Wykorzystanie Emacsa, Haskella i TEX-a w pracach nad słownikiem języka staro-cerkiewno-słowiańskiego

Halina Wątróbska, Ryszard Kubiak
finhv@univ.gda.pl, R.Kubiak@guests.ipipan.gda.pl

Streszczenie

W Katedrze Slawistyki Uniwersytetu Gdańskiego powstaje dwujęzyczny słownik staro-cerkiewno-słowiańsko-polski. Jego podstawą jest rękopis XIII-wiecznego egzegetycznego frolilegium, czyli wyboru tekstów autorów chrześcijańskich, z komentarzem. Wszystkie formy wyrazowe z tego rękopisu mają zostać objaśnione w hasłach słownika.

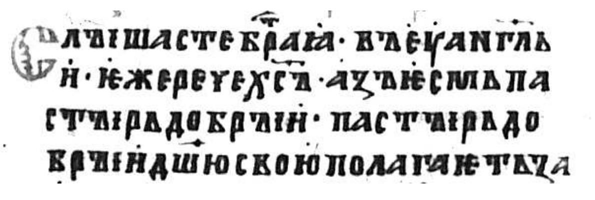
Autorka słownika wprowadziła rękopis do komputera w postaci pliku tekstowego, w którym średniowieczne znaki cyrylickie tak zwanego ustawu zakodowała za pomocą znaków dostępnych na typowej klawiaturze komputera. Oryginalny tekst oznakowała dodatkową informacją o podziale na wiersze, strony i jednostki logiczne, tak zwane incipity. Do przetwarzania danych w takiej reprezentacji znakomicie nadają się tytułowe narzędzia programistyczne: edytor Emacs, język programowania Haskell oraz system składu TEX.

TEX oczywiście służy do eleganckiego prezentowania informacji słownikowej. Konieczne do tego jest przekodowanie znaków cyrylickich z ich reprezentacji klawiaturowej na kody w foncie cyrylickim. Takie przekodowanie może być przeprowadzane w foncie wirtualnym, który w tym celu specjalnie opracowano.

Wstęp

W Katedrze Slawistyki Uniwersytetu Gdańskiego powstaje dwujęzyczny słownik staro-cerkiewno-rusko-polski. Jego autorką jest prof. Halina Wątróbska. Podstawą słownika jest XIII-wieczny zabytek piśmiennictwa starosłowiańskiego [1], który ma charakter *egzegetycznego florilegium*, czyli wyboru tekstów autorów chrześcijańskich, z komentarzem. Koncepcja słownika jest taka, że mają w nim zostać objaśnione wszystkie formy wyrazowe spotykane w zabytku. Form takich jest ponad 11 tysięcy.

W niniejszym artykule zabytek, liczący 392 strony, będziemy w skrócie nazywali Izbornikiem. Oto fragment strony 59v: Do tego fragmentu jeszcze



kilka razy wrócimy w niniejszym artykule, w którym zamierzamy przedstawić nasze doświadczenia z wykorzystania narzędzi komputerowych do prac nad słownikiem.

Komputerowa reprezentacja materiału źródłowego

Pełny tekst Izbornika został ręcznie przepisany do komputera, z użyciem opracowanej przez autorkę słownika notacji. Każdy znak cyrylicki jest w tej notacji kodowany jednoznaczną kombinacją znaków z klawiatury komputera, ściślej – kombinacją znaków z zakresu ASCII. Tym kodowaniem objęte zostały wszystkie znaki używane w średniowiecznej cyrylicy.

Plik komputerowy z tekstem Izbornika zawiera informacje o tym, gdzie zaczynają się kolejne strony i kolejne wiersze. Ciekawostką dla niejęzykoznawców może być to, że trzeba było oznaczyć granice wyrazów, gdyż w średniowiecznych tekstach takiego podziału nie było. Zapisu, w którym wyrazy są rozdzielane odstępami, zaczęto używać dopiero w pierwszej połowie XVII wieku.

W oryginale Izbornika występuje natomiast wydzielenie fragmentów zdań – za pomocą znaku kropki, która odczytującemu tekst, a właściwie go wyśpiewującemu wskazywała, gdzie może on zrobić przerwę na oddech. Za pomocą tego samego znaku wyrażano to, do czego dzisiaj używamy rozmaitych znaków interpunkcyjnych: kropek, przecinków, myślników itp. Oczywiście wszystkie kropki z rękopisu zachowano w reprezentacji komputerowej.

W reprezentacji komputerowej autorka wprowadziła też znaczniki podziału tekstu na jednostki strukturalne: strony, tytuły części rękopisu i tak zwane incipity, oznaczające początki fragmentów tekstu, które dzisiaj nazwalibyśmy wypisami.

Oto komputerowy zapis powyższego kilkuwierszowego fragmentu Izbornika:

```
<I 571> Slyshaste bra/t/ja. v7 euangl//6-
i. jezhe reche 'Xs//7. az7 jesm6 pa-
styr6 dobryi. pastyr6 do-
bryi dsh//ju svoju polagajet6 za
ovc8. sam7 stvori 'Xs//7 soboju.
```

Widoczne są: oznaczenie incipitu o numerze 571, wyraźny podział na słowa, łacińska transliteracja znaków cyrylickich, przenoszenie wyrazów, konwencje notacyjne do wyrażania elementów graficznych, jak `/t/` oznaczająca wyniesienie litery т oraz konwencja `//` do oznaczania tyld, używanych do skracania zapisu słów.

Dobór narzędzi do przetwarzania

Programów komputerowych, które można by wykorzystać do prac nad słownikiem, jest tak wiele, że bardzo trudno jest podjąć decyzję, które z nich zastosować. Na ten problem natknęła się też autorka słownika, która kilka lat temu zdecydowała się zapisywać hasła słownika w edytorze Word Perfect, popularnym wówczas konkurencie edytora MS Word. W edytorach tego typu kuszące jest to, że już podczas edycji widzimy na ekranie tekst w postaci zbliżonej do tej, która ukaże się w druku: są wytłuszczenia i inne wyróżnienia, czcionka mniejsza i większa, są fragmenty w cyrylicy obok zapisanych w alfabecie łacińskim, itp. Początkowy entuzjazm użytkownika takich narzędzi szybko jednak ustępuje narastającemu zdenerwowaniu, że nie wszystkie efekty graficzne daje się uzyskać, a na dodatek, że poświęca on coraz więcej czasu nie na pracę merytoryczną, lecz właśnie na graficzne formatowanie materiału. Coraz trudniejsze staje się też korygowanie już utworzonego tekstu. Autorka słownika natknęła się właśnie na tę trudność, jak też na to, że edytor Word Perfect praktycznie wyszedł z użycia i konieczna stała się konwersja całego materiału słownikowego do innego formatu.

Jasne się stało, że w doborze narzędzi komputerowych trzeba patrzeć w dłuższej perspektywie i że trzeba szukać narzędzi które zapewnią lepszą jakość zarówno samej pracy nad słownikiem, jak też efektu końcowego. Korzystając z porad informatyków, w tym współautora niniejszego artykułu, autorka słownika zdecydowała się skorzystać w dalszych pracach z edytora Emacs i systemu składu TEX. Podstawową zaletą obu tych narzędzi jest to, że są one programowalne w znacznie szerszym zakresie niż wspomniane edytory typu WYSIWYG. Programowalność jest tu bardzo potrzebna, bo w merytorycznej pracy nad słownikiem możliwość doboru czcionek w celach prezentacyjnych ma w istocie znaczenie marginalne, znacznie ważniejsze jest, by można było materiał komputerowo przetwarzać na wiele sposobów.

Z wykorzystaniem programowalności narzędzi jest taka trudność, że wymaga ona przygotowania informatycznego, znajomości języków programowania i doświadczenia w ich używaniu. Wynikł stąd naturalny podział pracy: autorka słownika koncentruje się na pracy merytorycznej, zlecając informatykowi opracowywanie dodatkowych narzędzi, potrzebnych na różnych etapach prac.

Konteksty wystąpień wyrazów

Narzędziem, które okazało się niezwykle pomocne w pracach nad słownikiem, jest program do wyszukiwania kontekstów, w których dany wyraz występuje w Izborniku. Istotne jest przy tym, że wszystkie wystąpienia wyrazu możemy odszukać w bardzo krótkim czasie, rzędu sekundy lub kilku. Znając te wystąpienia, możemy się łatwiej zorientować w znaczeniu wyrazu i opracować właściwe hasło słownikowe. Dysponujemy też od ręki przykładami użycia objaśnianego hasła, z których jeden lub kilka możemy zamieścić w słowniku.

Wystąpienia wyrazu można wyszukiwać podczas sesji pracy z Emacsem. W tym celu po ustawieniu się na wyrazie wywołujemy odpowiednią komendę. W reakcji Emacs zleca wyszukanie wyrazu zewnętrznemu programowi, który analizuje Izbornik i wyszukuje w nim wyraz. Każde wystąpienie wyrazu wraz z jego kilkuwierszowym otoczeniem jest umieszczane w pliku zewnętrznym, którego zawartość Emacs pokazuje, dodatkowo podkolorowując ważniejsze miejsca. Oto zasadniczy fragment programu, odpowiedzialnego w Emacsie za wywoływanie szukania kontekstów:

```
(defun find-Izb-contexts ()
 (interactive)
 (let ((call-buffer-name (buffer-name)))
  (message (concat "Konteksty dla formy: "
                   (current-word)))
  (shell-command (concat konteksty-exe
                   " " (current-word)))
  (set-buffer (get-buffer-create
                   "*Konteksty*"))
  (izb-mode)
  (goto-char (point-max))
  (let ((insert-point (point)))
```

```
        (insert-file (concat work-dir
                             "Tmp/znajdy.izb"))
    (if (not (string= call-buffer-name
                            "*Konteksty*"))
        (switch-to-buffer-other-window
             (get-buffer "*Konteksty*")))
   (goto-char insert-point))))
```

Program ten zapisano w języku ELisp, przeznaczonym do programowania rozszerzeń Emacsa. Klauzula `interactive` oznacza, że procedura `find-izb-contexts` ma być poleceniem Emacsa, czyli że użytkownik uzyskuje prawo do jej wywoływania podczas sesji pracy z edytorem. Zapamiętywana jest nazwa bufora aktywnego w chwili wywołania szukania, a u dołu ekranu wyświetlana jest informacja o szukanym słowie, które będzie słowem `current-word` – tym, na którym w chwili wywołania ustawiony jest kursor. Szukane słowo jest przekazywane – za pomocą procedury `shell-command` – do zewnętrznego programu `konteksty-exe`. Wyniki swojej pracy program ten zapisuje w zewnętrznym pliku `znajdy.izb`, który Emacs wczytuje – procedurą `insert-file` – do roboczego bufora `*Konteksty*`. Jeśli ten bufor wcześniej nie istniał, to jest tworzony i przełączany w tryb `izb-mode`, zapewniający podkolorowywanie ważniejszych fragmentów tekstu. Na koniec następuje przełączenie się Emacsa na pracę w buforze `*Konteksty*`, w miejscu `insert-point` – początku znalezionych kontekstów.

Oto, w jaki sposób Emacs pokaże wystąpienie słowa `bra/t/ja` (bracia) na stronie 59v z powyższego przykładu:

```
== bra/t/ja <L 2585> <F 59v> <V 8> <I 571>
imouwtemou . OH+ ’ EUANGL//6*JA
SKAZAN6*JE STG//O ’ GRIGOR6*JA PAPY
Slyshaste |bra/t/ja| . v7 euangl//6i
. jezhe reche ’ Xs//7 . az7 jesm6 pastyr6
dobryi . pastyr6 dobryi
```

W Emacsie, już bez udziału programów zewnętrznych, zaprogramowano też inną funkcję: przejścia do miejsca w Izborniku, określonego przez kontekst. Po ustawieniu się w buforze z kontekstami na jednym z nich wywołujemy klawiszem funkcyjnym akcję otwarcia pliku Izbornika wraz z ustawieniem się na kontekście. Z tej możliwości możemy skorzystać, gdy kilka wierszy kontekstu nie wystarcza do ustalenia znaczenia słowa i chcemy poznać jego szersze otoczenie.

Indeks form wyrazowych

Przewiduje się, że słownik będzie się składał z dwóch zasadniczych części: właściwego słownika, z polskimi tłumaczeniami wyrazów staro-cerkiewno-ruskich, oraz z indeksu, w którym każda forma wyrazowa zostanie podana z odpowiadającym jej jednym bądź kilkoma hasłami. Jak we wszystkich wydawnictwach słownikowych, hasła będą występowały w ich formie podstawowej, to znaczy rzeczowniki w mianowniku, czasowniki w bezokoliczniku itd. Z kolei formy wyrazowe w indeksie będą wiernymi cytatami z Izbornika. Taki układ słownika będzie pomocny w dalszych pracach językoznawczych nad Izbornikiem, gdyż czytelnik będzie mógł łatwo ustalić znaczenie każdej formy.

Oczywiście użycie komputera do przygotowania słownika w takiej postaci ma wielkie znaczenie. Co więcej, oprócz wydania książkowego nietrudno będzie uzyskać wydanie elektroniczne. Jego użytkownik, analizując wyświetlony na ekranie fragment Izbornika, będzie mógł ustawić się na konkretnym wyrazie i uzyskać objaśnienie tej formy dotrzeć do innych miejsc w Izborniku, w których występuje ta forma lub jej pokrewne.

Jest utartą praktyką, że indeksy są tworzone w ostatniej, redakcyjnej fazie pisania książek lub innych publikacji. Przed autorem stoi wówczas zadanie ustalenia, które wyrazy mają się znaleźć w indeksie. Ten na ogół trudny problem decyzyjny ma w naszym wypadku radykalne rozwiązanie: w indeksie mają się znaleźć wszystkie formy wyrazowe, które występują w Izborniku. Zrąb takiego indeksu został już komputerowo wygenerowany; niewielki program wyszukał występujące w Izborniku formy wyrazowe umieścił je w porządku alfabetycznym w pliku dyskowym. Na obecnym etapie autorka pracuje głównie na pliku indeksu, dopisując do każdej formy wyrazowej hasło słownikowe, w którym zostanie ona objaśniona, a w niektórych wypadkach również krótki komentarz o polskim znaczeniu słowa. Oto niewielki fragment tego indeksu:

```
brat6ja       brat6ja (brać) albo brat7
brat6je       brat6ja
bra/t/ja      brat7 (brat)
brateh        brat7
brach6ny      brach6n7
brashna       brash6no (jedzenie)
```

Haskell jako język obróbki danych słownikowych

Język programowania rozszerzeń, w który wyposażony jest edytor Emacs, chociaż można w nim wyrazić praktycznie wszystko to, co da się wyrazić w językach ogólnego zastosowania jak C lub Pascal, nie jest najwygodniejszy w użyciu. Język ten, nazwany ELispem, jest odmianą Lispu – języka, którego prototyp powstał we wczesnych latach sześćdziesiątych. Zarówno pod względem składni, jak też koncepcji wyrażania myśli programistycznej Lisp bardzo odbiega od języków współczesnych. I chociaż ELisp świetnie się nadaje do sterowania strukturami wewnętrznymi Emacsa oraz do programowania kontaktów Emacsa z programami zewnętrznymi, jest wiele zadań, które znacznie wygodniej jest realizować w bardziej nowoczesnej technologii.

Stanęła kwestia, jaki język najlepiej nadawałby się do zapisu programów przetwarzających dane słownikowe. Wybór padł na język Haskell, na przekór modzie na języki obiektowo zorientowane, jak Java, czy języki skryptowe, jak Perl. O wyborze Haskella zadecydowała jego nowoczesna konstrukcja matematyczna, dostępność polimorfizmu oraz funkcji wysokich rzędów. Cechy te pomagają zwięźle i ściśle formułować najrozmaitsze przekształcenia danych.

Poniżej cytujemy fragment już powstałego oprogramowania okołosłownikowego – zrąb głównej funkcji do wyszukiwania kontekstów wyrazów w Izborniku. Zaczynamy od definicji Haskellowego typu danych, opisującego jednostki leksykalne, wydzielane w pliku źródłowym Izbornika, za każdym razem, kiedy trzeba przeanalizować jego zawartość. W typie `Token` występuje dziesięć rodzajów jednostek:

```
type Label = String
data Token
  = TWord String      -- słowo Izbornika
  | TOther String     -- tekst bez znaczn.
  | TBook Label       -- tytuł całości
  | TSection Label    -- tytuł części
  | TSubsection Label -- ozn. psalmu
  | TPhrase Label     -- ozn. incipitu
  | TNewPage Label    -- nr strony
  | TNewLine Integer  -- nr wiersza pliku
  | TNewVers Integer  -- nr wersa strony
  | TNewUnit Integer  -- nr wersa pliku
```

Funkcja wyszukująca konteksty wyrazu w Izborniku jest następującego typu:

```
findContexts
  :: IzbWord  -- wyszukiwane słowo
  -> Int      -- żądana wielkość otoczenia
  -> Izbornik -- tekst źródłowy
  -> [(Context, Neighbourhood)]
```

Wynik funkcji jest parą. której pierwszy element to abstrakcyjny kontekst, zawierający informacje o pozycji wyrazu w tekście: numer wiersza pliku, numer strony, numer wersu, czyli widocznego wiersza na stronie oraz oznaczenie incipitu. Drugi element wynikowej pary to fragment Izbornika, w jakim znaleziono wyraz. Wielkość tego fragmentu – liczbę wierszy przed i za znalezionym wyrazem – określa drugi parametr funkcji `findContexts`. Oto definicje typów danych, użytych do wyrażenia typu funkcji `findContexts`:

```
type Izbornik = String
type IzbWord = String
type Context = [Token]
type Neighbourhood = ((String, [String]),
                      IzbWord,
                      (String, [String]))
```

Konstrukcja typu `Neighbourhood` ma prawo – w kontekście wcześniejszych wyjaśnień – wydawać się tu najbardziej tajemniczą. W trójkach `((u, us), w, (v, vs))`, będących elementami tego typu `w` jest szukanym wyrazem, `u`, to części znalezionego wersu przed słowem `w`, a `v` – część tego wersu za słowem `w`. Lista `us` wiersze poprzedzające znaleziony wers, a napisy `vs` to wiersze następujące po nim.

Definicję funkcji `findContexts` można zapisać jako złożenie kilku innych funkcji:

```
findContexts word k
    = map (processPreContext k)
    . filter (sameWord word . head . snd)
    . rts []
    . glueHyphens
    . verser (1, 1)
    . lexer
    . liner
```

Znak kropki gra tu rolę znanego ze zwykłej matematyki symbolu złożenia funkcji:

$$(f \circ g)(x) = f(g(x))$$

Odczytujemy to tak, że argument x złożenia funkcji f i g jest najpierw przekazywany do g, a do wyniku jej zadziałania stosowana jest funkcja f.

W naszym wypadku wejściowy ciąg znaków – typu `Izbornik` – jest przekazywany najpierw funkcji `liner`, która do tekstu dodaje oznaczenia wierszy. Następnie funkcja `lexer` dzieli plik wejściowy na jednostki leksykalne, na których działa z kolei funkcja `verser`, numerująca widoczne wiersze na stronach.

Aby ułatwić wyszukiwanie wyrazów w tekście, następnym krokiem – wykonywanym za pomocą funkcji `glueHyphens` – jest połączenie fragmentów wyrazów przenoszonych. Z grubsza chodzi o to, że z przykładowych cząstek `leh-` i `to` następuje połączenie w formę `lehto-3`, gdzie cyfra 3 wskazuje miejsce przeniesienia. Tak więc funkcja `glueHyphens` nie gubi informacji o przeniesieniu wyrazu, co się przydaje, kiedy trzeba przekształcić znalezione otoczenie wyrazu z powrotem do postaci tekstowej.

Wywoływana kolejno funkcja `rts` wykonuje coś, czego w tradycyjnych językach programowania raczej byśmy starali się uniknąć – z powodu narzutu w kodzie źródłowym, potrzebnego na organizację struktur pomocniczych. Z listy elementów funkcja `rts` buduje listę par – dla każdego elementu odwrócony ciąg tych, które go poprzedzają, oraz resztę, z nim włącznie. Na przykład z listy `[1, 2, 3]` zostałaby utworzona lista `([], [1, 2, 3])`, `([1], [2, 3])` i `([2, 1], [3])`. Funkcja `rts` jest zdefiniowana polimorficznie, tak że można jej użyć do list elementów dowolnego typu, w naszym wypadku – do listy jednostek leksykalnych.

Wynik działania funkcji `rts` – lista par list jednostek leksykalnych Izbornika – jest następnie przeszukiwany funkcją `filter`. Wybrane zostają te pary `u, v`, w których na początku listy `v` jest szukane słowo. Lista `u` to kontekst „wsteczny” słowa, czyli to, co w Izborniku jest przed słowem, czytane od końca. Lista `v` to reszta Izbornika, zaczynająca się od szukanego słowa. Pozostaje prezentacja tak wyszukanych wystąpień słowa, za pomocą funkcji `processPreContext`.

Powyższy fragment pozwoliliśmy sobie zacytować jako znakomitą ilustrację sposobu formułowania rozwiązań w Haskellu. Istotnie zostały wykorzystane takie elementy języka jak polimorfizm i dostępność funkcji wyższych rzędów. Istotna z obliczeniowego punktu widzenia jest też jeszcze jedna właściwość Haskella: obliczenia odbywają się w trybie „leniwym”, to znaczy każde wyrażenie jest ewaluowane dopiero wówczas, gdy jego wynik jest potrzebny do uzyskania wyniku ostatecznego. W naszym wypadku zasada ta ma ogromny wpływ na to, jak realizowane jest złożenie funkcji. Powoduje ona, że każda cząstka wyniku działania funkcji `liner` jest od razu przekazywana do wykorzystania funkcji `lexer`, która z kolei każdą wygenerowaną przez siebie jednostkę leksykalną natychmiast przekaże dalej. Przypomina to znany z systemów typu Unix mechanizm *pipe-lining*.

XML, TEI

Jak wspomnieliśmy, graficzna prezentacja materiału to właściwie tylko jeden z aspektów prac nad słownikiem. W świecie publikacji elektronicznych od dawna wiadomo, że przygotowanie ostatecznej prezentacji warto oddzielić od opracowania warstwy informacyjnej. W ostatnich latach do zapisywania wyników prac w warstwie informacyjnej popularne stało się użycie standardu XML. Jest to metoda podawania informacji wraz ze znacznikami, które ją klasyfikują. Siłą standardu XML są proste, a zarazem ścisłe i ujednolicone reguły oznakowywania informacji. Istotne w XML-u jest to, że zestaw znaczników nie został ustalony raz na zawsze – czyli dla wszystkich dokumentów XML-owych, które już utworzono lub które powstaną w przyszłości – lecz że można je dobierać odpowiednio do wykonywanych zadań.

Pisząc dokument XML-owy, zawsze musimy wiedzieć, z jakiego konkretnego zestawu znaczników wolno nam korzystać. Używając nomenklatury XML-owej – musimy wiedzieć, jaka jest Definicja Typu Dokumentów (ang. *Document Type Definition*), w skrócie DTD.

W wypadku słownika zdecydowaliśmy już, że jego warstwa informacyjna zostanie zapisana w standardzie XML, nie podjęliśmy jednak jeszcze decyzji, którą DTD się posłużymy. Mamy w zasadzie dwie możliwości. Pierwsza to skorzystać ze standardowej DTD, opracowanej przez TEI Consortium [2]. TEI DTD ma tę zaletę, że opracował ją zespół profesjonalistów, przez wiele lat pracujących nad rozwojem standardu XML i dostosowywaniem go do potrzeb naukowców-humanistów. W zestawie znaczników tej DTD znajduje się spora część przeznaczona do zapisywania informacji słownikowej.

Inna rozpatrywana przez nas możliwość to opracować własną DTD, opartą na podzbiorze TEI DTD. Taka DTD byłaby skrojona na miarę zadania, które przed nami stoi. To, że nie byłoby w niej całego bogactwa znaczników, dostępnego w TEI DTD, może mieć ten korzystny skutek, że wymuszałoby to większą dyscyplinę i precyzję oznakowywania informacji słownikowej. Zadanie utworzenia DTD, dostosowanej do specyfiki konkretnego zadania – nie jest trudne dla profesjonalisty-informatyka, dysponującego w tym względzie bogatą dokumentację standardu XML.

Niezależnie od tego, która z wersji DTD zostanie ostatecznie przyjęta, autorka słownika będzie dysponowała znakomitym narzędziem do wprowadzania informacji słownikowej do komputera. Tym narzędziem jest tryb `nxml` edycji dokumentów XML w edytorze Emacs. Tryb ten prowadzi niejako piszącego za rękę, na bieżąco wskazując, których spośród znaczników dostępnych w DTD, można użyć w danym miejscu dokumentu. Jeśli się pomylimy w znakowaniu dokumentu, Emacs natychmiast o tym poinformuje.

Po zapisaniu słownika w standardzie XML, trzeba go przetworzyć do postaci publikacyjnej, czyli przygotować skład. Do uzyskiwania składu zostanie wykorzystany TEX, potrzebne będzie jednak oprogramowanie, które zapis XML będzie potrafiło zamienić na dane w postaci, którą zaakceptuje TEX.

Szczęśliwie istnieje biblioteka funkcji w języku Haskell, za pomocą której nie jest trudno zapisać niezbędne przekształcenia danych, wykorzystując przy tym znaczną część kodu, który powstał na potrzeby generowania kontekstów wyrazów w Izborniku.

Wieloaspektowe zagadnienie fontowe

Zadanie opracowania prezentacji graficznej słownika wymaga podjęcia ważnej decyzji o tym, jakiego fontu należy użyć do pokazywania tekstów staro-cerkiewno-słowiańskich. Nie nadają się do tego fonty opracowane dla współczesnego języka rosyjskiego, gdyż średniowieczne znaki cyrylickie bardzo odbiegają kształtem od dzisiejszych. Niektóre dźwięki, posiadające jeszcze w średniowieczu pisemne odpowiedniki, nie występują w dzisiejszym języku rosyjskim. Na przykład znany nam dźwięk, zapisywany literą ę, występował w języku staro-cerkiewno-słowiańskim, zapisywany w postaci znaku ѧ, nie ma go już jednak w języku rosyjskim.

O doborze fontu decydować będą dwa czynniki. Po pierwsze powinny się w nim znajdować wszystkie potrzebne znaki. Po drugie chcielibyśmy, aby teksty staro-cerkiewno-słowiańskie zostały w słowniku zaprezentowane w postaci graficznie jak najbliższej oryginałowi.

Autorka słownika zakupiła kilka lat temu komercyjny font Izhitsa typu TrueType, moskiewskiej firmy Paragraf. Niestety zawiera on nie wszystkie znaki, a brakujące trzeba pobierać z innego komercyjnego fontu tej samej firmy – Evangelie. Niestety pod względem kształtu liter fonty te znacznie się między sobą różnią, co fatalnie rzutuje na jakość składu wyrazów z kombinacją znaków z obu fontów.

W archiwum CTAN można znaleźć MetaFontowe źródła fontu o nazwie izhitsa, opartego graficznie na Izhitsy, ale zawierającego już wszystkie znaki staro-cerkiewno-ruskie, które występują w Izborniku. Oto zacytowany we wstępie fragment w jego reprezentacji komputerowej, zapisanej z użyciem izhitsy:

Слышасте бра҃тїа. въ еѵангл҃ьи. ѥже рече Х҃съ. азъ ѥсмь пастырь добрыи. пастырь добрыи д҃шю свою полагаѥть ꙁа овцѧ. самъ створи Х҃съ собою.

Niestety nie jest jasna sytuacja prawna izhitsy, z czego zdaje sobie sprawę autor fontu, który zwrócił się do zarządzających CTAN o usunięcie izhitsy z archiwum.

Na eleganckiej stronie Internetowej Irmologion [3] można znaleźć komercyjny zestaw PostScriptowych fontów, z bogactwem znaków cyrylickich. Pod względem kroju znaki w tych fontach odpowiadają jednak piśmiennictwu z wieku XVII i czasów jeszcze późniejszych, znacznie więc różnią się kształtem od liter w Izborniku.

W tej sytuacji może się okazać, że nasze dwa życzenia będzie mógł spełnić jedynie font specjalnie zaprojektowany i dostosowany do specyfiki zabytku. Decyzja o wyborze fontu ciągle jeszcze przed nami. Na rozstrzygnięcie czeka też kwestia, czy powinniśmy zdefiniować font wirtualny, by TEX-owi można było bezpośrednio przekazywać znaki staro-cerkiewno-ruskie w ich transkrypcji łacińskiej. Z użyciem takiego fontu zapis `lehto` byłby przekształcany przez TEX-a automatycznie w **лѣто**. W konstrukcji fontu wirtualnego wykorzystany zostałby mechanizm ligatur do przekodowania kombinacji jak `eh` z zapisu `lehto` na pojedyncze znaki z fontu docelowego.

Plan dalszych działań

Prace nad słownikiem są już bardzo zaawansowane, ale też wiele pozostaje jeszcze do zrobienia. Dobiegają końca prace nad indeksem, z którego w następnym etapie zostanie automatycznie wygenerowany XML-owy dokument – zrąb słownika. Również sam indeks uzyska postać dokumentu XML-owego. Oba dokumenty autorka wypełni zasadniczą treścią słownikową, posługując się oprogramowaniem, które pomoże efektywnie pracować na wielu dokumentach. Równocześnie będą się toczyły prace nad konwersją dokumentów XML-owych do TEX-a, tak aby uzyskać skład książkowy. Przygotowane zostanie też oprogramowanie, za pomocą którego słownik stanie się dostępny w postaci elektronicznej w Internecie.

Literatura

[1] Izbornik XIII w., sygn. QpI18 ze zbiorów Biblioteki Publicznej w Petersburgu.

[2] `[http://www.tei-c.org]` 2004:3:9.

[3] `[http://www.mtu-net.ru/irmologion]` 2004:3:9.