Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Mikołaj Dądela
Nr albumu: 262484

Corthus – korpus równoległy z interfejsem internetowym i przeszukiwarką fonetyczną

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dra hab. Adama Przepiórkowskiego
Instytut Informatyki

Wrzesień 2012

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

W swojej pracy stworzyłem korpus równoległy z tekstów liturgicznych używanych przez Cerkiew Prawosławną w językach polskim, cerkiewnosłowiańskim i starogreckim. Korpus można przeglądać i przeszukiwać poprzez dwa interfejsy: wiersza poleceń oraz internetowy. Przeszukiwanie opiera się na przybliżonej wymowie słów, aby można było było znaleźć słowo, którego zna się tylko wymowę.

Słowa kluczowe

tłumaczenie, alignment, teksty równoległe, wyszukiwanie fonetyczne, metaphone, Python

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

H. Information Systems
H.3 Information Storage and Retreival
H.3.1 Content Analysis and Indexing

Tytuł pracy w języku angielskim

Corthus - a parallel corpus with internet interface and a phonetic search engine

Spis treści

Rozdział 1

Wprowadzenie

W swojej pracy stworzyłem korpus równoległy z tekstów liturgicznych używanych przez Cerkiew Prawosławną w językach polskim, cerkiewnosłowiańskim i starogreckim. Korpus można przeglądać i przeszukiwać poprzez dwa interfejsy: wiersza poleceń oraz internetowy. Przeszukiwanie opiera się na przybliżonej wymowie słów, aby można było było znaleźć słowo, którego zna się tylko wymowę.

1.1. Cel

Głównym celem mojej pracy było ułatwienie szerszej publiczności dostępu do tekstów liturgicznych w różnych językach i sprawienie, żeby były zrozumiałe dzięki możliwości równoległego ich czytania i łatwo dostępne dzięki wyszukiwarce.

Pragnąłem również ułatwić zainteresowanym naukę języka cerkiewnosłowiańskiego, poczynając już od pierwszego jej etapu, którym jest nauka czytania. Chciałem również poprzez zestawienie tekstów unaocznić analigoczny, a często identyczny szyk zdań w językach starogreckim i cerkiewnosłowiańskim.

Trwa obecnie dyskusja na temat zasadności tłumaczenia tekstów liturgicznych na język polski i poprawności istniejących tłumaczeń - chciałem więc stworzyć pomoc do ich badania, jak również do potencjalnego tłumaczenia kolejnych podobnych tekstów na język polski.

1.2. Wizja

Planowałem stworzyć serwis internetowy, w którym można łatwo przeglądać, przeszukiwać i porównywać teksty w różnych językach, w pisowni oryginalnej lub w transkrypcji fonetycznej na język polski.

Ważnym elementem projektowanego serwisu była przeszukiwarka wspomnianych tekstów. Głównym jej celem była nie tyle precyzja wyszukiwania, co łatwość jej użycia, gdyż pisownia cerkiewnosłowiańska i starogrecka jest na tyle skomplikowana, że nie można założyć, że użytkownik bezbłędnie wpisze w wyszukiwarkę frazę z szukanego tekstu.

Ciekawą możliwością dalszego rozwinięcia projektu byłaby implementacja rytmu kalendarza prawosławnego i udostępnienie linku „tekst na dzisiaj”.

Główne cele użycia serwisu byłyby takie, jak wymieniłem w rozdziale 1.1 - starałem się jednak stworzyć na tyle ogólną implementację, żeby można było jej

później użyć do innych zastosowań, np. stworzenia bazy tłumaczeń tekstów technicznych. Niektóre algorytmy w sposób nieunikniony używają różnych wersji dla trzech języków, z których składała się moja baza, ale nic nie stoi na przeszkodzie, żeby dodać analogiczne funkcje do obsługi innych języków.

Rozdział 2

Przygotowania i eksperymenty

Zacząłem swoją pracę od poszukiwań gotowych narzędzi, które odpowiadałoby przynajmniej części założeń opisanych powyżej. Postaram się tu krótko opisać swoje doświadczenia z nimi.

2.1. Hunalign

Hunalign jest narzędziem zestawiającym 2 teksty na poziomie zdań (zakładając, że są już podzielone na zdania), bazując na ich długościach i (opcjonalnie) na słowniku. Jeśli jednak nie dysponujemy słownikiem, można uruchomić Hunaligna z opcją `–realign` - Hunalign wówczas po pierwszym, wstępnym dopasowaniu próbuje wygenerować przybliżony słownik, a następnie dopasowuje teksty ponownie z użyciem tego słownika. Dopasowania generowane przez Hunaligna są niemal w 100% poprawne, jeśli teksty dokładnie sobie odpowiadają. Niestety teksty, z którymi pracowałem, bardzo często różniły się od siebie dużymi fragmentami tekstu, co powodowało przesunięcia w dopasowaniu.

Mimo to Hunalign okazał się bardzo przydatny, gdyż, bazując głównie na długości zdań, generuje dopasowanie, które może zostać wykorzystane do wygenerowania przybliżonego słownika fraz, który następnie może posłużyć do polepszenia dopasowania.

2.2. GIZA++

GIZA++ [10] jest narzędziem tworzącym dopasowanie na poziomie słów z już dopasowanych par zdań. Nie użyłem jej w mojej pracy, ponieważ przede wszystkim potrzebowałem dopasowania na poziomie zdań. Dopasowanie na poziomie słów jest jednak poza zakresem niniejszej pracy (chociaż byłoby niewątpliwie ciekawym i użytecznym rozszerzeniem tej pracy).

2.3. Poliqarp

Poliqarp [12] jest darmowym (swobodnym) zestawem narzędzi do przeszukiwania dużych korpusów. Wspiera obsługę korpusów otagowanych znacznikami morfosyntaktycznymi i pozwala na przeszukiwanie korpusu za pomocą złożonych

zapytań. Jednak teksty, którymi się zajmowałem w swojej pracy, nie były otagowane, więc nie mógłbym też wykorzystać pełni możliwości Poliqarpa. Poliqarp nie posiada również obsługi korpusów równoległych.

2.4. Wzorce odmiany i lematyzacja

Próbowałem uprościć Hunalignowi generowanie przybliżonego słownika poprzez usunięcie końcówek morfosyntaktycznych ze słów. Nie znalazłem jednak gotowej biblioteki, wykonującej takie zadanie. Rozważałem ręczną implementację tej funkcjonalności, ale po przyjrzeniu się wzorcom odmiany języka cerkiewnosłowiańskiego [13] zrezygnowałem. Początkowo przybliżałem lematyzację skracaniem słowa do 5 liter, a po implementacji Metaphone zauważyłem, że Metaphone również łączy pod wspólnym kluczem wiele form fleksyjnych jednego słowa, więc w końcowej wersji projektu dopasowuję do siebie nie oryginalne teksty, lecz ciągi kluczy Metaphone z nich utworzone.

2.5. Słowniki

Szukałem również słownika cerkiewnosłowiańsko-polskiego, lecz znalazłem tylko jeden [14], autorstwa O. Aleksego Znosko, i nie udało mi się go zdobyć w wersji cyfrowej - na mail do wydawcy otrzymałem odpowiedź, że autor zmarł przed wydaniem tego słownika, i plik z zawartością słownika nigdzie się nie zachował.

Rozdział 3

Architektura projektu

Cała implementacja została napisana w języku Python.

3.1. Katalog *toolkit* - główna funkcjonalność

TODO

toolkit/translit

Wiele skryptów z tego folderu można wywołać z linni komend, aby wykonać różne zadania (patrz rozdział 5).

3.2. Katalog *www* - interfejs internetowy

Interfejs internetowy został szczegółowo opisany w rozdziale 6.

3.3. Katalog *texts* - baza tekstów

Katalog *texts* zawiera wszystkie teksty.

Moja decyzja o używaniu plików tekstowych zamiast bazy danych była powodowana prostotą operacji na plikach w konsoli, i możliwością łatwej zmiany w organizacji tekstów. Łatwość zmian była dla mnie bardzo ważna, bo w początkowej fazie rozwoju projektu nie było jasne, w jaki sposób system będzie korzystał z tekstów.

Nazwa pliku z tekstem źródłowym ma postać `texts/`*książka*`/`*rozdział*`/`*język*`.txt` , gdzie *język* jest jednym spośród “pl”, “cu” [1] i “el”.

3.4. Katalog *external* - zewnętrzne biblioteki i narzędzia

Katalog *external* zawiera zewnętrzne biblioteki, z których korzysta projekt. Na chwilę obecną są to Hunalign [11] i Whoosh [**?**].

[1] “cu” oznacza język cerkiewnosłowiański, zgodnie ze standardem ISO 639-1.

Rozdział 4

Przygotowanie danych

4.1. Konwersja danych wejściowych

Teksty pobrałem ze stron [1], [2] i [3]. Pliki z tych stron były odpowiednio w formatach HIP [1] , HTML i DOC.

Zdecydowałem wszystkie te pliki przekonwertować do zwykłego tekstu kodowanego w UTF-8. Napisałem w tym celu skrypt w Pythonie, który korzystał dodatkowo z Elinksa i AbiWorda. Zdecydowałem się na kodowanie UTF-8, bo dzięki niemu mogłem używać jednego kodowania dla wszystkich plików, i uznałem, że to uproszczenie rekompensuje 2 razy większy rozmiar pliku. Decyzja o przejściu na zwykły tekst oznaczała również rezygnację z formatowania. Połamałem akapity na linie nie dłuższe niż 70 znaków, rozdzielając je pustymi liniami (tak jak w TeXu), tak żeby można było je wygodnie przeglądać w dowolnym edytorze tekstu lub w konsoli.

4.2. Dopasowywanie tekstów

Na początku potrzebne było dopasowanie tekstów na poziomie plików. Nie było to trywialne, bo teksty były różnie zredagowane, i bardzo często jeden plik odpowiadał kilku plikom w innym języku. Wykonałem to zadanie ręcznie, z pomocą prostego skryptu dzielącego plik na części w odpowiednich miejscach.

Stworzyłem wstępne dopasowania Hunalignem.

Z nich wydorębniłem pary tłumaczeń, które się powtarzały (bardzo duża poprawność dopasowań na tych parach!!!).

Następnie sprawdziłem, ile procent zdań stanowią te, które udało się w ten sposób sparować. Okazało się, że to ok. 20%!

TODO: ALGORYTM alignera!!! (i może wykresy)

TODO: Dopasowanie 3 języków!

TODO: WYNIKI!!! ewaluacja!!!

[1] HIP - standard zapisu tekstu cerkiewnosłowiańskiego. Więcej szczegółów w dodatku A.1

4.3. Transformacje tekstu (pakiet `translit`)

Aby poprawnie wyświetlić tekst w języku cerkiewnosłowiańskim, potrzebna była specjalna czcionka, która oprócz liter zawierała również zestaw ligatur, potrzebnych do poprawnego ułożenia znaków diakrytycznych nad konkretnymi literami. Przygotowanie tekstu do wyświetlenia w takiej czcionce również wymagało napisania prostego skryptu.

Język cerkiewnosłowiański w zapisie często występujących słów korzysta ze skrótów. Polegają one zazwyczaj na opuszczeniu samogłosek (zazwyczaj tytło proste, ◌҃) lub środkowej części wyrazu (zazwyczaj tytła literowe,◌᷉ ◌᷊ ◌᷎). Język cerkiewnosłowiański korzysta ponadto z własnego systemu zapisu liczb. Wobec tego, aby dokonać transkrypcji fonetycznej takiego tekstu, należało najpierw rozwinąć wszystkie skróty i zamienić liczby na cyfry arabskie. Napisałem w tym celu moduł `translit.expand_cu`, zawierający listę 130 różnych skrótów i obsługujący liczby do 10 tysięcy (większe liczby nie występują w tekstach). Aby konwersja mogła działać wydajnie, skompilowałem wszystkie wzorce skrótów do jednego wyrażenia regularnego.

Następnie zaimplementowałem transkrypcję fonetyczną na polski w module `cu2pl`. Największy problem sprawiło mi tutaj dostosowanie wyniku do polskiej pisowni (przede wszystkim chodziło tu o litery „i” i „j” w różnych kontekstach).

Zaimplementowałem również transkrypcję fonetyczną z języka starogreckiego na polski. Dokonuję tej transkrypcji jednak zgodnie z regułami wymowy współczesnej Greki, dlatego, że w ten sposób są zwykle czytane. Konwersja zachodzi w dwóch etapach: najpierw upraszczam zapis, usuwając znaki przydechu (dasia, prosgegrammeni, ypogegrammeni) i zamieniając wszystkie rodzaje akcentu (tonos, oxia, varia, perispomeni) na jeden (tonos), i dopiero wtedy dokonuję zamiany greckich liter i dyftongów na polskie odpowiedniki fonetyczne. Pierwszy etap konwersji wykonuje moduł `translit.simplify_el`, a drugi `translit.el2pl`.

4.4. Metaphone

Do rozmytego wyszukiwania według wymowy słów potrzebowałem algorytmu, działającego podobnie do Soundex [ref], generującego dla słów klucze odpowiadające przybliżonej wymowie. Z jednej strony Soundex był do tego celu zbyt mało dokładny, z drugiej strony transkrypcja fonetyczna wszystkich tekstów na polski nie pozwoliłaby na znalezienie słowa, jeśli zostało błędnie wpisane w wyszukiwarkę.

Zainspirował mnie algotytm Metaphone [ref]. Nie zastosowałem jednak dokładnie tego algorytmu, lecz stworzyłem jego wersję dostosowaną do moich danych[2]. Działanie algorytmu dla danego słowa przedstawia się następująco:

1. Jeśli dane słowo jest liczbą, zwróć tę liczbę dodając “#” na początku.

2. Jeśli język nie został podany, wykryj język tekstu. Jeśli tekst zawiera znaki, których nie udało się rozpoznać, zwróć “?”.

[2]W dalszej części pracy nazywam tę wersję po prostu “metaphone”

Tabela 4.1: Uproszczona tablica podstawień dla mojej wersji algorytmu Metaphone

metaphone	pl	cu	el
a	a	ѧ	α
e	e, ę	є, ѣ, ѣ	ε, αι
i	i, y	ѷ, и, ы, і	ι, η, υ
o	o, ą	ѡ, ѻ	ο, ω
u	u, ó	ꙋ, ѹ	ου
b	b	в	μπ
c	c, ć, cz	ц, ч, ҷ	τσ
d	d, dz, dż	д	δ
f	f	ф, ѳ	φ, θ
h	g, h, ch	г, ҳ	γ, χ
j	j	й	
k	k	к	κ
l	l	ль	λ
7	ł	л	
m	m	м	μ
n	n, ń	н	ν
p	p	п	π
r	r	ꙗ	ρ
s	s	ѕ	σ, ς
t	t	т	τ
v	w	в, ѵ	β, υ
z	z, ź	ꙁ, ѕ	ζ
2	ż, rz, sz, ś	ж, ш	

3. Użyj tablicy podstawień dla danego języka (patrz tabela 4.1). Uproszczona wersja tej tablicy znajduje się poniżej. W języku cerkiewnosłowiańskim dodatkowo następuje tu rozwijanie skrótów i konwersja liczb. Jeśli okazało się, że mieliśmy do czynienia z liczbą, zwracamy jej wartość poprzedzoną znakiem “#”.

4. Usuń wszystkie samogłoski, zaczynając od drugiej litery wynikowego ciągu znaków.

4.5. Wyszukiwarka

Lucene, szuka po kluczach

Rozdział 5

Interfejs wiersza poleceń

Większość modułów pakietu `toolkit` udostępnia swoją funkcjonalność poprzez prosty interfejs linii komend.

5.1. `fetcher`

Fetcher pobiera odpowiedni tekst lub dopasowanie tekstów i wypisuje go (w ładnie sformatowanej postaci) na standardowe wyjście.

5.2. `search`

Skrypt przeszukuje bazę tekstów i wyświetla wynik. Wywołanie go z opcją `--create-index` powoduje utworzenie indeksu potrzebneg do realizacji wyszukiwania. Wyszukiwarka opiera się na pakiecie Whoosh [16].

5.3. `aligner`

TODO!

5.4. `Alignment`

Skrypt Alignment.py zawiera klasę obsługującą dopasowania wielu tekstów. Wywołany samodzielnie z wiersza poleceń, wyświetla podany plik dopasowania.

Przykład użycia:

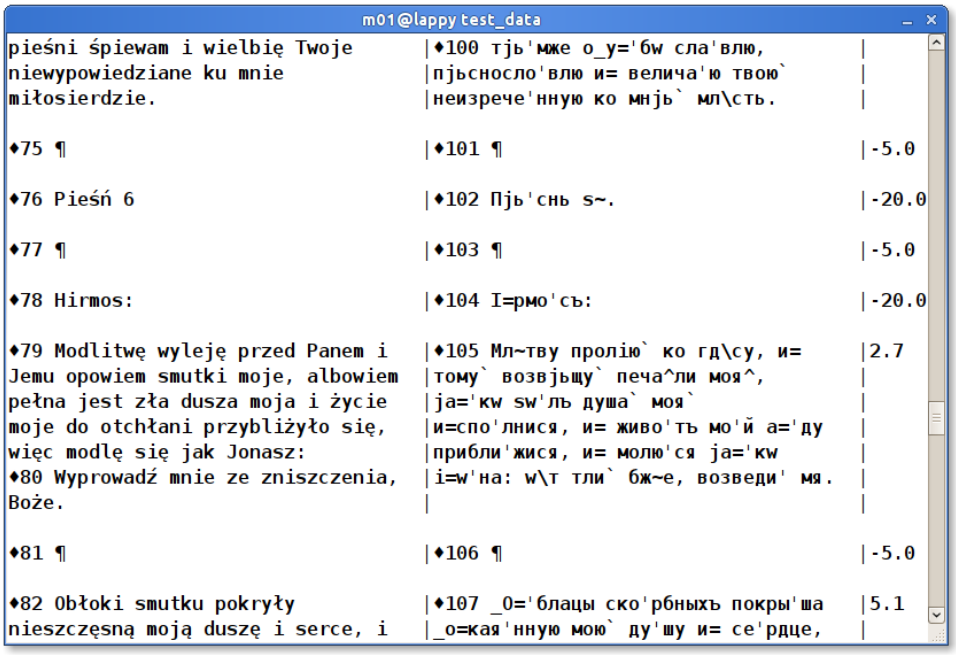
```
./toolkit/Alignment.py test_data/kanon_izr.pl-cu.golden
```

Fragment przykładowego wyniku działania skryptu można zobaczyć na rysunku 5.1.

5.5. Skrypty z pakietu `translit`

Wszystkie skrypty z pakietu `translit` można uruchomić z linii poleceń, aby dokonać odpowiednią transliterację na wybranym tekście. Jako argument należy podać plik z tekstem, lub –, aby korzystać ze standardowego wejścia.

Rysunek 5.1: Przykładowe dopasowanie tekstów wyświetlone w konsoli.



- `expand_cu.py` – rozwijanie skrótów języka cerkiewnosłowiańskiego i konwersja liczb na cyfry arabskie,
- `cu2pl.py` – transkrypcja fonetyczna języka cerkiewnosłowiańskiego na polski,
- `render_cu.py` – konwersja tekstu cerkiewnosłowiańskiego do formatu UCS (patrz dodatek A.1),
- `simplify_el.py` – ujednolicanie greckich akcentów i usuwanie innych znaków diakrytycznych,
- `el2pl.py` – transkrypcja fonetyczna z (“uproszczonego”) greckiego na polski,
- `metaphone.py` – zamiana tekstu na ciąg kluczy Metaphone.

Rozdział 6

Interfejs internetowy

Do utworzenia interfejsu internetowego użyłem frameworka Django [15]. Zdecydowałem się na ten wybór, ponieważ cały kod mojego projektu został napisany w Pythonie, a Django jest najpopularniejszym frameworkiem do tworzenia aplikacji internetowych w Pythonie, i miałem z nim styczność podczas zajęć na MIM UW.

Interfejs internetowy można uruchomić do przetestowania wpisując polecenie `./www/manage.py runserver` w terminalu w głównym katalogu projektu. Powoduje to uruchomienie serwera HTTP, który odpowiada na zapytania pod adresem `http://127.0.0.1:8000/`.

Wygląd przykładowej strony można zobaczyć na rys. 6.1.

6.1. Model

Model danych i operacje na nich obsługuje napisany przeze mnie pakiet `toolkit`. Nie potrzebowałem używać baz danych, które zwykle są pomocne w podobnych projektach.

6.2. Widoki

Cały serwis internetowy obsługują 4 widoki.

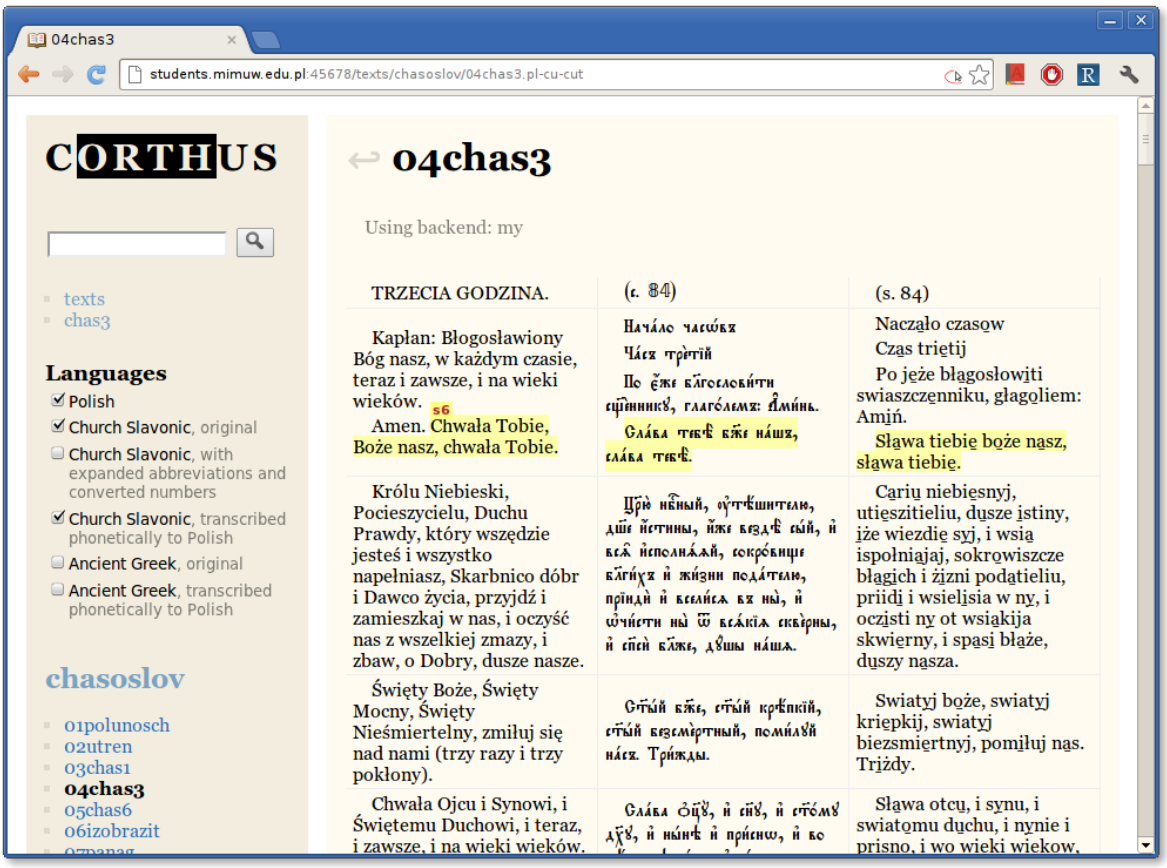
Widok główny - strona główna, TODO!

Widok folderu Wyświetla spis treści książki lub spis książek. To zadanie sprowadza się do wyświetlenia odpowiednio sformatowanej listy plików.

Widok tekstu Wyświetla tekst równoległy. Po lewej stronie znajdują się pola wyboru języków, których zmiana powoduje przeładowanie strony z nowymi ustawieniami.

Widok wyników wyszukiwania

Rysunek 6.1: Widok tekstu w interfejsie internetowym.



6.3. Szablony

Szablony są prostymi plikami HTML, korzystającymi z zewnętrznych arkuszy stylów CSS i skryptów w JavaScripcie.

Aby umożliwić przeglądanie tekstów cerkiewnosłowiańskich osobom, które nie mają zainstalowanej w systemie odpowiedniej czcionki, użyłem deklaracji `@font-face` z języka opisu formy prezentacji stron WWW CSS. Deklaracja ta wprowadzona została dopiero w wersji 3 standardu, dlatego może nie działać w starszych przeglądarkach. Jednak nowe wersje wszystkich wiodących przeglądarek (Seamonkey 2.7.2, Opera 10.60, Firefox 3.6, Safari 4.0, Internet Explorer 9) obsługują ją poprawnie. Starsze wersje Internet Explorera wyświetlają odstępy pod znakami diakrytycznymi, jednak zawsze można ten problem obejść, wyświetlając tekst w transkrypcji fonetycznej.

Zmiana języków jest obsługiwana przez krótki skrypt w języku JavaScript. Preferencje użytkownika są zapisywane w pliku cookie, i odczytywane przy następnej wizycie.

Podświetlanie odpowiadających sobie fragmentów tekstu jest również obsługiwane przez skrypt JavaScript.

Dodatek A

Formaty plików

A.1. HIP i UCS - tekst cerkiewnosłowiański

W czasach, kiedy więszość tekstów cerkiewnosłowiańskich została zdigitalizowanych, standard Unicode nie zawierał wszystkich znaków potrzebych do zapisu tekstu w języku cerkiewnosłowiańskim. W odpowiedzi na to powstał standard HIP (opisany dokładnie w internecie na stronie [4]). Pliki HIP to pliki tekstowe kodowane w Windows-1251, używające cyrylicy wraz z niektórymi literami łacińskimi, aby w ten sposób wyrazić wszystkie litery alfabetu cerkiewnosłowiańskiego. Zapis poszczególnych liter alfabetu (różne warianty zapisu) zostały przedstawione w tabeli A.1.

Brakujące znaki były stopniowo dodawane do Unicode w wersjach 5.0.0 (2006), 5.1.0 (2008) i 5.2.0 (2009). Można więc obecnie zapisać tekst cerkiewnosłowiański w Unicode - głównym problemem jest jednak dostępność czcionek: czcionki obsługujące tytła literowe (znaki `2DE0–2DFF`) i literę ꙗ (`A64B`) są wielką rzadkością. Na chwilę obecną udało mi się w internecie znaleźć tylko jedną czcionkę obsługującą w pełni język cerkiewnosłowiański, oznaczoną jako “wersja beta” [5]. Używam wobec tego starszych czcionek, dostosowanych do kodowania UCS. Kodowanie UCS jest oparte na Windows-1251. Znaki cyrylicy są w nim kodowane tak samo, jednak wszystkie pozostałe pozycje strony kodowej są zajęte przez pozostałe znaki języka cerkiewnosłowiańskiego oraz ligatury tych znaków z tytłami lub znakami diakrytycznymi (tylko w zestawieniach, gdzie takie ligatury są potrzebne).

Dużą wadą takiego kodowania jest to, że jest ono niekompatybilne z ASCII, i tak zakodowany tekst jest czytelny tylko jeśli zostanie wyświetlony odpowiednią czcionką. Przykładowe czcionki dla tego kodowania można pobrać ze strony *Irmologion* [6].

A.2. TXT - pliki tekstowe

Wszystkie pliki TXT w projekcie są kodowane w UTF-8. Zdecydowałem jednak nie konwertować tekstów cerkiewnosłowiańskich z zapisu HIP na znaki odpowiadające im wedug standardu Unicode, ze względu na problemy z wyświetlaniem tych znaków (patrz sekcja A.1). Ta decyzja pozwoliła mi na duże uproszczenie logiki programu, bo mogłem bardzo łatwo przeglądać te pliki i wprowadzać w nich zmiany. Wadami tego rozwiązania była konieczność rezygnacji z formatowa-

Tabela A.1: Kodowanie znaków cerkiewnosłowiańskich w standardzie HIP

A a	Ѧ ѧ
Б б	Ѣ ѣ
В в	Ⱒ ⱒ
Г г	Ґ ґ
Д д	Ⅾ ⅾ
Е е	Ꙥ ꙥ
_E _e <E> <e>	Ꙥ ѥ Ꙥ ѥ
Ж ж	Ж ж
S s	Ꙁ ꙁ
З з	З з
И и	Ꙇ и
I i	І ї
_i <i>	ı ı
Й й	Й й
К к	К к
Л л	Л л
М м	М м
Н н	Н н
О о	Ѳ ѳ
_O _o <O> <o>	Ѻ ѻ Ѻ ѻ
W w	Ѡ ѡ
_W _w <W> <w>	Ѽ ѽ Ѽ ѽ
П п	П п
Р р	Ꝑ ꝑ
С с	Ꙅ ꙅ
Т т	Т т
У у	Ꙋ ꙋ
_У _у <У> <у>	ᲂ ᲂ ᲂ ᲂ
О_У О_у о_у О<У> О<у> о<у>	Ꙋ Ꙋ ꙋ Ꙋ Ꙋ ꙋ
Ф ф	Ф ф
Х х	Х х
W\т w\т	Ѿ ѿ
Ц ц	Ц ц
Ч ч	Ч ч
Ш ш	Ш ш
Щ щ	Щ щ
Ъ ъ	Ъ ъ
Ы ы	Ы ы
Ь ь	Ь ь
Jь Jь jь	Ъь Ъь ъь
Ю ю	Ю ю
JA Ja ja	Ꙗ Ꙗ ꙗ
Я я	Ѧ ѧ
_КС _Кс _кс <КС> <Кс> <кс>	ҀҀ ҀҀ ҁ ҀҀ ҀҀ ҁ
_ПС _Пс _пс <ПС> <Пс> <пс>	Ѱ Ѱ ѱ Ѱ Ѱ ѱ
F f	Ф ф
V v	Ѵ ѵ

nia i nieco większy rozmiar plików.

Połamałem akapity na linie nie dłuższe niż 70 znaków, rozdzielając akapity od siebie pustymi liniami (analogicznie do TeXa), tak żeby można było je wygodnie przeglądać w dowolnym edytorze tekstu lub w konsoli.

A.3. Pliki dopasowań

Dopasowania tekstów są zapisywane w formacie CSV, z tabulatorem pełniącym rolę separatora pól. Jest to dokładnie taki format, jaki zwraca Hunalign.

Nazwy plików dopasowań kończą się rozszerzeniem `.hunalign`, `.my` lub `.golden`, w zależności od swojego pochodzenia. Kolejne rozszerzenia oznaczają odpowiednio pliki wygenerowane przez Hunaligna, pliki utworzone przez aligner z projektu i pliki dopasowań utworzone ręcznie.

Bibliografia

[1] Teksty cerkiewnosłowiańskie:
http://orthlib.ru/worship/

[2] Teksty starogreckie:
http://analogion.gr/glt/

[3] Teksty polskie:
http://www.liturgia.cerkiew.pl/page.php?id=14

[4] Opis standardu HIP (w jęz. rosyjskim):
http://orthlib.ru/hip/hip-9.html

[5] Projekt Ponomar
http://www.ponomar.net/cu_support.html

[6] Czcionki z ligaturami potrzebne do wyświetlania tekstów w jęz. cerkiewnosłowiańskim:
http://www.irmologion.ru/fonts.html

[7] The Unicode Consortium. *The Unicode Standard*.
http://www.unicode.org/charts/PDF/U0400.pdf (Cyrillic)
http://www.unicode.org/charts/PDF/U2DE0.pdf (Cyrillic Extended-A)
http://www.unicode.org/charts/PDF/UA640.pdf (Cyrillic Extended-B)

[8] Algorytm Soundex:
http://en.wikipedia.org/wiki/Soundex

[9] Algorytm Metaphone:
http://en.wikipedia.org/wiki/Metaphone,
Hanging on the Metaphone, Lawrence Philips. Computer Language, Vol. 7, No. 12 (December), 1990.

[10] GIZA++
http://code.google.com/p/giza-pp/
Franz Josef Och, Hermann Ney. *"A Systematic Comparison of Various Statistical Alignment Models"*,
Computational Linguistics, volume 29, number 1, pp. 19-51 March 2003.

[11] Hunalign
http://mokk.bme.hu/resources/hunalign/
D. Varga, L. Németh, P. Halácsy, A. Kornai, V. Trón, V. Nagy (2005). *Parallel corpora for medium density languages*, In Proceedings of the RANLP 2005, pages 590-596.

[12] Poliqarp
http://poliqarp.sourceforge.net/about.html
Adam Przepiórkowski. (2004). *Korpus IPI PAN. Wersja wstępna*. IPI PAN, Warszawa.

[13] Stanisław Strach, *Krótka gramatyka języka cerkiewnosłowiańskiego*. Prawosławna Diecezja Białostocko-Gdańska, 1994.

[14] Aleksy Znosko, *Słownik cerkiewnosłowiańsko-polski*, Prawosławna Diecezja Białostocko-Gdańska, 1996.

[15] Django core team (2011). *Django: A Web framework for the Python programming language*. Django Software Foundation, Lawrence, Kansas, U.S.A.
http://www.djangoproject.com

[16] Whoosh Python Search Library
https://bitbucket.org/mchaput/whoosh/wiki/Home