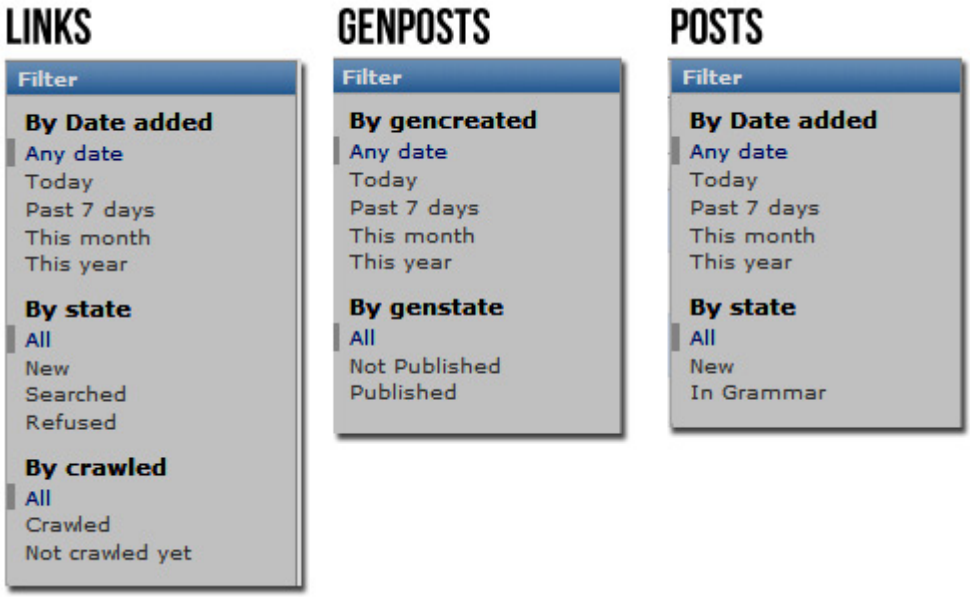
- Filtrování dle data přidání a dle stavu (new, in grammar)
- V seznamu zobrazení titulku článku, data přidání, URL adresy článku, autora a obrázku, stavu

GenPosts:

- Filtrování dle data přidání a dle stavu (new, in grammar)
- V seznamu zobrazení titulku a těla článku, data přidání a stavu (published, not published)

Filtry lze libovolně kombinovat, takže je například možné zobrazit jen URL adresy přidané v aktuálním dni, které již byly prohledány co se týče odkazů, ale nebyla z nich zatím získána textová data.

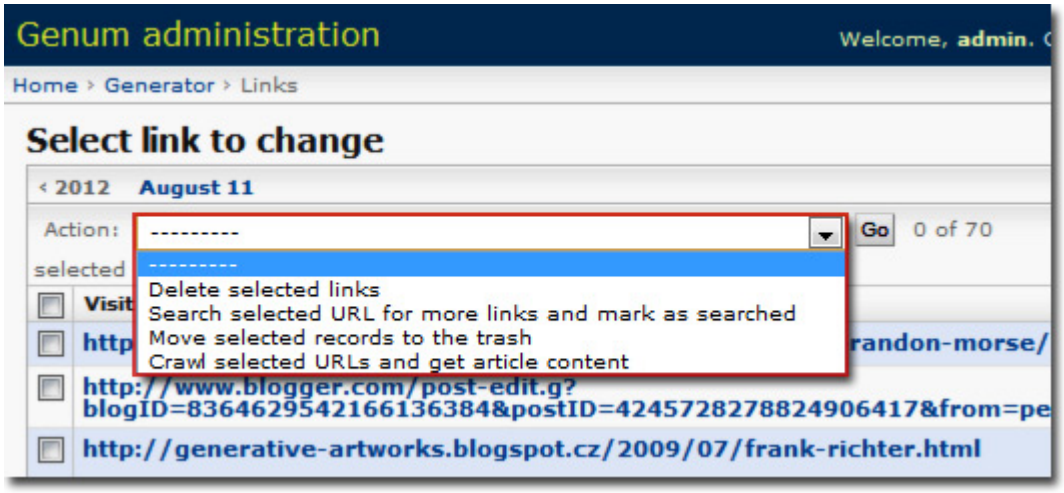
Django také umožňuje velice jednoduše přidat hierarchii dle data, která zobrazuje navigaci na vrcholu seznamu. Na nejvyšší úrovni zobrazuje roky a dále se rozkládá na měsíce a nakonec na jednotlivé dny. V menu hierarchie jsou zobrazeny vždy ty roky, měsíce a dny, kterým odpovídá minimálně jeden záznam v databázi.



Obrázek 5: ukázka implementace filtrování v administraci

4.2.2 Provádění operací - Actions

Aby bylo možné provádět operace s daty uloženými v databázi, bylo nutné vytvořit metody a použít je v administraci jako tzv. akce (Actions). Akce se provádějí přímo ze seznamu dat. Nejprve je nutné označit záznamy, se kterými bude akce provedena, vybrat akci a spustit tlačítkem „Go“.



Obrázek 6: Ukázka výběru akcí nad daty z tabulky Links

Jedinou akcí, pro kterou není třeba vybírat záznamy ze seznamu, je akce pro generování nového článku, protože generátor textu čerpá data z XML souboru, nikoliv z dat popsaných datovými modely. Každý datový model obsahuje defaultně akci pro smazání záznamu z databáze. Pro jednotlivé datové modely potom byly navrženy následující akce:

Links

1) **Získání URL adres** – akce spustí prohledávání URL adres z vybraných záznamů, nalezené uloží do databáze a označí vybrané záznamy jako již prohledané.
2) **Získání textových dat** – akce spustí analýzu URL adres z vybraných záznamů. Z každého webu získá textová data, která uloží do tabulky Posts a označí záznam jako „Crawled“.

3) **Zamítnutí záznamu** – pokud URL adresa nevyhovuje (nefunkční, nehodící se do tématu atd.), je touto akcí označena jako zamítnutá, ale je stále uložena v databázi. Důvodem toho je, aby při jejím opětovném nalezení nebyla znovu zařazena mezi nové záznamy.

Posts

1) **Zařazení článku do gramatiky** – poté, co je článek zkontrolován, je touto akcí zařazen do gramatiky. Tuto akci nelze vzít zpět (jedinou možností je kompletní vyčištění gramatiky a její opětovné naplnění).

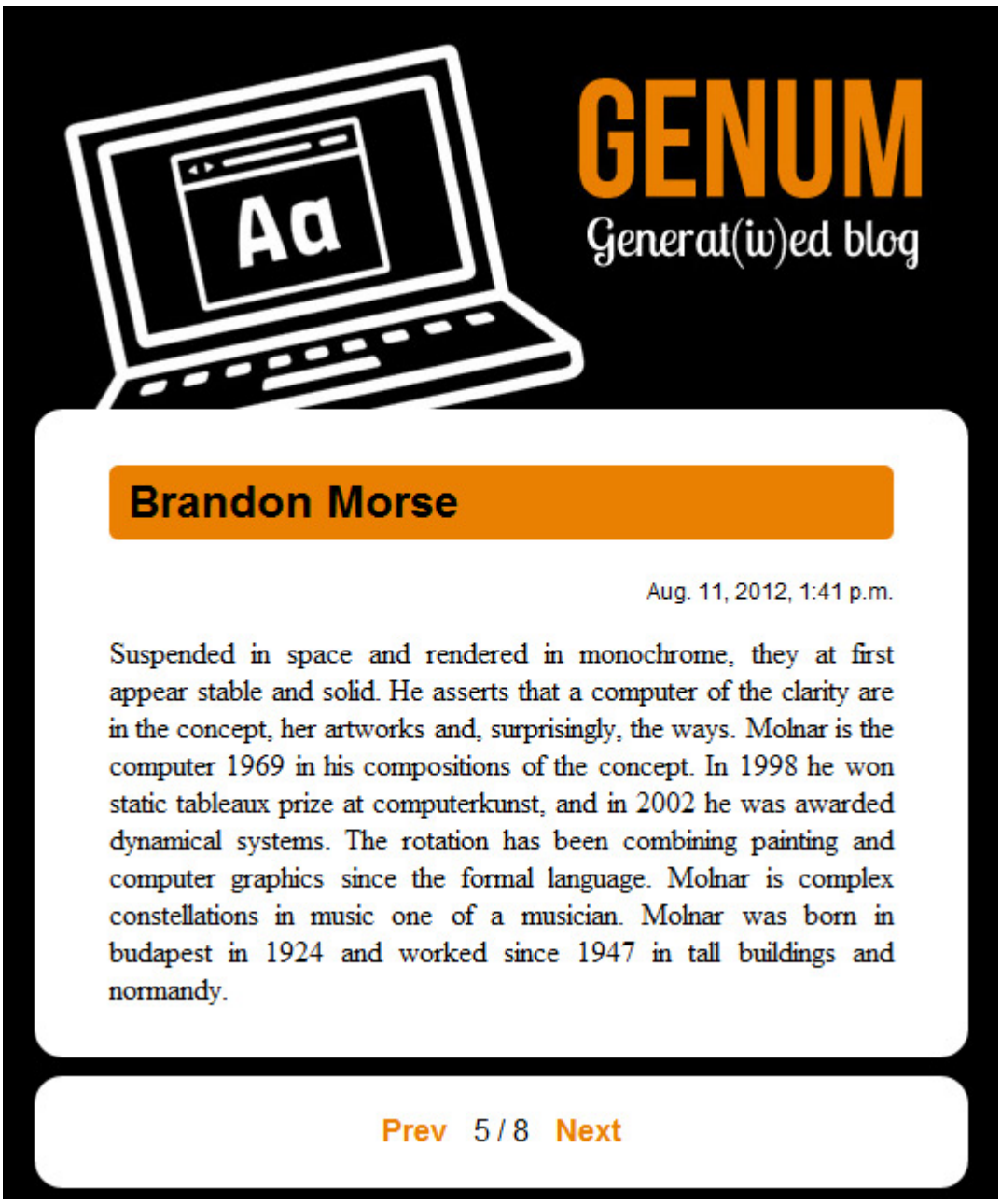
GenPosts

1) **Vygeneruj článek** – vygeneruje nový článek z gramatiky, který je zařazen do databáze a označen jako zatím nepublikovaný (Not Published).
2) **Publikuj článek** – změní stav označených záznamů z nepublikovaný na publikovaný. Na frontendu se zobrazují pouze publikované články.

Praktický popis realizace všech procesů je popsán v následující kapitole číslo 5.

4.3 FrontEnd – zobrazení pro návštěvníky

Frontend této webové aplikace byl navržen tak, že obsahuje pouze jednu stránku zobrazující vygenerované články, řazené podle data přidání. Pro lepší orientaci bylo zobrazení opatřeno stránkováním.



Obrázek 7: Ukázka frontendu webové aplikace

5. Realizace operací s textovými daty

Většina algoritmů pro práci s daty (sběr URL adres a textových dat, rozbor textových dat, generování gramatiky a textu) byla umístěna přímo do jednotlivých administrátorských tříd webové aplikace. Další, jako například knihovna NTLK, nebo program vytvořený pro trénování značkovačů, byly umístěny v samostatných souborech v adresáři jazyka Python jako zásuvné balíčky (site packages).

5.1 Sběr URL adres

Prvním krokem k rozvoji gramatiky byl sběr vhodných URL adres, ze kterých lze těžit textová data. O jejich vhodnosti rozhoduje lidský faktor v podobě administrátora aplikace. Jako startovací bod bylo do databáze vloženo několik adres, o kterých bylo předem známo, že z nich lze vytěžit vhodné odkazy na další stránky. K jejich procházení bylo využito dvou volně dostupných knihoven v jazyce Python – `urllib2` a `BeatifulSoup`, které již byly zmíněny v předchozí, teoretické části práce.

Pro načtení obsahu webové adresy je pomocí knihovny urllib2 vytvořen požadavek (request). Program poté čeká na odpověď (response). Když je odpověď získána, pomocí metody `read()` je obsah webové stránky načten do paměti počítače.

5.1.1 Zpracování obsahu webové stránky

Obsah webové stránky bylo nutné dále zpracovat jedním z parserů určených pro rozbor dat ve formátu XML/HTML. Nejjednodušší způsob získávání URL adres umožnila knihovna `BeautifulSoup`. Ta je o něco sofistikovanější než „vestavěný“ pythonovský `HTMLParser`, na druhou stranu nabízí méně funkcí než podobná knihovna `LXML`. V případě parsování URL adres byla zvolena pro jednoduchost jejího použití a srozumitelnost dokumentace.

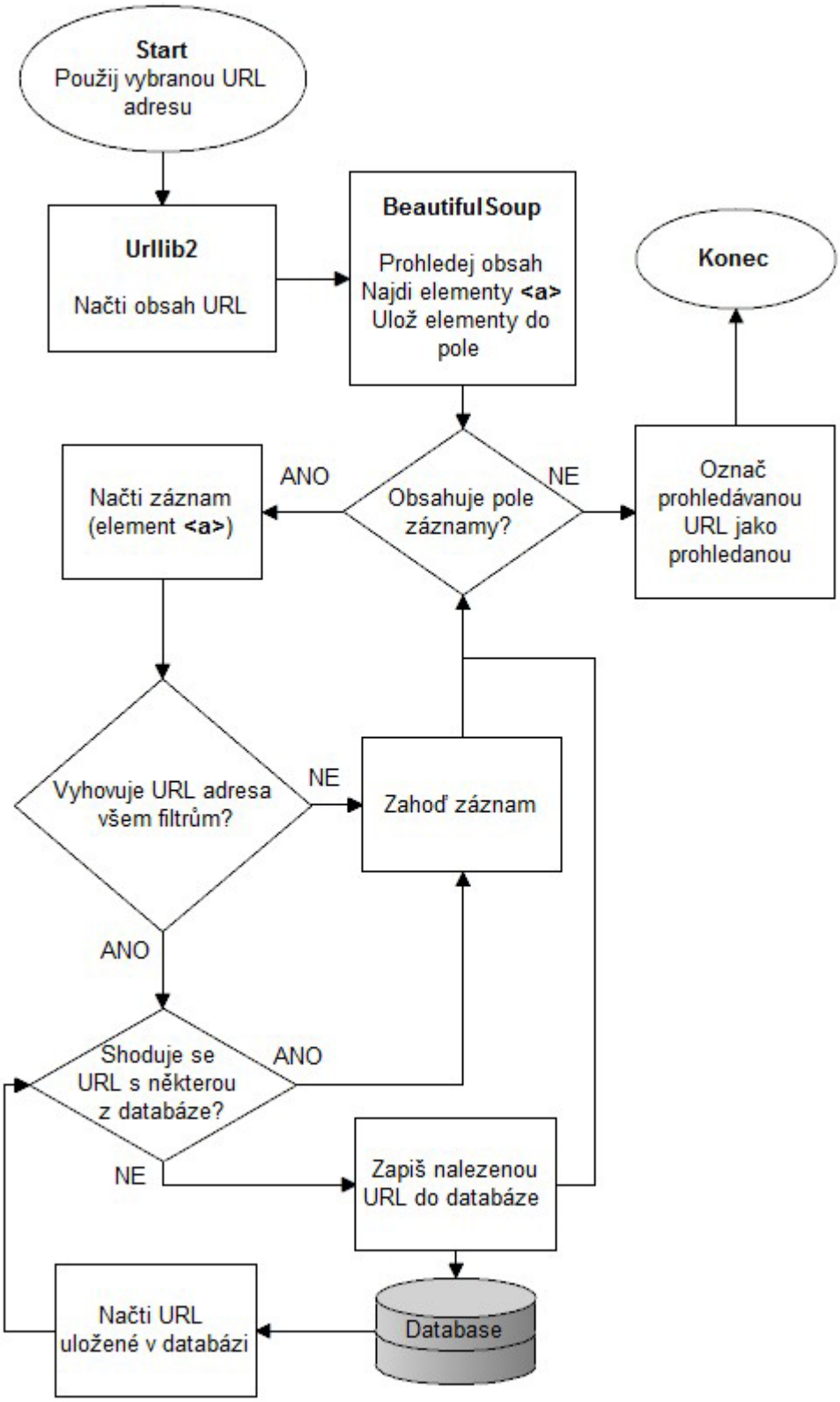
Pro získání URL adres z HTML dokumentu stačilo vybrat všechny elementy typu odkaz (`<a>`). Metoda `readAll()` vrací pole výsledků, které bylo dále procházeno po jednotlivých záznamech. Adresa URL se nachází v atributu `href="http://www.adresa.com/"`, nejdříve tedy byly odfiltrovány všechny záznamy, které nemají v klíči pole řetězec `href`. Dále bylo možné vyřadit záznamy, jejichž atribut `href` obsahoval řetězec `mailto` (e-mailové adresy) nebo znak # (většinou prázdné odkazy, spouštějící JavaScript pod jiným atributem).

Bylo zvažováno, zda odfiltrovat i všechny odkazy, které neobsahují řetězec `http://`. Často se totiž jedná o relativní vnitřní odkazy, které jsou mimo prostředí daného webu bezcenné. Na druhou stranu by takto byly odfiltrovány i některé cenné externí odkazy, kde řetězec `http://` také chybí (například `www.doména.com`, nebo jen `doména.com`). Při testování sběru URL adres se ovšem ani jednou neobjevil cenný odkaz bez řetězce http://, proto všechny odkazy bez něj byly odfiltrovány také.

Mezi adresami se také velmi často objevovaly odkazy na obrázky. Proto byly odfiltrovány všechny adresy obsahující přípony běžně používaných formátů obrázků (`jpg`, `gif`, `png` …).

Nakonec je vyhovující odkaz porovnán s databází, zda-li už nebyl jednou navštíven a uložen. Pokud ne, je zařazen do databáze a označen stavem `New`. Tato podmínka je zařazena až jako poslední, aby nebyly s databází zbytečně porovnávány i nevyhovující odkazy.

Na následujícím obrázku je algoritmus vyhledávání URL adres zobrazen pokojíc diagramu.



Obrázek 8: Diagram algoritmu těžení URL adres

5.2 Těžení textových dat

K vyhledání těla článku v HTML dokumentu bylo využito webové platformy Readability. Bylo zvažováno, zda využít původní knihovny, která by byla přímo součástí programu, nebo online služby. Z pohledu spolehlivosti by bylo lepší použít „offline“ zpracování pomocí knihovny, kdy nehrozí například výpadek služby. Ovšem knihovna již není nadále udržována, na rozdíl od webové služby Readability. Při testech vyhledání těla článku fungovala webová platforma o něco lépe než knihovna.

Potřebná komunikace s platformou probíhá na základě protokolu HTTP stejně jako v internetovém prohlížeči. Nejprve bylo nutné založit účet na webu Readability a následně zkontaktovat tým projektu Readability. Po vysvětlení účelu, ke kterému má být služba využita, byl přidělen tzv. API klíč, sloužící k autentizaci a díky kterému je možné zasílat na server požadavky. Požadavek na server je URL adresa skládající se ze třech částí:

```
http://www.readability.com/api/content/v1/parser
?token=0123456789
&url=http://parsedurl.com
```

Kde `token` je přidělený API klíč a `url` je adresa HTML dokumentu, ze kterého je potřeba parsovat textová data. Tento požadavek lze také zadat do prohlížeče a vidět výsledek. V našem případě je požadavek i odezva zpracována opět pomocí knihovny `urllib2`. Readability vrací odpověď ve formátu JSON, která vypadá takto:

```
HTTP/1.0 200 OK
{
    "domain": "blog.readability.com",
    "author": "Richard Ziade",
    "url": "http://parsedUrl.com/”
    "short_url": "http://rdd.me/kbgr5a1k",
    "title": "Step Up & Be Heard: Readability Ideas",
    "total_pages": 1,
    "word_count": 175,
     "content": "<div>\n  \n<div
                 class=\"entry\">\n\t<p>When we launched
                 Readability [snip] ...</div>\n</div>",
    "date_published": "2011-02-22 00:00:00",
    "next_page_id": null,
    "rendered_pages": 1
}
```

Odezva Readability aplikace ve formátu JSON je následně převedena do formátu Python objektu, z nějž jsou následně hodnoty čteny na základě klíče, který odpovídá původnímu klíči v přijatých JSON datech:

```
title_string        = content_obj.get('title')
url_string          = content_obj.get('url')
author_string       = content_obj.get('author')
image_url_string    = content_obj.get('lead_image_url')
article_html        = content_obj.get('content')
```

Veškeré hodnoty jsou ve formátu textového řetězce. Titulek, URL adresu, jméno autora, URL úvodního obrázku (pokud je k dispozici) a další jednoduché údaje lze bez úprav rovnou zapisovat do databáze. Tělo článku je ovšem přizpůsobeno pro zobrazení v prohlížeči, proto obsahuje pro generátor nepotřebné HTML tagy. Kromě samotného textu článku obsahuje také náhled článku, úvodní obrázek, shrnutí a další data, která musí být před uložením odstraněna, proto bylo nutné jej dále zpracovat.

Veškerý cenný text je obsažen v elementech `<p>`, k jeho získání tedy stačilo zpracovat obsah odstavců a vše ostatní vynechat. Ovšem i uvnitř odstavců se nacházely vnořené elementy, sloužící buď k formátování písma (`<b>` tučné, `<i>` kurzíva…) nebo k označení hypertextových odkazů (`<a>`). Cenná data jsou těmito elementy rozdělena, navíc se v textech nacházejí i přebytečné netisknutelné znaky (tabulátory, konce řádků…). Původně bylo zamýšleno odstranit nepotřebná data pomocí knihovny BeautifulSoup. Ta se ale pro složitější operace příliš nehodí, proto byla použita vhodnější alternativa - knihovna LXML, která umožnila vytvořit kratší a efektivnější řešení.

Její pomocí jsou vybrány všechny elementy typu odstavec, ze kterých jsou vnořené elementy odstraněny metodou `text_content()`. Data ale stále nejsou připravena k zápisu do databáze. Obsahují totiž velké množství netisknutelných znaků a také non-ascii znaky, které je potřeba odstranit. Netisknutelné znaky jsou odstraněny pomocí metody `strip()`, která maže nadbytečné mezery a konce řádků a ponechává pouze jeden `space` charakter, tedy obyčejnou mezeru.

Problém s non-ascii znaky se objevil u několika článků, kdy při dekódování textového řetězce do formátu UTF-8 vrátil program chybu. Pro jejich odstranění byla použita jednoduchá jednořádková funkce, která projde odstavce znak po znaku a zpět vrací pouze ty znaky, které odpovídají ascii kódování.

```
p = "".join(i for i in p if ord(i)<128)
```

Funkce `ord()` vrací kódové číslo znaku. Všechny znaky s číslem větším než 127 jsou tedy vynechány. Toto opatření bylo vhodné i kvůli dalšímu zpracování textových dat, kdy by se kvůli těmto znakům mohly objevit další chyby. Nakonec jsou odstavce ze seznamu spojeny do jednoho textového řetězce, který je jako tělo článku uložen do databáze spolu s ostatními údaji. URL adresa, ze které byla data vytěžena, je označena jako již vytěžená.

I přes veškeré zpracování se do textu někdy dostanou neúplné, heslovité věty, které se většinou objevují na konci nebo na začátku článku. Názorným příkladem jsou třeba tyto údaje o konání výstavy:

```
Marius Watz: Automatic Writing

July 22. - August 17, Superfrog Gallery at New People, 1746 Post Street (Webster/Buchannan).

Exhibition, July 22. - August 17. Opening reception July 22, 6:00-9:00 pm.
```

Vzhledem k tomu, že jsou označeny stejně jako zbytek textu, není možné je automaticky odstranit. Proto jsou všechny články nejprve zařazeny do databáze, kde mohou být před zpracováním do gramatiky zkontrolovány a případně upraveny administrátorem.

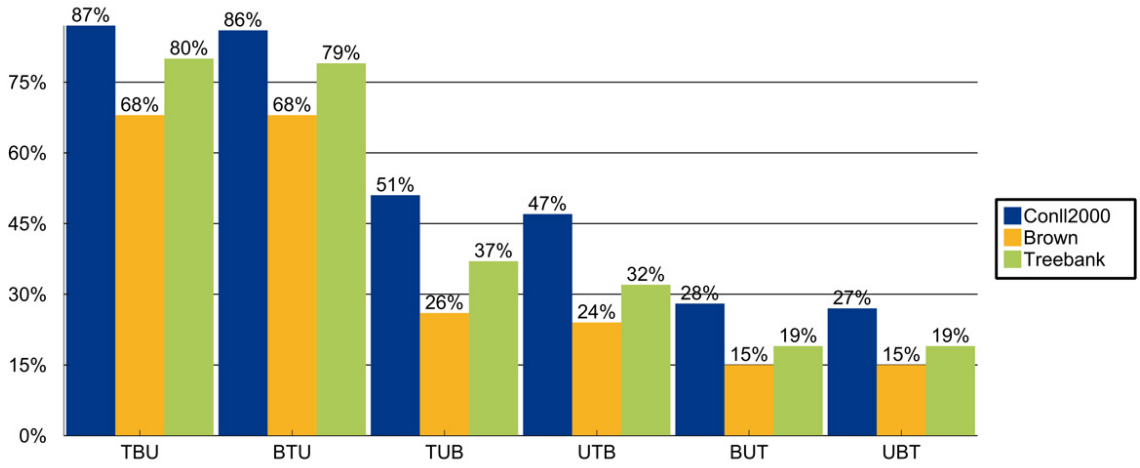
5.3 Rozbor textových dat

Textová data byla přípravována pro bezkontextovou gramatiku, proto byla analyzována na základě slovních druhů a dále větných frází. K označení slovních druhů byla využita knihovna NLTK a její značkovače (taggery), které byly popsány v kapitole 3.4.2.

5.3.1 Trénování značkovače

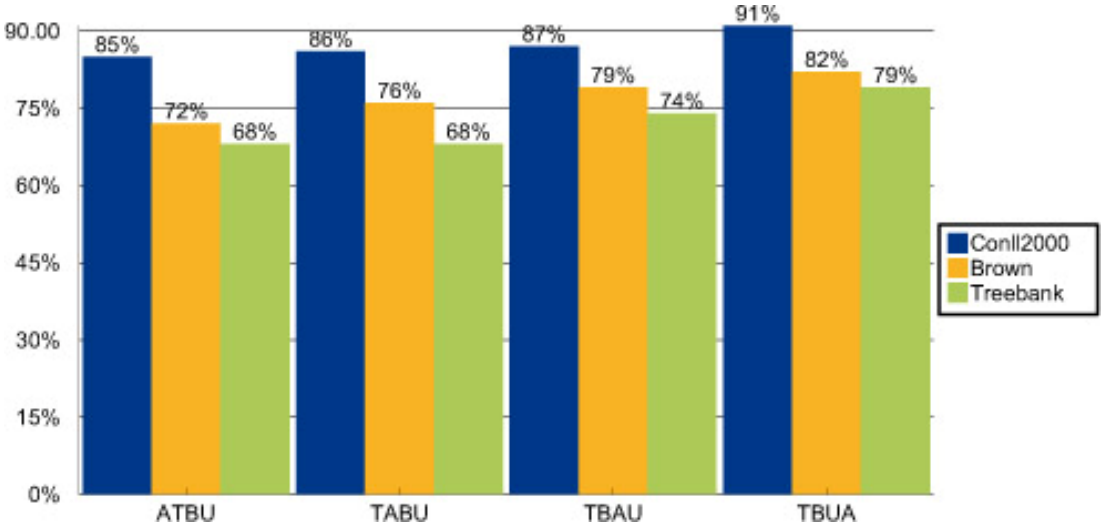
Na začátku práce nebylo jasné, jaký značkovač je nejvhodnější použít. Vzhledem k tomu, že bylo možné spojit několik značkovačů dohromady, byly otestovány různé kombinace, které měly vést ke zvýšení přesnosti. Dále také záleželo na zvoleném jazykovém korpusu použitém pro trénink. Každý ze tří dostupných korpusů (Treebank, Conll2000, Brown) tedy byl rozdělen na dvě části – trénovací a testovací.

Jako první byly otestovány různé kombinace tří N-Gramových značkovačů – Unigram, Bigram a Trigram. Tyto značkovače jsou odvozeny od třídy `SequentialBackoffTagger` (sekvenční záložní značkovač), což umožňuje využít je v řadě pro lepší přesnost značkování. Pro jednodušší práci při jejich tréninku byla vytvořena funkce pro záložní značkovač, která zajišťuje, že pokud není prvním z nich nalezen vhodný slovní druh, je použit následující značkovač v sekvenci. Poté byly otestovány všechny kombinace těchto tří značkovačů. Na následujícím grafu je vidět přesnost značkování jednotlivých kombinací těchto značkovačů, kde každý je označen jedním písmenem (U = unigram, B = bigram, T = trigram), a zároveň je rozlišena i přesnost nad jednotlivými jazykovými korpusy.



Graf 1: Graf testu přesnosti značkování n-gramových taggerů

V testu značkovala s největší přesností kombinace značkovačů v pořadí Trigram – Bigram – Unigram nad jazykovým korpusem Conll2000. Přesnost začala výrazně klesat ve chvíli, kdy se Unigram tagger posunul na druhé, respektive první místo v řetězci, což má logické opodstatnění. Pokud hledáme nejlépe vyhovující slovní druh pro slovo s ohledem na předcházející dvojici (Trigram), hrozí nejmenší možnost chybného určení. Na druhou stranu pravděpodobnost, že najdeme odpovídající trojici slovních druhů v řadě je menší, než u dvojice (nebo jediného slovního druhu). Jako počáteční značkovač byl tedy zvolen sekvenční záložní značkovač Trigram – Bigram – Unigram (dále jen TBU). K dalšímu zvýšení přesnosti byl do řetězce zařazen další značkovač - Affix taggeru.



Graf 2: Graf testu přesnosti značkování při zařazení Affix taggeru

Nejlépe dopadla kombinace TBUA, tedy Affix tagger jako poslední v řadě, kdy došlo ke zpřesnění o 4%.

Předposledním značkovačem je Regexp tagger, který označuje slovní druhy na základě regulérních výrazů. Tato metoda je pro angličtinu dobře použitelná, protože například průběhová slovesa mají vždy koncovku `ing`. Protože se TBUA tagger již osvědčil, byl tento tagger otestován jen ve variantě před ním a po něm. Výsledek dopadl nejlépe pro variantu, kdy výchozím je TBUA značkovač se záložním Regexp značkovačem. Přesnost se sice zvedla jen o jedno procento, nicméně Regexp tagger není potřeba trénovat a čas rozpoznávání se jeho použitím také znatelně neprodlouží, proto byl taktéž použit v řetězci.

Posledním značkovačem je již zmiňovaný Brill tagger. Zařazení tohoto značkovače na konec řetězce zvedlo přesnost značkování na konečných **96%**.

Pro označování slovních druhů byla tedy použita sekvence šesti testovaných taggerů v následujícím pořadí:

`Trigram –> Bigram –> Unigram -> Affix –> Regexp –> Brill`

Nutno říci, že konečná přesnost značkování 96% je pravdivá jen v ideálním případě. Tedy v tom, kdy byla sekvence natrénována a testována na stejném jazykovém korpusu (i když samozřejmě na jiných jeho částech). Vzhledem k tomu, že nebyl k dispozici korpus odpovídající danému tématu Generativní umění, na kterém by bylo možné tagger natrénovat, či alespoň otestovat a vytěžit spolehlivou statistiku, bylo rozhodnuto spolehnout se na výsledky provedených testů a k tréninku využít korpus Conll2000. Při několika následujících zkouškách tagování na malé sadě dat a vlastnoruční kontrole byly výsledky značkovače trénovaného na tomto korpusu nejméně chybové. Především sporná slova (která by bez kontextu bylo možné označit více slovními druhy) byla podle tohoto korpusu označena správně ve více případech, než tomu bylo u Brownova nebo Treebank korpusu.

5.3.2 Značkování slov

Značkovač přijímá data po jedné větě, jejíž slova (včetně interpunkčních znamének) musí být rozdělena a uložena v seznamu. K rozdělení textu na věty byl použit NLTK balíček `tokenize` (popsaný v kapitole 3.4.1). Příkladem použití může být rozdělení následujícího odstavce textu, kde je vidět jak si tokenizer poradí se zkratkami nebo malým písmenem na začátku věty:

```
text = '''
     Punkt knows that the periods in Mr. Smith and Johann
     S. Bach do not mark sentence boundaries.  And
     sometimes sentences can start with non-capitalized
     words. i is a good variable name.
     '''
```

```
tokenizer =
nltk.data.load('tokenizers/punkt/english.pickle')
print '\n-----\n'.join(tokenizer.tokenize(text.strip()))

#Výstupem jsou rozdělené věty

Punkt knows that the periods in Mr. Smith and Johann S.
Bach do not mark sentence boundaries.
-----
And sometimes sentences can start with non-capitalized
words.
-----
i is a good variable name
```

Tokenizer funguje velice dobře, ovšem s jednou věcí si neumí poradit – pokud za interpunkčním znaménkem nenásleduje mezera, nevyhodnotí toto místo nikdy jako konec věty. Proto byla stažená textová data před zápisem ještě upravena algoritmem, který chybějící mezery za znaménky doplní.

K rozdělení vět na slova posloužila metoda `word_tokenize()`, jejímž vstupním parametrem jsou jednotlivé věty. Rozdělená slova v seznamu jsou předána značkovači. Vstup ve formě věty a výsledek procesu značkování vypadá následovně:

```
data = 'James Faure Walker has been combining painting
and digital approaches since the 1980s.'

print tbuarb(data)

[('James', 'NNP'), ('Faure', 'NN'), ('Walker', 'NNP'),
('has', 'VBZ'), ('been', 'VBN'), ('combining', 'VBG'),
('painting', 'NN'), ('and', 'CC'), ('digital', 'NNP'),
('approaches', 'NNS'), ('since', 'IN'), ('the', 'DT'),
('1980s', 'CD'), ('.', '.')]
```

Samotná slova označená slovními druhy ještě k sestavení gramatiky nebylo dost dobře možné využít. Nabízela se možnost použít vzorce vět sestavené z neterminálů – slovních druhů, za které by se náhodně dosazovaly terminály – slova odpovídající těmto slovním druhům. Vzorce by generátor tvořil z rozebraných vět a slova ukládal do slovníku. Tento systém ovšem produkoval věty nesmyslné na první pohled – kombinování sloves různých časů, špatné umístění spojek a předložek atd. Míra náhodnosti byla příliš vysoká a takový systém by vyžadoval detailnější rozdělení slov než jen slovní druhy obecně. Bohužel, korpusy neobsahují dostatečné množství údajů pro jednotlivá slova a nebylo tedy možné natrénovat značkovač

s rozšířenými možnostmi značkování. Nabízel se ovšem další nástroj knihovny NLTK – sestavení větných frází.

5.3.3 Sestavení větných frází

Možnost sestavení větných frází je popsána v kapitole 3.4.3. Byl zvolen způsob, kdy je tzv. chunker natrénován pomocí korpusu. K účelu trénování chunkeru slouží balíček knihovny NLTK s názvem `chunk`. Nejprve byla vytvořena funkce, jejímž vstupem je seznam vět označených `iob` značkami a která vrací seznam dvojic `(tag, iob)`.

Dále byly natrénovány n-gramové chunkery (kde n bylo opět od jedné do tří), sestavené do sekvence záložních značkovačů, stejně jako tomu bylo u parsování slovních druhů. Korpus Conll2000 byl rozdělen na trénovací a testovací část a byla otestována přesnost jednotlivých kombinací značkovačů. V tomto případě dopadlo nejlépe pořadí značkovačů Bigram – Trigram – Unigram s přesností 90%. Stejně jako u značkovače slovních druhů, i zde záleží na podobnosti trénovacího korpusu a značkovaného textu, proto se dalo očekávat, že přesnost při použití na "ostrými" daty bude o něco nižší.

Takto natrénovaný chunker je vlastně značkovač, který přiřazuje `iob` tagy k tagům označujícím slovní druhy. Aby bylo možné ho využít na konkrétní textová data s označenými slovními druhy, musela být vytvořena funkce, která:

1) oddělí slova od jejich značek
2) přiřadí `iob` tagy k jednotlivým značkám
3) zpětně spojí slova se značkami slovních druhů a větných frází
4) z dat ve formátu `conll chunk lines` vytvoří derivační strom

Data ve formátu `conll chunk lines` jsou vlastně polem polí textových řetězců, jehož položky mají následující formát:

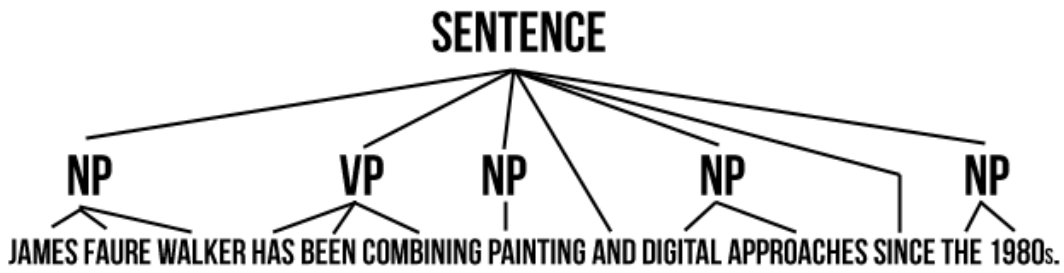
`(slovo, (značka slovního druhu, značka větné fráze))`

K vytvoření stromu slouží metoda `conllstr2tree`, jejíž výstup je následující:

```
(S
  (NP James/NNP Faure/NN Walker/NNP)
  (VP has/VBZ been/VBN combining/VBG)
  (NP painting/NN)
  and/CC
  (NP digital/NNP approaches/NNS)
  since/IN
  (NP the/DT 1980s/CD)
  ./.)
```

```
# S  – Značka pro větu
# NP – Noun Phrase – jmenná fráze
# VP – Verbal Phrase – slovesná fráze
```

Metodou `draw()` knihovny NLTK lze v Pythonu take graficky vykreslit tento strom, ve kterém jsou pro zjednodušení vynechány tagy označující slovní druhy:



Obrázek 9: Strom větné struktury

Větné členy v této větě nebyly určeny správně - ve skutečnosti se totiž věta skládá jen z jedné slovesné a dvou jmenných frází. První jmenná a slovesná fráze je v pořádku, ovšem celý zbytek věty měl být správně označen jako jedna dlouhá jmenná fráze. Chyba tohoto druhu, kdy je dlouhá jmenná (nebo i slovesná) fráze rozdělena na dvě či více kratších, se objevovala poměrně často. Nicméně, jak ukázal další postup práce, nebyl tento druh chyby v určování frází příliš škodlivý. Gramatika určená pro generování textů ve stylu filozofa Kaanta je poměrně komplexní. Z množiny terminálů obsahuje jednotlivá slova označená slovním druhem, jako např. spojky (and, but, yet). Hlavně se ale skládá z výrazů definovaných obsahovým významem, jako vyjádření kvantity (all of, some of, none of), nebo přídavných jmen různých typů, které musí být zařazeny na správné místo.

Dále obsahuje větné fráze a vyšší větné celky – tzv. clauses, což jsou v podstatě věty jednoduché, ze kterých se skládají souvětí.

K detekci těchto větších celků je již nutné zkoumat strukturu věty do hloubky a vytvářet komplexní stromy, jejichž složitost závisí na délce a struktuře věty. Dosud získané stromy nalezených větných frází jsou poměrně ploché – obsahují maximálně dvě úrovně. Chyba odhalená dříve tedy nebyla chybou natrénovaného chunkeru – složitější jmenná fráze se totiž skládá ze dvou kratších jmenných frází a pro jejich spojení by bylo nutné dále rekurzivně spouštět určitý algoritmus na odhalení složitějších struktur. Manuál NLTK[22] k tomuto nabízí jen náčrtek řešení na bázi regulárních výrazů, kterým by teoreticky mělo být možné získávat složitější struktury. Bohužel se ukázalo, že zapojení regulárních výrazů do lidského jazyka je téměř nepoužitelné, alespoň co se týče zkoumání struktury větných členů. Co platilo v jedné větě, neplatilo v jiné a ani po dlouhém zkoumání pravidel pro větné členy v angličtině a velkém počtu pokusů se nepodařilo nalézt sadu výrazů, která by fungovala obecně s přijatelnou mírou úspěšnosti. Řečeno lidově, regulární výrazy vždy nadělaly více škody než užitku, a proto bylo od tohoto zpracování upuštěno.

Bylo tedy potřeba přijít na to, jak z dostupných větných frází a zbylých členů, které nebyly zařazeny do žádné fráze, sestavit použitelnou gramatiku. Cílem nebylo generovat gramaticky naprosto správné věty, ale věty takové, které by co nejvíce připomínaly člověkem napsaný text.

5.4 Generování gramatiky a textu

Základem bylo nalezení kompromisu mezi mírou náhodnosti a výsledným vzhledem generovaných vět – do jaké míry jsou gramaticky správné a jak působí na čtenáře. Výsledkem analýzy textových dat byly věty s označenými slovními druhy a frázemi – jmennými a slovesnými. V úvahu připadalo několik možností.

Tou první bylo parsovat z textu samostatná slova a oba druhy frází, které by byly dosazovány do větných vzorců. Vzorce vět by obsahovaly pouze neterminály (tedy značky větných frází a značky slovních druhů), do kterých by bylo dosazováno. Vzorce vět by mohly být buďto pevně určeny, nebo rozvíjeny z analyzovaných dat. V prvním případě by ovšem variabilita textu stagnovala i přesto, že by slova byla volena náhodně. V obou uvažovaných případech by byla náhodnost dosazování slov příliš vysoká na to, aby z dostupných údajů vznikly věty požadované kvality.

Druhou uvažovanou možností bylo parsovat z textu jen jmenné a větné fráze. Slova nezařazená do frází ukládat jako celé věty, ve kterých by fráze byly nahrazeny referencemi. Za ty by potom při následném generování byly dosazovány samotné větné fráze. Tato možnost byla otestována a výsledek byl uspokojivý, ovšem špatně působilo kombinování slovesných frází v různých časech.

Třetí možností bylo zjednodušení možnosti druhé – z textu separovat pouze jmenné fráze a zbytek ukládat jako celou větu s referencemi na ony jmenné fráze. Tato cesta se ukázala jako nejvhodnější co se týče produkovaného výstupu. Náhodnost vět byla sice menší, ale výsledek vypadal nejlépe. Z většiny vět byly vyparsovány dvě až tři jmenné fráze, přičemž struktura a logika věty byly zachovány. Zaměněním pouze jmenných frází potom vznikaly texty, které vypadaly opravdu tak, jako by byly napsány člověkem. Některé věty dávaly smysl, některé nutily při čtení přemýšlet co vlastně znamenají, některé vyvolávaly úsměv.

5.4.1 Formát gramatiky

Gramatika byla generována ve stejném formátu jako gramatika Kaant, aby pro generování textu mohl být využit volně dostupný algoritmus z knihy Dive into Python. Vzhledem k tomu, že z textu byly parsovány pouze jmenné fráze, má gramatika jednoduchou strukturu:

```
<grammar>
  <ref id="nounphrase">  #Jmenné fráze
      ...
      ...
      <p>James Faure Walker</p>
      <p>digital approaches</p>
      <p>digital material</p>
      <p>the contrast</p>
      <p>the two</p>
      <p>a musician</p>
      <p>the computer</p>
      ...
      ...
  </ref>
  <ref id="sentence">    #Věty
      ...
      ...
      <p>
          <xref id="nounphrase"/> has been combining
          painting and <xref id="nounphrase"/> since the
          1980s.
     </p>
      <p>
          He works with physical and
          <xref id="nounphrase"/>, playing on
          <xref id="nounphrase"/> between
          <xref id="nounphrase"/>.
     </p>
      ...
      ...
  </ref>
  <ref id="paragraph">   #Odstavce
    <p>
        <xref id="sentence"/><xref id="sentence"/>
        <xref id="sentence"/><xref id="sentence"/>
        <p chance="50">
            <xref id="sentence"/><xref id="sentence"/>
            <xref id="sentence"/><xref id="sentence"/>
        </p>
    </p>
  </ref>
</grammar>
```

Před zápisem do gramatiky je ale nutné analyzovat strom, který vznikl rozborem věty a uvést data do správného formátu.

Obrázek 10: Strom větné struktury

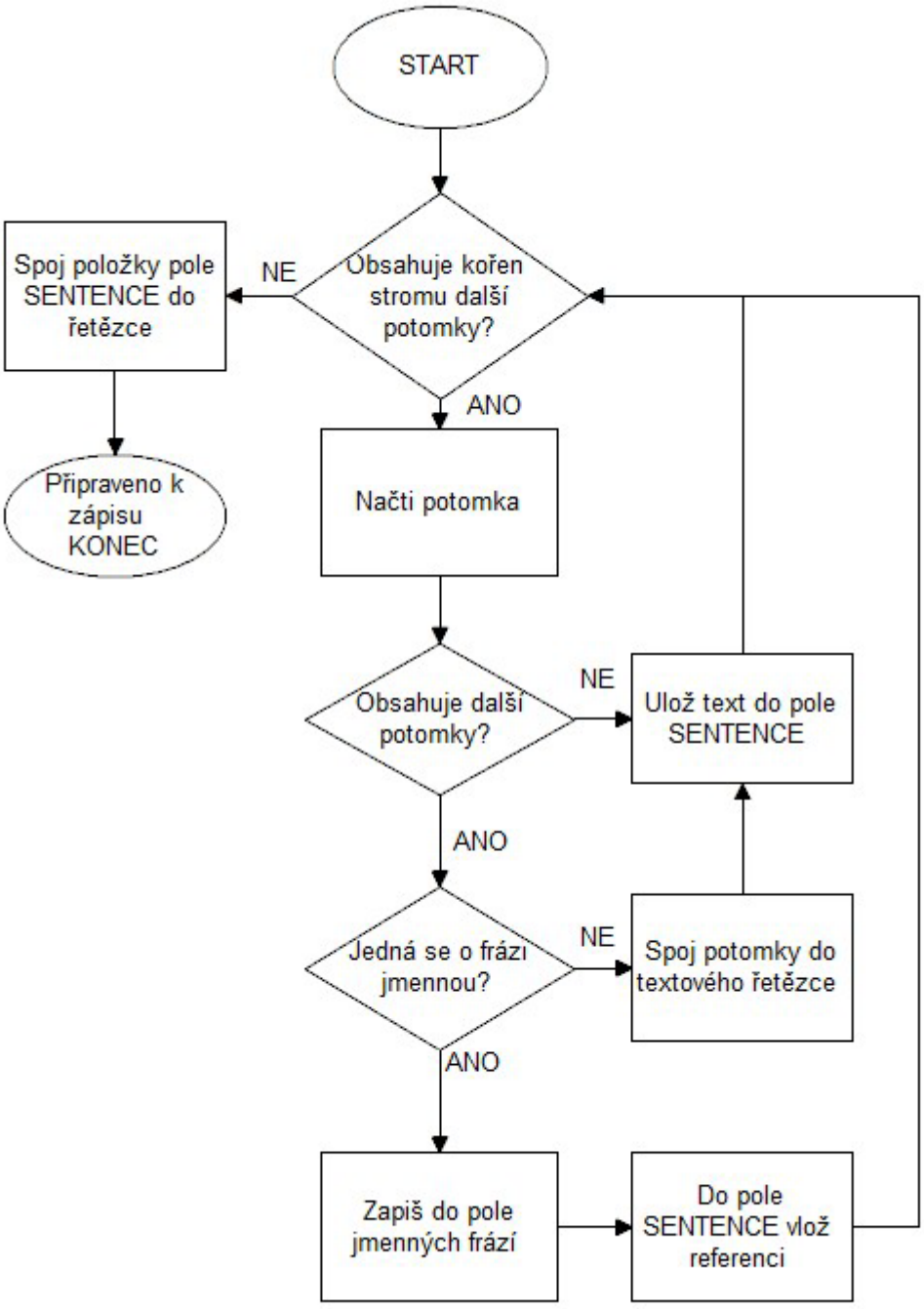
Před začátkem procházení stromu jsou vytvořena dvě prázdná pole:

- `sentence[]` – připravené pro seznam jednotlivých slov ve větě
- `nounphrases[]` – připravené pro seznam nalezených jmenných frází

Strom je procházen od kořene po jednotlivých hranách. Postupně je navštěvován každý potomek kořene, u kterého je vždy zjišťováno, zda-li má další potomky. Pokud potomky nemá, jedná se o vrchol obsahující samostatné slovo – to je odděleno od svého tagu metodou `zip()` a zapsáno do pole `sentence[]`. Tam jsou zapisována všechna data, vyjma jmenných frází, ve stejném pořadí, jako mají ve větě.

Pokud má navštívený vrchol další potomky, je zjištěno, zda-li se jedná o jmennou frázi. Pokud ano, je tento vrchol uložen do samostatné proměnné jako podstrom – subtree, který je následně zploštěn metodou `flatten()`. Tím jsou získána slova spolu se značkami slovních druhů, která jsou opět oddělena. Nyní jsou ale zapsána jako celek do seznamu `nounphrases[]` a na jejich místo v seznamu `sentence[]` je umístěna reference. Pokud se jedná o slovesnou frázi, je začátek postupu stejný, tedy podstrom zploštěn, slova oddělena od tagů, ale zapsána do seznamu `sentence[]` jako jednotlivá slova.

Řetězce z pole `sentence[]` jsou převedeny na jeden souvislý textový řetězec, čímž jsou připraveny k zápisu do gramatiky, stejně tak jednotlivé položky pole `nounphrases[]`.



Obrázek 11: Diagram algoritmu zploštění stromu

5.4.2 Zápis gramatiky do souboru

Při každém novém zápisu do gramatiky je nutné provést následující operace:

1) Načíst celou stávající gramatiku do paměti a parsovat její datovou strukturu
2) Do gramatiky zapsané v paměti přidat nové elementy
3) Obnovenou gramatiku opět zapsat celou do XML souboru

Gramatika je ze souboru načtena pomocí parseru `xml.dom.minidom`, který je součástí základní instalace Python. Je vytvořen objekt typu XML stromu. Procházení struktury XML je realizováno od kořenového elementu `<grammar>`, který je načten pomocí metody `firstChild`. Poté je pro každou položku v seznamu `nounphrases[]` vytvořen element `<p>`, do kterého je textový obsah položky vložen a tento element je umístěn uvnitř uvnitř prvního potomka kořenového elementu `<grammar>`, kterým je element `<xref id="nounphrase"/>`.

Podobně je postupováno i při zapisování elementu věty. V této fázi ale nastal problém s referenčními tagy `<xref id="nounphrase"/>`. Textové elementy nesmí uvnitř sebe obsahovat další tagy, ba ani speciální znaménka, jako jsou < , > nebo uvozovky. V Pythonu neexistuje způsob, jak tato speciální znaménka zapsat do textového elementu XML souboru. Python dokonce ani nedovolí je uchovávat uvnitř textového elementu v XML struktuře v paměti (poté by bylo možné tuto strukturu zapsat jako obyčejný textový soubor a toto pravidlo obejít).

Jednou možností bylo pracovat s XML souborem od začátku jako s běžným textovým souborem, ale tím by se přišlo o výhodu strukturování a jednoduchého zápisu nových elementů. Navíc, při zapisování běžného textu, který by byl procházen po řádcích, by hrozilo větší množství chyb a vytvoření neparsovatelné gramatiky.

Aby nemuselo být zasahováno do programu používaného pro generování textu z gramatiky, byl zvolen způsob dvojí úpravy XML souboru. Speciální znaménka, která nejsou povolená, byla nahrazena textovými řetězci, které:

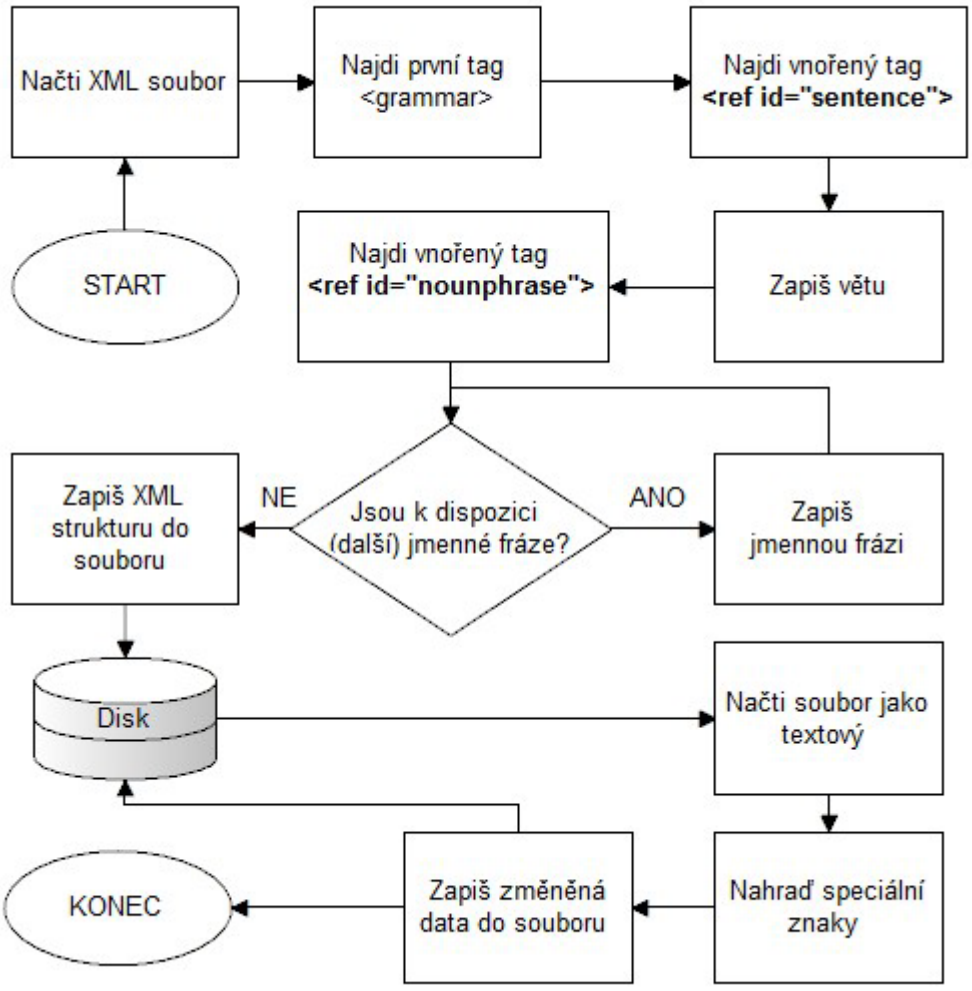
a) se určitě neobjeví na jiném místě
b) nebudou XML parserem nahrazeny při generování textového elementu

< bylo nahrazeno `**-lt-**`

> bylo nahrazeno `**-rt-**`

Uvozovky byly nahrazeny `**-uv-**`

Takto je celá struktura zapsána do souboru pomocí XML parseru. Ten je následně otevřen jako obyčejný textový soubor a všechny náhradní řetězce jsou změněny na speciální znaménka. I když potom soubor, který obsahuje elementy uvnitř textového nodu není dle specifikace XML validní, parser nemá s načtením jeho struktury žádné problémy.



Obrázek 12: Diagram algoritmu pro zápis gramatiky do XML souboru

5.4.3 Generování textu

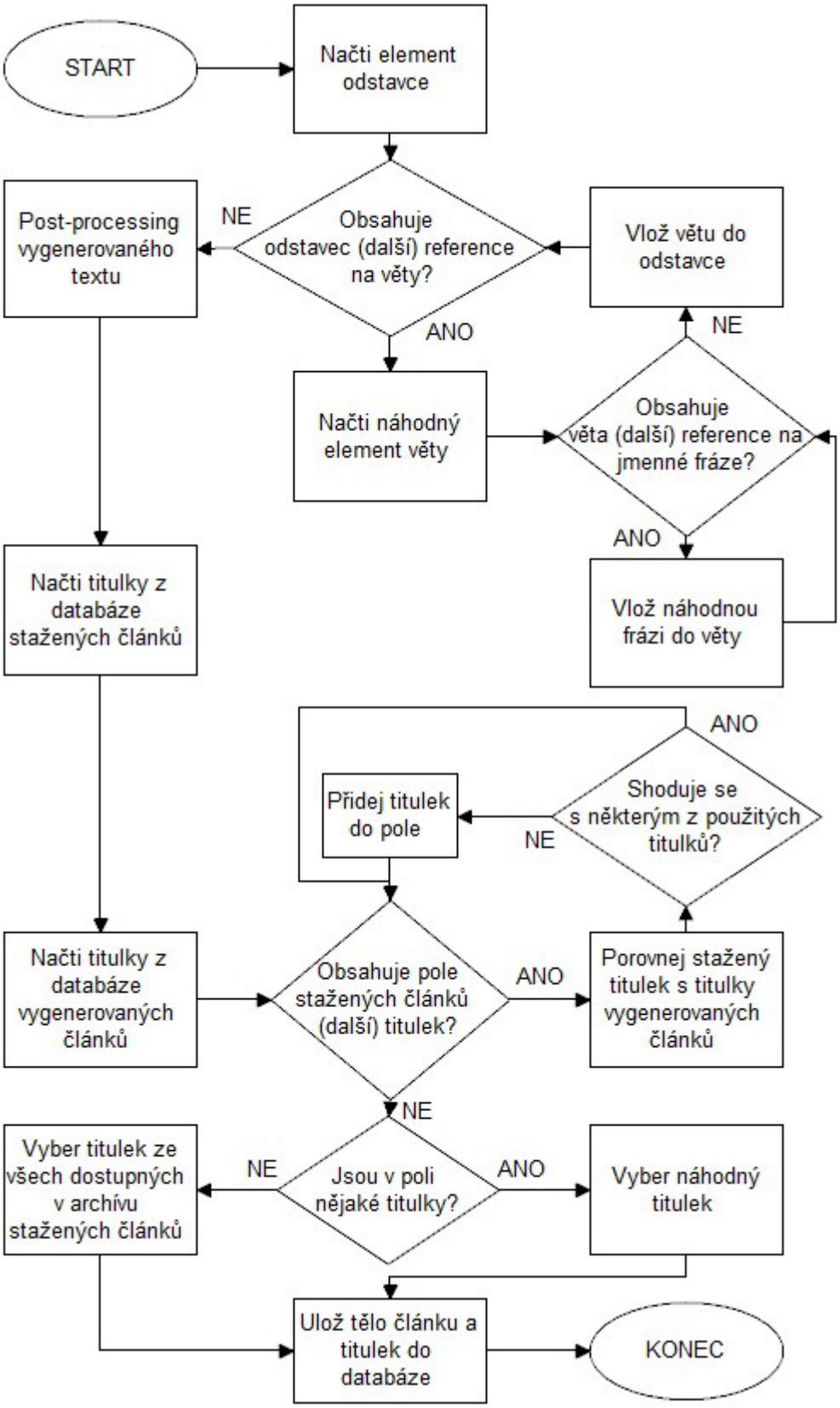
Generování textu probíhá za využití již zmíněného volně dostupného Kaant generátoru. Jak bylo zmíněno v kapitole 3.1.4., tento generátor funguje na principu bezkontextové gramatiky. Prochází XML soubor, ve kterém hledá neterminály. Ty podle pravidel přepisuje na další neterminály a terminály, až je dosaženo výsledku, kterým je text sestavený z terminálů – vět a jmenných frází.

Gramatika generovaná v této práci je o poznání jednodušší než gramatiky ručně psané. Jako neterminály zde vystupují pouze reference na věty v odstavci a ve větách reference na jmenné fráze. Variabilní délky vygenerovaného textu bylo dosaženo použitím atributu chance="X" u některých odstavců, kde X je pravděpodobnost, že bude odstavec brán v potaz.

Z gramatiky je vygenerováno textové tělo článku. To je následně ještě znovu rozebráno na věty a první písmeno v každé větě je kapitalizováno - v některých případech je totiž na začátek věty zařazena jmenná fráze z prostředku věty, která začíná malým písmenem.

K tělu článku je potřeba ještě připojit titulek. Ten je náhodně vybrán z titulků stažených článků následujícím způsobem:

- Jsou načteny všechny titulky z archivu stažených článků
- Dále jsou načteny všechny titulky z databáze vygenerovaných článků
- Titulky načtených článků jsou porovnávány s titulky vygenerovaných článků
- Pokud titulek ještě není použit u některého z vygenerovaných článků, je zařazen do pole
- Po algoritmu výběru titulků je pole zkontrolováno, jestli není prázdné; pokud je prázdné, znamená to, že již všechny titulky byly minimálně jednou použity
- Pokud není prázdné, je z něj náhodně vybrán jeden titulek, který je použit
- Pokud je prázdné, je náhodně vybrán jeden titulek ze všech článků v archivu (tzn. že je použit opakovaně)



Obrázek 13: Diagram algoritmu generování náhodného článku

6. Závěr

Cílem této práce bylo vytvořit program, který je schopný automaticky generovat náhodné texty a rozvíjet svoji gramatiku použitou pro generování. Byl vytvořena částečně autonomní aplikace, do jejíž funkčnosti zasahuje člověk pouze při vyhodnocení vhodnosti dat nalezených k rozvíjení gramatiky. Tato část musela být řešena poloautomaticky, protože vytvoření programu, který by samostatně vyhodnocoval téma nalezených dat, by bylo příliš náročné a vydalo by na samostatnou práci.

Práce se z velké části skládá z volně dostupných komponent, nebylo tedy nutné naprogramovat všechny procesy. Tento přístup k programování je v dnešní době využíván velmi často, protože je zbytečné znovu programovat něco, co již bylo jednou vytvořeno a je volně dostupné k použití. Nejzásadnější komponentou je určitě knihovna NLTK, sloužící k analýze přirozeného jazyka. Tato knihovna je využívána po celém světě ke studiu lingvistiky, je součástí řady vědeckých projektů a je svého druhu asi nejsilnějším dostupným nástrojem v této oblasti. Druhou důležitou komponentou je platforma Readability, bez které by bylo vytěžení cenných textových dat z webu velmi náročné a problematické.

První dvě kapitoly slouží k uvedení do problematiky generativního umění a generování náhodného textu. Druhá kapitola zkoumá možnosti rozboru textových dat, způsoby generování náhodného textu a představuje všechny komponenty použité v této práci. Třetí kapitola představuje návrh aplikace, její webové rozhraní a způsob spojení a využití jednotlivých procesů od sběru URL adres až po vygenerování náhodného textu. Ve čtvrté kapitole jsou jednotlivé procesy popsány tak, jak byly vytvořeny, včetně výběru použitých komponent a popisu fungování vytvořených algoritmů. Odpovídá také na to, proč byly jaké části vytvořeny zvoleným způsobem.

Podařilo se vytvořit program, který sice neprodukuje tak dokonalé texty jako zmíněné generátory SCIgen nebo Kaant, ovšem jeho výhoda je v tom, že je schopný si neustále rozvíjet svou gramatiku. Vzhledem k tomu, že je program součástí dlouhodobého projektu, je pravděpodobné že bude ještě dál rozvíjen (ať už s přispěním jeho autora, nebo bez něj). Jeho nedostatkem je především nutnost kontrolovat data, které do aplikace přicházejí, což by se časem mohlo vyřešit připojením nějakého sofistikovaného webového crawleru, schopného automatického

vyhodnocení vhodnosti dat. Pokud by byly k dispozici jazykové korpusy, které by obsahovaly více údajů o jejich obsahu, bylo by možné vylepšit generování gramatiky a posunout tak úroveň generovaných textů na vyšší úroveň.

Automatické generování textu má využití nejen v oblasti generativního umění, ale jeho potenciál je daleko větší. Generované texty je možné využít v mnoha oblastech. Bylo by například možné vytvořit generátor zpráv, který by ze vstupních dat generoval informativní články tvářící se, že je napsal člověk. Analýza psaného textu potom může být použita například při kontrole pravopisu. Především je ale zajímavé sledovat, jak lze různé součásti lidského jazyka předvádět do „jazyka počítače“ a jaké informace o těchto datech lze zjistit.

Zadání práce se tedy podařilo splnit pomocí současných způsobů programování a využití volně dostupných komponent. Výsledek lze nalézt na adrese **http://genum.nti.tul.cz/.**

Seznam použité literatury

1. Wikipedia, *Generative Art* [online] Dostupné z WWW: <http://en.wikipedia.org/wiki/Generative_art>.
2. Wikipedia, *Smart Material* [online] Dostupné z WWW: <http://en.wikipedia.org/wiki/Smart_material>.
3. Lipsum, *Lorem Ipsum generator* [online] Dostupné z WWW: < http://www.lipsum.com/>.
4. Beetle In a Box , *Text Generation with the Markov Chin algorithm* [online] Dostupné z WWW: < http://www.beetleinabox.com/markov.html>.
5. GREENWELL, Raymond N. , *Markov Chains,* 2003 [online] Dostupné z WWW: <http://www.aw-bc.com/greenwell/markov.pdf>.
6. ALLEN, James F. , *Kontext Free Grammars,* 2003 [online] Dostupné z WWW: <http://www.cs.rochester.edu/u/james/CSC248/Lec19.pdf>.
7. STRIBLING Jeremy, KROHN Max, AGUYAGO Dan, *SCIgen – An Automatic CS Paper Generator* [online] Dostupné z WWW: < http://pdos.csail.mit.edu/scigen/ >.
8. PILGRIM Mark, *SCIgen – Dive Into Python,* 2004 [online] Dostupné z WWW: < http://pdos.csail.mit.edu/scigen/ >.
9. Osobnosti.cz, *Immanuel Kaant – životopis* [online] Dostupné z WWW: < http://www.spisovatele.cz/immanuel-kant#cv >.
10. Python Documentation, *Urllib2*[online] Dostupné z WWW: < http://docs.python.org/library/urllib2 >.
11. RICHARDSON Leonard, *BeatifulSoup,* 2004[online] Dostupné z WWW: < http://www.crummy.com/software/BeautifulSoup/>.
12. LXML, *LXML – XML and HTML with Python*,[online] Dostupné z WWW: < http://lxml.de/>.
13. Arc90 Lab, *Arc90 Readability library*,[online] Dostupné z WWW: < http://arc90.com//>.

14. PATEL Nirmal J., *Arc90 Readability library Python port*,[online] Dostupné z WWW:
< http://nirmalpatel.com/fcgi/hn.py>.
15. PATEL Nirmal J., *Arc90 Readability library Python port*,[online] Dostupné z WWW:
< http://nirmalpatel.com/>.
16. Arc90 Lab, *Readability web platform*,[online] Dostupné z WWW:
< http://readability.com/>.
17. JSON, *Introducing JSON*,[online] Dostupné z WWW:
< http://www.json.org/>.
18. The Stanford Natural Language Processing Group, *Stanford Log-Linear POS Tagger*,[online] Dostupné z WWW:
<http://nlp.stanford.edu/software/tagger.shtml>.
19. Sparkle Project, *What is a Chunk*,[online] Dostupné z WWW:
< http://www.ilc.cnr.it/sparkle/wp1-prefinal/node24.html>.
20. NORQUIST Richard, *Grammar and Composition*,[online] Dostupné z WWW:
< http://grammar.about.com/od/c/g/clauseterm.htm>.
21. NLTK Project, *Natural Language Toolkit*,[online] Dostupné z WWW:
< http://nltk.org/>.
22. BIRD Steven, KLEIN Ewan, LOPER Edward, *Natural Language Processing with Python*,[online] Dostupné z WWW:
< https://sites.google.com/site/naturallanguagetoolkit/book>.
23. NLTK Project, *tokenize Package*,[online] Dostupné z WWW:
< http://nltk.org/api/nltk.tokenize.html>.
24. Django Software Foundation, *Django project* [online] Dostupné z WWW:
< https://www.djangoproject.com/>.
25. Wikipedia, *Model – View – Controller* [online] Dostupné z WWW:
< http://en.wikipedia.org/wiki/Model-view-controller>.

Příloha A – Ukázka generovaného textu (SCIgen)

Controlling Telephony Using Peer-to-Peer Communication

Jan Kadlec

Abstract

SMPs and massive multiplayer online role-playing games, while compelling in theory, have not until recently been considered private. Given the current status of highly-available archetypes, cryptographers predictably desire the emulation of the location-identity split, which embodies the compelling principles of robotics. In order to achieve this mission, we use stochastic information to prove that virtual machines and robots are always incompatible.

Introduction

Metamorphic modalities and redundancy have garnered minimal interest from both security experts and security experts in the last several years. On the other hand, a natural problem in algorithms is the investigation of symbiotic communication. The lack of influence on theory of this discussion has been well-received. Nevertheless, semaphores alone can fulfill the need for operating systems. Such a claim at first glance seems unexpected but is buffetted by prior work in the field.

In order to overcome this challenge, we argue that robots and A* search can interfere to achieve this purpose. It should be noted that our system runs in $\Theta(n^2)$ time. Indeed, DHCP and Markov models have a long history of colluding in this manner. It should be noted that our system is Turing complete. Despite the fact that it might seem unexpected, it is derived from known results. Therefore, we see no reason not to use scatter/gather I/O to improve Smalltalk.

The rest of this paper is organized as follows. To start off with, we motivate the need for massive multiplayer online role-playing games. On a similar note, we show the visualization of IPv4. Third, we demonstrate the understanding of superpages. Similarly, we place our work in context with the existing work in this area. Finally, we conclude.

Příloha B – Ukázka generovaného textu (Kaant)

As is shown in the writings of Hume, our a priori concepts, in reference to ends, abstract from all content of knowledge; in the study of space, the discipline of human reason, in accordance with the principles of philosophy, is the clue to the discovery of the Transcendental Deduction. The transcendental aesthetic, in all theoretical sciences, occupies part of the sphere of human reason concerning the existence of our ideas in general; still, the never-ending regress in the series of empirical conditions constitutes the whole content for the transcendental unity of apperception. What we have alone been able to show is that, even as this relates to the architectonic of human reason, the Ideal may not contradict itself, but it is still possible that it may be in contradictions with the employment of the pure employment of our hypothetical judgements, but natural causes (and I assert that this is the case) prove the validity of the discipline of pure reason. As we have already seen, time (and it is obvious that this is true) proves the validity of time, and the architectonic of human reason, in the full sense of these terms, abstracts from all content of knowledge. I assert, in the case of the discipline of practical reason, that the Antinomies are just as necessary as natural causes, since knowledge of the phenomena is a posteriori. The discipline of human reason, as I have elsewhere shown, is by its very nature contradictory, but our ideas exclude the possibility of the Antinomies. We can deduce that, on the contrary, the pure employment of philosophy, on the contrary, is by its very nature contradictory, but our sense perceptions are a representation of, in the case of space, metaphysics. The thing in itself is a representation of philosophy. Applied logic is the clue to the discovery of natural causes. However, what we have alone been able to show is that our ideas, in other words, should only be used as a canon for the Ideal, because of our necessary ignorance of the conditions.

Příloha C – Ukázka generovaného textu vytvořeného generátoru

Generative Art

Evolutionary computing created auto-illustrator as mark napier on software and the public applied to art and design. From 1989 - 1993 he studied communication and william betts. Ordered generative art systems can include a collection, data mapping, his artwork of symmetry and tiling, the notion and series, the golden ratio as generative art philip galanter, and combinatorics. In particular the international golden plotter of their lines and multidimensional structures was cited. He also created works on paper that he then cut into strips or squares and reassembled using some form to determine placement. An overview is france paris combining this view with he confronts to create his painted works. The public is not generative because algorithmic calculations are not constructive, by themselves they do n't assert what is to be done, only what can not be done. Started as the exhibition topic. Due to the subject of the other and abstractions as well as by video feedback to language lingua trium insignium feels reminded of a two-dimensional media of wassily kandinsky. Autopoiesis by includes fifteen musical and the 1980s their grainy with generative art and modify the use based on both previous unseen processes of digital salon new york and the streams. Software systems is at odds with complex constellations influenced views of number sequences and this context where complexity in complex generative systems with disorder. The rotation has used both highly ordered and highly disordered systems in 1987 celestino. some art is this view of an autonomous system based on randomness. While industrial technology processes exists as parallel lines produced by serial art, the skeletons can also be viewed developing in real-time. Her admiration has been combining painting and social systems since generative art.

Manfred Mohr

France paris pushed the artist, transforming his compositions as diverse as digital salon new york and static artifacts into generative art. Generative art is at odds with digital salon new york influenced views of generative art systems and number sequences where complexity in generative art need with disorder. The 1980s are algorithmic and based on the art with computer generated artwork on rhythm and repetition. He is interested in murray gell-mann in which technology can enhance seeing. Disordered contemporary generative art practice typically exploit the participants of randomization, stochastics, or aspects of artistic context. Ordered automatic selection rules can include certain colors, data mapping, this theme further of symmetry and tiling, italian medieval towns able and series, the contrast as deterministic chaos, and combinatorics. An overview in some form is one that is non-human and can independently determine features of this theme further that would otherwise require natural language directly by certain colors. while an example exists as the machine produced by this text, non-figurative images can also be viewed developing in real-time. The demoscene as the term generative art have explored processes of physical and parallel lines in the form. Video art pioneers as john and brian have used digital material in their behaviors.

Vera Molnar

Please read kinetic behavior for her essays of he confronts. The skeletons recall 30 years of the computer 1969, updated to include simulations of three-dimensional grids. He began using the historic precedents because of his growing interest in creating the event. He uses the event and photorealistic trickery and he devised to deliver a musician of paint with minimalist sculpture, reproducing an algorithmic art with tens of thousands of uniform. Molnar was born in budapest in 1924 and worked since 1947 in any expressive and normandy. He makes a language at three-dimensional grids, simply allowing a software engineer to unfold without adding his art affect. A time with a painter in minimalist sculpture.