Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 812 19 Bratislava

Dokumentácia k inži-nierskemu dielu 3

Šprint 5, 6 a 7

Tím č. 4 – aDictIT

Bc. Róbert Horváth
Bc. Peter Jurčík
Bc. Peter Macko
Bc. Vladimír Ruman
Bc. Peter Sládeček
Bc Maroš Ubreži
Bc. Matúš Vacula

Vedúci: Ing. Dušan Zeleník
Akademicky rok: 2010/2011

Obsah

Piaty šprint – čajíček

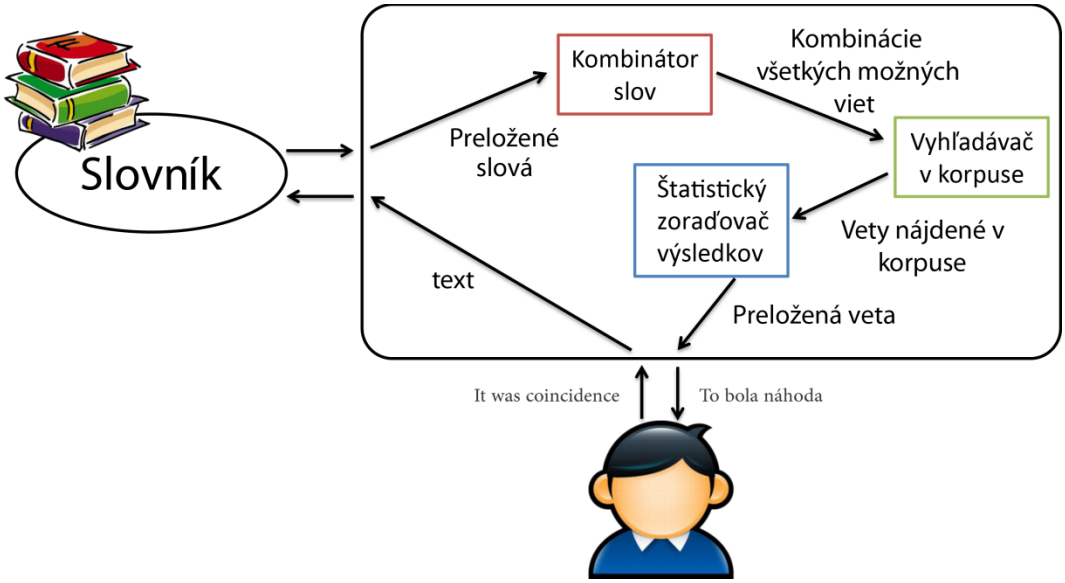
Ako používateľ chcem vedieť, ako dobre prekladá môj prekladač v porovnaní s konkurenciou.

Nová štruktúra prekladača

Chceme mať možnosť odčleniť jednotlivé časti prekladu

Analýza

Pôvodná štruktúra prekladača je zviazaná a nedovoľuje samostatné vyvíjanie jeho častí. Na nasledujúcom obrázku je zobrazený momentálny spôsob fungovania. Prekladaný text postupne prechádza cez jednotlivé komponenty, pričom ich fungovanie je pevne zviazané.



Obr. 1 - Schéma prekladu v aktuálnej verzii prekladača

Pre ďalší postup a prácu v tíme je preto nutné projekt rozčleniť.

Návrh

Počas vývoja prekladača sme identifikovali dve samostatné časti:

Prekladač viet

Vyhľadávač viet

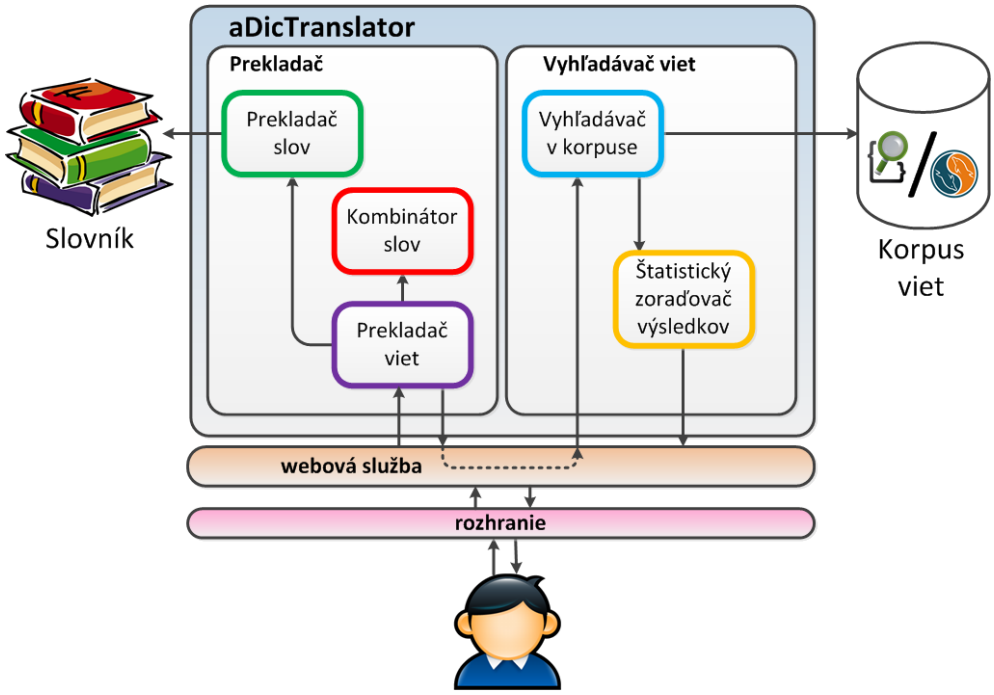
Tieto dve časti sú samostatné, a preto je ich možné dobre odčleniť. Odčlenenie je predprípravou pre ďalšiu úlohu, na ktorej budeme pracovať, a ktorou je vytváranie oddelených verzií týchto častí.

Implementácia

Pri implementácii sme preto vytvorili dve samostatné triedy - Translator a SentenceFinder. Tieto dve triedy na seba nijakým spôsobom nerefrencujú. SentenceFinder využíva preklady získané z Translator-a. Tie je však možné nahradiť iným generátorom viet.

Translator dostáva na svojom vstupe zadaný text, ktorý rozdelí na vety, neskôr na slová a vyhľadá v korpuse. V ďalšej fáze ich skombinuje a vytvorí všetky možné skupiny slov (teda vety).

Tieto putujú do SentenceFindera, ktorý sa ich snaží nájsť v korpuse a používateľovi vrátiť najsprávnejší preklad.



Obr. 2 - Schéma prekladu s odčinenými časťami prekladača a vyhľadávača viet

Extraktor faktov z korpusu viet

Návrh

V našom projekte sme sa na základe analýzy Fact-Extractora rozhodli vytvoriť vlastné pozmenené riešenie, ktoré by pokrývalo naše potreby. Hlavnou vlastnosťou nášho riešenia by mala byť schopnosť rozoznať čísla a fakty nachádzajúce sa v korpuse. Rozpoznané čísla a fakty potom vymeníme za príslušný tag. Takto vymeníme všetky relevantné termy za tagy v korpuse. Takéto riešenie by nám v konečnom dôsledku malo zlepšiť vyhľadávanie viet v korpuse.

Napríklad ak máme vetu:

„Pondelok 13.5.2010 bolo po výbuchu v Bagdade zranených 10 ľudí.“

Táto veta bude zmenená nasledovne:

„#=?+ #=? bolo po výbuchu v #=?+ zranených #=? ľudí.“

Takto bude ElasticSearch schopný hľadať relevantné preklady bez prihliadania na čísla a faktografické údaje. Vďaka tomu bude schopný objaviť s väčšou pravdepodobnosťou vhodný preklad v korpuse.

Aby sme mohli relevantné termy pretagovať musíme vhodným spôsobom identifikovať tieto termy. Ako navrhované riešenie pre nájdenie čísiel použijeme regulárne výrazy. Zároveň okrem základných čísloviek takto nájdeme aj radové číslovky, dátumy alebo výsledky športových zápasov. Tieto termy označíme dohodnutým tagom. Vzhľadom na to, že v korpuse sa nenachádza „#=?“, môže sa tento tag použiť ako identifikátor pre označenie všetkých čísiel. Tag „#=?+“ môže byť označenie pre fakty.

Pre objavovanie faktov je potrebný omnoho sofistikovanejší prístup. Analyzovaný Fact-Extractor použiť nemôžeme, nakoľko nemá pre nás dostatočné rozpoznanie relevantných faktov. Vedúci navrhol využiť práve vyvíjané riešenie jedného študenta, ktorý vyvíja podobné riešenie ako svoj bakalársky projekt.

Implementácia

Pre rôzne typy čísiel používame rôzne regulárne výrazy.

Pre nájdenie základných a radových čísloviek:

„([^a-z,A-Z][0-9]+((\\.|,)[0-9]+)?)|(^[0-9]+((\\.|,)[0-9]+)?)“

Dátum:

„((0[1-9]|[1-9]|[12][0-9]|3[01])[\\/\\-\\.])?((0[1-9]|1[012]|[1-9])[\\/\\-\\.])?([0-9]{1,4})?“

„((0[1-9]|1[012]|[1-9])[\\/\\-\\.])?((0[1-9]|[1-9]|[12][0-9]|3[01])[\\/\\-\\.])?([0-9]{1,4})?“

Športový zápas:

„[0-9]{1,3}[\\s]?:[\\s]?[0-9]{1,3}“

Na základe iných urgentnejších úloh sme sa rozhodli túto úlohu odložiť na neskôr a v implementácii pokračovať po vyriešení dôležitejších úloh. V rámci implementácie je ešte nutné použiť vytvorené regulárne výrazy a implementovať spomínané vyvíjané riešenie.

Skupiny slov v korpuse

Chceme vedieť preložiť aj slovné spojenia, ktorých preklad je jedno slovo a neprekladať ich po slovách

Analýza

Momentálne sa nám všetky slová prekladajú po slovách, čo nemusí byť vždy to najsprávnejšie. Napríklad slovné spojenie „*prime minister*“ sa do slovenčiny prekladá ako „*premiér*“. Náš prekladač však toto slovné spojenie prekladá po slovách a tak sa k prekladu *„premiér“* nikdy nedostaneme.

Tento stav by sme chceli zmeniť tak, aby náš prekladač hľadal aj takéto slová. Pričom tieto slová by mali mať vyššiu relevanciu ako samostatný preklad slov.

Návrh

Pri prekladaní viet je nutné do prekladača vložiť celý zvyšok vety, teda od aktuálne prekladaného slova až po koniec. Týmto spôsobom vieme získať nielen preklad daného slova, ale aj väčšieho slovného spojenia.

Údaj o tom, koľko slov nahradil daný preklad, je nutné uchovať tak, aby sa v nasledujúcich fázach spájania a generovania viet dalo povedať, ktorá veta je výhodnejšia a preto má byť vo výsledkoch vyššie umiestnená.

Implementácia

Túto metódu sme vo výsledku implementovali jednoduchým rozšírením SELECT príkazu, ktorý získava dáta zo slovníka.

```
SELECT DISTINCT
      W_TO.word AS translate,
      W_FROM.word AS source
FROM
      word W_FROM
      JOIN translation T ON W_FROM.id_word = T.id_word
      JOIN word W_TO ON W_TO.id_word = T.id_translation
WHERE
      T.id_language_to = <LangTo> AND
      W_FROM.word LIKE <WordToTranslate>% AND
      W_FROM.id_language = <LangFrom> AND
      <SentencePart> LIKE CONCAT(W_FROM.word,'%')
```

V tomto príkaze sa namiesto parametrov dosádzajú v prvej fáze jazyky, z ktorého(LangForm), a do ktorého(LangTo) sa prekladá text.

Ďalej sa do príkazu dosádza slovo, ktoré chceme preložiť (WordToTranslate) s tým, že na jeho koniec sa umiestňuje znak %, ktorý nahrádza akúkoľvek ďalšiu sentenciu slov. Na koniec sa do príkazu dosádza celá postupnosť slov v prekladanej vete (SentencePart) nasledujúca za aktuálne prekladaným slovom vrátane tohto slova. To znamená, že sa vo výsledku vyhľadajú nielen slová, ktoré sú prekladom len jedného slova, ale aj celej sentencie, ktorá sa zhoduje so zadanou vetou.

Automatický tester

Analýza

V šprinte 3 sme opísali implementáciu automatického testera. Google však v decembri prestal poskytovať službu, ktorá nám zabezpečovala ich preklad. Preto je potrebné nájsť iné riešenie, ktorým by sme spomenutý preklad získali.
Existuje niekoľko riešení ako sa dopracovať k požadovaným informáciám. Jedným z nich je objavenie takej stránky, z ktorej by bolo možné tento prekladu extrahovať. Ďalšou možnosťou je odchytávať HTTP komunikáciu, v ktorej jej výsledok prekladu takisto uložený. Posledným riešením je nájsť spôsob, ako Google Translate prinútiť, aby nám tento výsledok vrátil sám.

Návrh

Z vyššie opísaných možností sme nakoniec využili posledné riešenie. Cez skrátené vyhľadávanie v prehľadávači sa nám podarilo nájsť takú adresu URL, ktorá vo svojom kontexte obsahovala požadovaný preklad. Následne bol vytvorený nový Google parser, ktorý odstránil zo získaného zdrojového kódu všetok nepotrebný text.

Implementácia

Implementovaný parser otvorí stránku Google Translate prostredníctvom URL adresy

*$url="http://translate.google.sk/?js=n&prev=_t&hl=".**$to**."&ie=UTF-8&layout=2&eotf=1&sl=".**$from**."&tl=".**$to**."&text=".**$text**."&file=#".**$from**."|".**$to**."|".**$text**." %0D%0A";*

Parameter **$to** definuje jazyk, do ktorého sa požadovaný text (**$text**) prekladá. Jazyk, z ktorého sa prekladá je opísaný parametrom **$from**.

V získanom zdrojovom kóde stránky je následne hľadaný podreťazec

*span title="'.**$what**.'" onmouseover="this.style.backgroundColor=\'#ebeff9\'" onmouseout="this.style.backgroundColor=\'#fff\'">*

kde parameter **$what** predstavuje upravený tvar vstupného textu **$text**.

Testovanie

Získanie prekladu z Google Translate

Tento testovací scenár overuje schopnosť funkcie *get_google($from,$to,$what)* získať požadovaný preklad.

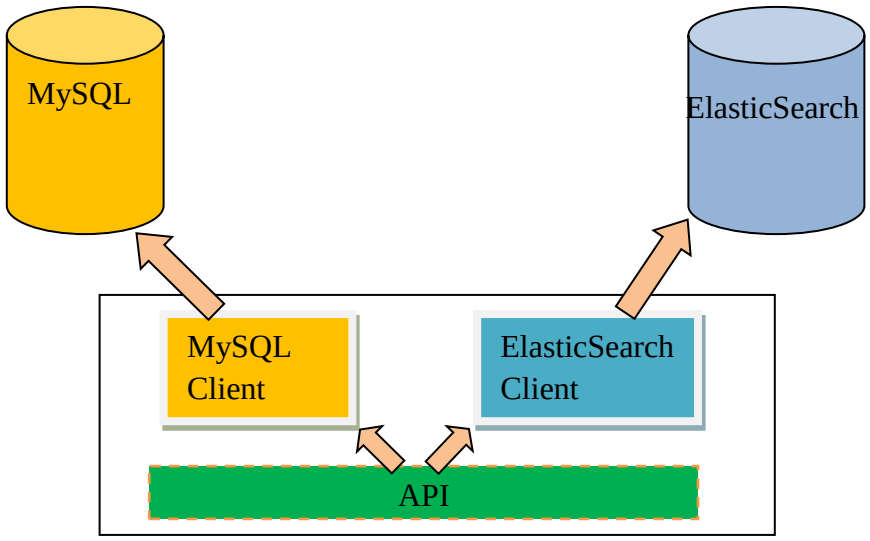
Názov	Získanie prekladu z Google Translate	ID Testu	**0x-0x**
Rozhranie	translate_checker/TranslateClasses/google.php	ID UC	**0x**
Účel	Overenie správnej extrakcie prekladu zo zdrojového kódu		

Vstupné podmienky	Zadaný text na preklad, vstupný a výstupný jazyk
Výstupné podmienky	Funkcia *get_google($from,$to,$what)*vráti rovnaký preklad ako je zobrazený na stránke translate.google.com

Krok	**Akcia**	**Očakávaná akcia**	**Skutočná reakcia**
1.	Zavolanie funkcie	Funkcia stiahne požadovanú webovú stránku Google Translate a extrahuje z nej požadovaný preklad.	Preklad z Google Translate bol korektne získaný a zobrazený na stránke automatického testera.

Návrh nástroja pre automatické pridávanie zmien do ElasticSearch a MySQL

Po tom ako sme preniesli korpus viet z MySQL do ElasticSearch vznikla požiadavka na jednotný nástroj pomocou ktorého by sme vedeli vykonávať zmeny v obidvoch spomínaných systémoch Obr. 3. Vďaka tomuto nástroju by sme nemuseli vykonávať zmeny v každom z nich samostatne a eliminovali by sme zbytočnú duplicitu práce.



Obr. 3 – Nástoj na pridávanie zmien do ElasticSearch

Nástroj poskytuje rozhranie, ktoré umožnuje vykonávať nasledujúce úlohy s oboma databázami:

Pridávanie nových viet

Úpravu existujúcich viet

Zmazanie zvolených viet

Implementácia nástroja na pridávanie zmien

Nástroj je implementovaný v jazyku Java. Poskytuje rozhranie pomocou, ktorého je možné volať metódy na výkon úloh identifikovaných v časti návrh.

Zdrojový kód poskytovaného rozhrania nástroja

Pridávanie nových viet:

Vstup:

Id prvého nového záznamu

pole viet, ktoré chcete vložiť do databáz

Návratová hodnota:

0: podarilo sa vložiť všetky požadované vety

-1: vety sa nepodarilo vložiť

```
public int insertNewSentences(int startingId,String[] newSentences) {
      }
```

Úprava existujúcich viet:

Vstup:

Id záznamu, ktorý chcete zmeniť

Požadovaná nová veta

Návratová hodnota:

0: podarilo sa zmeniť záznam

-1: záznam sa nepodarilo zmeniť

```
public int updateSentence(int id,String sentence) {
      }
```

Zmazanie zvolených viet:

Vstup:

Id záznamu, ktorý chcete zmazať

Návratová hodnota:

0: záznam bol zmazaný

-1: záznam sa nepodarilo zmazať

```
public int deleteSentence(int id) {
	}
```

Šiesty šprint – radler

Ako používateľ chcem mať možnosť si prekladač sám škálovať.

Konfigurovateľnosť prekladača

Chceme, aby bol prekladač jednoducho konfigurovateľný bez zmeny kódu služby

Analýza

Pri ladení nášho prekladača je nutné pracovať s konfiguráciou služby. Doteraz bolo nutné meniť tieto parametre priamo v zdrojovom kóde. Chceme tieto parametre teda presunúť do jedného konfiguračného súboru.

Návrh

Keďže náš prekladač je implementovaný v prostredí Java, využijeme na konfigurovanie Property súbory. Tieto súbory majú jednoduchú štruktúru:

```
Meno_premennej  hodnota
```

Property súbor bude v prekladači prístupný pomocou typu ConfigInfo. Tento obsahuje všetky aktuálne podporované premenné. Konfiguračný súbor sa bude získavať pomocou vzoru Singleton.

Implementácia

Načítavanie súboru je náročnejšie, pretože súbor chceme mať sprístupnený aj vo webovej službe, kedy java pracuje v inej časti súborového systému, ako by sme čakali.

```
private ConfigInfo() throws IOException
{
	final Properties prop = new Properties();

	try
	{
		prop.load(ConfigInfo.class.getResourceAsStream("/config.properties"));
	}
	catch (NullPointerException ex)
	{
			prop.load(new FileInputStream("config.properties"));
	}
	mySqlAddress  = prop.getProperty("MYSQL_ADDRESS");
	...
}
```

Po samotnom načítaní súboru sa inicializujú všetky potrebné parametre. Vzor SIngleton je implementovaný pomocou statickej metódy *instance*:

```
public static ConfigInfo instance() throws IOException
{
	if(instance == null)
	{
		instance = new ConfigInfo();
	}

	return instance;
}
```

Vďaka tomu sa súbor načítava vždy iba raz.

Pomocné funkcie webovej služby

Chceme mať prístupný zoznam všetkých verzií a aktuálne využívané premenné

Analýza

Keďže nás prekladač ma niekoľko rôznych verzií, je dobré ich mať popísané a zdokumentované. Tento popis môže byť prístupný aj verejne, keďže pri zadávaní prekladu je potrebné zadať verziu podľa jej čísla. Pri ladení je okrem toho dobré vedieť, s akými parametrami pracuje služba.

Návrh

Na tento účel chceme využiť ďalšie webové služby, ktoré budú prístupné našim zákazníkom.

Pre možné verzie bude dopyt vyzerať takto:

http://localhost:8080/aDicTranslator/support/versions

Výstup bude vo formáte JSON takto:

```
{
	"count":7,
	"versions":[{
		"id":1,
		"name":"vyhladajVMysql",
		"comment":"Verzia používa základný prekladač a vyhľadáva dáta ..."
		},
		...
	]
}
```

Pre získanie parametrov bude dopyt vyzerať takto:

http://localhost:8080/aDicTranslator/support/config

Výstup bude vo formáte JSON takto:

```
{
	"MYSQL_ADDRESS":"localhost",
	"MYSQL_USER":"root",
	...
}
```

Implementácia

Pomocné servisy budú teda takisto realizované pomocou Jersy knižnice nasledovne:

```
@Path("support")
public class SupportService
{
	@GET
	@Produces("application/json; charset=UTF-8")
	@Path("versions")
	public String getVersions() throws JSONException
	{
		...
	}
}
```

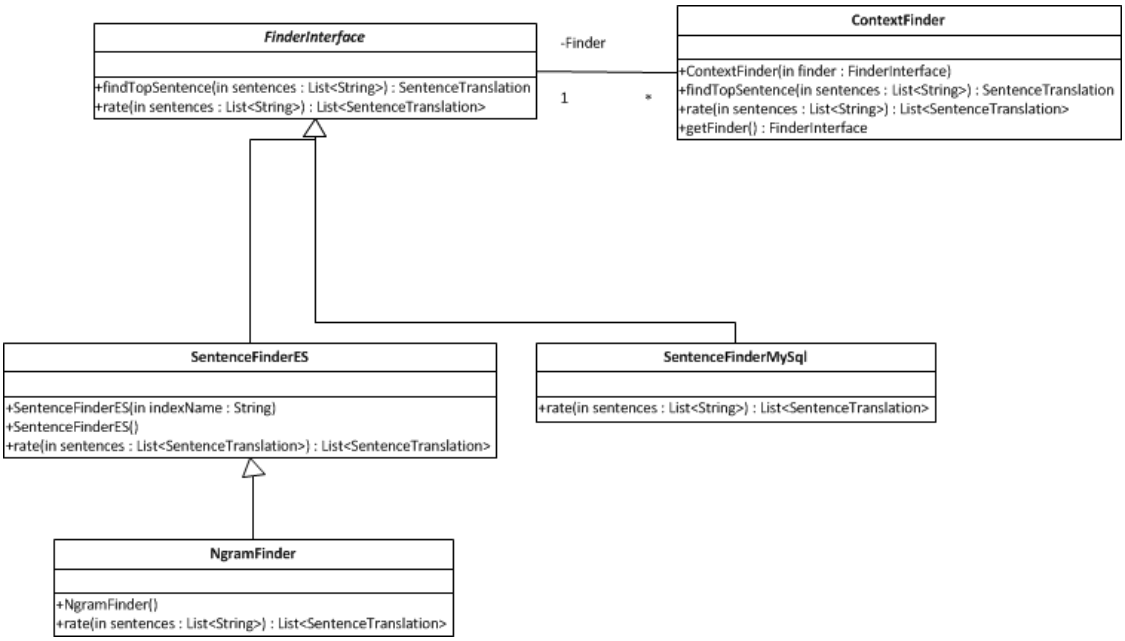
Vytvorenie Stratégií

Analýza

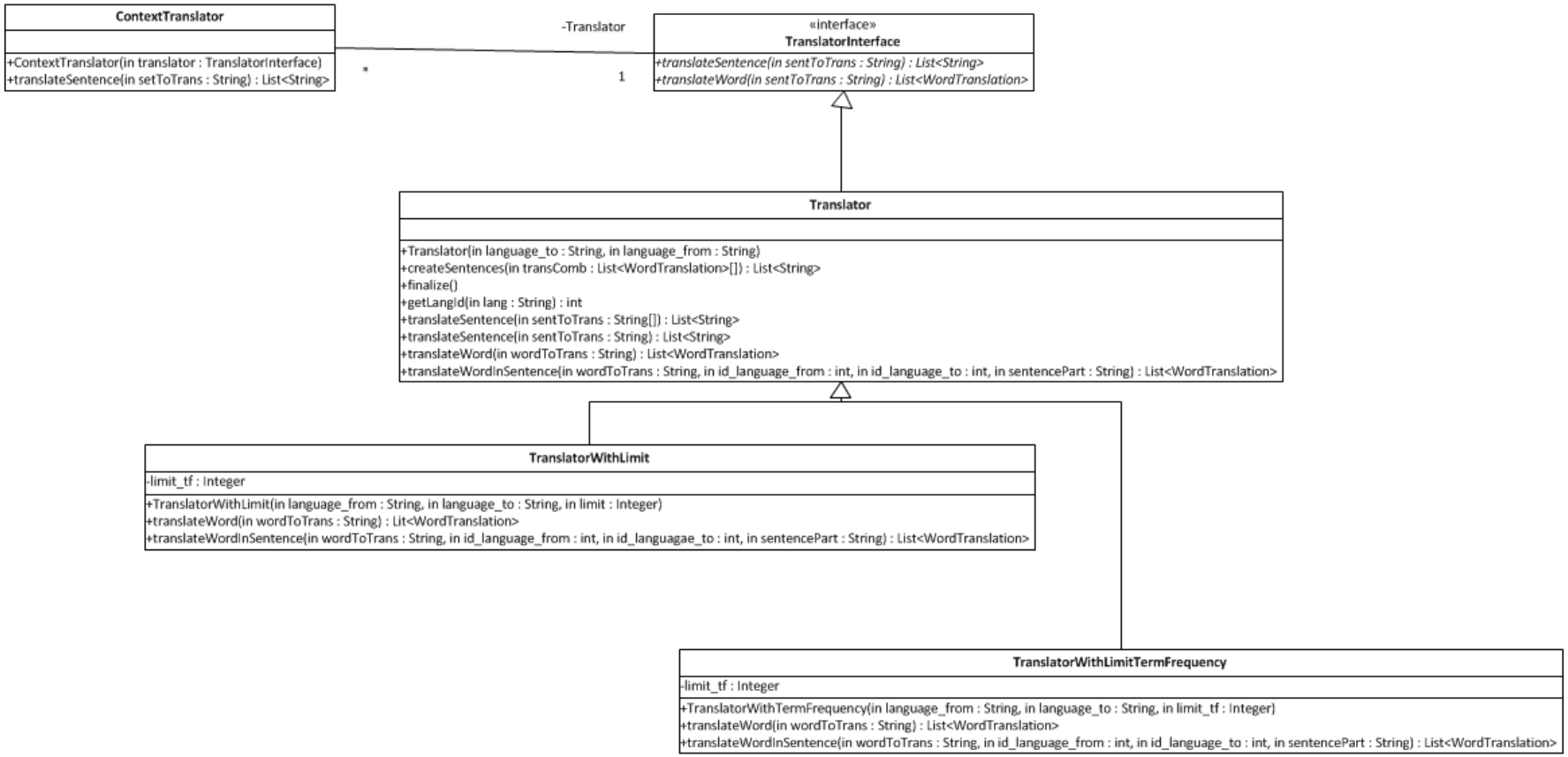
Vzhľadom na to, že vyvíjame viacero alternatív, ktoré chceme medzi sebou porovnávať a testovať je potrebné aby zdrojový kód RESTovej služby bol čo možno najviac nezávislý od kódu samotného prekladača. Tiež potrebujeme, aby jednotlivé časti prekladača boli čo najmenej ovplyvnené pri modifikácií alebo pridaní ďalšej časti a v konečnom dôsledku potrebujeme sprehľadniť náš zdrojový kód.

Návrh

Preto sme sa rozhodli využiť návrhový vzor Strategy, ktorý toto umožňuje. Implementovaním tohto vzoru sme schopný pridávať ďalšie verzie prekladača pričom to len minimálne ovplyvní ostatný kód.

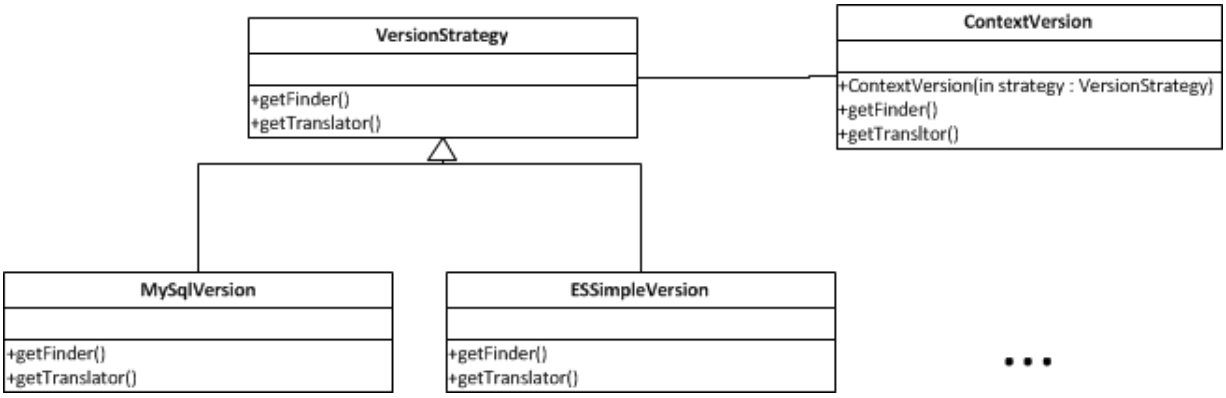


Obr. 4 - Diagram stratégie pre objekty súvisiace s vyhľadávaním viet



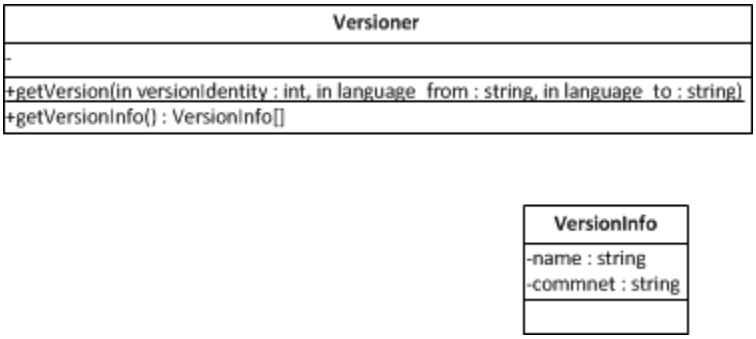
Obr. 5 - Diagram stratégie pre objekty súvisiace s prekladom textu

Následne sme vytvorili stratégiu verzionovania. Každá verzia má v tomto prípade priradený jeden Translator a jeden SentenceFinder podľa toho, aké činnosti chce realizovať.



Obr. 6 - Stratégia pre verzie prekladača

Okrem toho sme vytvorili objekt Verzioner, ktorý vytvára verziu podľa zadaného ID a takisto vie získať informácie o aktuálnych verziách ktoré sú uchovávané vďaka objektu VersionInfo.



Obr. 7 - Verziner pre získavanie verzie podľa zadaného ID a VersionInfo pre uchovávanie informácií o verziách

Implementácia

Pri implementácií bola umiestnená stratégia pre Prekladač do balíčka sk.stuba.fiit.adictit.translator. Stratégia pre SentenceFinder bola umiestnená v sk.stuba.fiit.adictit.translator.sentences.

Pre verzie bol vytvorený nový balíček: sk.stuba.fiit.adictit.utils.versioning. Výber verzií podľa ID je realizovaný nasledovne:

```
public static ContextVersion getVersion(final int versionIdent,final String langua-
ge_from,final String language_to) throws CannotCreateTranslatorException, CannotCrea-
teSenteceFinderException
	{
		ContextVersion version;
```

```
            switch (versionIdent)
            {
                  case 1:
                        version = new ContextVersion(new MySqlVer-
sion(language_from, language_to));
                        break;
                  case 3:
                        version = new ContextVersion(new ESVersionWithLi-
mit(language_from, language_to));
                        break;
                  case 4:

....
```

Optimalizácia služby typu REST

Chceme mať pohodlne prístupnú webovú službu so zadaním verzie prekladu

Analýza

Naša rest služba prijíma momentálne 4 parametre prekladu a to:

Metóda výstupu (najlepší preklad, množina najlepších prekladov, všetky relevantné preklady)

zdrojový jazyk

cieľový jazyk

text

štruktúra URL adresy:

```
http://adresaServera/aDicTranslator/metoda/jazykZ/jazykDo/prekladanyText
```

Všetky tieto parametre sú časťou URL adresy a tak nie je možné žiaden vynechať. Takisto dĺžka adresy je limitovaná, a tak by v prípade prekladu dlhšieho textu nastali problémy. Do adresy je preto nutné pridať nový parameter a to verziu prekladača a prerobiť štruktúru tak, aby bola viac prispôsobivá.

Návrh

V samotnej URL adrese dopytu na získanie prekladu zostanie už len metóda prekladu, ku ktorej sa pridá číslo verzie, ktorou budeme prekladať. Zdrojový, cieľový jazyk a samotný text sa presunú do formy parametrov. Volanie služby bude prístupné nielen HTTP metódou GET, ale aj POST pre prekladanie dlhších textov.

Nová štruktúra adresy bude vyzerať takto:

```
http://adresaServera/aDicTranslator/VERZIA/METODA?
      text=PREKLADANY_TEXT&
```

```
from=JAZYKY_Z&
to=JAZYK_DO
```

Implementácia

Služba typu REST bola na našom serveri momentálne realizovaná za pomoci knižnice WINKS. Tá nám však nedovoľovala pracovať s parametrami formulára, a tak sme boli nútení prejsť na inú knižnicu. Použili sme preto Jersy knižnicu.

Kód pre vytvorenie webovej služby s danou štruktúrou vyzerá takto:

```
@GET
@Produces("text/plain; charset=utf-8")
@Path("{version:\\d}/first")
public String getFirstTrasnalte(
            @PathParam("version")                  final  int    versionIdent,
            @QueryParam("text")                    final  String text,
            @QueryParam("from") @DefaultValue("en") final  String lang_from,
            @QueryParam("to")   @DefaultValue("sk") final  String lang_to)
      throws  CannotCreateTranslatorException,
            CannotCreateSenteceFinderException, CannotTranslateTextException
{
      ...
}
```

Tento kód zobrazuje metódu „first“, pričom definícia ostatných metód vyzerá analogicky. Ako bolo spomenuté vyššie, v URI (@Path) ostali zachované iba verzia a metóda, pričom verzia je definovaná číslom. Verzia je potom parameter funkcie, čo definuje štruktúra @PathParam(„version“). Ďalšie parametre sú získané z parametrov typu Query, teda z formulárových premenných.

Siedmy šprint – pivečko

Ako používateľ chcem prekladať rýchlejšie a presnejšie.

Vyhľadávanie cez n-gramy

Analýza

Pri prekladaní vety spôsobom preloženia každého slova a vyhľadávaní všetkých viet, ktoré vznikli kombináciou týchto prekladov, vzniklo veľké množstvo dopytov na korpus v ElasticSearch. Taktiež táto metóda pre úspech predpokladala, že výsledná veta sa v korpuse bude nachádzať. Tieto dva problémy je možné vyriešiť tým, že sa prekladaná veta bude deliť na n-gramy (n-tice slov). Týmto spôsobom je možné vetu preložiť aj keď sa celá v korpuse nenachádza, ale nachádzajú sa v ňom jej jednotlivé časti.

Ako príklad majme vetu o dĺžke 6 slov, kde priemerný počet možných prekladov slova je 5 prekladov. Pri vytváraní všetkých potencionálnych viet vznikalo pri klasickom preklade teda 5^6 (15625) možných kombinácii. Veľký počet kombinácií predstavuje veľa požiadaviek na vyhľa-

dávanie a tak spomaľuje rýchlosť celého prekladu. Pri použití n-gramov o dĺžke 3 dostaneme pre uvedený príklad len $4*5^3$ (500) kombinácií, čo predstavuje 31 násobne menšiu náročnosť.

Náročnosť doterajšieho prekladu sa dá vyjadriť vzorcom: x^y

Náročnosť prekladu využitým n-gramov vzorcom: $(x - n + 1) \times x^n$

Kde x je počet slov vo vete, y je priemerný počet prekladov jednotlivých slov a n je veľkosť n-gramu. Z uvedeného vyplýva náročnosť $O(x^Y)$ v porovnaní s $O(x^3)$ pri n-gramoch a teda efektivita n-gramov sa prejaví najmä pri dlhých vetách.

Nájdené n-gramy je následne nutné spojiť do jednej vety. Vhodným spôsobom je n-tice spájať na základe ich spoločného prieniku. Teda budú sa spájať n-gramy, ktoré majú po svojich krajoch spoločný prienik. Tým pádom je vyššia pravdepodobnosť, že výsledná veta bude zmysluplná a nebude to len zlepenec slov bez významu. Dosiahnuteľné to je úpravou hodnotenie, kde sa zvýhodnia vety, ktoré vznikli spojením n-gramov s väčším prienikom.

Návrh

Na základe skúseností s rýchlosťou existujúceho riešenia a zvážení minimálnej veľkosti n-gramu sme sa dohodli, že riešenie bude pracovať s trigramami. Táto hodnota však bude konfigurovateľná a bude ju možné v budúcnosti meniť.

Aby bolo možné prekladať vety prostredníctvom trigramov je nevyhnutné upraviť nie len dopyty na korpus, ale aj samotný korpus viet. Upravíme preto existujúci korpus, rozsekaním viet na vety o maximálnej dĺžke troch slov.

Nový proces prekladu bude pozostávať z piatich hlavných krokov:

Preloženie vety po slovách (získanie všetkých možných prekladov pre každé slovo).

Vyskladanie trigramov.

Vyhľadanie trigramov v korpuse.

Spájanie výsledkov z korpusu a vypočítanie skóre na základe skóre z SlasticSearch a prekryvu spájaných trigramov.

Zobrazenie výsledkov podla dosiahnutého skóre.

Implementácia

Základom pre implementáciu bolo rozdeliť vety v korpuse na n-gramy. Z hľadiska výkonu bolo pre nás najideálnejšie rozdeliť vety na trojice slov. K tomu sme použili nasledujúcu metódu.

```
private static ArrayList<String> generateNgrams(int n, String senten-
ce){
      String words[] = sentence.split(" ");
```

```
        String generatedSentence = null;
        ArrayList<String> sentGroup = new ArrayList<String>();

        for (int i=0; i<words.length-n+1; i++){
              generatedSentence = "";
              for (int j=0; j<n; j++){
                    generatedSentence = generatedSentence.concat (words[i
                    + j]).concat(" ");
              }
              sentGroup.add(generatedSentence.trim());
        }
        return sentGroup;
}
```

Preklad viet cez n-gramy sme implementovali ako novú verziu prehliadača, pričom sme vychádzali z existujúcej verzie hľadajúcej v ElasticSearch.

```
public class NgramFinder extends SentenceFinderES{

      public NgramFinder() throws CannotCreateSenteceFinderException {
            super("shingleindex");
      }
```

Pre optimalizovanie výkonu sme najprv identifikovali všetky unikátne trigramy a dopyt na ElasticSearch sme vykonali pre všetky hromadne. Neskôr sme k prekladom pristupovali cez objekt triedy HashMap.

```
HashMap<String, List<SentenceTranslation>> hmap = new HashMap<String,
...
for(SentenceTranslation st: translations){
      if(hmap.containsKey(st.getOriginalTranslate())){
            hmap.get(st.getOriginalTranslate()).add(st);
      }
      else{
            List<SentenceTranslation> newlist = new Array-
            List<SentenceTranslation>();
      newlist.add(st);
      hmap.put(st.getOriginalTranslate(),newlist);
      }
}
```

Pre spájanie jednotlivých trigramov a ich ohodnocovanie je nevyhnutné odhaliť veľkosť prekryvu dvoch ngramov. Slúži na to nasledujúca funkcia *getSentenceSimilarity*.

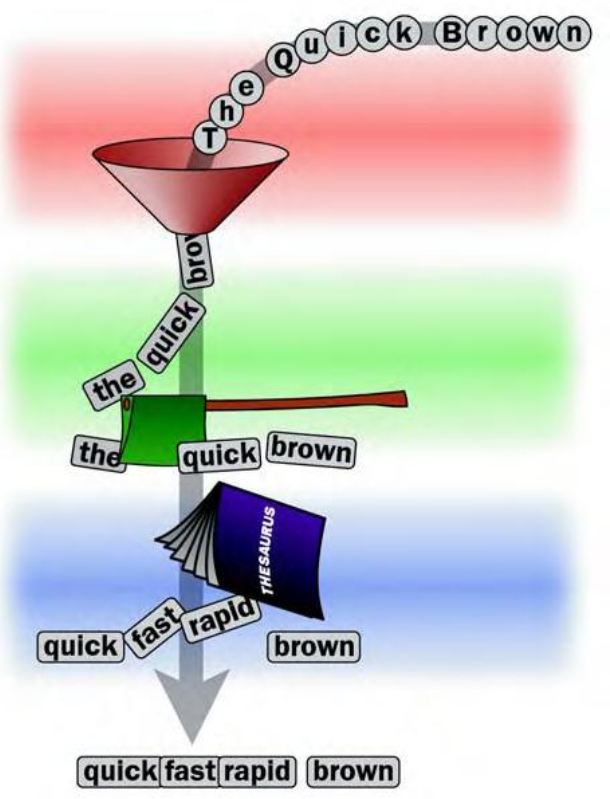
```
private static int getSentenceSimilarity(String s1, String s2){
          String a1[] = s1.toLowerCase().split(" ");
          String a2[] = s2.toLowerCase().split(" ");

          int val = 0;
          for (int i=0; i<a1.length; i++){
               if (a1[i].equals(a2[0])){
                    for (int j=0; j<a1.length-i; j++){
                         if (a1[i+j].equals(a2[j])){
                              val++;
                         }
                         else{
                              val = 0;
                              break;
                         }
                    }
                    break;
               }
          }
          return val;
     }
```

Vyhľadávanie pomocou synoným

Analýza

Elasticsearch umožňuje využívanie zoznamu synoným ako podpory pre vyhľadávanie v indexovaných dokumentoch. To znamená, že pri vyhľadávaní Elasticsearch neprehľadáva len dané slovo ale aj všetky synonymá k tomuto slovu. Táto funkcionalita nám umožní zrýchliť a zlepšiť vyhľadávanie prekladu.



Obr. 8 - princíp fungovania token filtra „synonym“ v elasticsearch.1

Elasticsearch podporuje dva typy formátov synonymických slovníkov:2

Solr format

WordNet

Návrh

Pre využite token filtra “synonym” potrebujeme vytvoriť textový súbor obsahujúci synonymá v tvare , ktorý podporuje Elasticsearch. Vhodným formátom je Solr formát.

Kedže máme k dispozícii slovník z PC Translatora v textovom formáte nebude problém ho upraviť do správneho tvaru. Rovnako bude vhodné získať iný slovník, ktorý by sme tiež použili a mohli tak porovnať rôzne verzie a vybrať tú lepšiu. Ako ďalší slovník na úpravu sa dá použiť otvorený slovník Thesaurus, ktorý obsahuje okolo 14000 synoným.

1 Prevzaté z HATCHER, E. et al.: Lucene in Action. 2. Vyd. 2010. ISBN 1933988177.

2 http://www.elasticsearch.org/guide/reference/index-modules/analysis/synonym-tokenfilter.html

Implementácia

Vytvorili sme dva súbory s názvami “synonymspc.txt” a “synonymsthesaurus.txt”, ktoré slúžia ako synonymiské slovníky pre elasticsearch.

Format súborov:

```
administratíva, správa
administratívny, byrokratický, správny, úradný
administrácia, riadenie, správa
administrátor, správca, superpoužívateľ
adopcia, osvojenie, prijatie
adoptovanie, osvojenie si
adoptovať, osvojiť si, prijať za vlastné
adresovať, obrátiť sa na, osloviť
adresár, priečinok
adresát, príjemca
aerodróm, letisko
aeronaut, aviatik, letec, pilot
afekt, afektovanosť, strojenosť, umelosť
afektovanosť, afekt, strojenosť, umelosť
afektovaný, neprirodzený, prehnaný, strojený
agent, dohadzovač, sprostredkovateľ, zástupca, vyzvedač, špión
agregovaný, agregovaný , akcesorický , nepodstatný, nepričlenený, vedľajší
```

Obr. 9 - Časť zo súboru synonymsthesaurus.txt

Taktiež sme vytvorili dva nové indexi, ktoré sme nazvali “synpc” a “synthesaurus”. Názvy korelujú s použitými slovníkmi. V oboch indexoch sme vytvorili token filter s názvom “synonym”.

Nastavenia indexov:

{"synpc":{"settings":{"index.analysis.analyzer.synonym.filter.0":"synonym","index.analysis.filter.synonym.type":"synonym","index.analysis.analyzer.synonym.tokenizer":"whitespace","index.analysis.filter.synonym.synonyms_path":"analysis/synonymspc.txt","index.number_of_shards":"5","index.number_of_replicas":"1","index.version.created":"190053"}}}

{"synthesaurus":{"settings":{"index.analysis.analyzer.synonym.filter.0":"synonym","index.analysis.filter.synonym.type":"synonym","index.analysis.analyzer.synonym.tokenizer":"whitespace","index.analysis.filter.synonym.synonyms_path":"analysis/synonymsthesaurus.txt","index.number_of_shards":"5","index.number_of_replicas":"1","index.version.created":"190053"}}}

Reindexácia údajov z indexu sentences do indexov synoným

Po vytvorení požadovaných indexov, bolo potrebné ich naplniť dátami. Namiesto opätovného manuálneho vkladania viet z MySQL databázy do oboch nových indexov, sme sa rozhodli reindexovať dáta z už existujúceho ElasticSearch indexu menom sentences. Toto riešenie bolo zvolené z dôvodu vyššej efektívnosti a nižšej dobe trvania.

Implementácia reindexácie údajov

Reindexáciu sme implementovali v jazyku Perl s použitím knižnice *ElasticSearch - An API for communicating with ElasticSearch*3 vo verzii 0.52.

Zdrojový kód skriptu:

```
#!/usr/bin/perl


use ElasticSearch();

my $es = ElasticSearch->new(

   transport => 'http',

   servers => '127.0.0.1:9200' -- host:port na ktorom beží Elas-
ticSearch

);


my $source = $es->scrolled_search(

   index => 'sentences',   -- index z ktorého chceme získať dáta

   search_type => 'scan',

   scroll => '15m',

   version => 1

);


$es->reindex(

   source => $source,           -- dáta ktoré chceme vložiť do
indexu

   dest_index => 'synpc',       -- názov indexu kam chceme vlo-
žiť dáta
```

3 https://metacpan.org/module/ElasticSearch

```
   bulk_size => 5000              -- veľkosť dávky po akej sa budú
dáta

);                            -- vkladať
```

Uvedený skript bol spustený pre obidva nové indexy, synpc a synthesaurus.

Logovanie v prekladači

Analýza

Počas procesu prekladanie slov a viet prostredníctvom našej služby sa niekoľkokrát stalo, že náš prekladač nevrátil požadovaný výsledok. Táto neočakávaná situácia bola spôsobená internou chybou, ktorá vznikla v niektorej fáze prekladu. Elimináciu týchto chýb je možné dosiahnuť logovaním činnosti prekladača.

java.util.logging

Java nám poskytuje balík *java.util.logging*, ktorý však predstavuje iba okliešteną verziu log4j. Základným objektom je trieda typu Logger. Správy, ktoré prostredníctvom nej zaznamenávame, môžu mať rôzne úrovne dôležitosti (FINEST, FINER, FINE, CONFIG, INFO, WARNING, SEVERE). Na začiatku logovania je nutné nastaviť jej povolenú úroveň – štandardne je INFO. Všetky správy sú posielané do handlera, ktorý overuje jej dôležitosť. V prípade, že je nižšia ako vopred definovaná úroveň, handler správu neakceptuje a zahodí ju. Správy je možné formátovať triedou formater, ktorá poskytuje ich výstupy do štandardných súborov typu .txt či .xml.

log4j

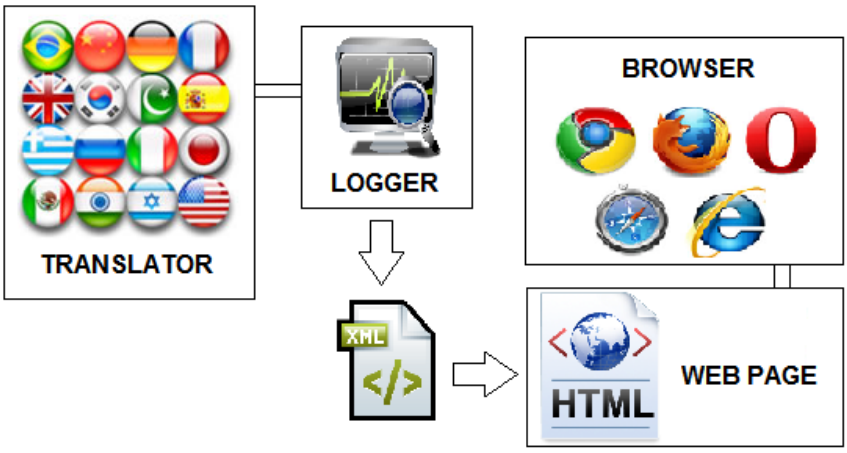
Log4j je balík, ktorý slúžil ako základ pri tvorbe JUL. Publikácia správ sa vykonáva implementáciou rozhrania Appender, ktorý je ekvivalentom vyššie spomínaného handleru. Na rozdiel od neho je však možné použiť mapovanie do jednotlivých stratégií výstupu. Zdedením abstraktnej podtriedy Layout vytvoríme vlastnú štruktúru výstupu.

Výsledky získané logovaním je nutné mapovať na webovú stránku tak, aby bolo umožnené ich zobraziť v prehľadnej forme. Z tohto dôvodu je potrebné vytvoriť vlastný formát výstupu do XML. V tomto prípade je preto vhodnejšie použiť log4j.

Návrh

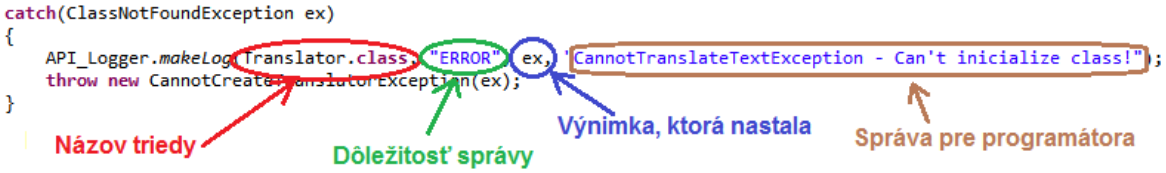
Vytvárané riešenie logovanie musí spĺňať niekoľko základných požiadaviek. Medzi ne patrí hlavne plná automatizovateľnosť procesu. Ten zhromaždí množinu výsledkov, ktoré následne poskytnú programátorovi relevantné informácie o príčinách pádu služby.

Nami navrhnuté riešenie je opísané na Obr. 10. K nášmu prekladaču je pripojený logovací balík log4j, ktorý monitoruje jeho činnosť. Správy, ktoré mu prekladač posiela, sú mapované do pripraveného XML formátu. Vytvorená webová stránka načíta informácie zo vstupného súboru. Tie následne zobrazí vo svojom obsahu prostredníctvom zoznamu umiestneného v tabuľke. Keďže predpokladáme, že XML súbor bude obsahovať veľké množstvo záznamov, používateľ musí mať možnosť tento zoznam filtrovať.



Obr. 10 - Návrh logovania našej služby

Do nášho prekladača bola pridaná knižnica *log4j-1.2.16.jar*, ktorá zabezpečuje logovanie do súboru. Súčasne bol vytvorený balík *sk.stuba.fiit.adictit.xml_logger*, ktorý obsahuje triedu *API_Logger*. Tá implementuje viditeľnú funkciu *makeLog* prostredníctvom ktorej je realizované logovanie do súboru. Parametre funkcie opisuje Obr. 11.

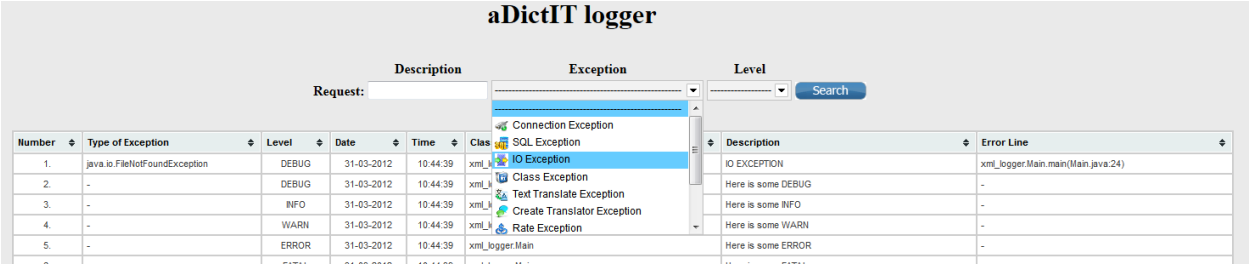


Obr. 11 - Volanie vytvoreného loggera

Vyššie opísaná funkcia využíva nami vytvorenú XML šablónu dokumentu (Obr. 12). Takto uložený súbor je následne poskytnutý webovej stránke (Obr. 13).

```
<EXCEPTION>
    <TYPE>java.io.FileNotFoundException</TYPE>
    <LEVEL>DEBUG</LEVEL>
    <DATE>31-03-2012</DATE>
    <TIME>10:44:39</TIME>
    <CLASS>xml_logger.Main</CLASS>
    <DESCRIPTION>IO EXCEPTION</DESCRIPTION>
    <ERROR_LINE>xml_logger.Main.main(Main.java:24)</ERROR_LINE>
</EXCEPTION>
```

Obr. 12 - Správa z prekladača zapísaná v súbore XML.



Obr. 13 - Webová stránka s načítaným XML súborom.

Testovanie

Zobrazenie vstupného XML súboru na webovej stránke

Pre overenie úspešnosti mapovania XML súboru sme použili nasledovný testovací scenár, podľa ktorého sme otestovali funkčnosť implementácie.

Názov	Mapovanie XML súboru	ID Testu	**0x-0x**
Rozhranie	logger/index.php	ID UC	**0x**
Účel	Overenie úspešnosti mapovania XML súboru		
Vstupné podmienky	XML súbor umiestnený v rovnakom priečinku ako webová stránka.		
Výstupné podmienky	Používateľ vidí na obrazovke vyplnenú tabuľku.		

Krok	**Akcia**	**Očakávaná akcia**	**Skutočná reakcia**
1.	Načítanie XML súboru.	XML bude načítaný v tabuľke webovej stránky.	Dáta z XML súboru boli v tabuľke zobrazené v požadovaných stĺpcoch.

Filtrovanie správ načítaných z XML súbora

Nasledovný testovací scenár overuje funkčnosť implementácie filtrov.

Názov	Filtrovanie	ID Testu	**0x-0x**

Rozhranie	logger/index.php	ID UC	**0x**
Účel	Overenie úspešnosti filtrovanie XML súboru		
Vstupné podmienky	XML súbor umiestnený v rovnakom priečinku ako webová stránka.		
Výstupné podmienky	Používateľ vidí na obrazovke vyplnenú tabuľku podľa zvoleného kritéria.		

Krok	Akcia	Očakávaná akcia	Skutočná reakcia
1.	Načítanie XML súboru.	XML bude načítaný v tabuľke webovej stránky.	Dáta z XML súboru boli v tabuľke zobrazené v požadovaných stĺpcoch.
2.	Používateľ nastaví filter.	V tabuľke sa zobrazia iba tie kritéria, ktoré vyhovujú filtru.	Na webovej stránke sa zobrazili iba tie dáta z XML súboru, ktoré boli filtrom vybraté.

Nové webové rozhranie prekladača

Chceme implementovať viacero moderných prvkov do rozhrania prehliadača. Cieľom je využívať možnosti škálovania prekladača ako aj zrýchliť odozvu servera na používateľove dopyty.

Analýza

Súčasná funkcionalita webového rozhrania prekladača je zabezpečovaná iba na strane servera prostredníctvom PHP. To spôsobuje, že pri každom preklade sa musí stránka znovu načítať. Keďže stránka rozhrania je v súčasnosti veľmi jednoduchá, toto načítavanie nespôsobuje významné zdržanie. V budúcnosti ak sa však rozhodneme pridať nejaké grafické komponenty na stránku, nebude ich potrebné načítavať opätovne pri každom preklade. To určite skráti čakanie používateľa na zobrazenie odpovede servera. S využitím technológie AJAX by sa pri preklade načítali znova dáta iba do tých častí, kde sa zobrazujú prekladané výrazy. Tento prístup úspešne využíva aj konkurenčný prekladač Google Translate. Jeho rozhranie je aj vďaka tomu veľmi rýchle na odozvu.

Samotný preklad je v súčasnosti vykonávaný tak, že text zadávaný do textového poľa sa odošle funkcii, ktorá vráti jeden najlepší preklad danej vety. Pre zobrazenie alternatívnych prekladov je volaná iná funkcia, ktorá vráti všetky ostatné získané preklady. Spustením prekladu sa znovu načíta aj celá stránka prekladača. Aby sme na stránke popri prekladu zobrazili aj pôvodný prekladaný zdrojový text, zobrazí sa v ňom obsah globálnej premennej $_POST[‘type_here’], do ktorej bol text po spustení prekladu uložený.

Návrh

Používateľ má mať možnosť vybrať, s ktorou verziou prekladača chce vykonať preklad. Pre túto činnosť navrhujeme pridať do rozhrania rozbaľovacie menu, ktoré dynamicky načíta zoznam dostupných verzii z danej URL. Použijeme pritom nejaké dostupné efekty s využitím JavaScript knižnice jQuery a CSS.

Na zrýchlenie samotného prekladu zavoláme iba raz funkciu, ktorá zabezpečí preklad a jej prvú hodnotu zobrazíme v rámčeku pre najpravdepodobnejší preklad. Prípadné ďalšie hodnoty zobrazíme v časti pre alternatívne preklady. Výsledok zobrazíme na stránke, bez potreby jej znovu načítania. Pomocou technológie AJAX načítame iba získané dáta do častí, ktoré zobrazujú výsledky prekladu.

Implementácia

Dostupné verzie prekladača sú do rozbaľovacie menu načítavané priamo z webovej služby. Vytvorením novej verzie prekladača je táto verzia automaticky viditeľná aj v rozhraní a to bez potreby akéhokoľvek zásahu do zdrojového kódu rozhrania. Z URL adresy zobrazujúcej JSON je pomocou natívnej PHP funkcie file_get_contents("URL") získaný obsah, ktorý je následne skonvertovaný do asociatívneho reťazca. Jednotlivé hodnoty tohto reťazca sú zobrazené v príslušných HTML elementoch a tvoria rozbaľovacie menu a aj obsah pomocníka pre verzie prekladača.

Samotné odoslanie požiadavky s prekladaným textom na server vykonávame s použitím technológie AJAX. Pre vykonanie AJAX volaní a následné spracovanie odpovedí zo servera využívame JavaScript knižnicu jQuery. Vzhľadom na fakt, že volaná webová služba beží na inom porte ako rozhranie prekladača, nie je možné vykonať toto volanie na službu priamo. Namiesto toho pomocou AJAX voláme, pre tento účel vytvorený, súbor s názvom service_connect.php. Jedinou úlohou tohto súboru je zabezpečenie komunikácie s príslušnou webovou službou. Keďže chceme prekladať aj veľké úryvky textu využívame pri tom HTTP metódu POST. Pre jej využitie v PHP bolo potrebné použiť niektoré curl funkcie dostupné v PHP knižnici libcurl.

Využitím tohto prístupu na obídenie tzv. cross-site scripting prevencie sme docielili, že rozhranie prekladača je jednoducho prenositeľné. To všetko bez potreby menenia nejakých serverových nadstavení. Stačí totiž zmeniť iba dva linky v PHP súboroch a rozhranie môže byť implementované na inom serveri.

Samotná odpoveď webovej služby na zadaný prekladaný text je vlastne JSON obsahujúci príslušné preklady. JSON je u klienta skonvertovaný na asociatívny reťazec a jeho prvé hodnoty sú zobrazené v rámčeku pre najviac pravdepodobný preklad. Ostatné hodnoty reťazca sú zobrazené v časti pre alternatívne preklady. Celé toto spracovanie prebieha v klientskom okne prehliadača pomocou JavaScript-u.