MASARYKOVA UNIVERZITA

Filozofická fakulta

Ústav českého jazyka

Český jazyk se specializací počítačová lingvistika

David Klement

Grafické rozhraní pro prediktivní zápis textu

Bakalářská diplomová práce

Vedoucí práce: RNDr. Adam Rambousek, Ph.D.

2015

Prohlašuji, že jsem bakalářskou diplomovou práci vypracoval samostatně s využitím uvedených pramenů a literatury.

.....
autor práce

Na tomto místě bych rád poděkoval Adamu Rambouskovi za veškerou pomoc, trpělivost a všechny odborné rady, jež mi ochotně poskytoval po celou dobu psaní této práce.

Děkuji také své rodině za morální podporu, Janu Šedovi za cenné rady v oblasti programování v JavaScriptu a Marii Staré za pomoc s jazykovými korekturami.

Abstrakt

Tato bakalářská práce shrnuje vývoj systémů pro predikci textu, porovnává je a popisuje technologie, na nichž jsou tyto systémy postaveny. Dále se zabývá hodnocením účinnosti vybraných metod zadávání textu a možnostmi jejího zlepšení použitím různých prediktivních systémů. Praktická část práce je zaměřena na vývoj grafického uživatelského rozhraní pro prediktivní zápis textu, které je vhodné pro zařízení bez dotykové obrazovky a s klasickou klávesnicí QWERTY. Toto rozhraní zobrazuje predikce generované serverovou částí aplikace, která využívá jazykový model ve formě n-gramů vytvořených z největšího českého korpusu czTenTen.

Abstract

This bachelor thesis summarizes development of text prediction systems, compares these and describes technologies these systems are built upon. Furthermore, the thesis outlines methods of evaluation for selected text input systems as well as text prediction systems. The part about implementation is focused on the development of the graphical user interface for predictive text entry that is well suited for devices without a touchscreen and with a benefit of a full QWERTY keyboard. This user interface presents a user with predictions produced by the server-side application which uses n-grams generated from the largest Czech corpus, czTenTen, as a language model.

Klíčová slova

predikce textu, jazykový model, zadávání textu, KSPC

Keywords

text prediction, language model, text input, KSPC

Obsah

Úvod

Různé implementace predikce textu se objevují v mobilních telefonech a elektronických osobních asistentech již přes dvě dekády, přičemž jejich hlavním úkolem je usnadnit zadávání textu na klávesnicích mobilních zařízení. V poslední době se s rozmachem chytrých telefonů objevily systémy predikce textu (a automatické opravy slov, která s ní souvisí) i pro jednoznačné klávesnice typu QWERTY, a to z důvodu zvýšení komfortu psaní. Tyto systémy mají obvykle dvě nevýhody. První z nich je závislost na konkrétním operačním systému, konkrétním zařízení (především v případě starších telefonů) či na konkrétní aplikaci, která zajišťuje zobrazení klávesnice na dotykovém displeji. Takové systémy tudíž nejsou použitelné například pro psaní na stolním počítači na klasické klávesnici QWERTY. Druhou nevýhodou je fakt, že systémy většinou musí spoléhat pouze na lokálně uložená jazyková data, která tak musí být relativně kompaktní co do datové velikosti a nemusí tedy obsahovat takové množství dat, aby uživatel nebyl omezován nedokonalostmi jazykového modelu.

Protipólem velkého množství aplikací pro predikci textu pro mobilní zařízení je téměř prázdná množina takových aplikací pro klasické počítače se standardní klávesnicí QWERTY. Textové procesory typu LibreOffice Writer[1] mívají vestavěné primitivní způsoby doplňování, které ale čerpají data pouze z textu, který je v danou chvíli v editovaném dokumentu. Textové editory typu Atom[2] mívají podobnou funkcionalitu, navíc například při editaci programového kódu nabízí relativně komplexní doplňování, ale pouze podle předem připravených šablon.

Cílem této práce je tedy vytvořit implementaci systému predikce textu, která bude čerpat data z velkého korpusu textů, a navíc bude nezávislá na platformě. Vzhledem k tomu, že korpusová data není možné uchovávat na straně klienta, bylo logickým krokem zvolit model klienta a serveru, který bude zajišťovat generování predikcí. Tento systém na serveru již v době tvorby této práce existoval, což umožnilo soustředit se především na vývoj uživatelského prostředí.

Pro zajištění co nejvyšší kompatibility s různými systémy byla zvolena forma zásuvného modulu do CKEditoru[3], webového editoru HTML (Hyper Text Markup Language). Použitými technologiemi tedy byl především JavaScript pro programovou logiku, Ajax (asynchronous JavaScript and XML (Extensible Markup Language)) pro komunikaci se serverem a HTML a CSS (Cascading Style Sheets) pro uživatelské rozhraní.

[1] `https://www.libreoffice.org/discover/writer/`
[2] `https://atom.io/`
[3] `http://ckeditor.com/`

Část I

Teoretická část

Kapitola 1

Obecně o predikci textu

1.1 Definice pojmu predikce textu

Predikce textu, neboli automatické doplňování textu (angl. autocomplete) je funkce v aplikaci, která na základě uživatelského vstupu (většinou prvních několika znaků slova) doplňuje zbytek části textu (většinou slova). V ideálním případě funkce jednoznačně rozpozná, co chtěl uživatel napsat, a nabídne mu právě to doplnění, jež uživatel chce. V praxi obvykle uživatel dostává na výběr z několika možností, které systém vyhodnotil jako nejpravděpodobnější doplnění vstupu. Systémy se liší způsobem zužování výběru možností a jejich řazením, stejně jako vizuálním podáním nabídky.

Systémy automatického doplňování textu obecně fungují nejlépe v implementacích na omezených doménách, tzn. například doplňování e-mailových adres v e-mailových klientech, klíčových slov určitého programovacího jazyka v programátorských editorech či doplňování ve formulářových polích s předem vymezeným očekávaným vstupem.

1.2 Disambiguace, automatické opravy a predikce textu

Automatické opravy a predikce textu jsou dva velmi úzce svázané pojmy. Prvním se myslí opravení textu poté, co jej uživatel chybně zadal, druhým pak nabídnutí textu uživateli předtím, než jej zadal. Na pomezí těchto dvou technik se ještě vyskytuje třetí, a to disambiguace vstupu z nejednoznačné klávesnice. Všechny tyto tři techniky jsou úzce spjaty proto, že je nutné nejdříve nějakým způsobem odhadnout, co uživatel chtěl, případně bude chtít, napsat.

Ačkoliv je praktická část této práce zaměřena na predikci textu a disambiguací se nezabývá, zdálo se vhodné popsat i vývoj a některé způsoby řešení problematiky disambiguace vstupu, protože řeší podobné problémy.

1.3 Úskalí jazyků s rozvinutým tvaroslovím

Systémy pro predikci textu určené k tvorbě volného textu jsou obvykle, jak bude zřejmé z dalšího popisu jednotlivých aplikací, založeny na více či méně rozsáhlém lokálně uloženém slovníku jednotlivých slov či slovních spojení, ze kterého vybírají možná doplnění. Tyto systémy zpravidla umožňují uživateli přidávat manuálně slova, která napsal a nejsou ve slovníku, což slouží jako základní uživatelské přizpůsobení. Lokální slovník má ovšem tu nevýhodu, že jeho velikost je značně limitována možnostmi zařízení, tj. jeho paměťovou a výpočetní kapacitou. To je problém především pro jazyky s rozvinutým tvaroslovím, jako je čeština, které potřebují na rozdíl od analytických jazyků, např. angličtiny, poněkud větší množství tvarů jednotlivých slov.

Rozdílnost angličtiny a češtiny v ohledu množství slovních tvarů připadajících na jedno lemma[4] lze porovnat například na dvou velkých korpusech, enTenTen13 a czTenTen12 (Suchomel, 2012). Jde o korpusy vytvořené z textů získaných na Internetu, které byly vyčištěny a zbaveny duplikátů nástrojem Onion[5]. Anglický korpus enTenTen13 obsahuje 39 452 714 slov a 37 366 565 lemmat (Sketch Engine, 2015a). Naproti tomu český korpus czTenTen12 obsahuje slov 18 725 879 a lemmat pouze 13 976 481 (Sketch Engine, 2015b). V češtině tedy podle výše uvedených dat připadá na jedno lemma 1,339 slovního tvaru, naproti tomu v angličtině na jedno lemma připadá pouze 1,056 slovního tvaru. Je tedy vidět, že slovník pro češtinu by v ideálním případě měl být minimálně 1,27× větší než slovník o stejném množství lemmat pro angličtinu. Zde je na místě uvést, že to je ve srovnání s údaji, jež uvádí Németh a Zainkó (2001), velice málo. Ti totiž píší, že pro udržení míry chybovosti v rozpoznávání řeči je pro němčinu potřeba slovník zhruba 4× větší a pro maďarštinu dokonce 20× větší než pro angličtinu.

Vezmou-li se v potaz teoretická data, vychází poměr slovních tvarů na lemma i pro češtinu výrazně vyšší. Pro účely této práce byla provedena empirická analýza cca 500 nejčastějších slov v češtině pomocí morfologického analyzátoru Majka. Tento experiment ukázal, že čeština je v teoretickém poměru (unikátních) slovních tvarů na lemma skutečně relativně blízko maďarštině s poměrem 17,537. Nutno podotknout, že se jedná o tvary vzniklé nejen skloňováním a časováním, ale i záporkou *ne-*, stupňováním adjektiv či jejich kombinacemi.

Množství slovních tvarů na lemma ve flektivních jazycích komplikuje implementaci slovníkových jazykových modelů do aplikací pro predikci textů v těchto jazycích, protože je nutno uchovávat větší množství slovních tvarů než pro angličtinu. Stejně tak je žádoucí zohledňovat kontext, aby byly slovní tvary vybírány správně, protože například bez zohlednění syntaxe mohou predikce vést k negramatickým větám. (Ghayoomi a Momtazi, 2009, s. 5234–5235)

[4] základní slovní tvar

[5] `http://corpus.tools/wiki/Onion`

Kapitola 2

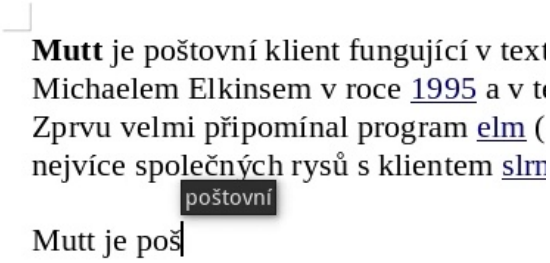
Klíčové vlastnosti systému pro predikci textu

Navzdory velmi odlišným způsobům použití systémů pro doplňování textů jsou některé jejich klíčové vlastnosti pro většinu implementací společné. Studie, které zkoumaly efektivitu těchto systémů, obecně docházejí k podobným závěrům, co se požadavků na tyto systémy týče. Ať jde o vyhledávač typu Google[6] nebo o predikci volného textu v textovém editoru, musí systém mít několik vlastností, aby měl uživatel při užívání aplikace pocit, že mu doplňování textu skutečně pomáhá. Primární je rychlost, s níž aplikace možná doplnění uživateli zobrazuje. Nandi a Jagadish (2007) uvádí, že by odezva neměla převyšovat 100 ms. Tento faktor je rozhodující pro to, zda se uživatel rozhodne nabídnutá doplnění použít, nebo raději napíše zbytek textu sám. V případě implementace predikce textu do editoru, kde uživatel zadává volný text, jímž si je relativně jist, je tato vlastnost pro uživatelovo rozhodnutí patrně klíčová. Naproti tomu ve vyhledávání lze očekávat, že uživatel bude ochoten počkat delší chvíli na zobrazení doplnění. S tímto souvisí i další vlastnost – umožnit uživateli pohodlně nabízené predikce ignorovat (Ward et al., 2012). Rozhodne-li se totiž uživatel, že např. bude rychlejší text manuálně napsat a predikce nepoužít, je žádoucí, aby mu grafické zpracování nabídky žádným způsobem neztěžovalo tak učinit. Z tohoto důvodu je ve většině systémů nabídka předvídaných výrazů zobrazována pod textovým polem (např. automatické doplňování ve vyhledávači Google (Google, 2015)).

Zobrazení nabídky v textových editorech, kde se nejedná o jednořádkové pole s očekávaným vstupem pouze několika jednotlivých slov, je předmětem několika studií, z nichž např. jedna navrhuje zobrazovat nabídku na spodní části obrazovky, aby uživatel nebyl nucen přesouvat pozornost příliš daleko od klávesnice. Lze ale polemizovat o tom, jak je tato argumentace validní pro uživatele bez motorického nebo jiného postižení, jelikož ti často při psaní na klávesnici nehledí. Textové editory pro psaní programového kódu

[6] `https://www.google.com`

tento problém většinou řeší zobrazováním nabízených doplnění v menu nad nebo pod kursorem (tedy zhruba podobně jako jednořádková vyhledávací a jiná pole). Podobně to řeší i kancelářský balík LibreOffice, který zobrazuje po napsání prvních tří písmen slova jednu predikci nad kursorem. Výběr predikce se odvíjí od frekvence, přičemž velký důraz je kladen na délku nabízeného slova (obrázek 1). Tento systém uživateli umožňuje udržovat pozornost na relativně malé oblasti obrazovky, čímž se zvyšuje pravděpodobnost, že se rozhodne doplnění použít.



Obrázek 1: Implementace predikce textu v LibreOffice Writer 4.2.8.2

Lze tedy říci, že nabídka slov, která daný systém vyhodnotil jako vhodná doplnění, by neměla být obtěžující, mělo by být snadné ji ignorovat a měla by se nacházet co nejblíže místu, kam je soustředěna uživatelova pozornost. To se, jak bylo naznačeno výše, může lišit v závislosti na cílové kategorii uživatelů.

Kapitola 3

Smysl automatického doplňování pro uživatele

3.1 Predikce jako způsob zjednodušení zápisu

Důvody, pro které si může uživatel vybrat užívat predikce textu, se liší podle přístroje, na němž píše, a dalších okolností. Na mobilních zařízeních, která disponují nejednoznačnou klávesnicí (typicky starší mobilní telefony s klávesnicí ITU, tedy typickou 12tlačítkovou klávesnicí), je predikce výhodná proto, že umožňuje zadávat text jediným stisknutím kláves, které reprezentují znaky obsažené v psaném slově. Bez predikce by byl uživatel nucen v případě písmene, které na klávese není na první pozici, stisknout tutéž klávesu vícekrát po sobě[7].

Dalším případem jsou mobilní zařízení, která mají dotykový displej, kde je zase problém s přesností, s jakou je uživatel schopen se na klávesy trefit prstem nebo stylem. Tento případ sice spadá do kategorie automatických oprav, ale může být kombinován s predikcí textu. (Kocienda et al., 2012)

Při psaní na jednoznačné fyzické klávesnici (typicky klasická klávesnice QWERTY, jež je pro jednoduchost v této práci považována za reprezentanta všech ostatních lokalizovaných rozložení) obvykle není predikce potřeba, ale například pro uživatele handicapovaného může být výhodné mít k dispozici systém, který mu ušetří stisky kláves nutné k zapsání celého slova.

3.2 Predikce jako inspirace

Vedle původního určení predikce textu, kterým je zrychlit zadávání textu a tím komunikaci na rozhraní člověk–počítač, může pro běžného uživatele automatické doplňování textu plnit ještě několik dalších funkcí.

[7] více o klávesnicích ITU v kapitole 6.1.

Takovým případem může být doplňování ve vstupním poli vyhledávače typu Google, případně jeho obdoby v nějaké menší síti, například na elektronickém portálu knihovny. (Google, 2015)

Ward et al. (2012) uvádí, že studenti, na kterých testoval automatické doplňování ve vyhledávání v rámci informačního systému knihovny, poukázali v následném dotazníku na několik dalších funkcí, které pro ně predikce plnila.

Na otázku, jak by svým přátelům vysvětlil, k čemu je našeptávač dobrý, odpověděl jeden student, že rozhodně jako asistence v pravopisu. Studenti totiž často nemají z ústních zadání v hodinách úplně přesnou představu, jak se např. jméno autora, jehož díla jim jejich vyučující doporučil jako zdroj informací k práci, přesně píše. Autoři studie ze záznamů hledání vyvodili, že někteří studenti si ani nemuseli být vědomi faktu, že začátek jména autora zadali špatně, protože predikce jim nabídla rovnou správnou verzi a oni ji vybrali, aniž by přemýšleli, v čem se nabídka liší od jejich vstupu.

Mezi další výhody, které studenti uvedli ve zmíněné studii, patřilo například to, že automatické doplňování jim potvrdilo, že podobné téma již někdo před nimi hledal a jejich postup je tedy pravděpodobně správný.

S tímto souvisí i jedna odpověď od dalšího účastníka studie, která uvádí, že našeptávač pro něj fungoval jako nástroj pro dotvoření myšlenky, jako brainstorming od počítače.

Z výše uvedeného lze tedy vyvozovat, že automatické doplňování textu na klasickém počítači skutečně nemá význam pouze pro uživatele tělesně či jinak handicapované, kteří mohou mít problém se samotným zadáváním textu, ale může posloužit i běžným uživatelům jako nástroj pro kontrolu toho, zda to, co píšou, je správně, k tématu a podobně. Základním rozdílem oproti předchozím případům využití predikce je fakt, že uživatel nemusí přesně vědět, co chce napsat, před tím, než to vidí jako doplnění.

Kapitola 4

Jak predikce textu funguje

4.1 Slovníkový model

Pokud prediktivní systém využívá slovníkového modelu, znamená to, že má v paměti zařízení uložený slovník (seznam slov) společně s jejich frekvencemi v referenčním korpusu, tedy tzv. frekvenční slovník[8]. Implementace frekvenčních slovníků v aplikacích pro predikci textu se pravděpodobně značně liší a vzhledem k tomu, že jde většinou o uzavřené komerční systémy, není možné konkrétní implementace zjistit. (Henry, 2014)

U systémů na mobilních telefonech s ITU klávesnicí výběr kandidátského slova probíhá tak, že systém vytvoří kombinace všech slov, která mohou vzniknout z písmen na stisknutých klávesách, a vzniklá slova porovná se slovníkem. Průnik těchto dvou množin pak je uživateli nějakým způsobem prezentován.

Systémy s plnou klávesnicí QWERTY na mobilních zařízeních čelí, jak bylo naznačeno výše, jinému problému. Jelikož je velikost zařízení omezená a klávesy jsou tudíž relativně malé, je vysoká pravděpodobnost, že se uživatel dopustí chyby při zadávání vstupu. (Kocienda et al., 2012)

Řešení je podle Kocienda et al. (2012) následující. Po získání vstupu z klávesnice se vytvoří permutace, které jsou pak porovnávány se slovníkem. Permutace jsou v tomto případě řetězce, jež byly vytvořeny tak, že pro každý znak A v původním zadaném řetězci O je vytvořen nový řetězec P obsahující na pozici původního znaku A znak A nebo znak, který byl na klávesnici původnímu znaku A sousedem. Pro slovo pes by tak vytvořené permutace při použití klávesnice QWERTY vypadaly na anglické QWERTY takto:

pes, oes, les, prs, pds, pws, pss, pew, pee, pea, ped, pex, pez

Dalším řešením je hledání slov ve slovníku, která mají od zadaného slova určitou editační vzdálenost. Editační neboli Levenshteinova vzdálenost je vyjádření míry odlišnosti dvou řetězců minimálním množstvím operací **vložit** znak, **smazat** znak, **prohodit**

[8] více o frekvenčních slovnících v kapitole 4.1.1

dva sousední znaky a **nahradit** jeden znak jiným, které je nutno provést, aby z prvního řetězce byl vytvořen druhý (Levenshtein, 1966). Tato metoda také využívá frekvenčního slovníku, který navíc kombinuje se statistickým modelem pravděpodobnosti výskytu a pravděpodobnosti souvýskytu. (Norvig, 2015)

4.1.1 Frekvenční slovníky

Frekvenční slovník ke každému heslu uvádí hodnoty spojené s frekvencí daného slova ve zdrojovém korpusu. Obvykle to je hodnota absolutní frekvence daného slova, tedy počet výskytů všech tvarů daného slova v korpusu. Například *Frekvenční slovník češtiny* (Čermák a Blatná, 2004) vedle ní uvádí také průměrnou redukovanou frekvenci, což je hodnota, jež vychází z absolutní frekvence, avšak zohledňuje rozložení daného slova v korpusu. Pokud tedy mají dvě slova stejnou frekvenci, avšak první z nich se vyskytuje pouze v malém množství odborných textů, zatímco výskyty druhého jsou rovnoměrně rozloženy přes celý korpus, druhé slovo dostane vyšší číselnou hodnotu ARF (average reduced frequency) než slovo první. Podle hodnoty ARF tak lze vyvodit, že druhé slovo je obecně známější a užívanější širším publikem (Čermák a Blatná, 2004). V kontextu s predikcí slov to může například znamenat, že je pravděpodobnější, že uživatel chce napsat spíše druhé slovo nežli první, které je pravděpodobně značně odborné.

Tvorba frekvenčních slovníků byla v dobách před rozvojem počítačů schopných zpracovat velká množství textů poněkud komplikovaná. Databáze textů bylo obtížné jak vytvářet, tak aktualizovat a jejich výroba byla náročná a pomalá. Nástup počítačů však většinu těchto problémů odstranil, takže je možné velmi snadno z korpusu vyrobit frekvenční slovník, který bude tak aktuální, jako je samotný korpus. Na rozdíl od ručního zpracování dat navíc počítačové zpracování dovoluje pracovat s mnohem většími korpusy, takže výsledný frekvenční slovník může být i přesnější. Pro srovnání, jeden z prvních frekvenčních slovníků pro angličtinu, *The Teachers Word Book of 30 000 words*[9] (Lorge, 1944), měl 30 tisíc lemmat (z 13 tisíc slovních rodin) a byl vyroben z ručně psaného korpusu, který obsahoval 18 milionů slov (Nation a Waring, 1997). Naproti tomu *A frequency dictionary of contemporary American English*[10] (Davies a Gardner, 2010) byl vytvořen z korpusu současné americké angličtiny, *The Corpus of Contemporary American English*[11], který obsahuje přes 400 milionů slov. Samotný slovník obsahuje pouze 5 000 slov, protože je určen především uživatelům, kteří nejsou rodilí mluvčí a anglicky se učí, takže potřebují výrazně menší množství slov.

Alderson (2007) tvrdí, že ani velké korpusy nemusí být dostačující pro adekvátní zhodnocení frekvenčního rozložení slov v jazyce tak, jak jej vnímají mluvčí daného jazyka. Vychází z výsledků výzkumů zaměřujících se na porovnání hodnocení frekventovanosti

[9] učitelský slovník 30 000 slov

[10] frekvenční slovník současné americké angličtiny

[11] korpus současné americké angličtiny, `http://corpus.byu.edu/coca/`

určitých slov v jazyce jeho zkušenými uživateli s daty získanými z korpusu. Tyto hodnoty se totiž značně rozchází a je tedy otázkou, zda z toho vyplývá, že ani zkušení mluvčí nedokážou správně posoudit pravděpodobnost výskytu daného slova, či že data získaná z korpusů nereprezentují zkušenosti mluvčích s jazykem dostatečně přesně.

4.2 Statistický model

Statistické modelování je založeno na výběru predikce na základě pravděpodobnosti, s jakou se řetězec může vyskytnout v jazyce a případně v daném kontextu. (Ghayoomi a Momtazi, 2009, s. 5233)

Řetězcem mohou být jednotlivá písmena či celá slova. V případě jednotlivých písmen má systém uložen databázi n-gramů s jejich frekvenčním rozložením, z nichž pak vybírá ty, které mají shodný prefix s již zadaným textem. Práce s celými slovy probíhá podobně, ale jako databázi má systém uložen frekvenční slovník.

Hranice mezi statistickým a slovníkovým modelem není v tomto případě zcela pevná, protože slovníkový model také využívá frekvenčního slovníku a rozdíl se tak smazává. Například Ghayoomi a Momtazi (2009, s. 5234) slovníkový model neuvádí samostatně, ale řadí jej pod statistické modelování. Z hlediska této práce se však jevilo vhodnějším modely rozdělit, aby tak lépe vynikl rozdíl mezi jednotlivými metodami predikce textu, které jsou uvedeny v kapitole 6.

4.3 Model se znalostní bází

Často se stává, že návrhy učiněné aplikacemi se slovníkovým či statistickým modelem jsou v daném kontextu nevhodné a mnohdy i negramatické. Tento problém se snaží řešit systémy, které kombinují další metody zúžení výběru návrhů.

Gramatické správnosti lze docílit využitím syntaktické analýzy zadaného řetězce (věty) a n-gramů slovních druhů vytvořených z korpusu (které tvoří databázi pravděpodobností, s jakou je slovní druh X následován slovním druhem Y). Tento model může dále využívat gramatiky pro generování korektních vět. (Ghayoomi a Momtazi, 2009, s. 5234–5235)

Aby bylo docíleno kontextové správnosti návrhů, je nutno zapojit do predikce i sémantickou analýzu. Pro takovou analýzu je možno využít například rozsáhlé sémantické sítě Wordnet[12]. (Ghayoomi a Momtazi, 2009, s. 5235)

Jakkoliv gramaticky správná a sémanticky vhodná, predikce stále nemusí být pro uživatele použitelná, protože je v rozporu s diskursem. Pro eliminaci takových situací může systém využívat korpus obsahující pragmatická data a vyřadit z výběru slova, která jsou pravděpodobně nevhodná. (Ghayoomi a Momtazi, 2009, s. 5235)

[12] `https://wordnet.princeton.edu/`

4.4 Heuristický model

Heuristické modely jsou takové, které se buď v dlouhodobém, nebo krátkodobém měřítku adaptují konkrétnímu uživateli. Činit tak mohou analýzou textu, který v minulosti uživatel zadal, a podle toho uzpůsobovat další predikce. Uzpůsobením se rozumí například upřednostnění slov, která již uživatel zadal před těmi, jež ještě nikoliv. (Ghayoomi a Momtazi, 2009, s. 5235)

Aplikace popisovaná v této práci jistým způsobem využívá heuristického modelování, protože preferuje určitá již uživatelem zadaná slova před predikcemi vrácenými serverovou částí aplikace.

Kapitola 5

Měření účinnosti zadávání textu

Vstupní metody mívají zpravidla dva cíle, jichž se snaží jejich vývojáři dosáhnout. Jedním z nich je omezení počtu nutných úhozů při zadávání textu, druhým pak je rychlost samotného zadávání. Omezování počtu nutných úhozů je aktuální u nejednoznačných klávesnic, které zpomalují zadávání textu především nutností stisknout jednu klávesu vícekrát pro zadání některých znaků. Tento problém u jednoznačných klávesnic QWERTY není tak podstatný, takže záleží spíše na rychlosti zadávání.

Měřit rychlost zadávání textu pak lze několika odlišnými způsoby. Uživatelé, na kterých se testuje, mohou buď psát text předem nepřipravený, tedy ho mohou komponovat sami, nebo mohou kopírovat již hotový text. Kopírování se obecně považuje za lepší, protože eliminuje zpoždění nutné k samotnému vymyšlení psaného textu. (Tarvainen, 2010, s. 2)

5.1 Měření počtu úhozů na znak

Metoda měření počtu úhozů na znak, neboli KSPC[13] (keystrokes per character), je nejužitečnější metrikou pro vyhodnocování těch metod zadávání textu, které se snaží o snížení množství úhozů na znak, jako je například prediktivní systém T9[14]. KSPC tedy udává průměrný počet úhozů nutný k zadání jednoho znaku.

5.1.1 Primitivní výpočet

Naivní přístup k výpočtu KSPC je takový, že se spočítá počet písmen ve slově (i s mezerou za slovem) a počet úhozů, kterými se na dané klávesnici slovo zadá. Nutno počítat i se stisky speciální klávesy NEXT, kterou je nutno použít v případě, že se dvě po sobě jdoucí písmena nachází na stejné klávese (Silfverberg et al., 2000, s. 10). Pro metodu

[13] Přestože název obsahuje stisky kláves, veličina se používá i pro měření účinnosti zadávání textu např. pomocí stylu na dotykovém zařízení.

[14] více o predikci T9 v kapitole 6.1.2.2

multitap na standardní ITU klávesnici by to pro slovo kniha byla následující sekvence kláves:

kniha_: 5566444N442S,

kde N je NEXT a S je znak mezery. KSPC vypočteme jako poměr úhozů na znak:

$$\frac{12}{6} = 2,000.$$

Je ovšem evidentní, že pokud se nyní stejným způsobem provede výpočet pro slovo data, bude sekvence úhozů mnohem jednodušší:

data_: 3282S

a KSPC tedy vyjde:

$$\frac{5}{5} = 1,000.$$

Z toho vyplývá, že je nutno provést výpočet tak, aby výsledná hodnota byla průměrná pro daný jazyk. Takový výpočet má dvě prerekvizity. První z nich je jednoznačný popis postupu, kterým lze dojít k zadání jednotlivých znaků, druhou pak jazykový model. Ten je nutný proto, aby výsledné KSPC bylo nezávislé na konkrétním textu v daném jazyce (tedy aby pro daný jazyk bylo průměrné). (MacKenzie, 2002, s. 196)

5.1.2 Jazykový model

Jazykovým modelem je v případě měření KSPC korpus a jeho omezené formy. Tyto formy se liší podle toho, jaká metoda zadávání je testována. Pokud například jde o metodu, která vybírá znaky nezávisle na kontextu, stačí mít informace o frekvenčním rozložení jednotlivých písmen v korpusu. Pokud by byl výběr znaku závislý na kontextu jednoho písmene, byl by jazykový model frekvenční slovník bigramů. Pro vyhodnocení metod, které využívají k odhadu dalšího písmene kontext celých slov, se použije korpus zredukovaný na unikátní slova a jejich frekvence. (MacKenzie, 2002, s. 197)

5.1.3 Obecný výpočet

Při výpočtu průměrného KSPC pro daný jazyk a metodu se pak postupuje podle následujícího vzorce:

$$KSPC = \frac{\sum (K_c \times F_c)}{\sum (C_c \times F_c)},$$

kde K_c je počet úhozů nutných pro zadání znaku, C_c je velikost znaku a F_c je frekvence daného znaku v korpusu. Pokud se tedy tento vzorec aplikuje na situaci, kdy metoda zadávání textu vybírá písmeno bez ohledu na kontext, platí, že $C_c = 1$, pokud se vybírá v závislosti na jednom předešlém znaku, je $C_c = 2$. Pokud metoda zohledňuje celá slova, počítá se s průměrnou délkou slov v daném jazyce. Ta se počítá jako vážený průměr počtu znaků ve slovech daného jazyka, který pro BNC[15] vychází na 4,59 znaku; pro srovnání, Arif a Stuerzlinger (2009, s. 2) počítají při svých výpočtech s 5 úhozy na slovo, což se považuje za obecný standard. (Tarvainen, 2010, s. 3)

5.1.4 Obecné dělení metod zadávání dle počtu úhozů na znak

Metody zadávání textu lze podle KSPC rozdělit do tří kategorií: ty, pro které je KSPC větší než 1 (např. původní metoda multitap), ty, pro které je rovno 1 (např. klávesnice QWERTY) a nakonec ty, jejichž KSPC dosahuje hodnot nižších než 1 (např. klávesnice QWERTY kombinovaná s predikcí).

Jako základ pro porovnání jednotlivých metod podle KSPC je vhodná standardní klávesnice QWERTY (Arif a Stuerzlinger, 2009). Uvažují-li se pouze malá písmena anglické abecedy, platí pro ni, že KSPC = 1, tedy na jeden znak připadá jeden úhoz, protože pro každý znak má dedikovanou právě jednu klávesu.

5.2 Měření počtu slov za minutu

Počet slov za minutu (WPM, Words per minute) je běžně užívaná metrika pro vyhodnocení rychlosti psaní (Tarvainen, 2010). Tato metoda nezohledňuje počet úhozů, který byl během měření proveden, pouze výslednou délku přepsaného řetězce. Vzorec pro výpočet WPM (Wobbrock, 2007, s. 48):

$$WPM = \frac{T-1}{S} \times 60 \times \frac{1}{5},$$

kde T je délka přepsaného řetězce, S počet sekund od prvního úhozu do posledního úhozu. Konstanta 60 reprezentuje počet sekund za minutu a 1/5 počet znaků na slovo. Aby byla metoda univerzální, počítá se slovo jako pět znaků nezávisle na tom, kolik znaků slova ve výsledném řetězci skutečně mají. Tato standardizace se vyskytuje už od roku 1905 (Yamada, 1980). Odečtení konstanty 1 z počtu znaků ve výsledném řetězci je důležité proto, že většina systémů začíná počítat čas ve chvíli zadání prvního znaku. (Wobbrock, 2007, s. 49)

[15] Britský národní korpus, `http://www.natcorp.ox.ac.uk/`

5.3 Vyhodnocování chyb

Při vyhodnocování rychlosti psaní je důležité zohlednit také chyby ve výsledném textu, jelikož jinak by bylo možné dosáhnout velice dobrých výsledků pouhým rychlým náhodným mačkáním kláves.

Metoda KSPC může být použita k vyčíslení chybovosti tak, že porovná počet znaků zadaných a počet znaků ve výsledném řetězci. Jednou z nevýhod je fakt, že žádným způsobem nerozlišuje chybné znaky od správných, takže vyčísluje pouze množství oprav.

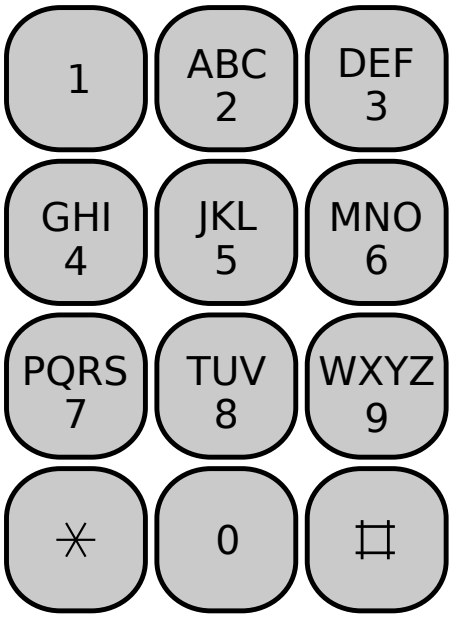
Dalším způsobem, jak změřit chybovost, s jakou uživatel text zadal, je vyhodnocení Levenshteinovy vzdálenosti originálního a přepsaného řetězce. (Soukoreff a MacKenzie, 2001)

Kapitola 6

Rozšířené metody zadávání textu

6.1 Klávesnice ITU

Klávesnice ITU je textová klávesnice na tónovém telefonu, která má na jednu klávesu přiřazeno více písmen. V průběhu vývoje se vyskytovalo několik různých způsobů, jak byla písmena na klávesnici rozřazena, podle mezinárodního standardu ITU E.161/ISO 9995-8 ISO/IEC (2009) je rozložení následující:



Obrázek 2: Klávesnice ITU (ISO 9995-8)[16]

[16]převzato z Wikimedia Commons

6.1.1 Multitap

Na těchto klávesnicích musí uživatel stisknout klávesu s požadovaným písmenem tolikrát, jaké je pořadí daného písmena na klávese. Například na klávese 3 jsou písmena D, E, F, tudíž chce-li uživatel zadat písmeno F, musí klávesu 3 stisknout třikrát. Např. pro napsání slova nejlepsi_ tak je nutno stisknout 21 kláves:

nejlepsi: 66335N555337N7777444S.

Tento systém zadávání se nazývá multitap podle toho, že na jeden znak připadá několik úhozů. Při psaní tímto systémem může KSPC dosahovat hodnot přes 2, protože většina kláves musí být stisknuta vícekrát než jednou. (MacKenzie, 2002, s. 202)

Aby bylo možné zbavit se tohoto problému u klávesnic, které mají méně kláves, než je písmen v abecedě, je nutné nějakým způsobem disambiguovat vstup vzniklý tak, že uživatel stiskl klávesy, na nichž se vyskytovala potřebná písmena, vždy jen jednou. Termín predikce textu je v tomto případě poněkud nepřesný, ale je použitelný, protože systém na základě nejednoznačného vstupu předvídá, odhaduje, co uživatel chtěl napsat. Toto odhadování může být prováděno dvěma způsoby – na základě slovníku a na základě prefixů.

6.1.2 Slovníková disambiguace

Slovníková disambiguace umožňuje uživateli zadávat text jediným stiskem každé klávesy s požadovaným písmenem. Systém potom vyhledává ve slovníku záznamy korespondující se sekvencí stisknutých kláves, tzn. slova, která obsahují písmena ze stisknutých kláves. Pokud sekvenci stisknutých kláves odpovídá více slov, je uživateli prezentován seznam těchto slov seřazený obvykle podle frekvence výskytu.

Tento způsob disambiguace samozřejmě není dokonalý, protože mnoho různých slov sdílí stejné kombinace kláves. V takovém případě pak uživatel musí pomocí speciálních kláves požadované slovo vybrat (MacKenzie et al., 2001). Například kombinace kláves 7725 má v češtině možné min. tři interpretace: prak, pral, psal.

Slova, která se zapisují stiskem stejných kláves, se nazývají textonyma (Zorn, 2007) a v angličtině jsou jimi např. slova book a cool. Zatímco obvykle tento jev pouze snižuje efektivitu zápisu textu, v roce 2010 záměna dvou textonym v textové zprávě vedla ve Spojených státech k vraždě. (The Bolton News, 2011)

6.1.2.1 První disambiguační systém

První disambiguační systém byl určen pro lidi s částečně nebo zcela poškozeným sluchem. Tento systém, navržený v roce 1985, si kladl za cíl umožnit neslyšícím telefonní komunikaci, a to za přijatelnou pořizovací cenu (na rozdíl od předchozích existujících

řešení) a s tím, že speciální přístroj bude potřebovat pouze neslyšící strana. (Feinson, 1988)

Zpráva se neslyšící straně předává tak, že protistrana na svém přístroji stiskne pro každé písmeno ve zprávě klávesu, na níž se písmeno vyskytuje (a to pouze jednou), tyto stisky se převedou na tón o odpovídající frekvenci (podobně jako u tónové volby) a tón se přenese po telefonní lince neslyšící straně. Tam je sekvence tónů převedena pomocí přístroje vybaveného mikroprocesorem na sekvenci skupin znaků, které odpovídají daným klávesám, a provede se prohledání slovníku lokálně uloženého v přístroji. Nalezená shoda je pak uživateli zobrazena na obrazovce telefonního přístroje.

6.1.2.2 T9

T9 – akronym pro *Text on 9 keys* – je systém, který se společně s větším rozvojem mobilních telefonů s klasickými 12klávesovými klávesnicemi se stal všeobecně známým pojmem pro predikci textu. Byl vyvinut společností Tegic Communications, jež později přešla pod firmu Nuance Communications (Nuance Communications, 2007), a patentovaný ve Spojených státech (Grover et al., 1998).

Systém T9 umožňuje uživateli rozšiřovat vestavěný slovník tak, že uživatel nové slovo zadá klasickým způsobem multitap a nové slovo se pak uloží do uživatelské databáze (Nuance Communications, 2015c). Problém nastává ve chvíli, kdy uživatel zjistí, že slovo není ve slovníku, až poté, co zadá příslušnou sekvenci kláves, která by k onomu slovu měla vést. V tu chvíli totiž musí smazat zadaný řetězec, který je buď nesmyslný, nebo to není požadované slovo, a zadat slovo znova pomocí multitapu.

Celkově je však T9 velmi jednoduchá na pochopení a použití a pro její nezávislost na kontextu u ní lze očekávat vždy stejný výstup po stisknutí určité sekvence kláves, takže se zkušený uživatel nemusí při psaní dívat na klávesnici. (MacKay, 2015)

6.1.2.3 iTap

Firma Motorola vyvinula iTap jako systém konkurenční k systému T9. Přestože oba systémy využívají slovníkového modelu, iTap je na rozdíl od T9 kontextově závislý a jeho slovník obsahuje kromě jednotlivých slov i běžně užívané fráze.

MacKay (2015) dále uvádí, že ve srovnání se systémem T9 je systém iTap pro uživatele méně výhodný v tom smyslu, že jsou nuceni činit rozhodnutí ohledně toho, zda v určitém momentu zvolit nabízenou predikci, či pokračovat v manuálním psaní. Pro nezkušené uživatele navíc může u systému iTap být demotivující, když jim predikce nabídne slova, která nemínili napsat. Systém iTap je také celkově mírně náročnější na ovládání.

6.1.3 Prefixová disambiguace

6.1.3.1 LetterWise

LetterWise je systém prediktivního psaní vyvinutý a patentovaný firmou Eatoni Ergonomics. Tento systém nepracuje se slovníkem, ale s databází pravděpodobností prefixů pro každý používaný jazyk. Které písmeno ze stisknuté klávesy zvolit tedy systém vybírá porovnáním pravděpodobností, s jakými mohou písmena z dané klávesy následovat po předcházejících písmenech. V angličtině je například nepoměrně pravděpodobnější, že po sekvenci písmen T a H bude následovat při stisku klávesy 3 písmeno E, nikoliv D či F. Prediktivní systém LetterWise má tedy v paměti zařízení uloženy pouze tyto pravděpodobnosti, nikoliv celá slova. (MacKenzie et al., 2001) Díky tomu je paměťová náročnost systému LetterWise velmi nízká a navíc jeho účinnost nedegraduje se snižující se velikostí jazykového modelu tolik, jako např. u systému T9. Jazykový model pro šest jazyků (angličtina, dánština, italština, španělština, francouzština, němčina) zabíral na implementaci pro procesor Intel 80C51 pod 5 kB. (Eatoni Ergonomics, 2006)

Další, pro uživatele výrazně citelnější výhodou oproti slovníkovým systémům, je možnost relativně jednoduše zapisovat i neslovníková slova. Pokud se totiž uživatel pokouší zadat slovo, které by sice ve slovníku neexistovalo, ale má skladbu hlásek odpovídající jazyku, jehož databáze prefixů je aktivní, je pravděpodobné, že systém vybere správné písmeno. Není tedy nutné se při zápisu neslovníkových slov uchylovat k multitapu, pouze k jeho zmírněné verzi: čím odlišnější je slovo od aktivního jazykového modelu, tím častěji bude odhad písmene chybný a uživatel bude muset stisknout klávesu vícekrát, aby zvolil správné písmeno. Tento systém tedy uživatele nenutí v případě zjištění, že jím požadované slovo není ve slovníku, celé slovo smazat a zadat jej klasickým způsobem. (Ghayoomi a Momtazi, 2009, s. 5234; MacKenzie et al., 2001, s. 112)

6.1.3.2 WordWise

Méně známý systém od firmy Eatoni Ergonomics, WordWise, je podobný systému LetterWise. WordWise je však na rozdíl od systému LetterWise založený na doplňování slov ze slovníku, v čemž je podobnější systému T9. Od toho se ale liší tím, že se vrací k systému LetterWise, pokud se uživatel pokusí zadat slovo, které nemá WordWise ve slovníku. U LetterWise, jak je uvedeno výše, záleží počet nutných stisků kláves na podobnosti slova s právě používaným jazykovým modelem. (Eatoni Ergonomics, 2007)

Modifikací tohoto systému je také systém WordWise Shift, který využívá jednu klávesu na telefonu jako klávesu Shift, a na každé klávese se skupinou písmen je vybráno jedno písmeno, jež lze napsat pouze se stiskem klávesy Shift a dané klávesy. Toto má podle výrobce za cíl minimalizovat problémy s predikcí slov, která sdílejí stejnou kombinaci kláves. Společnost Eatoni na svých stránkách uvádí, že uživatel při používání tohoto systému nemusí vůbec sledovat display svého zařízení, a přesto může jím napsaný text

být zcela správně, protože chyba v predikcích slov nastává jednou za 440 slov. (Eatoni Ergonomics, 2007)

Paměťová náročnost jazykového modelu pro angličtinu je v aplikaci WordWise zhruba 75 kB. (Eatoni Ergonomics, 2007)

6.2 Klávesnice QWERTY na chytrých telefonech

S rozvojem chytrých telefonů s velkými dotykovými obrazovkami se do širšího používání dostaly plné klávesnice QWERTY. Tyto klávesnice, které se zobrazují na displeji zařízení, a nemají tudíž po hmatu rozlišitelné klávesy, se také nazývají soft keyboards nebo softwarové klávesnice.

Pokud by se účinnost takové klávesnice jako metody vstupu textu porovnávala pouze podle hodnoty KSPC, byly by plné klávesnice QWERTY na chytrých telefonech stejně účinné jako standardní klávesnice k počítačům. Realita je ovšem jiná, protože velikost klávesnic na chytrých telefonech je v řádu centimetrů namísto desítek centimetrů, a také proto, že na rozdíl od fyzické klávesnice nejsou jednotlivé klávesy na dotykovém displeji rozpoznatelné po hmatu. Zadávání textu na takové klávesnici proto vyžaduje ze strany uživatele větší zkušenosti a pozornost. Text navíc nemůže zadávat všemi deseti prsty, ale při nejlepším dvěma prsty, což rychlost psaní také rapidně snižuje. Nemožnost poznat hranice kláves po hmatu navíc vede k zásadnímu nárůstu chybovosti (překlepů), které je uživatel nucen opravovat. Ačkoliv je pravděpodobně nereálné snažit se bez empirického výzkumu zahrnout tyto faktory do teoretického výpočtu KSPC pro tyto klávesnice, lze logickým úsudkem dospět k tomu, že i pro tuto metodu vstupu je žádoucí zvýšit rychlost a účinnost zadávání nějakým typem predikce textu.

6.2.1 Barevné zvýrazňování kláves

Jedním z navrhovaných způsobů, jak pomoci uživatelům snížit množství překlepů při psaní na softwarových klávesnicích QWERTY, bylo barevně odlišovat nejpravděpodobnější následující klávesy (Gonçalves, 2015). Tento způsob byl navržen především pro starší uživatele, kteří jednak nemusejí být seznámeni s rozložením klávesnice QWERTY, a jednak nemají takovou přesnost motoriky a rychlost reakcí, jako uživatelé mladí.

Klávesy byly odlišeny různými odstíny šedé barvy, a to lineárně podle toho, jak pravděpodobné bylo dané písmeno v souvislosti s předchozím zadaným znakem. Vedle barevného odlišení pozadí klávesy byl ještě zvětšen popisek klávesy (obrázek 3).

Výsledky studie, která byla provedena na 15 ženách a 5 mužích, ovšem předpoklad, že zvýrazňování kláves starším uživatelům pomůže, popřely. Ukázalo se, že snaží-li se klávesnice vizuálně nějakým způsobem odlišit některé prvky a přitáhnout k nim pozornost uživatele, působí to na něj spíše rušivě a výrazně to snižuje rychlost zadávání textu. Jeden uživatel dokonce napsal místo slova, které zamýšlel, slovo, které mu nabízela pre-

Obrázek 3: QWERTY – barevné zvýrazňování

dikce. Funkčnost systému zvýrazňování navíc byla omezena tím, že starší uživatelé dělali v textu hodně chyb a systém pak nebyl schopen správně rozpoznat daná slova.

6.2.2 Predikce celých slov

Druhým navrhovaným způsobem ve výše zmíněné studii byla metoda, kterou používá v současné době většina klávesnic na chytrých telefonech.

Jde o systém, kde po zadání znaku či znaků je ze slovníku známých slov vybráno několik slov, která jsou v souvislosti s již zadanými znaky nejpravděpodobnějšími řetězci, jež uživatel chce zadat (obrázek 4). Tento seznam je obvykle nabízen ve formě horizontálního seznamu nad klávesnicí a uživatel může ušetřit několik stisků kláves tak, že vybere požadované slovo ještě před tím, než jej celé dopíše (za předpokladu, že jej systém našel ve svém slovníku a správně navrhl).

Obrázek 4: QWERTY – seznam predikcí

Oproti zvýrazňování jednotlivých nejpravděpodobnějších kláves má tento systém minimálně dvě výhody. Jednak nevyžaduje, aby si uživatel nestínil prsty, kterými píše, výhled na klávesnici, a jednak uživateli zobrazuje celá slova, ne pouze další znak z nich.

Nevýhodou této metody na druhou stranu je fakt, že je na uživatele kladena vyšší kognitivní zátěž v tom smyslu, že je nucen procházet zrakem seznam navrhovaných slov a přemýšlet, zda je mezi nimi i to, které zamýšlel napsat. Z tohoto důvodu je nutné délku seznamu návrhů omezit na optimum. Například Nevěřilová a Ulipová (2014) ve

své implementaci používají šest návrhů a implementace popisovaná v praktické části této práce používá pět návrhů.

6.2.3 Swype

Další systém zadávání textu, který je poněkud odlišný od všech dosud zmíněných, používají mj. aplikace Swype[17], Fleksy[18], oficiální klávesnice od Google, etc. Jde o systém původně vyvinutý firmou Swype Inc. určený výhradně pro dotyková zařízení. Uživatel v něm nezadává text ťukáním na jednotlivé klávesy, ale přejíždí prstem či stylem po klávesnici od prvního písmena zadávaného slova k poslednímu tak, aby na cestě přejel pokud možno přes všechna písmena obsažená ve slově. (Nuance Communications, 2015b)

Swype se skládá ze tří základních částí – analyzátoru trasy dotyku, systému na vyhledávání slov v databázi výrazů a uživatelského prostředí (Kushler a Marsden, 2006). V raných verzích databáze obsahovala 65 000 výrazů. (Needleman, 2008)

Nutno podotknout, že firma Nuance, jež systém Swype v současnosti vyvíjí, nezahrnuje do své klávesnice pouze tento systém zadávání, ale i klasickou metodu zadávání s nabídkou predikcí v horizontálním seznamu. Navzdory faktu, že hlavním důvodem popularity klávesnicové aplikace Swype je systém zadávání Swype, i klasická klávesnice v této aplikaci kombinuje různé technologie na zlepšení uživatelského komfortu. Mezi ně patří například schopnost rozšiřovat virtuální (nikoliv vizuální) velikost klávesy podle toho, jaké klávesy předcházely. Pokud uživatel tedy např. často klepne na klávesu N, smaže zadaný znak N a místo něj napíše B, pak se to klávesnice naučí a příště, pokud je klepnutí ne zcela jednoznačné, vybere pravděpodobnější klávesu, v tomto příkladu tedy B. (Nuance Communications, 2015b)

Swype také používá Andvance Language Model (AML), „pokročilý jazykový model“, který pracuje s bigramy a trigramy vytvořenými mimo jiné na základě textů, které už uživatel zadal. To aplikaci umožňuje účinněji disambiguovat slova, která vychází jako možná řešení uživatelova vstupu, či předvídat další slovo, které by mohlo přijít po již napsaném. Oficiální dokumentace uvádí, že díky AML aplikace dokáže na základě kontextu správně rozlišit slova, která mají stejnou trasu tahu, tedy v angličtině např. put, pit a pot. (Nuance Communications, 2015a)

6.2.4 Predikce na fyzických klávesnicích QWERTY

Jak bylo naznačeno v kapitolách výše, většina implementací predikce textu, automatického doplňování textu a automatických oprav založených na predikci textu je určena pro zařízení buď s nejednoznačnou klávesnicí, kde má predikce eliminovat nutnost opakovaných stisků stejné klávesy, nebo pro zařízení s dotykovou obrazovkou a softwarovou

[17] `http://www.swype.com/`
[18] `https://fleksy.com/`

klávesnicí, na nichž si systémy kladou za cíl zvýšit rychlost a snížit chybovost zadávání textu.

Existují ale případy, kdy je predikce textu vhodná i pro použití s klávesnicemi fyzickými. Lze usuzovat, že se tyto případy nebudou týkat uživatelů, kteří nemají fyzický nebo mentální handicap a jsou na fyzickou klávesnici QWERTY zvyklí, ale spíš uživatelů, kteří z nějakého důvodu nemohou zadávat text normální rychlostí. Takoví uživatelé pravděpodobně trpí nemocemi, jako je například mozková obrna (Gadomski, 2015, 1–2). Pro ně může predikce textu mít smysl v tom, že nemusí zadávat celá slova, ale stačí zadat jeho menší část a poté vybrat požadované slovo z nabídky předvídaných slov. To pro ně patrně bude jednodušší, než muset zadávat celá slova, ať už z důvodu neschopnosti dostatečně rychlého psaní, či problémům se správným psaním (Penfriend, 2014). Taková predikce textu může napomoci např. lepší sociální interakci těchto uživatelů, protože rychlost komunikace může být značnou překážkou v účinné výměně informací s protistranou, která takovým handicapem netrpí, a nemá tudíž například trpělivost čekat na odpověď v chatu nadstandardní dobu.

Část II

Praktická část

Kapitola 7

Analýza

Jak bylo naznačeno v úvodu, cílem této práce je implementovat tradiční, byť poněkud zjednodušený, model automatického doplňování textu pro uživatele s fyzickou klávesnicí QWERTY s tím, že na rozdíl od tradičních systémů popsaných v práci výše nebude mít tato aplikace lokální slovník nebo databázi n-gramů, ale veškeré operace produkující možná doplnění budou prováděny na vzdáleném serveru.

Základní výhodou tohoto modelu je fakt, že na vzdáleném serveru, na němž se provádí všechny operace spojené s analýzou vstupu a tvorbou predikcí, je možné skladovat výrazně vyšší množství jazykových dat, které slouží jako databáze známých slov.

Při umístění jazykových dat na vzdálený server je sice nutno počítat s neustálým připojením k internetové síti, na druhou stranu lze počítat s tím, že v implementaci určené pro klasické počítače, která je součástí této práce, by to neměl být omezující faktor.

7.1 Backend

Zde popisovaná aplikace je závislá na serverovém skriptu, který tvoří takzvaný backend, tedy aplikaci, která se stará o produkci doplnění. Tento backend byl, společně s předchozími verzemi popisované aplikace, vyvinut již dříve a zcela nezávisle na této práci. (Nevěřilová a Ulipová, 2014)

V této práci je tedy backendem skript naprogramovaný v dynamickém programovacím jazyce Python, který s webovou aplikací komunikuje pomocí protokolu CGI.

Jako zdrojová data pro skript produkující doplnění slouží předem vygenerované n-gramy o délce 3 až 12 tokenů. Tyto n-gramy pochází z nejrozsáhlejšího českého korpusu czTenTen, který je kolekcí textů z různých webových zdrojů (Suchomel, 2012). Texty v korpusu neprošly korekturami, takže nemusí nutně obsahovat pouze spisovnou češtinu, což může být závažným problémem, pokud by aplikace pro predikci textu měla být využívána pro kontrolu správnosti psaní. Vzhledem k tomu, že to ale není primární

cíl a ani to přímo nesouvisí s cílem praktické části této práce, nebyl na tuto skutečnost během vzniku zde popisované aplikace brán zřetel. Nicméně v případě, že by byl zájem aplikaci užívat za takovým účelem, stačí vyměnit korpus, z něhož se generují predikce, za takový, který bude vyhovovat potřebám užití.

Vygenerované n-gramy jsou vyfiltrovány podle různých kritérií s cílem vytvořit databázi n-gramů, která bude mít přijatelné pokrytí textů korpusu se zachováním rozumné datové velikosti databáze. Pro uložení výsledné databáze jsou pak použity konečné automaty. (Nevěřilová a Ulipová, 2014, s. 13)

Vstupem backendového skriptu je posledních několik slov (záleží na konfiguraci aplikace) z textového editoru na frontendu. Serverový skript pracuje v základu ve dvou režimech, a to v závislosti na tom, zda vstup, který dostává, končí mezerou, či nikoliv. Pokud končí mezerou, předpokládá se, uživatel poslední slovo vstupu dokončil a chystá se psát další. Pokud mezera na konci vstupu není, je rozumné předpokládat, že uživatel slovo ještě nedokončil a chystá se v něm pokračovat. Podle tohoto jsou také rozdělené režimy, ve kterých serverový skript pracuje.

V prvním případě, tedy pokud na konci vstupu mezera je, jsou výstupem skriptu nejpravděpodobnější celá slova, která se v bigramech vyskytovala za slovem, které je posledním slovem vstupu. Pokud mezera na konci vstupu není, výstupem skriptu jsou slova, která se v bigramech vyskytovala na pozici za předposledním slovem vstupu a zároveň začínají řetězcem, který je shodný s posledním (nedokončeným) slovem na vstupu. Je důležité si povšimnout, že výstupem nejsou pouze doplnění posledního nedokončeného slova, ale celá slova obsahující i vstupní řetězec. Tímto se druhý režim odlišuje od prvního, který vrací doplnění, jež lze jednoduše připojit za text bez nutnosti odstraňovat část v editoru již existujícího textu.

7.1.1 Forma komunikace

Co se formy vstupu a výstupu týče, má tento serverový skript velmi jednoduché aplikační rozhraní. Vstupní řetězce přijímá přes požadavek HTTP, v němž jsou data přenášena metodou GET, tedy jako parametry adresy URL, například `http://nlp.fi.muni.cz/projekty/predictive/predict.py?input=sk%C3%A1kal+pes+`. Vstupem je v uvedeném příkladu tedy skákal_pes_ (podtržítka značí mezery). Jelikož posledním znakem je mezera, očekávaným výstupem budou slova, která lze doplnit za pes.

Skript pak vrací aplikaci, z níž byl požadavek odeslán, text s maximálně 13 doplněními, která jsou oddělená mezerou.

Díky tomuto jednoduchému konceptu komunikace tedy stačí, aby aplikace, která serverového skriptu využívá, byla schopna odesílat požadavky HTTP na konkrétní adresu URL a zpracovat prostý text.

7.2 Frontend

Frontend, neboli část aplikace, se kterou interaguje uživatel, byla v této práci pojata jako zásuvný modul do webového editoru CKEditor, což je vizuální editor HTML, jehož cílem je zjednodušit produkci obsahu pro webové stránky. Editor umožňuje pracovat s formátováním ve značkovacím jazyce HTML, který se používá pro tvorbu webových stránek a webového obsahu. Práce s formátováním je v CKEditoru založena na principu WYSIWYG[19], což znamená, že uživatel nemusí mít žádné znalosti HTML, protože pro aplikaci formátování lze použít vizuální prvky ve formě tlačítek a rozbalovacích menu, podobně jako v klasických kancelářských textových procesorech typu LibreOffice Writer nebo Microsoft Office Word.

CKEditor je open source aplikace, což znamená, že jeho zdrojový kód je veřejně dostupný a každý jej může libovolně používat, modifikovat a distribuovat za dodržení jistých podmínek. Výhodou CKEditoru je jeho modularita, která spočívá v možnosti rozšiřovat jej pomocí zásuvných modulů. Tvorba zásuvných modulů není vyhrazena pouze pro vybrané autory, ale díky otevřenosti kódu editoru se jí může zabývat takřka kdokoliv (CKSource, 2015). Proto je CKEditor vhodným výběrem pro tuto práci.

[19] what you see is what you get

Kapitola 8

Návrh

Jelikož uživatelské rozhraní zde popisované aplikace je určeno pro použití ve webovém prohlížeči, byly při vývoji využity výhradně webové technologie, tedy JavaScript v kombinaci s knihovnou Ajax, značkovací jazyk HTML a kaskádové styly (CSS).

8.1 JavaScript

JavaScript je dynamický, interpretovaný, objektově orientovaný programovací jazyk široce využívaný pro vývoj webových aplikací. Z hlediska komunikačního modelu klient-server se jedná o jazyk pro skriptování na straně klienta, takže se používá pro programování uživatelského rozhraní webových stránek. (Flanagan, 2006, s. 2–4)

Jeho vývoj, který probíhá od roku 1995 dodnes, začal ve společnosti Netscape Communications Corporation. V roce 1997 se JavaScript stal průmyslovým standardem poté, co společnost Ecma International vydala první standardizovanou verzi pojmenovanou ECMAScript. (Eich, 2011)

Ve zde popisovaném zásuvném modulu slouží JavaScript k obstarání hlavní logiky rozhraní, tedy extrakci textových dat editoru a jejich zpracování do formy vhodné k odeslání serverovému skriptu, sledování aktivity uživatele a případnému zajištění, že data budou odeslána. Po přijetí odpovědi serveru zpracovává JavaScript přijatý text, zajišťuje uživateli nabídku doplnění a jejich případné vložení na správné místo v editoru.

8.2 Ajax

Ajax je souhrnné označení webových technologií, které se používají pro komunikaci webových aplikací se serverem. Hlavním využitím Ajaxu bývají právě aplikace, které potřebují komunikovat se vzdáleným serverem bez nutnosti načítat celou webovou stránku znovu. (Garrett et al., 2005)

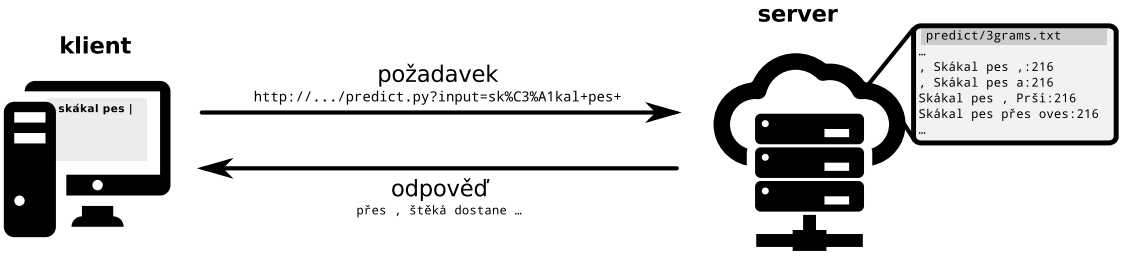
Asynchronní výměna dat se serverem probíhá pomocí aplikačního rozhraní XMLHttpRequest (XHR), které skriptovacím jazykům na straně klienta, jako je například

JavaScript, umožňuje odesílat data na vzdálený server. Po přijetí odpovědi se pak stará o dodání dat zpět do skriptu.

Pro komunikaci se serverem se využívá vzhledem k velmi malým objemům dat, které je nutno přenášet, prostého textu, a to ve dvou formách.

První z forem se používá při odesílání požadavku, kdy jsou data serveru dodávána v rámci URL jako parametry. Jedná se vždy o posledních několik slov (záleží na konfiguraci zásuvného modulu), která jsou zakódována globální funkcí JavaScriptu `encodeURI()`, aby byla zajištěna správná funkčnost i v případě výskytu zvláštních znaků, jako jsou například písmena s diakritikou nebo jakékoliv jiné znaky, které v adrese URL nemohou být přímo. Jednotlivá slova jsou oddělována pomocí znaku `+`, který v adrese URL může nahrazovat mezeru.

Druhá forma prostého textu se využívá v odpovědích serveru. Skript na požadavky odpovídá textovým řetězcem, který obsahuje navrhovaná slova oddělená mezerou. Tento textový řetězec je odesílán klientu jako dokument obsahující prostý text, který neobsahuje žádné přídavné strukturní nebo designové prvky, takže je opět velmi snadno zpracovatelný.



Obrázek 5: Schéma komunikace klienta a serveru[20]

8.3 jQuery

Knihovna jQuery[21] je navržená pro usnadnění programovaní v jazyce JavaScript. Zaměřuje se především na zjednodušení zpracování prvků v HTML dokumentu, práci s událostmi (např. stisk klávesy) a komunikaci mezi klientem a serverem pomocí Ajaxu.

8.4 HTML

HTML je standardní značkovací jazyk velice podobný jazyku XML. Je hojně používaný při tvorbě elektronických dokumentů pro webové prohlížeče, jako je kupříkladu Mozilla

[20] schéma obsahuje ikony od Freepik vytvořené pod licencí Creative Commons BY 3.0
[21] `https://jquery.com/`

Firefox[22]. Webové prohlížeče jsou schopny HTML zpracovat a zobrazit uživateli výslednou stránku, která obsahuje vizuální prvky a textové formátování. (Raggett et al., 1999, s. 19–22)

Standard HTML vychází ze obecného značkovacího jazyka SGML (Standard Generalized Markup Language), který je založen na dvou principech, a to že by značkovací jazyk měl být deklarativní, tedy popisovat strukturu a další atributy, spíše než definovat způsob, jakým je dokument zpracováván, a že by definice v dokumentu měly být jednoznačné a pevně definované, aby byl dokument zpracovatelný stejně jako například programový kód. (Rubinsky, 1990)

HTML popisuje strukturu stránky po sémantické stránce a definuje (částečně či úplně) její vzhled. Rozdíl mezi sémantickou informací a informací o tom, jak by měla prezentace vypadat, lze ilustrovat na dvojici značek, jejichž výsledná prezentace je zpravidla identická, a to `<strong>` a `<b>`. Značka `<strong>` webovému prohlížeči říká, že text uvnitř této značky je důležitější než okolní text a měl by podle toho také být naformátován (tedy zpravidla tučným řezem písma), zatímco značka `<b>` pouze indikuje, že by text v ní uzavřený měl být naformátován tučně.

Zde popisovaná aplikace využívá HTML relativně málo, protože se jedná o zásuvný modul, který ze své podstaty nesmí ovlivňovat vzhled stránky, do níž si uživatel vloží CKEditor s tímto modulem. V rámci editačního pole se však vyskytla potřeba HTML použít pro zobrazení nabídky s výběrem jednotlivých možných doplnění, z nichž si uživatel může vybrat. Jakkoliv je tedy co do množství kódu HTML zastoupeno nejmenší částí ze zde popisovaných technologií, je jeho použití důležité pro komfort koncového uživatele.

8.5 CSS

Druhou technologií, která je relativně málo využita, ale pro koncového uživatele je zásadní z hlediska uživatelské přívětivosti rozhraní, jsou kaskádové styly. Ty jsou také jednou ze základních technologií používaných pro webové prezentace a zajišťují definice vizuálního vzhledu stránek napsaných ve značkovacím jazyce. (Meyer, 2004)

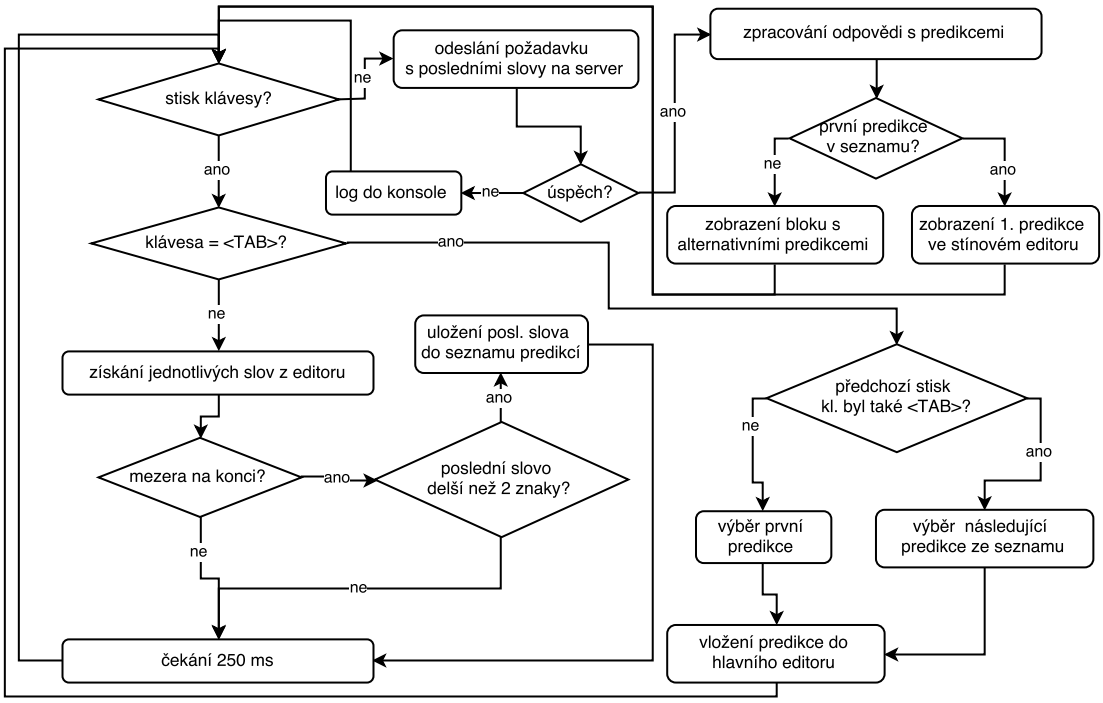
Hlavním smyslem CSS je oddělení obsahu webové stránky od její vizuální prezentace. To je důležité především pro snazší udržování stránky a větší flexibilitu. Použití CSS navíc umožňuje zobrazit jednu stránku různými způsoby, jedním z nichž může být například možnost zobrazení pro barvoslepé uživatele.

Ve zde popisované aplikaci je tedy CSS využíváno k definici vizuální prezentace nabídky doplnění.

[22] `https://www.mozilla.org/en-US/firefox/desktop/`

Kapitola 9

Implementace



Obrázek 6: Vývojový diagram pluginu

9.1 Uživatelské rozhraní

Mluvit o uživatelském rozhraní jako celku je v tomto případě poněkud problematické, protože jde o zásuvný modul a není tudíž možné předvídat, jak přesně bude koncový uživatel modulu mít CKEditor nakonfigurován. V této práci však postačí uvažovat základní

konfiguraci CKEditoru tak, jak je poskytována na oficiálních webových stránkách[23]. Tam jsou ke stažení tři předpřipravené balíčky, jež se liší množstvím zásuvných modulů, které jsou obsahují. Pro účely vývoje zde popisovaného pluginu i pro účely samotného popisu byl vybrán nejmenší balíček *Basic Package*, který obsahuje 17 zásuvných modulů obstarávajících základní funkcionalitu. V popisu práce s editorem pak bude uvažována webová stránka, na níž se bude nacházet pouze CKEditor v konfiguraci *Basic Package* s přidaným pluginem, který je předmětem této práce.

Práce v takto nakonfigurovaném editoru probíhá tak, že uživatel zadává text do editoru pomocí klasické klávesnice a vybírá (případně nevybírá) si doplnění, která jsou mu nabízena. Doplnění jsou nabízena vždy, když uživatel na určitou dobu přestane vyvíjet aktivitu (pro účely testování se jako vhodné ukázalo být 250 ms). Aktivitou se myslí události `keydown` a `keyup`, které reprezentují stisky kláves – konkrétně stisknutí klávesy (`keydown`) a její uvolnění (`keyup`). Nabídka se také nezobrazuje, když pro danou kombinaci slov a písmen před pozicí kursoru nejsou nalezena doplnění. Ve chvíli, kdy si uživatel některé doplnění z nabídky vybere, skript jej v závislosti na situaci vhodným způsobem doplní do editoru a posune na její místo kursor. Pokud uživatel za doplněné slovo vloží mezeru a poté nevyvine žádnou další aktivitu po 250 ms, zobrazí se mu další doplnění zvolené na základě předchozího slova zadaného (či vybraného) slova. V ideálním případě by tak mělo na základě prvního slova být možné doplnit určité standardní věty pouhým vybíráním vhodných predikcí. Tomu ovšem zabraňuje nastavení aplikace, které filtruje z nabídky predikcí příliš krátká slova (méně než tři znaky), protože ta většinou bývá rychlejší napsat celá manuálně.

9.2 Zpracování textu v editoru

Aby mohla být serverovému skriptu odeslána data, na jejichž základě budou vyhledána možná doplnění, musí být na straně klienta zpracován text, který uživatel napsal do editoru. To probíhá v několika krocích, jejichž výsledkem je řetězec předem definovaného počtu posledních slov v editoru, který je vhodný k odeslání na server.

Všechny operace spojené s modifikacemi textu v editoru i prezentací nabídky predikcí jsou prováděny pouze v případě aktivity uživatele.

Prvním krokem je rozdělení obsahu editoru na jednotlivá slova. To zajišťuje funkce `sliceContent()`, jež přijímá text editoru ve formě řetězce. Text rozděluje postupně na řádky podle tagu `<br>`, kterým končí v CKEditoru každý řádek, a dále zpracovává pouze poslední řádek. To znemožňuje používat predikci na jiném než posledním řádku, což ale nevadí, protože momentálně plugin nepodporuje práci s predikcemi na jiných místech než na úplném konci vkládaného textu. To je jeden ze známých nedostatků, který bude rozebrán v kapitole 10.

[23] `http://ckeditor.com/`

Následně funkce odstraní tagy HTML, nahradí nedělitelné mezery normálními a rozdělí zpracovávaný řádek na jednotlivá slova. Jako datový typ pro uložení řetězce rozděleného na slova bylo zvoleno pole (array), jež má v jazyce JavaScript připraveno velké množství metod vhodných pro tuto aplikaci.

Jakmile má skript k dispozici slova z aktuálního (posledního) řádku v editoru, se v případě, že současné slovo u kurzoru má délku alespoň tři znaky, zjišťuje pomocí funkce `findLastSimiliarWord()`, zda se v editoru už nevyskytuje slovo, které začíná na řetězec shodný se současným slovem a má délku alespoň pět znaků. Pokud ano, přidá se do seznamu predikcí, který později bude uživateli nabídnut. Tento krok se provádí proto, že lze předpokládat, že pokud uživatel napsal jednou nějaké delší slovo a znova začal psát slovo, které začíná stejnými znaky, bude chtít napsat téže slovo. Jde o funkcionalitu podobou té z textového procesoru LibreOffice Writer (obrázek 1).

Po tomto kroku funkce `getSuggestions()` provádí ajaxové volání na server za účelem získání predikcí vytvořených z korpusových dat. Tato funkce přijímá několik hodnot, a to pole se slovy z posledního řádku, počet slov, která mají být odeslána na server, a dále callback na funkci `handleSuggestion()`, která zpracovává získaná doplnění.

Ve funkci `getSuggestions()` proběhne připravení řetězce se slovy, která mají být odeslána na server. To znamená, že funkce metodou `slice()` vyřízne z pole posledních N slov, kde N je počet slov odesílaných na server a výsledné pole N prvků spojí do řetězce s tím, že mezi jednotlivé prvky (slova) vloží znak `+` (plus). Znak `+` byl zvolen proto, že slova jsou odesílána, jak již bylo popsáno v kapitole 8, metodou GET, tedy jako parametry adresy URL. V rámci dat odesílaných metodou GET lze reprezentovat mezeru buď znakem `+` nebo řetězcem `%20`, přičemž tyto způsoby jsou v praxi rovnocenné.

Následně funkce čeká předem nastavenou dobu (250 ms), zda nenastane další aktivita uživatele, a v případě, že nenastane, odešle ajaxový požadavek na server.

V případě, že ajaxové volání na server uspělo (návratový kód `202 Success`), zavolá se pomocí callbacku, který byl funkci `getSuggestions()` předán dříve, funkce `handleSuggestion()`. Pokud volání z jakéhokoliv důvodu skončilo chybou (kupříkladu nedostupnost server způsobí, že volání skončí návratovým kódem `404 Not Found`), vypíše se chyba do konzole a z pohledu uživatele se nic nestane.

Funkce `handleSuggestion()` přijímá jako argumenty mj. současné slovo pod kursorem, jeho případné doplnění z existujícího textu v editoru a odpověď serveru. Odpověď serveru je řetězec znaků, který obsahuje slova oddělená od sebe znakem mezery. Funkce odpověď rozdělí na jednotlivá slova a vyřadí slova kratší než 3 znaky včetně (z důvodů uvedených v kapitole 9.1). Pokud byla dříve nalezena predikce z textu, je doplněním ze serveru předřazena, aby uživateli byla prezentována jako první. Poté jsou predikce nabídnuty uživateli.

9.3 Zobrazení predikcí

Nabídka predikcí sestává ze dvou různých částí. První z nich je predikce (jíž může být celé slovo po mezeře, či část slova doplňující část již zadanou v editoru), která se zobrazuje méně výrazně (šedě) ihned za zadávaným textem v hlavním editoru (obrázek 7). Takové doplnění lze přijmout pouhým stiskem tabulátoru. Tento způsob zobrazení doplnění je inspirován řešením zobrazování automatického doplňování ve vyhledávači Google, které funguje na podobném principu.

Pro toto řešení je využíváno tzv. stínového editoru, který je pomocí CSS umístěn přesně pod hlavní editor, v němž uživatel zadává text. Do něj se pomocí JavaScriptové funkce `copyContent()` ve zde popisovaném zásuvném modulu kopíruje text z hlavního editoru. Pro zobrazení predikce je pak využíváno toho, že text zkopírovaný z hlavního editoru má ve stínovém editoru identické prostorové vlastnosti jako text hlavního editoru a na jeho konec lze snadno připojit aktuální predikci, která se tak zobrazuje uživateli fontem stínového editoru (ve zde popisované implementaci má stínový editor šedou barvu písma a hlavní editor černou).

Obrázek 7: Zobrazení predikcí

9.3.1 Alternativní predikce

Vzhledem k tomu, že predikce nejsou vždy dokonalé nebo není možné jednoznačně rozhodnout, které slovo uživatel zamýšlí napsat, ukázalo se nutným zobrazovat i alternativní predikce.

Tyto alternativní predikce jsou prezentovány pod pozicí kursoru ve formě seznamu. Pokud chce uživatel využít některou predikci zobrazenou v seznamu, stačí mu opětovně stisknout klávesu tabulátor, pomocí níž lze v seznamu jednosměrně přecházet mezi položkami. Aktuální položka se pak zároveň kopíruje do hlavního editoru, takže po dosažení požadované predikce uživatel vloží mezeru a může pokračovat v psaní. Tato metoda volby byla navržena především s ohledem na původní cíl aplikace, tedy usnadnit psaní uživatelům, kteří z mají z nějakého důvodu problém zadávat text na klávesnici QWERTY. Cílem tedy není zrychlení psaní, jako tomu bývá u implementací predikce textu například pro dotyková zařízení, ale co nejvíce snížit množství stisků různých kláves pro volby predikce.

Nabídka alternativních predikcí je zobrazována vždy pod kursorem, aby uživatel nebyl nucen dělit svou pozornost na různá místa obrazovky. O zobrazení nabídky v místě

kursoru se stará funkce `showAlternatives()`, která je závislá na funkci `getSelectionCoords()`. Z pohledu problematiky implementace je zajímavá především funkce `getSelectionCoords()`, která pomocí zjišťování pozice aktuálního výběru[24] textu získává absolutní pozici kursoru na obrazovce, jež je nutná pro správné zobrazení alternativ. Implementace výběrů se může lišit podle konkrétního prohlížeče, avšak obecně lze říci, že aktuální výběr vždy končí v místě, kde je právě umístěn kursor. S ohledem na tuto skutečnost a na to, že API (application program interface) moderních prohlížečů umožňují získat pozice začátku a konce výběru, lze pozici konce aktuálního výběru považovat za pozici kursoru, na níž se má zobrazit nabídka alternativních doplnění.

9.4 Vkládání predikcí

Poté, co si uživatel vybere predikci tím, že stiskne klávesu tabulátor, zbývá ji vložit na správné místo, tedy na pozici kursoru.

V případě vkládání celého slova, nikoliv doplnění zbytku již zadaného slova, je vložení predikce relativně triviální. Jde pouze o připojení dané predikce za pozici kursoru, přidání mezery a přesun kursoru za ni. Jak bylo naznačeno výše, pokud se uživatel rozhodne cyklovat v seznamu alternativních predikcí, vkládají se tyto do hlavního editoru automaticky pomocí funkce `printCompletions()`. Chce-li uživatel využít první predikce, použije se při stisku klávesy tabulátor stejná funkce.

V případě doplňování pouhé části slova jde vzhledem k tomu, že serverový skript ve všech případech vrací celá slova, o úkon, který představuje zásah do již existujícího textu v hlavním editoru. Pokud tedy uživatel napíše jako poslední slovo řetězec `p` a předcházející slovo je `vážený`, navrácená slova jsou `paciente`, `pane`, etc. Je tedy zřejmé, že nelze zvolenou predikci vložit přímo za existující text, ale je nutné nějakým způsobem zabránit duplikaci částí slov. Tento problém je řešen ve funkci `insertCompletion()` smazáním již napsané části slova z predikce, jejíž zbytek je pak vložen za napsaný text. Do hlavního editoru se tedy nevkládá celá predikce `paciente`, nýbrž pouze její zbývající část po odtržení již napsané části, tedy například `aciente`.

Úplně poslední úkon, který se provádí, je přesunutí kursoru na korektní pozici poté, co byla predikce vložena do hlavního editoru. Je evidentní, že kursor nemůže zůstat na původním místě, na němž byl před vložením predikce, protože pak by uživatel buď psal další text před vloženou predikci, nebo by musel manuálně přejít na konec vkládaného textu.

[24] více o výběrech v kapitole 9.4.0.1

9.4.0.1 Výběry

Manipulace s kursorem v textu se v prohlížečích provádí pomocí výběrů. Výběr je objekt[25], který reprezentuje text označený v textovém poli, přičemž může mít i nulovou délku v případě, že není označen žádný text. Výběr má různé vlastnosti včetně pozice, s nimiž lze manipulovat (Mozilla Developer Network, 2015). Pozice výběru může být vyjádřena dvěma způsoby, a to jako absolutní pozicí na obrazovce, tedy pixelovými souřadnicemi, nebo jako pozice v textu, tedy počet znaků relativní k určitému místu v textu.

Přesun kursoru je tedy řešitelný přemístěním výběru v textu o délku predikce dále, ale v praxi se ukazuje, že existuje nezanedbatelné množství případů, kdy zjišťování pozice výběru v textu není přesné, a tudíž výsledná pozice kursoru nemusí být správná. Z toho pramení i nedokonalosti v implementaci popsané v kapitole 10.

[25] míněno jako objekt v objektově orientovaném programování

Kapitola 10

Vyhodnocení

Z testování během vývoje zásuvného modulu, který je předmětem této práce, lze potvrdit, že jeho použití šetří počet úhozů nutný k napsání textu, takže cíl práce byl splněn. Vzhledem k tomu, že se zásuvný modul zatím nedočkal nasazení v reálných situacích, nelze jeho skutečný přínos uživatelům hodnotit.

Množství ušetřených úhozů značně závisí na povaze textu, který uživatel píše. V rámci vývoje bylo provedeno krátké manuální testování na odlišných textech, které ukázalo, že kvalita predikcí je velmi závislá na podobnosti zdrojového korpusu a psaného textu. Vzhledem k tomu, že při testování byly použity n-gramy z korpusu czTenTen12, nejlepších výsledků dosahoval systém na textech ze zpravodajských serverů, kde bylo dosaženo KSPC cca 0,4. Naopak v případě, že text obsahoval například osobní jména, byla účinnost predikce výrazně omezena. Nejhoršího výsledku bylo dosaženo na textu mírně technického rázu (manuál k instalaci počítačového programu), a to KSPC cca 0,7. Do úhozů byly počítány všechny klávesy.

Nutno podotknout, že se v predikcích mnohdy vyskytovalo správné slovo, ale bylo ve špatném tvaru. Je tedy zřejmé, že by predikci značně vylepšila syntaktická analýza, případně kdyby doplňování probíhalo po částech, tedy nejdříve kmen slova a poté jeho koncovka.

Další problémy, které potenciální uživatel zaznamená, jsou technické nedostatky frontendu. Hlavním z nich je ten, že implementace momentálně neumožňuje práci vprostřed těla textu, ale pouze na jeho konci. Není tak možné mít text, do jehož části chce uživatel napsat odstavec s využitím predikce. Zde je tedy značný prostor pro zlepšení.

Technicky příbuzný problém je pak omezená podpora formátovaného textu. Umístění predikcí a pozice kursoru mohou být při výskytu formátovaného textu v editoru nepřesné. To je vzhledem k faktu, že CKEditor je editor typu WYSIWYG, další limitace, kterou je před reálným nasazením nutno vyřešit.

Celkově ale je implementace frontendu pro psaní s výrazně sníženým průměrným množstvím úhozů na znak použitelná již nyní.

Závěr

Byla vyvinuta aplikace, která umožňuje zadávání textu bez nutnosti stisku všech kláves korespondujících s písmeny ve slovech zadávaného textu. Pokud uživatel chce zadat text, jehož slova existují v korpusu, z něhož jsou vybírány predikce, má možnost z těchto predikcí vybrat vhodně ty, které odpovídají jeho záměru, a snížit tak počet úhozů na znak na hodnotu menší než jedna. Vzhledem k velikosti korpusu, z něhož jsou predikce vybírány, je rozumné předkládat, že takového výsledku lze dosáhnout téměř s libovolným textem v českém jazyce, což prokázaly i krátké testy. Současná implementace navíc umožňuje snížit počet stisknutých kláves na počet znaků v textu také tím, že mezi predikce zařazuje již zadaná delší slova. Tato funkcionalita může být užitečná například pro uživatele, kteří zadávají text s větším množstvím dlouhých odborných termínů (dobrým příkladem budiž lékařské texty).

V dalším vývoji by bylo vhodné zaměřit se především na problém s tvary slov, jehož řešením může být buď syntaktická analýza nebo nabízení predikcí po částech, tedy oddělit kmen a koncovku. Tímto krokem by bylo možné dále snížit hodnotu KSPC a navíc by se tím eliminoval problém frustrace uživatele, který, vidí-li požadované slovo ve špatném tvaru, musí přemýšlet, zda je výhodnější predikci v chybném tvaru zvolit a opravit koncovku, nebo celé slovo napsat bez využití predikce.

Seznam literatury

ALDERSON, J. Charles, 2007. Judging the frequency of English words. *Applied Linguistics* [online], 28(3): s. 383–409, [cit. 2015-10-09]. Dostupné z: ⟨http://applij.oxfordjournals.org/content/28/3/383.short⟩.

ARIF, Ahmed S. a STUERZLINGER, Wolfgang, 2009. Analysis of text entry performance metrics. In: *2009 IEEE Toronto International Conference Science and Technology for Humanity (TIC-STH)* [online]. Toronto: IEEE, s. 100–105, [cit. 2015-09-29]. Dostupné z: ⟨http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5444533⟩.

CKSOURCE, 2015. *CKEditor – About* [online]. [Cit. 2015-10-25]. Dostupné z: ⟨http://ckeditor.com/about⟩.

ČERMÁK, František a BLATNÁ, Renata, 2004. *Frekvenční slovník češtiny*. Praha: Nakladatelství lidové Noviny.

DAVIES, Mark a GARDNER, Dee, 2010. *A frequency dictionary of contemporary American English: word sketches, collocates, and thematic lists*. New York: Routledge.

EATONI ERGONOMICS, 2006. *LetterWise Requirements and Performance* [online]. [Cit. 2015-10-22]. Dostupné z: ⟨http://ns2.eatoni.com/wiki/index.php/LetterWise_Requirements_and_Performance⟩.

EATONI ERGONOMICS, 2007. *WordWise* [online]. [Cit. 2015-10-22]. Dostupné z: ⟨http://ns2.eatoni.com/wiki/index.php/WordWise⟩.

EICH, Brendan, 2011. *New JavaScript Engine Module Owner* [online]. [Cit. 2015-10-25]. Dostupné z: ⟨https://brendaneich.com/2011/06/new-javascript-engine-module-owner/⟩.

FEINSON, Roy W., 1988. *Interpretive tone telecommunication method and apparatus* [online]. US Patent 4,754,474. Dostupné z: ⟨https://www.google.com/patents/US4754474⟩.

FLANAGAN, David, 2006. *JavaScript: the definitive guide*. Sebastopol, CA: O'Reilly Media. Dostupné také z: ⟨https://books.google.cz/books?id=k0CbAgAAQBAJ⟩.

GADOMSKI, Phil. *Predictive Text Technology* [online]. [Cit. 2015-10-22]. Dostupné z: ⟨http://www.ece.gatech.edu/academic/courses/ece4007/09fall/ece4007l01/ws1/trp2.pdf⟩.

GARRETT, Jesse J. et al., 2005. *Ajax: A new approach to web applications* [online]. [Cit. 2015-10-25]. Dostupné z: ⟨https://courses.cs.washington.edu/courses/cse490h/07sp/readings/ajax_adaptive_path.pdf⟩.

GHAYOOMI, Masood a MOMTAZI, Saeedeh, 2009. An overview on the existing language models for prediction systems as writing assistant tools. In: *Proceedings 2009 IEEE International Conference on Systems, Man and Cybernetics* [online]. San Antonio, TX: IEEE, s. 5233–5237, [cit. 2015-10-24]. Dostupné z: ⟨http://hpsg.fu-berlin.de/~ghayoomi/Publishedpapers/ghayoomi-2009[3].pdf⟩.

GONÇALVES, Daniel. *Accessible Virtual Keyboard for Seniors* [online]. [Cit. 2015-10-22]. Dostupné z: ⟨http://web.ist.utl.pt/~daniel.j.goncalves/research/paelife_textentry/paelife_textentry.html⟩.

GOOGLE, 2015. *Automatické doplňování* [online]. [Cit. 2015-10-23]. Dostupné z: ⟨https://support.google.com/websearch/answer/106230?hl=cs%5C&vid=1-635809595515389355-3739668669⟩.

GROVER, Dale L. et al., 1998. *Reduced keyboard disambiguating computer* [online]. US Patent 5,818,437. Dostupné z: ⟨https://www.google.com/patents/US5818437⟩.

HENRY, Alan, 2014. How Predictive Keyboards Work (and How You Can Train Yours Better). *Lifehacker* [online], [cit. 2015-10-23]. Dostupné z: ⟨http://lifehacker.com/how-predictive-keyboards-work-and-how-you-can-train-yo-1643795640⟩.

ISO/IEC, 2009. *ISO/IEC 9995-8:2009: Information technology – Keyboard layouts for text and office systems – Part 8: Allocation of letters to the keys of a numeric keypad* [online]. [Cit. 2015-11-15]. Dostupné z: ⟨http://www.iso.org/iso/catalogue_detail?csnumber=51641⟩.

KOCIENDA, Kenneth et al., 2012. *Method, device, and graphical user interface providing word recommendations for text input* [online]. US Patent 8,232,973. Dostupné z: ⟨https://www.google.com/patents/US8232973⟩.

KUSHLER, Clifford A. a MARSDEN, Randal J., 2006. *System and method for continuous stroke word-based text input* [online]. US Patent 7,098,896. Dostupné z: ⟨https://www.google.com/patents/us7098896⟩.

LEVENSHTEIN, Vladimir I., 1965. Dvoičnyje kody s ispravlenijem vypaděnij, vstavok i zameščenij simvolov. *Doklady Akaděmij Nauk SSSR*, 163(4): s. 845–848.

LEVENSHTEIN, Vladimir I., 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10(8): s. 707. [Překlad Levenshtein, 1965]. Dostupné také z: ⟨http://adsabs.harvard.edu/abs/1966SPhD...10..707L⟩.

LORGE, Irving, 1944. *The teacher's word book of 30,000 words*. New York: Teachers College, Columbia University.

MACKAY, David. *Some notes on iTap / T9 / predictive text* [online]. [Cit. 2015-10-22]. Dostupné z: ⟨http://www.inference.phy.cam.ac.uk/itprnn/itap/⟩.

MACKENZIE, Scott I., 2002. KSPC (keystrokes per character) as a characteristic of text entry techniques. In: *Proceedings of the 4th International Symposium on Mobile Human-Computer Interaction*. Londýn: Springer-Verlag, s. 195–210. Dostupné také z: ⟨http://www.yorku.ca/mack/hcimobile02.PDF⟩.

MACKENZIE, Scott I. et al., 2001. LetterWise: prefix-based disambiguation for mobile text input. In: *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technologye* [online]. New York, NY: ACM, s. 111–120, [cit. 2015-10-22]. Dostupné z: ⟨http://www.yorku.ca/mack/uist01.pdf⟩.

MEYER, Eric A., 2004. *Cascading style sheets: The definitive guide*. Sebastopol, CA: O'Reilly Media. Dostupné také z: ⟨https://books.google.cz/books?id=nU4EiJ9ZPf8C⟩.

MOZILLA DEVELOPER NETWORK, 2015. *Selection* [online]. [Cit. 2015-10-22]. Dostupné z: ⟨https://developer.mozilla.org/en-US/docs/Web/API/Selection⟩.

NANDI, Arnab a JAGADISH, Hosagrahar V., 2007. Assisted querying using instant-response interfaces. In: *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*. New York, NY: ACM, s. 1156–1158. Dostupné také z: ⟨http://www.eecs.umich.edu/db/files/prompt07.pdf⟩.

NATION, Paul a WARING, Robert, 1997. Vocabulary size, text coverage and word lists. *Vocabulary: Description, acquisition and pedagogy* [online], s. 6–19, [cit. 2015-10-22]. Dostupné z: ⟨http://www.fltr.ucl.ac.be/fltr/germ/etan/bibs/vocab/cup.html⟩.

NEEDLEMAN, Rafe, 2008. *Move over T9, here comes Swype* [online]. [Cit. 2015-10-22]. Dostupné z: ⟨http://www.cnet.com/news/move-over-t9-here-comes-swype/⟩.

NÉMETH, Géza a ZAINKÓ, Csaba, 2001. Word unit based multilingual comparative analysis of text corpora. In: DALSGAARD, Paul et al. (ed.). *EUROSPEECH 2001 Scandinavia* [online]. Aalborg: ISCA, s. 2035–2038, [cit. 2015-10-22]. Dostupné z: ⟨http://perso.telecom-paristech.fr/~chollet/Biblio/Congres/Audio/Eurospeech01/CDROM/papers/page2035.pdf⟩.

NEVĚŘILOVÁ, Zuzana a ULIPOVÁ, Barbora, 2014. A System for Predictive Writing. In: HORÁK, Aleš a RYCHLÝ, Pavel (ed.). *8th Workshop on Recent Advances in Slavonic Natural Language Processing* [online]. Brno: NLP Consulting, s. 11–18, [cit. 2015-10-22]. Dostupné z: ⟨http://nlp.fi.muni.cz/raslan/raslan14.pdf⟩.

NORVIG, Peter. *How to Write a Spelling Corrector* [online]. [Cit. 2015-10-23]. Dostupné z: ⟨http://norvig.com/spell-correct.html⟩.

NUANCE COMMUNICATIONS, a. *Swype – How To* [online]. [Cit. 2015-10-22]. Dostupné z: ⟨http://www.nuance.com/ucmprod/groups/corporatecomms/@web-enus/documents/webasset/nc_024737.pdf⟩.

NUANCE COMMUNICATIONS, 2007. *Nuance Closes Acquisition of Tegic Communications* [online]. [Cit. 2015-10-24]. Dostupné z: ⟨http://www.nuance.com/news/pressreleases/2007/20070824_tegic.asp⟩.

NUANCE COMMUNICATIONS, 2015b. *Swype – Android Features* [online]. [Cit. 2015-10-24]. Dostupné z: ⟨http://www.swype.com/product-features/android/features.html⟩.

NUANCE COMMUNICATIONS, 2015c. *T9® – The Global Standard for Mobile Text Input* [online]. [Cit. 2015-10-24]. Dostupné z: ⟨http://www.nuance.com/for-business/by-product/t9/index.htm⟩.

PENFRIEND, 2014. *Who can Penfriend benefit?* [online]. [Cit. 2015-10-22]. Dostupné z: ⟨http://www.penfriend.biz/benefits.html⟩.

RAGGETT, Dave et al., 1999. HTML 4.01 Specification. *W3C recommendation* [online], 24 [cit. 2015-10-25]. Dostupné z: ⟨http://webx.ubi.pt/~hgil/utils/HTML/html40.pdf⟩.

RUBINSKY, Yuri, 1990. *The SGML Handbook*. New York: Oxford University Press. Dostupné také z: ⟨https://books.google.cz/books?id=RilvKya0EnwC⟩.

SILFVERBERG, Miika et al., 2000. Predicting text entry speed on mobile phones. In: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* [online]. Paříž: ACM, s. 9–16, [cit. 2015-10-22]. Dostupné z: ⟨http://www.yorku.ca/mack/p9-silfverberg.pdf⟩.

SKETCH ENGINE, 2015a. *Corpus czTenTen12 – statistics and info* [online]. [Cit. 2015-10-22]. Dostupné z: ⟨https://ske.fi.muni.cz/bonito/run.cgi/corp_info?corpname=preloaded/cztenten12_8⟩.

SKETCH ENGINE, 2015b. *Corpus enTenTen [2013] – statistics and info* [online]. [Cit. 2015-10-22]. Dostupné z: ⟨https://ske.fi.muni.cz/bonito/run.cgi/corp_info?corpname=preloaded/ententen13⟩.

SOUKOREFF, R. William a MACKENZIE, Scott I., 2001. Measuring errors in text entry tasks: an application of the Levenshtein string distance statistic. In: *CHI'01 extended abstracts on Human factors in computing systems* [online]. Seattle, WA: ACM, s. 319–320, [cit. 2015-10-22]. Dostupné z: ⟨http://www.yorku.ca/mack/CHI01a.PDF⟩.

SUCHOMEL, Vít, 2012. Recent Czech Web Corpora. In: HORÁK, Aleš a RYCHLÝ, Pavel (ed.). *6th Workshop on Recent Advances in Slavonic Natural Language Processing* [online]. Brno: NLP Consulting, s. 77–83, [cit. 2015-10-22]. Dostupné z: ⟨http://nlp.fi.muni.cz/raslan/raslan12.pdf⟩.

TARVAINEN, Jussi, 2010. Beginner Performance with the GKOS Chorded Keyboard. [online], s. 2–5, [cit. 2015-10-24]. Dostupné z: ⟨http://urn.fi/urn:nbn:fi:uta-1-20834⟩.

THE BOLTON NEWS, 2011. *Indefinite sentence for killing his friend* [online]. [Cit. 2015-10-24]. Dostupné z: ⟨http://www.theboltonnews.co.uk/news/8950811.Indefinite_sentence_for_killing_his_friend/⟩.

WARD, David et al., 2012. Autocomplete as research tool: A study on providing search suggestions. *Information Technology and Libraries* [online], 31(4): s. 6–19, [cit. 2015-10-22]. Dostupné z: ⟨http://ejournals.bc.edu/ojs/index.php/ital/article/view/1930⟩.

WOBBROCK, Jacob O., 2007. Measures of text entry performance. In: MACKENZIE, Scott I. a TANAKA-ISHII, K. (ed.). *Text Entry Systems: Mobility, Accessibility, Universality* [online]. San Francisco, CA: Morgan Kaufmann, s. 47–74, [cit. 2015-10-22]. Dostupné z: ⟨https://books.google.cz/books?id=XWSc3b_gkX8C⟩.

YAMADA, Hisao, 1980. A Historical Study of Typewriters and Typing Methods : from the Position of Planning Japanese Parallels. *Journal of Information Processing*, 2(4): s. 175–202. Dostupné také z: ⟨http://ci.nii.ac.jp/naid/110002673261/en/⟩.

ZORN, Eric, 2007. *Slang early-warning alert: 'Book' is the new 'cat's pajamas'* [online]. [Cit. 2015-10-24]. Dostupné z: ⟨http://blogs.chicagotribune.com/news_columnists_ezorn/2007/01/slang_earlywarn.html⟩.