Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Tomasz Olejniczak
Nr albumu: 236111

Obsługa formatu PDF/A na potrzeby dygitalizacji tekstów

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
prof. dra hab. Janusza S. Bienia
Katedra Lingwistyki Formalnej

Czerwiec 2011

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 pracy przedstawiono stworzony zestaw narzędzi (PDFAUtilities) do obsługi formatu PDF/A: przeglądarkę struktury dokumentów, konwerter do formatu hOCR oraz walidator PDF/A.

Słowa kluczowe

PDF/A, PDF, hOCR, konwersja, walidacja, układ strony, struktura logiczna dokumentu, czcionki

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

I. Computing methodologies
I.7. Document and text processing
I.7.4 Electronic Publishing

Tytuł pracy w języku angielskim

PDF/A support for text digitalization

Nota licencyjna

License Note

Spis treści

Wprowadzenie

Pakiet PDFAUtilities to zestaw narzędzi umożliwiających obsługę plików PDF i PDF/A. W jego skład wchodzi przeglądarka, umożlwiająca podgląd struktury logicznej dokumentów oraz informacji o zawartych w nich czcionkach i obrazach, konwerter plików PDF do formatu hOCR oraz walidator plików w formaci PDF/A.

Format PDF/A jest jednym z najważniejszych formatów rekomendowanych dla bibliotek jako międzynarodowa norma ISO. Jest to format bardzo skomplikowany, jego specyfikacja liczy co prawda jedynie 36 stron, ale sama odwołuje się do specyfikacji PDF liczącej 968 stron.

Celem pracy było zaimplementowanie obsługi tych aspektów formatu, które mają bezpośrednie praktyczne znaczenie w prowadzonych w Polsce pracach dygitalizacyjnych. Niektóre przedstawione rozwiązania znalazły już praktyczne zastosowanie w Katedrze Lingwistyki Formalnej w postaci Korpusu "font-sensitive" dla słownika Lindego[1].

Zamierzeniem przyświecającym powstaniu pakietu była obsługa plików PDF/A tworzonych przez program FineReader[2], wykonujący OCR na skanowanych dokumentach. Zapisuje on w nich informacje o układzie strony dokumentu traktując go jako strukture logiczną. By umożliwić zobaczenie wygenerowanego układu strony została stworzona przeglądarka. Konwerter umożliwia konwersję utworzonych przez FineReder plików na format hOCR, bardziej dogodny do przechowywania informacji o układzie strony. Ponadto format ten jest obsługiwany przez istniejące narzędzia (zob. 3.1) i wygenerowane przez konwerter pliki mogą być wykorzystane do dalszego przetwarzania.

Ponadto pakiet umożliwia wykorzystanie programu `pdf2txt.py` z pakietu PDFMiner do analizy układu strony. Przy konwersji pliku PDF na hOCR taki układ strony może być użyty zamiast zapisanego w strukturze logicznej dokumentu.

Pliki źródłowe w pakiecie PDFAUtilities nie korzystające z oprogramowania na licencji GNU General Public License są udostępniane na licencji Expat[3]. Dotyczy to większości plików, m.in. konwertera i walidatora. Pliki korzystające z oprogramowania na licenji GNU General Public License (przeglądarka) są udostępniane na licencji GNU General Public License version 3.0. Oprócz tego rozszerzenie pakietu FontTools jest udostępniane na licencji Digital Equipment Corporation[4].

[1] http://poliqarp.wbl.klf.uw.edu.pl/extra/linde/index.djvu
[2] http://www.finereader.pl
[3] http://www.jclark.com/xml/copying.txt
[4] http://www.xfree86.org/current/LICENSE5.html

Rozdział 1

Elementy formatu PDF

1.1. Wprowadzenie

PDF jest szeroko używanym formatem zapisu dokumentów. Posiada on bardzo rozbudowaną funkcjonalność, umożliwia m.in. przechowywanie sformatowanego tekstu, grafiki rastrowej i wektorowej oraz multimediów. Szczegółowy opis formatu PDF można znaleźć w [PDF01] i [PDF08], tutaj omówimy jedynie elementy istotne z punktu widzenia niniejszej pracy.

Istnieje kilka wersji formatu PDF. Omówimy tutaj wersję 1.3, opisaną w [PDF01], która jest podstawą formatu PDF/A. Aktualną wersją formatu PDF jest wersja 1.7 (zob. [PDF08]). Jeżeli przetwarzany przez PDFAUtilities plik zawiera elementy z wersji późniejszych niż 1.3 to program wykorzystuje niektóre z nich przy przetwarzaniu czcionek.

1.2. Ogólna struktura dokumentu PDF

Podstawowym elementem dokumentu PDF jest obiekt. Szczegóły niskopoziomowej struktury pliku i sposobu przechowywania obiektów można znaleźć w [PDF01] (rozdziały *3.4 File Structure* oraz *3.2 Objects*).

PDFAUtilities przyjmuje założenie, że fonty i obrazy w słownikach zasobów oraz obiekt pod kluczem *Mask* w obiekcie obrazka są przechowywane jako referencje do obiektów (są reprezentowane w PDFMinerze przez obiekt klasy PDFObjRef). W przeciwnym przypadku program nie zadziała poprawnie. Podobne założenie dotyczy obiektu pod kluczem *StructTreeRoot* w katalogu dokumentu oraz elementów struktury pod kluczem *K* w korzeniu drzewa struktury i elementach strutkury. Jak dotąd nie napotkano na pliki które nie spełniałyby tego założenia, natomiast upraszcza ono implementacje.

Poza tym PDFAUtilities przyjmuje założenie, że wszystkie obiekty maja numer generacji 0, co powinno być prawdą do plików PDF które po utworzeniu nie były edytowane. Ograniczenie to wynika z implementacji pakietu PDFMiner.

Najważniejszym obiektem w dokumencie PDF jest katalog dokumentu (*catalog dictionary*). Jest to słownik zawierający inne obiekty definijujące plik PDF, m.in. korzeń drzewa stron (*root of page tree*, klucz *Pages*) oraz korzeń drzewa struktury (*structure tree root*, klucz *StructTreeRoot*).

1.3. Drzewo stron

Drzewo stron jest podstawową strukturą danych w której przechowywane są informacje o zawartości dokumentu. Korzeń drzewa stron jest węzłem drzewa stron (*page tree node*) za-

wartym w katalogu dokumentu pod kluczem *Pages*. Węzeł drzewa stron to obiekt zawierający tablicę dzieci, które są albo kolejnymi węzłami drzewa stron albo obiektami strony (*page objects*). W ten sposób obiekty stron tworzą liście drzewa.

Obiekt strony jest słownikiem, którego ważniejsze elementy to (klucz i wartość):

Resources
: słownik zasobów, obiekt zawierający zasoby używane przez stronę (m.in. obrazy i czcionki, może być dzielony miedzy wieloma stronami)

MediaBox
: wymiary strony

CropBox
: obszar strony który będzie widoczny, definiuje układ współrzędnych na stronie[1] (element opcjonalny, jeśli go nie ma to traktujemy go jako równego obiektowi spod klucza *MediaBox*[2])

Contents
: zawartość strony, sekwencja operacji graficznych i tekstowych opisująca zawartość strony, lub tablica takich sekwencji (element opcjonalny)

Rotate
: określa o ile stopni obrócona jest strona[3] (element opcjonalny, musi być wielokrotnością 90)

Wszystkie przedstawione powyżej klucze (poza *Contents*) mogą wystąpić też w węźle drzewa stron. Wtedy wszystkie obiekty stron znajdujące się w jego liściach dziedziczą po nim ich wartości.

Jedną z możliwych operacji występujących w sekwencji pod kluczem *Contents* są operacje `BDC`/`BMC` i `EMC` oznaczające odpowiednio początek i koniec zawartości oznaczonej (*marked content*). Zawartość oznaczona może być identyfikowana przez parametr operacji `BDC` w którym znajduje się jednoznaczny identyfikator zawartości w obrębie strony (MCID). Zawartość oznaczona obejmuje wszystkie operacje między `BDC`/`BMC` a `EMC`.

1.4. Struktura logiczna dokumentu

Format PDF umożliwia przechowywanie informacji o strukturze logicznej dokumentów (zob. [PDF01], rozdział *9.6. Logical Structure*). Jest to funkcjonalność opcjonalna. Struktura ma charakter drzewa, którego korzeń reprezentuje cały dokument. Korzeń drzewa struktury jest słownikiem znajdującym się w katalogu dokumentu.

Ważniejsze elementy w słowniku korzenia:

K
: dziecko lub tablica dzieci (element opcjonalny)

ParentTree
: obiekt umożliwiający znajdowanie elementów struktury logicznej dokumentu w których znajduje się dana zawartość oznaczona lub obiekt (element wymagany jeżeli elementy struktury logicznej zawierają jakieś zawartości oznaczone lub obiekty)

[1] PDFAUtilities zakłada, że początek układu współrzędnych jest w lewym dolnym rogu.
[2] PDFAUtilities z góry zakłada, że *CropBox* jest równy *MediaBox*.
[3] PDFAUtilities zakłada, że strona nie jest obrócona

RoleMap
mapowanie typów (nazw) elementów struktury na zbiór standardowych typów (element opcjonalny)

Dziećmi korzenia drzewa są elementy struktury (*structure elements*), słowniki o nieco innej postaci:

S
typ (nazwa) elementu struktury

Pg
strona na której znajduje się całość lub część zawartości elementu (element opcjonalny)

P
ojciec elementu w drzewie struktury

A
obiekt definiujący atrybuty elementu lub tablica takich obiektów (element opcjonalny)

C
klasa atrybutów do której należy element lub tablica takich klas (element opcjonalny)

Lang
język zawartości elementu[4] z wyjątkiem potomków które mają explicite zdefiniowany inny język (element opcjonalny)

Alt
opis tekstowy zawartości elementu niereprezentowalnego w postaci tekstowej (element opcjonalny)

ActualText
tekst równoważny zawartości tekstowej elementu reprezentującej tekst w nietypowy sposób (element opcjonalny)

K
definicja potomków lub zawartości (element opcjonalny)

Pod kluczem *K* mogą się znajdować:

- liczba będąca identyfikatorem zawartości oznaczonej w obrębie strony (MCID) - wtedy zawartością elementu struktury jest fragment strony definiowany przez tą zawartość oznaczoną,
- słownik odnośnika do zawartości oznaczonej (*marked content reference dictionary*) - zawiera on m.in. identyfikator zawartości oznaczonej MCID i stronę na której znajduje się ta zawartość (jeżeli inna niż znajdująca sie pod kluczem *Pg* w elemencie struktury) - wtedy zawartością elementu struktury jest fragment strony definiowany przez tą zawartość oznaczoną,
- słownik odnośnika do obiektu (*object reference dictionary*) - wtedy zawartością elemetu struktury jest obiekt do którego odnosi się słownik (np. obrazek lub adnotacja),

[4]Obecnie PDFAUtilities pokazuje tylko język *explicite* podany w elemencie, nie pokazuje języka dziedziczonego po przodkach.

- inny element struktury (potomek),
- tablica dowolnych wymienionych wcześniej elementów.

PDFAUtilities wspiera jedynie zawartość oznaczoną umieszczoną w liściach drzewa struktury, nie obsługuje natomiast odnośników do obiektów i zawartości oznaczonych w elementach struktury nie będących liśćmi. Na podstawie analizy różnych plików PDF nie powinno to stanowić problemu dla plików składających się z samego tekstu lub skanów i tekstu ukrytego.

1.5. Drzewo *ParentTree*

Drzewo umieszczone w korzeniu drzewa struktury pod kluczem *ParentTree* umożliwia znalezienie elementu struktury do którego należy dany obiekt lub zawartość oznaczona. Szczegółową implementację znajdywania poszukiwanego elementu omówiono w [PDF01], sekcja *Finding Structure Elements from Content Items* w rozdziale *9.6.3 Structure Content*.

Drzewo umożliwia znalezienie:

- Elementu struktury w którym znajduje się dany obiekt po jednoznacznym identyfikatorze umieszczonym w obiekcie pod kluczem *StructParent*.
- Tablicy elementów struktury w których znajdują się zawartości oznaczone umieszczone na pojedynczej stronie po jednoznacznym identyfikatorze umieszczonym w obiekcie tej strony pod kluczem *StructParents*. Indeks elementu struktury w takiej tablicy jest równy identyfikatorowi zawartości oznaczonej (MCID) w obrębie strony.

1.6. Obiekt obrazka

Obiekt obrazka (*image dictionary*) opisuje obraz rastrowy wyświetlany na stronie. Wszystkie obrazki wyświetlane na danej stronie można znaleźć w słowniku zasobów jej obiektu strony.

Obiekt obrazka jest strumieniem PDF. Zawartością strumienia jest właściwy obraz rastrowy, przechowywany np. jako bitmapa, obraz w formacie JPEG lub JBIG2. Oprócz tego w obiekcie znajduje się wiele atrybutów obrazka, np. przestrzeń kolorów (pod kluczem *ColourSpace*) i maska (pod kluczem *Mask*), która sama może być obrazkiem.

1.7. Obiekt czcionki

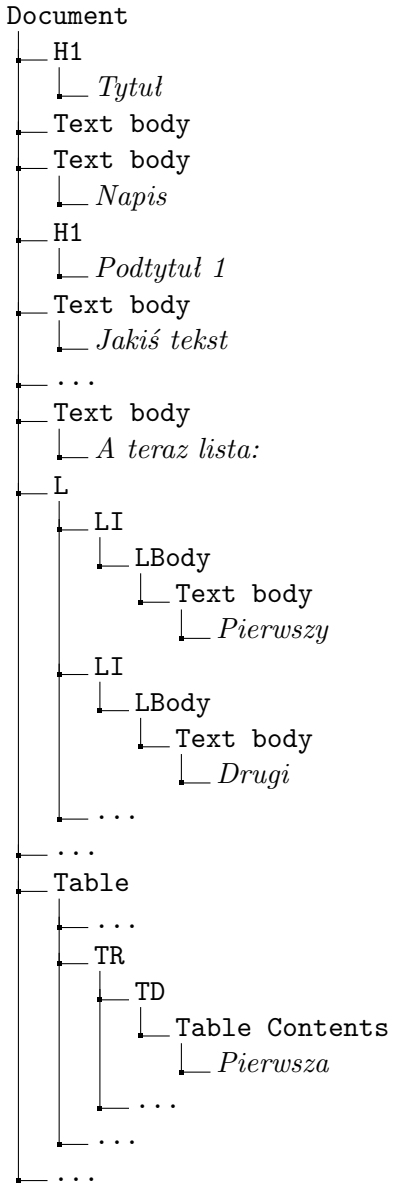
Obiekt czcionki (*font dictionary*) opisuje czcionkę wyświetlaną na stronie. Znajdują się w nim m.in. następujące informacje:

Subtype
: typ fontu (TrueType, Typ 1, Typ 3, Typ 0)

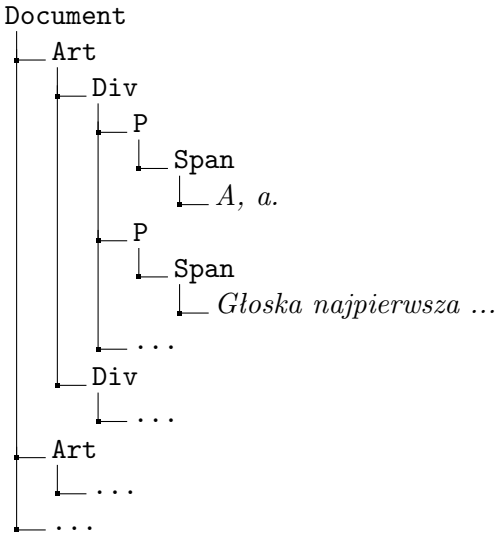
BaseFont
: nazwa postscriptowa fontu (nie ma jeśli font Typu 3)

FontDescriptor
: deskryptor fontu (patrz niżej, element wymagany, z wyjątkiem 14 standardowych fontów, nie ma jeśli font Typu 3 lub Typu 0)

CharProcs
: funkcje rysujące znaki (wymagany dla Typu 3, nie ma dla innych fontów)



Rysunek 1.1: Przykładowa struktura logiczna: dokument eksportowany z OpenOffice (kursywą podano zawartość tekstową elementów, obiekt pod kluczem *RoleMap* mapuje elementy o nazwie *Text body* i *Table Contents* na element standardowy *P*).



Rysunek 1.2: Przykładowa struktura logiczna: układ strony dokumentu eksportowanego z FineReadera (kursywą podano zawartość). *Art* oznacza pojedynczą stronę a *Div* przeważnie kolumnę lub element paginy.

DescendantFont
: definicje fontów składowych (*descendant fonts*, wymagany dla Typu 0, nie ma dla innych fontów)

Encoding
: specyfikacja kodowania fontu (element opcjonalny, wymagany dla Typu 3 i Typu 0)

ToUnicode
: mapowanie kodów znaków w danym foncie na Unicode (element opcjonalny)

PDFAUtilities obsługuje w pełni jedynie fonty TrueType, używane w wynikach FineReadera, choć potrafi też wyświetlać niektóre informacje dla innych fontów.

Wewnątrz deskryptora fontu (*font descriptor*) znajdują się:

FontName
: nazwa postcriptowa fontu, powinna być identyczna jak *BaseFont* w obiekcie czcionki, ale w plikach generowanych przez FineReadera tak nie jest

Flags
: flagi określające styl fontu (PDFAUtilities wykorzystuje informację, czy font jest kursywą)

FontFile
: program fontu Typu 0 (element opcjonalny)

FontFile2
: program fontu TrueType (element opcjonalny)

FontFamily
: rodzina fontu - jest to element obecny w specyfikacji [PDF08], nie ma go natomist w [PDF01], a zatem nie powinno go też być w plikach PDF/A, mimo to FineReader tworząc pliki PDF/A umieszcza w nich ten element (element opcjonalny)

FontWeight
liczba na podstawie której można ustalić czy font jest pogrubiony czy nie - jest to element obecny w specyfikacji [PDF08], nie ma go natomist w [PDF01], a zatem nie powinno go też być w plikach PDF/A, mimo to FineReader tworząc pliki PDF/A umieszcza w nich ten element (element opcjonalny)

Do określenia czy font jest kursywą lub jest pogrubiony. PDFAUtilities wykorzystuje obiekty pod kluczami *FontWeight* oraz *Flags* i, w przypadku fontów TrueType, program *FontFile*. Oprócz tego nazwa postscriptowa fontu jest przeszukiwana pod kątem zawierania w sobie ciągów "Bold", "bold", "Italic", "italic", "Oblique" i "oblique".

Do określenia rodziny fontu używany jest obiekt pod kluczem *FontFamily* oraz, w przypadku fontów TrueType, program *FontFile*. FineReader umieszcza nieprawidłowe nazwy rodziny fontów w programie fontu (są one równe nazwie pod kluczem *FontName* w deskryptorze).

Jeżeli nie można znaleźć rodziny to używana jest nazwa postscriptowa. Jeżeli konwertujemy plik PDF na hOCR możemy podać mapowanie nazw posctscriptowych na rodzinę fontu.

W chwili obecnej konwerter na hOCR wykorzystuje tylko informację o tym czy font jest kursywą z obiektu pod kluczem */Flags*. W razie potrzeby można by go rozszerzyć tak aby potrafił też umieszczać w wynikowym pliku informacje czy font jest szeryfowy, maszynowy (*monospace*) i czy używa kapitalików.

1.8. Tagowany PDF

Tagowany PDF (*Tagged PDF*) to zestaw wytycznych, które powinien spełniać dokument by można było z niego łatwo wyciągać zawartość oraz określać znaki, słowa i kolejność tekstu, zdefiniowanych w [PDF01], w rozdziałach *9.7 Tagged PDF* i *9.8 Accessibility Support*. Poniżej krótko omówiono zakres obsługi tagowanego PDF przez PDFAUtilities. Większość z pominiętych elementów tagowanego PDF jest mało przydatna dla aktualnych prac dygitalizacyjnych.

PDFAUtilities wspiera obsługę opisanego wcześniej drzewa struktury logicznej a także mapowanie (częściowo, ponieważ tylko bezpośrednie) nazw elementów struktury przy pomocy obiektu pod kluczem */RoleMap* w korzeniu drzewa struktury logicznej, pośrednio wspierając w ten sposób standardowe typy (nazwy) elementów.

Ponadto PDFAUtilities wspiera częściowo (tylko dla artefaktów[5] będących paginą) odróżnianie właściwej zawartości od paginy i artefaktów. Według specyfikacji tagowanego PDF artefakty nie powinny znajdować się wewnątrz struktury logicznej, ale w wynikach FineReadera tak się dzieje (jest pagina w strukturze logicznej). Informacje o tym, że dany element jest paginą nie są w żaden sposób eksportowane do hOCR, ale format ten udostępnia taką możliwość.

PDFAUtlities zakłada, że przy czytaniu dokumentu w kolejności struktury logicznej (*logical structure order*) kolejne strony następują po sobie w takiej kolejności jak w dokumencie i każda strona występuje tylko raz. W przeciwnym przypadku konwerter nie wytworzy prawidłowego hOCR.

PDFAUtilities nie wspiera w żaden sposób *soft hyphen* (dzielenie słowa z powodu końca wiersza) oraz nie eksportuje informacji o jej wystąpieniu do hOCR. Podobnie sytuacja ma się z hiperłączami.

PDFAUtilities nie wspiera systemów pisma pisanych od prawej do lewej[6].

[5] Artefakty to elementy nie stanowiące treści dokumentu, np. pagina, znaczniki drukarskie, linia oddzielająca przypisy od właściwego tekstu.

[6] Oraz systemów pisma pisanych pionowo.

PDFAUtilities korzysta z założenia, że tekst w pliku PDF zawiera spacje rozdzielające słowa (jest tak, jeżeli dokument jest tagowanym PDF).

Wsparcie dla paradygmatów struktury (paradygmat silny - struktura logiczna ma postać rozbudowanego wielopoziomowego drzewa, paradygmat słaby - struktura logiczna ma postać drzewa składającego się z korzenia i jego dzieci będących liśćmi) zdefiniowanych w specyfikacji tagowanego PDF jest zapewnione przez możliwość podania pliku z dowolnym mapowaniem (przedstawione poniżej mapowanie domyślne jest zgodne z paradygmatem silnym i nie obsługuje wszystkich nazw standardowych z tagowanego PDF).

PDFAUtilities potrafi wyświetlać atrybuty elementów struktury logicznej w przeglądarce, natomiast w żaden sposób nie próbuje wyeksportować zawartych w nich informacji do hOCR (w szczególności nie próbuje na ich podstawie ocenić, czy lista jest uporządkowana czy nie, patrzy tylko na zawartość elementu o nazwie *Lbl*).

Przy obliczaniu gabarytów (*bounding boxes*, współrzędnych prostokątów w których zawiera się element) elementów struktury PDFAUtilities nie używa algorytmu opisanego w specyfikacji tagowanego PDF, ale po prostu sumuje gabaryty zawartych w nich znaków.

PDFAUtilities nie wspiera komórek tabeli leżących w więcej niż jednym rzędzie lub w więcej niż jednej kolumnie.

PDFAUtilities wyświetla obiekty z pod kluczy *Alt*, *Lang* i *ActualText* ze słownika elementu struktury w przeglądarce. Nie wspiera on jednak obłsugi określania języka z użyciem zawartości oznaczonych. Nie eksportuje też informacji zawartych w tych obiektach do hOCR. Nie wspiera też w żaden sposób obsługi skrótów.

Rozdział 2

Format PDF/A

Format PDF/A to podzbiór formatu PDF przeznaczony do długotrwałego przechowywania dokumentów. Dokładna specyfikacja formatu jest zawarta w [PDFA05] i [PDFA07].

Jego głównym zadania to:

- przechowywanie dokumentów w sposób, który umożliwi jak najwierniejsze odtworzenie ich wyglądu,
- przechowywanie metadanych o kontekście i historii dokumentu,
- przechowywanie informacji o strukturze logicznej dokumentu i innych informacji semantycznych.

Pierwsze z postawionych przed nim zadań format PDF/A wypełnia nakładając dodatkowe ograniczenia na odpowiednie elementy formatu PDF. Drugie z zadań jest realizowane przy pomocy metadanych w formacie XMP (wariant XML opracowany przez Adobe do przechowywania metadanych) zawartych w pliku PDF/A. Trzecie zadanie format PDF/A realizuje dzięki zgodności ze zdefiniowanym wcześniej zestawem wytycznych tagowanego PDF.

Wchodzące w skład pakietu przeglądarka i konwerter zajmują się obsługą elementów formatu dotyczących trzeciego z powyższych zadań, natomiast konwerter częściowo sprawdza, czy dany plik spełnia wymogi PDF/A związane z pierwszym zadaniem.

PDFAUtilities potrafi przetwarzać także dokumenty PDF nie będące PDF/A. Jeżeli dokument nie zawiera struktury logicznej, to nie zostanie ona pokazana w przeglądarce (będzą widoczne jedynie słowniki zasobów) i nie zadziała konwersja do hOCR bez użycia opcji `-p`.

Rozdział 3

Format hOCR

3.1. Zarys formatu hOCR

Format hOCR jest modyfikacją formatu HTML na potrzeby przechowywania wyników hOCR. Wykorzystywany jest m.in. przez programy CuneiForm[1] i OCRopus[2]. Specyfikację formatu hOCR można znaleźć w [Breu].

Format hOCR wykorzystują także narzędzia stworzone na potrzeby prac dygitalizacyjnych w Katedrze Lingwistyki Formalnej UW. Są to:

- `djvu2hocr` - narzędzie eksportujące dane z plików w formacie DjVu (format pliku stworzony z myślą o przechowywaniu skanowanych tekstów ich wyników ich OCR) na format hOCR,
- `hocr2djvused` - narzędzie umożliwia dodawanie do plików DjVu informacji o gabarytach słów z plików hOCR,
- `hocr2xces` - narzędzie umożliwiające konwersję plików hOCR do formatu XCES (wariant formatu XML używany do przechowywania korpusów), który może być potem wykorzystany przez przeglądarkę korpusów Poliqarp[3].

Pliki hOCR generowane przez PDFAUtilities są akceptowane przez wspomniane powyżej narzędzia KLF UW.

Dane w plikach hOCR są zapisywane w drzewie elementów. Elementy i ich atrybuty (zwane właściwościami (*properties*) są zapisywane jako dodatkowe informacje w tagach HTML. Każdy element odpowiada jednemu tagowi HTML. Typ (nazwa) elementu jest zapisywany w atrybucie `class` tagu HTML. Zawartość tagu jest jednocześnie zawartością elementu. Właściwości elementu są zapisywane w atrybucie `title` tagu HTML.

Przykładowo następujący fragment pliku hOCR zapisuje informacje o podziale dokumentu na strony, kolumny i akapity za pomocą odpowiednio elementów *ocr_page*, *ocr_carea* i *ocr_par* zapisanych w tagach `div`. Właściwość *bbox* to gabaryt:

```
<div class="ocr_page">
  <div class="ocr_carea" title="bbox 300 300 600 1400">
    <div class="ocr_par">
      ABREWIACYA, yi. f. skrócenie, pismo prędkie z tytlami, skoropis.
```

[1] http://cognitiveforms.ru/products/cuneiform/, http://en.openocr.org/
[2] http://code.google.com/p/ocropus/
[3] http://poliqarp.sourceforge.net/

```
      Znajdują się w pismach polskich skrócenia czyli abrewiacye, tak przez
      opuszczenie głosek, jako przez zamianę ich na figury. Kopcz. Gr. 1.
      29.
    </div>
    <div class="ocr_par">
      ABROGACYA, yi. f. prawny obrządek, przez który prawo dawniejsze moc
      swoje traci. Kras. Zb. 1. 15. zniesienie prawa, uchylenie, odwołanie,
      cofnienie, cf. Derogacya.
    </div>
    <div class="ocr_par">
      ABSOLUCYA, yi. f. rozgrzeszenie, które wyznającemu grzechy, kapłan
      daje. Kras. Zb. 1. 15. wszelkie odpuszczenie, uwolnienie.
    </div>
  </div>
  <div class="ocr_carea" title="bbox 700 300 1100 1400">
    <div class="ocr_par">
      ABUCHT, a. m. Kontuz, salsenan, mortadella. Xiążę abuchty surowe,
      jeden chleb jadał z drugiemi. Tward. Wł. 192.
    </div>
    <div class="ocr_par">
      ABYSS, ABIS, u. m. miejsce bezdenne, bezdno, bezgruncie, otchłań,
      przetchliny, przepaść. Niech ćma ta nie zaraża powietrza, niech bieży
      do swych abissów, do swoich otchłani. P. Koch. J. 7. 239. I do abyssu
      prędko polecieli nieprzyjaciele wieczni ludzkiej duszy. ib. 241.
    </div>
  </div>
</div>
```

Specyfikacja formatu hOCR podaje zalecenie dla niektórych elementów tagi HTML (np. elementowi *ocr_par* powinien odpowiadać tag `p`), ale jeżeli ich zastosowanie nie jest możliwie lub pożądane można użyć innych tagów (np. na powyższym przykładzie tagu `div`).

Dostępne w hOCR elementy możemy podzielić na dwie grupy: dotyczące struktury logicznej i składu dokumentu. W powyższym przykładzie element *ocr_par* opisuje strukture logiczną. Inne elementy struktury logicznej to np. *ocr_chapter* (rozdział), *ocr_section* (podrozdział), *ocr_title* (tytuł) itp.

Elementy *ocr_page* i *ocr_carea* dotyczą z kolei składu dokumentu. Element *ocr_carea* w tym przykładzie oznacza kolumnę, ale może to być dowolny obszar spójnego tekstu podzielonego na wiersze. Element *ocr_page* musi się znajdować w każdym dokumencie hOCR. Oprócz tych dwóch elementów skład dokumentu opisuje element *ocr_line* (wiersz wewnątrz *ocr_carea*).

W dokumencie mogą występować elementy z obu grup. Ponieważ struktury logiczna i składu dokumentu mogą być ze sobą niezgodne (np. jeden podrozdział jest na kilku stronach, a na jednej stronie są fragmenty dwóch podrozdziałów) elementy mogą być podzielone na kilka części o takiej samej wartości właściwości *groupid*. Wtedy można utworzyć jedną hierarchię (np. podrozdział jest zawsze potomkiem strony, a jeżeli zajmuje więcej niż jedną stronę, to jest podzielony na części mieszczące sie na jednej stronie).

Dodatkowo istnieją elementy specyficzne dla silnika OCR (w naszym przypadku konwertera PDF/A do hOCR). Oznaczają one elementy struktury dokumentu używane przez silnik OCR których nie można reprezentować przu użyciu wymienionej wcześniej grupy elementów. Są to *ocrx_block* (blok tekstu, nie powinien zawierać tekstu z różnych *ocr_carea*), *ocrx_line*

(linia, element jest używany jeżeli definicja linii wykorzystywana przez silnik OCR różni się od wiersza w *ocr_carea* reprezentowanego przez *ocr_line*) i *ocrx_word* (słowo). Ich znaczenie zależy od silnika hOCR.

Informacje o czcionkach są zapisywane w standardowy dla HTML i CSS sposób. Specyfikacja wymaga, by informacje o fontach w postaci atrybutów HTML i CSS umieszczać w tagach będących jednocześnie elementami hOCR (tzn. mających atrybut `class` o wartości *ocr_<typ>* lub *ocrx_<typ>*). Obecnie PDFAUtilities umieszcza tą informację w osobnych tagach `span`, co jest niezgodne ze specyfikacją, ale nie wpływa to na użyteczność programu. Jednak w obrębie np. jednego elementu *ocr_line* lub *ocr_par* może być użytych kilka fontów, co uniemożliwia podanie definicji fontu w ich tagach. Żeby spełnić żądanie ze specyfikacji należałoby informacje o fontach wypisywać w elementach *ocrx_word*. Z drugiej strony w obrębie jednego słowa też mogą być użyte dwa fonty. W takiej sytuacji trzeba by rozbić słowo na części powiązane właściwością *groupid*[4]. Podobne rozwiązanie możnaby też zastosować dla umieszczania informacji o foncie w tagach nie będących elementami *ocrx_word*.

Specyfikacja formatu hOCR nie narzuca użytkownikowi dokładnej struktury pliku. Wybierając lub odrzucając poszczególne elementy i właściwości można uzyskać różne podformaty hOCR. W celu poinformowania narzędzi przetwarzających pliki hOCR o wykorzystywanych możliwościach (*capabilities*) hOCR informacja o tym powinna się znaleźć w nagłówku (`head`) pliku hOCR. Specyfikacja jest niejasna jeśli chodzi o określenie możliwości wypisywania informacji o fontach. Powinno się to odbyć przez podanie możliwości *ocrp_font* w nagłówku, natomiast sformułowane jest to tak, jakby istniała właściwość *font* opisująca font. Tymczasem nigdzie w specyfikacji taka właściwość nie jest wspomniana, użyte są natomiast inne mechanizmy określania fontów. Dlatego PDFAUtilities podaje możliwość *ocrp_font*, mimo tego, że używa innych niż domniemana właściwość *font* sposobów opisywania fontów.

Istnieją dwa standardowe podformaty (profile): fizyczny i logiczny. W profilu fizycznym podstawową hierarchią jest struktura składu dokumentu, natomiast elementy struktury logicznej będące potomkami kilku elementów struktury składu są podzielone na części (por. przykład ze stronami i podrozdziałami powyżej). W profilu logicznym sytuacja jest odwrotna. Dla specyficznych zastosowań można definiować konkretne profile, np. dla książek, gazet lub określonego programu do OCR.

3.2. Domyślny wynikowy format hOCR przy konwersji ze struktury logicznej PDF

Przy konwersji ze struktury logicznej PDF używane są elementy dotyczące struktury logicznej. Standardowe typy tagowanego PDF są mapowane na najbliższe im elementy profilu logicznego (np. *P* na *ocr_par* itd.). Jeżeli jakiś znajduje się na kilku stronach, to jest dzielony na części które są dziećmi elementu typu *ocr_page* reprezentującego stronę (czyli w zasadzie używamy wspomnianego wcześniej profilu fizycznego). Elementy *ocr_page* są bespośrednimi dziećmi tagu `body`.

Informacje o fontach zapisywane są w tagach `span`, w których są one zapisywane w postaci stylu CSS. Dodatkowo opcjonalnie można użyć specjalnych elementów *ocrx_italic* i *ocrx_bold* (są to elementy niestandardowe, nieobecne w dokumentacji hOCR) w oddzielnych tagach `span` oznaczających fragmenty tekstu pochyłego i pogrubionego. Poszczególne słowa są zapisywane w elementach *ocrx_word*.

[4]Co prawda specyfikacja nie przewiduje takiego użycia właściwości *groupid* ale wydaje się to rozsądne jej nadużycie. W chwili obecnej słowa w których jest wiele fontów są rozbijane na części, natomiast właściwość *groupid* nie jest używana.

Obecnie używany format niewiele odbiega od mapowania struktury dokumentu na tagi HTML o nazwie *html_simple* zdefiniowanego w dokumentacji hOCR. Ponieważ można modyfikować mapowanie podając odpowiedni plik z mapowaniem możliwe jest również uzyskanie mapowania zbliżonego do *html_none*. W razie potrzeby do konwertera można by dodać możliwość określania tagów HTML przy użyciu których zapisuje się informacje o fontach, ponieważ obecnie używany format bardziej pasuje do mapowania *html_none* (choć, jak wspomniano powyżej, informacje o fontach są umieszczane w niewłaściwych tagach), natomiast w *html_simple* fonty powinny być określane z użyciem tagów `b`, `i` i `font`. Użycie tagów `b`, `i` jest zalecanym sposobem określania właściwości fontów.

Obecnie konwerter na hOCR nie wspiera mapowania na elementy hOCR oryginalnych nazw elementów (umieszczonych pod kluczem *S*). Niektóre aplikacje mogą różnie traktować różne nazwy mimo że mapują na tą samą nazwę standardową, dlatego też w razie potrzeby należałoby rozważyć rozszerzenie pakietu o taką możliwość. Pozwala tylko na mapowanie nazw na które nazwy elementów mapują bezpośrednio z użyciem obiektu pod kluczem *RoleMap* w drzewie struktury logicznej dokumentu.

FineReader zapisując wyniki hOCR w strukturze logicznej PDF zapisuje tam w rzeczywistości układ strony: element *Art* to w rzeczywistości strona a *Div* to kolumna lub jej część. Dlatego domyślne mapowanie nie jest najlepszym pomysłem na konwertowanie wyników FineReadera do hOCR. Użytkownik konwertera może podać jako parametr własne mapowanie, np. wykorzystujące elemety opisujące układ strony lub specyficzne dla silnika hOCR.

3.3. Uwagi odnośnie użycia formatu hOCR przez PDFAUtilities

PDFAUtilities nie zapisuje informacji o skanach w plikach hOCR, chociaż są one wymagane przez specyfikację formatu. Można by spróbować przekonwertować plik PDF na zbiór obrazów reprezentujących poszczególne strony i informację o tak wygenerowanych skanach dodać do pliku hOCR.

W nagłówku powinny być podane informacje o użytych w pliku językach i systemach pisma.

Specyfikacja nakazuje umieszczanie w metadanych dokumentu informacji o mapowaniu elementów struktury na tagi HTML. PDFAUtilities nie spełnia tego wymagania.

Specyfikacja nakazuje, o ile to możliwe, umieszczanie informacji o końcu wiersza nie wynikającym z zawijania wierszy przy uzyciu tagu `br`. PDFAUtilitites nie udostępnia takiej możliwości.

Format hOCR umożliwia umieszczanie w nim metadanych, natomiast metadane z pliku PDF nie są w tej chwili eksportowane przez konwerter do hOCR.

Wszystkie przedstawione powyżej problemy nie są istotne z punktu widzenia użycia PDFAUtlities w Katedrze Lingwistyki Formalnej UW.

Rozdział 4

Używane narzędzia

4.1. PDFMiner

Podstawowym narzędziem używanym przez PDFAUtilities jest pakiet PDFMiner (zob. [PDFMin]). Pakiet ten umożliwia konwersję plików PDF do formatu HTML, XML i pliku tekstowego. Pozwala także na analizę układu strony w danym pliku PDF i zapis tej informacji we własnym formacie XML.

Programy pakietu PDFMiner kończą się z błędem dla niektórych plików PDF. Z powodu oparcia się PDFAUtilities na PDFMinerze spowoduje to dla tych plików zakończenie konwertera z błędem lub ich nieładowanie się w przeglądarce.

4.1.1. Analiza układu strony przez PDFMiner

Analizą układu strony zajmuje się program `pdf2txt.py`. Posiada on wiele opcji, nas interesuje eksport dokumentu do postaci pliku XML z układem strony. Żeby wyeksportować dokument do XML należy uruchomić program w następujący sposób:

```
pdf2txt.py -t xml -o wynik.xml plik.pdf
```

Wygenerowany plik ma następująca postać:

```
<?xml version="1.0" encoding="utf-8" ?>
<pages>
<page id="1" bbox="0.000,0.000,595.000,842.000" rotate="0">
<textbox id="0" bbox="56.800,761.497,94.071,783.828">
<textline bbox="56.800,761.497,94.071,783.828">
<text font="BAAAAA+Arial-BoldMT" bbox="56.800,761.497,66.621,783.828"
  size="22.331">T</text>
<text font="BAAAAA+Arial-BoldMT" bbox="65.494,761.497,74.446,783.828"
  size="22.331">i</text>
<text font="BAAAAA+Arial-BoldMT" bbox="74.301,761.497,79.662,783.828"
  size="22.331">t</text>
<text font="BAAAAA+Arial-BoldMT" bbox="79.807,761.497,89.628,783.828"
  size="22.331">l</text>
<text font="BAAAAA+Arial-BoldMT" bbox="89.612,761.497,94.071,783.828"
  size="22.331">e</text>
<text>
```

```
</text>
</textline>
...
<layout>
<textgroup bbox="56.800,62.992,537.800,783.828">
...
<textbox id="0" bbox="56.800,761.497,94.071,783.828" />
...
</textgroup>
</layout>
</page>
...
</pages>
```

Znaczenie użytych tagów:

page
: strona
możliwe dzieci: `textbox`, `layout`

textbox
: znaczenie podobne jak *ocr_carea* w hOCR
możliwe dzieci: `textline`

textline
: wiersz tekstu w obrębie `textbox`
możliwe dzieci: `text`

text
: pojedynczy znak (wraz z informacją o foncie, podana w tagu nazwa fontu jest brana spod klucza *FontName* z deskryptora fontu

layout
: tutaj znajdują się informacje dodatkowe o układzie strony
możliwe dzieci: `textgroup`, odwołanie do `textbox`

texgroup
: większe jednostki układu strony grupujące kilka elementów `textbox` lub `textgroup`
możliwe dzieci: `textgroup`, odwołanie do `textbox`

Oprócz tego mogą pojawić się inne tagi (np. `figure`, `polygon`), ale są one nieistotne z punktu widzenia niniejdzej pracy i ignorowane przez PDFAUtilities.

4.1.2. Domyślny wynikowy format hOCR przy analizie układu strony

Informacje o fontach i słowa są zapisywane w ten sam sposób co przy eksporcie struktury logicznej PDF. Jeśli chodzi o pozostałe tagi, to w następujący sposób są używane są tagi dotyczące składu dokumentu:

- `textline` przechodzi na *ocr_line*
- `textbox` przechodzi na *ocr_carea*

- `page` przechodzi na *ocr_page*
- `textgroup` przechodzi na *ocr_carea*, chociaż dokumentacja może sugerować, że *ocr_carea* nie może być zagnieżdżane

Na podstawie specyfikacji formatu hOCR trudno ocenić, czy lepszym rozwiązaniem nie byłoby zastosowanie zamiast tego oznaczeń specyficznych dla silnika OCR ([Breu], *8 OCR Engine-Specific Markup*).

4.1.3. Szczegóły implementacji pakietu PDFMiner związane z jego użyciem przez PDFAUtilities

Szczegóły implementacji przedstawimy na przykładzie programu `pdf2txt.py`.

Program ten składa się z jednej funkcji main. Po uruchomieniu tworzy ona obiekt klasy PDFResourceManager (pakiet *pdfinterp.py*). Obiekt ten na zmiennej PDFResourceManager.fonts pamięta fonty przechowywane w słownikach zasobów stron indeksując je ich identyfikatorami ich obiektu PDF (*object identifier*). Jeżeli zostanie na nim wywołana metoda PDFResourcesManager.get_font to jeżeli ma w tej zmiennej zapamiętany font o identyfikatorze danym jako argument to go zwraca. W przeciwnym przypadku tworzy obiekt klasy PDFFont (pakiet *pdffont.py*) reprezentujący dany font korzystając ze słownika fontu przekazanego jako drugi argument. Następnie umieszcza go w słowniku PDFResourceManager.fonts pod kluczem będącym identyfikatorem danym jako pierwszy argument i na koniec zwraca utworzony obiekt.

Po utworzeniu obiektu klasy PDFResourceManager program tworzy obiekt klasy XMLConverter [1].

Klasa XMLConverter jest rozszerzeniem klasy PDFLayoutAnalyzer. Klasa PDFLayoutAnalyzer udostępnia metodę PDFLayoutAnalyzer.receive_layout, do której będą przekazywane wyniki analizy układu strony. Metoda ta w klasie PDFLayoutAnalyzer nic nie robi, natomiast w klasie XMLConverter wypisuje przekazany układ strony do pliku wynikowego w opisanym powyżej formacie XML. Obie klasy znajdują sie w pliku *converter.py*.

Po utworzeniu obiektu klasy XMLConverter uruchamiana jest funkcja process_pdf z pakietu *pdfinterp.py*. Tworzone są w niej obiekty klasy PDFParser i PDFDocument. Następnie do każdego z tych obiektów jest przekazywana referencja do drugiego. Do obiektu klasy PDFDocument przekazujemy parser metodą PDFDocument.set_parser. W metodzie tej parser jest używany do odczytania z pliku PDF tabeli referencji (*cross-reference tables*), stopki (*trailer*) i katalogu dokumentu (zob. [PDF01] *3.4 File Structure*). Katalog dokumentu jest umieszczany na zmiennej PDFDocument.catalog, tabele referencji na liście PDFDocument.xrefs a niektóre informacje ze stopki na zmiennych PDFDocument.info i PDFDocument.encryption.

Następnie sprawdzane jest hasło do dokumentu[2]. Potem tworzony jest obiekt klasy PDFInterpreter (pakiet *pdfinterp.py*). Jako parametry jego konstruktora są przekazywane obiekty klas PDFDevice (pakiet *pdfdevice.py*) i PDFResourceManager które są zapisywane na zmiennych PDFInterpreter.device i PDFInterpreter.rsrcmgr. Pierwszy z nich to obiekt reprezentujący "urządzenie" na które będzie wypisywany plik PDF. W tym wypadku jest nim wspomniana wcześniej klasa XMLConverter (jej nadklasa PDFLayoutAnalyzer jest podklasą PDFDevice). "Urządzeniem" jest w tym wypadku docelowy plik XML.

[1] Dla innych opcji programu niż podana powyżej opcja `-t xml` tworzone są obiekty innych klas, ale są one nieistotne z naszego punktu widzenia.

[2] Obecnie PDFAUtilities nie wspiera dokumentów zabezpieczonych hasłem. Samo sprawdzanie hasła jest zaimplementowane, nie ma natomiast mechanizmu umożliwiającego jego podanie.

Następnie zostaje wywołana metoda PDFDocument.get_pages która odczytuje strony z katalogu dokumentu i zwraca je w postaci listy obiektów klasy PDFPage (pakiet *pdfparser.py*). Dla każdej ze stron zostaje wywołana metoda PDFInterpreter.process_page.

Metoda wywołuje na "urządzeniu" metodę PDFDevice.begin_page. Jej implementacja w klasie PDFLayoutAnalyzer tworzy obiekt klasy LTPage reprezentujący tag page w wynikowym pliku. Następnie w metodzie PDFinterpreter.process_page wywoływana jest metoda PDFInterpreter.render_contents, w której wywołane zostają metody PDFInterpreter.init_resources (ładująca zawartość słownika zasobów strony m.in. do zmienych PDFInterpreter.fontmap oraz PDFInterpreter.xobjmap i przy okazji umieszczająca ładowane fonty - o ile jeszcze ich tam nie ma, jeżeli są to je stamtąd pobiera - we wspomnianym obiekcie klasy PDFResourceManager), PDFInterpreter.init_state inicjalizująca stany (graficzny, tekstowy, aktualną macierz transformacji (*current transformation matrix*), stos stanów graficznych itp.) i zapisująca je na odpowiednich polach klasy PDFInterpreter oraz PDFinterpreter.execute, która przetwarza strumienie zawartości strony.

W metodzie PDFInterpreter.execute tworzony jest obiekt klasy PDFContentParser (pakiet *pdfinterp.py*) który następnie parsuje strumienie zawartości udostępniając kolejne polecenia PDF lub ich argumenty poprzez metodę PDFContentParser.next_object. Dla każdego z poleceń wywoływana jest przetwarzająca je metoda PDFInterpreter.do_<nazwa_polecenia>.

Najważniejszym poleceniem obsługującym wypisywanie tesktu jest PDFInterpreter.do_TJ, ponieważ wszystkie inne polecenia przekazują do niego sterowanie. Polecenie to wywołuje metodę PDFTextDevice.render_string [3]. Metoda ta pośrednio wywołuje metodę PDFTextDevice.render_char, która nic nie robi, natomiast jest przesłaniana przez metodę PDFLayoutAnalyzer.render_char. Metoda ta zamienia tekst na Unicode używając kodowania odpowiedniego fontu i tworzy obiekt klasy LTChar reprezentujący tag `text` w wynikowym pliku.

Na końcu metody PDFInterpreter.process_page wywoływana jest metoda PDFDevice.end_page. Jej implementacja w klasie PDFLayoutAnalyzer wywołuje metodę LTLayoutContainer.analyze na obiekcie reprezentującym tag page (LTPage jest podklasą LTLayoutContainer która analizuje układ strony. Następnie na "urządzeniu" jest wywoływana metoda PDFLayoutAnalyzer.receive_layout. Jej argumentem jest obiekt klasy LTPage ze zanalizowanym układem strony.

4.1.4. Rozszerzenia pakietu PDFMiner na potrzeby PDFAUtilities

Na potrzeby pakietu PDFAUtilities należało zmodyfikować niektóre z funkcji. Zmodyfikowane funkcje z PDFMinera zostały umieszczone w pakiecie *pdfminermod.py*.

Funkcja init_process_pdf wykonuje te same operacje co funkcja process_pdf do momentu utworzenia obiektu klasy PageInterpreter. Zamiast go tworzyć funkcja zwraca utworzony obiekt klasy PDFDocument i liczbę stron w dokumencie.

Funkcja continue_process_pdf wykonuje te same operacje co funkcja process_pdf poczynając od utworzenia obiektu klasy PageInterpreter. Ponadto, jeżeli jej dodatkowy argument verbose ma wartość True, wypisuje na terminal numery przetwarzanych stron.

Umieszone w pakiecie PDFMinera *pdftype.py* funkcje postaci ¡type>_value, jeżeli argument jest referencją do obiektu PDF (klasa PDFObjRef), zwracają obiekt na który ona wskazuje. W przeciwnym przypadku zwracają argument. Modyfikacje tych metod w pakiecie *pdfminermod.py* o nazwach ¡type>_value2 dodatkowo sprawdzają, czy obiekt który ma zwrócić metoda jest odpowiedniego typu i w przeciwnym przypadku zwracają None.

Modyfikacje o nazwach ¡type>_value_none zwracają wartość None jeżeli ich argument ma wartość None (w oryginalnych metodach tak nie było). Dodatkowo metoda literal_name z

[3] Klasa PDFTextDevice jest podklasą PDFDevice i nadklasą PDFLayoutAnalyzer.

pakietu *psparser.py* zamieniająca nazwę PDF na napis została zmodyfikowana w podobny sposób i ta modyfikacja otrzymała nazwę literal_name_none.

4.2. Pozostałe narzędzia

PIL

Pakiet PIL to biblioteka udostępniające różnorodne operacje na plikach graficznych w Pythonie (zob. [PIL]). Jest on wykorzystywany, wraz z biblioteką EXIF.py[4], do wyciągania niektórych informacji z zawartych w plikach PDF obrazów JPEG do pokazania w przeglądarce (metoda PhysList.__processImage). Ponadto jest używany do zamiany zawartych w PDF obrazów w formacie PPM na bitmapę (funkcja getWxImage).

Pdfimages

Program `pdfimages` jest częścią pakietu Poppler (zob. [Pop]). Umożliwia on wydobywanie obrazów z plików PDF. Na potrzeby PDFAUtilities program został zmodyfikowany jako biblioteka C++ z interfejsem umożliwiającym wywoływanie jej z poziomu Pythona (zob. 6.2.4).

FontTools

Biblioteka FontTools (zob. [TTX]) obsługuje fonty TrueType i jest używana do wyciągania informacji z fontów TrueType zawartych w plikach PDF (metody PhysList.__processFont i Font.__extractNameAndStyle).

Na potrzeby pakietu PDFAUtilties rozszerzono klasę TTFont z biblioteki FontTools. W pakiecie *ttFontMod.py* znajduje się klasa TTFontMod, będąca jej podklasą. W przeciwieństwie do TTFont do jej konstruktora nie przekazuje się nazwy pliku z programem fontu, ale bufor z zawartością programu fontu odczytanego z pliku PDF.

ICU

ICU to zestaw bibliotek w Javie i C++ zapewniających różnorodne funkcje związane z obsługą standardu Unicode i lokalizacji. PDFAUtilities korzysta z nich za pomocą PyICU, interfejsu do ICU w języku Python ([PyICU]). ICU jest wykorzystywane do podziału tekstu na słowa w zapisywanym przez konwerter pliku hOCR. Informacje na temat wykorzystywanych funkcji można znaleźć w [ICU].

wxWidgets

WxWidgets to biblioteka napisana w C++ która ma umożliwić stworzenie przenośnego interfejsu użytkownika (interfejs napisany w wxWidgets powinien działać zarówno pod Windows jak i pod Linuksem, choć okazuje się, że są pewne różnice w jego zachowaniu pod oboma systemami). W PDFAUtilities wykorzystano interfejs do wxWidgets o nazwie wxPython ([WX]) do stworzenia przeglądarki dokumentów.

[4] http://sourceforge.net/projects/exif-py/

SWIG

SWIG ([SWI]) to narzędzie umożliwiające tworzenie interfejsu do funkcji zaimplementowanych w C++ z poziomu Pythona. SWIG został użyty do zaimplementowania modyfikacji `pdfimages`.

Rozdział 5

Implementacja PDFAUtilities

5.1. Wprowadzenie

Pakiet jest zaimplementowany w języku Python. Język ten został wybrany, ponieważ w nim jest zaimplementowany PDFMiner, będący podstawą działania PDFAUtilties. Jedynym wyjątkiem jest modyfikacja programu `pdfimages` z pakietu Poppler, która została napisana w C++ i udostępniona w Pythonie przy użyciu narzędzia SWIG.

Opis implementacji zamieszczony w niniejszej pracy ma umożliwić rozeznanie się w zadaniach poszczególnych klas i przedstawić zarys przepływu sterowania podczas najważniejszych przypadków użycia. Szczegóły implementacyjne są wyjaśnione w komentarzach w kodzie źródłowym programu umieszczonym na załączonej płycie CD.

5.2. Szczegóły implementacji

W niniejszym opisie pominięto pakiet *utils.py* zawierający metody pomocnicze używane w różnych miejscach programu.

5.2.1. Pakiet *taglib.py*

Pakiet *taglib.py* zawiera cztery klasy: TagLib, PTree, Font i Node.

Klasa TagLib jest interfejsem do PDFMinera. Podczas otwierania każdego pliku PDF (przez przeglądarkę w metodzie MainWindow.__onOpen lub przez konwerter w metodzie main pakietu *pdfa2hocr.py*) tworzony jest nowy obieky tej klasy. Następnie jest na nim wywoływana metoda TagLib.initialize, która otwiera plik z użyciem PDFMinera[1] - otwarty dokument jest umieszczany w polu TagLib.__doc. W metodzie tej tworzony jest obiekt TagLib.__ptree, w którym są przechowywane słowniki zasobów poszczególnych stron. Fonty ze słownika zasobów są w nim przechowywane w obiektach klasy Font, które oprócz obiektu fontu PDFMinera zawierają różne dodatkowe informacje.

Utworzony obiekt TagLib.__ptree jest obiektem klasy PTree. Reprezentuje ona drzewo, którego korzeniem jest w domyśle dokument, jego dziećmi strony, a ich dziećmi elementy słownika zasobów tych stron (tylko fonty i obrazki). Struktura taka wynika z tego, że pierwotnie przeglądarka słownika zasobów w przeglądarce graficznej miała postać drzewa. Potem zmieniona została ją na prostszą postać listową, ale implementacja PTree pozostała bez zmian. Informacje o zasobach stron są przechowywane w postaci obiektów klasy Font, zawierających

[1] W podobny sposób jak funkcja process_pdf, z tym że klasa PageInterpreter nie jest używana do przetwarzania zawartości strony.

m.in. obiekt fontu PDF i obiekt klasy PDFFont którego PDFMiner używa do reprezentowania fontu, i obiektów obrazów PDF.

Po zakończeniu metody TagLib.initialize, wywołując metodę TagLib.getRoot, przeglądarka i konwerter pobierają z obiektu klasy TagLib korzeń drzewa struktury logicznej w postaci obiektu klasy Node.

Klasa Node reprezentuje pojedynczy element struktury lub korzeń drzewa struktury. Nowo utworzony obiekt tej klasy zawiera tylko obiekt klasy PDFObjectRef (pakiet *pdftypes.py* PDFMinera), będący odnośnikiem do wczytanego przez PDFMinera elementu struktury lub korzenia drzewa, w polu Node.__object. Inicjalizacja pozostałych pól odbywa się w metodzie Node.__initialize, która jest wywoływana za każdym razem, gdy na obiekcie wywołana jest metoda która wymaga żeby pozostałe pola były zainicjalizowane. W metodzie tej z obiektu Node.__object odczytywana jest informacja o potomkach elementu w drzewie lub o jego zawartości (jeżeli element jest liściem). Niezainicjalizowane obiekty Node reprezentujące potomków są umieszczane na liście Node.__children. W przypadku liści jest na niej umieszczana zawartość tesktowa przemieszana z fontami.

PDFAUtilities nie wspiera zawartości w elementach nie będących liśćmi (może się tak zdarzyć np. przy obrazkach umieszczonych w tekście), ponadto nie wspiera zawartości nie będącej tekstem - czyli adnotacji a także obrazków.

Korzeń zwrócony przez TagLib.getRoot jest początkowo niezainicjalizowany. Wywołanie na nim metod powoduje jego zainicjalizowanie (tworzone są niezainicjalizowane obiekty Node dzieci). Podobnie wywołanie metod na dzieciach inicjalizuje wnuki itd. Niektóre metody inicjalizują od razu wszystkich potomków.

Każdy obiekt klasy Node zawiera referencję do obiektu TagLib (pole Node.__lib) który utworzył korzeń drzewa w którym się znajduje. Dzięki temu obiekt Node ma dostęp do pliku PDF wczytanego przez PDFMiner. Krótko omówimy wykorzystywane przez Node w tym celu metody TagLib.

W czasie przetwarzania liścia metoda Node.__initialize wywołuje metodę TagLib.getMcrs która wykorzystuje obiekt klasy TagInterpreter do znalezienia wszystkich zawartości oznaczonych posiadających identyfikatory MCID na danej stronie. Metoda ta zwraca listę obiektów klasy MarkedContent reprezentujących zawartości oznaczone, które zawierają m.in. tekst występujący w zawartości przemieszany z informacją o foncie (kluczu fontu w słowniku zasobów strony i jego rozmiarze).

Metoda Node.__initialize wybiera następnie te z obiektów MarkedContent których identyfikatory MCID są takie te zawarte pod kluczem K w danym elemencie struktury. Potem przetwarza tekst w wybranych obiektach, w ten sposób, że po natrafieniu na informację o foncie pobiera obiekt typu Font za pomocą metody TagLib.findFont. Metoda ta umożliwia znalezienie w słowniku zasobów strony o danym identyfikatorze fontu znajdującego sie w nim pod danym kluczem. Font nie jest wyszukiwany w strukturach PDFMinera, tylko w obiekcie TagLib.__ptree. Zwrócony font jest kopiowany i kopia ta jest wstawiana na listę dzieci liścia z ustawioną informacją o rozmiarze. Oznacza ona, że cały tekst na liście dzieci do następnego fontu jest w foncie definiowanym przez kopię.

Po natrafieniu na tekst metoda Node.__initialize koduje go jako napis Unicode z użyciem metody TagLib.textof przy pomocy obiektu klasy PDFFont z ostatnio napotkanego fontu (pamięta klucz w słowniku zasobów i rozmiar ostatniego fontu, dzięki którym pobiera go przez TagLib.findFont) i również wstawia na listę dzieci.

Oprócz tego z obiektu MarkedContent są pobierane i zapisywane w obiekcie klasy Node informacje o boudning boxach znaków, a ich suma zapamiętywana jako gabaryt całego elementu reprezentowanego przez Node.

Metoda TagLib.getRoleMap umożliwia obiektowi klasy Node dostęp do opisanego wcze-

śniej słownika umieszczonego pod kluczem *RoleMap* w słowniku korzenia struktury logicznej dokumentu.

Oprócz korzenia zwracanego przez metodę TagLib.getRoot metody TagLib.getStructureToDrawById i TagLib.getStrucutreToDraw zwracają podrzewo drzewa struktury logicznej na potrzeby rysowania struktury w przeglądarce graficznej. Jest ono zupełnie niezależne od drzewa zwracanego przez TagLib.getRoot (tzn. temu samemu elementowi struktury logicznej odpowiadaja dwa różne obiekty Node w obu drzewach). Celem takiego rozwiązania miało być ułatwienie zarządzania pamięcią. Zwracane przez wspomniane metody podrzewo to pełne drzewo stuktury logicznej obcięte tylko do danej strony. Do znalezienia elementów struktury z danej strony wykorzystujemy obiekt pod kluczem *PageParent* w korzeniu drzewa struktury.

Metody te ponadto znajdują w zwróconym poddrzewie element o identyfikatorze TagLib.__selected i ustawiają w nim flagę Node.__selected, która powoduje, że będzie on podświetlony w przeglądarce graficznej.

Oprócz wspomnianych powyżej metod istnieje kilka innych używanych w przeglądarce i konwerterze, np. TagLib.getPageBBox umożliwiająca poznanie gabarytu strony o danym identyfikatorze, TagLib.getPageNo zwracająca liczbę stron czy funkcje mapujące identyfikatory stron na ich numery i odwrotnie.

W niektórych sytuacjach zachodzi konieczność podziału elementu na cześci z których każda znajduje się na jednej stronie. Zajmuje się tym metoda Node.split. Ma ona dwa tryby: w trybie parametru copyAll ustawionego na True element na którym wywołano metodę nie zmienia się, natomiast powstałe z podziału elementy są zapisywane w polu Node.__copies. Jest on używany przez metodę TagLib.getStructureToDrawById.

W trybie zwykłym (copyAll ustawione na False) element na którym wywołano metodę zostaje obcięty do jednej strony, natomiast w Node.__copies są zapisywane części dla pozostałych stron. Jest on używany przez metodę HOCRExporter.__processNode.

Obiekty wspomnianej wcześniej klasy Font reprezentują fonty. Poza obiektem fontu PDF i obiektem PDFFont z PDFMinera zawiera informacje o stylu i rodzinie fontu. Obiekty typu Font reprezentujące zmianę czcionki w zawartości tekstowej liści klasy Node posiadają dodatkowo informację o wielkości czcionki, natomiast obiekty Font reprezentujące w strukturze TagLib.__ptree elementy słownika zasobów strony nie. Informację o stylu i rodzinie fontu obiekt klasy Font próbuje uzyskać z deskryptora fontu, załączonego pliku fontu (w przypadku fontów TrueType) i w ostateczności przez analizę nazwy (1.7). Trzeba tutaj zaznaczyć, że w chwili obecnej PDFAutilities nie obsługują w pełni fontów nie będących fontami TrueType.

5.2.2. Pakiet *pdfreader.py*

Pakiet zawiera trzy klasy: MarkedContent, TagInterpreter i DummyConverter.

Klasa MarkedContent reprezentuje zawartośc oznaczoną. Zawiera m.in. gabaryty występujących w niej znaków i całej zawartości, informację czy znalazła się ona w obrębie zawartości oznaczonej będącej paginą (lub też czy sama jest artefaktem lub paginą) oraz własną zawartość tekstową wraz z informacją o czcionkach (jest ona przechowywana w postaci listy MarkedContent.els argumentów poleceń tekstowych przetwarzanych przez TagInterpreter, jest przetwarzana na napisy Unicode i obiekty klasy Font dopiero w metodzie Node.__initialize i zapisywana na liście dzieci obiektu typu Node).

Klasa TagInterpreter służy znajdowaniu na stronie zawartości oznaczonych. Nowy obiekt klasy TagInterpreter jest tworzony w metodzie TagLib.getMcrs na potrzeby przetwarzania jednej strony. Po przetworzeniu przezeń strony znalezione zawartości oznaczone są zapisywane na TagLib.__mcrs i każde kolejne wywołanie tej metody zwraca je bez użycia klasy TagInterpreter aż do momentu gdy zmieni się strona.

Klasa TagInterpreter jest podklasą klasy PDFPageInterpreter z programu PDFMiner, którego zadaniem jest przetwarzanie zawartości obiektu strony w PDF. Po utworzeniu obiektu tej klasy jest wywoływana na nim metoda TagInterpreter.process_page[2] - obiekt zapamiętuje stronę na której działa i przekazuje sterowanie do nadklasy.

Ponieważ klasa TagInterpreter implementuje metody dla poleceń dotyczących przetwarzania tekstu i zawartości oznaczonych, nadklasa przetwarzając stronę przekazuje sterowanie do tych metod.

Obiekt klasy TagInterpreter w momencie natrafienia na zawartość oznaczoną z identyfikatorem MCID tworzy obiekt klasy MarkedContent i zapamiętuje go w polu TagInterpreter.__bdcs. Po natrafieniu na operację ustawiającą font (Tf) jej parametry są zapamiętywane w polu MarkedContent.els zawartości zapamiętanej w TagInterpreter.__bdcs. Podobnie dzieje sie w przypadku operacji tekstowych (TJ i Tj), tylko wtedy dodatkowo zapamiętywane są informacje o bouding boxach znaków i na tej podstawie aktualizowany jest gabaryt zawartości oznaczonej. Informacje te są otrzymywane przez wywołanie metody DummyConverter.render_string na argumencie polecenia tekstowego i następnie pobranie obliczonych gabarytów z obiektu klasy DummyConverter.

Po natrafieniu na zawartość będącą paginą fakt ten jest zapamiętywany w polu TagInterpreter.__pagination, dzięki czemy można stwierdzić, czy dana zawartość jest w paginie czy nie.

Obiekt klasy DummyConverter jest przekazywany jako argument do konstruktora obiektu klasy TagInterpreter, który z kolei przekazuje go do konstruktora nadklasy. Klasa DummyConverter jest w zasadzie modyfikacją klasy PDFLayoutAnalyzer z programu PDFMiner. Modyfikacja polega na tym, że w czasie przetwarzania analizowanych napisów zapamiętywana jest informacja o gabarytach znaków i ich sumie. Jest to jedyna informacja wykorzystywana przez PDFAUtilities.

5.2.3. Pakiet *pdfminerparser.py*

Pakiet zawiera dwie klasy. Pierwszą z nich jest PDFMinerHandler który jest implementacją interfejsu ContentHandler z biblioteki SAX. Umożliwia on ładowanie plików XML z układem strony zanalizowanym przez pakiet PDFMiner do drzewa złożonego z obiektów PDFMinerNode.

Drugą z klas jest PDFMinerParser. Umożliwia on wczytanie pliku XML i przetworzenie go za pomocą klasy PDFMinerHandler (metoda PDFMinerParser.parse). Oprócz tego udostępnia trzy metody pozwalające na wczytanie pliku PDF i zanalizowanie w nim układu strony przez PDFMiner.

Pierwsza z nich to PDFMinerParser.extractFromPDF. Po wczytaniu pliku PDF z użyciem PDFMinera jest on przetwarzany przez obiekt klasy PDFMinerConverter, który jest używany jako "urządzenie" w funkcji PDFMinera process_pdf. Tworzy on reprezentujące zanalizowany układ strony drzewo obiektów PDFMinerNode które można uzyskać wywołując metodę PDFMinerParser.getResult. Metoda ta zwraca również drzewo obiektów uzyskane w wyniku wywołania metody PDFMinerParser.parse.

Drugą z metod jest PDFMinerParser.extractHOCRFromPDF. Działa ona podobnie jak poprzednia, ale używany przez nią obiekt klasy PDFMinerConverter zamiast tworzyć drzewo reprezentujące układ strony tworzy poddrzewa drzewa układu strony dla poszczególnych stron i każde z osobna eksportuje z użyciem klasy HOCRExporter (obiekt tej klasy jest argumentem metody i jest przekazywany do obiektu klasy PDFMinerConverter przez jego konstruktor).

[2]Por. 4.1.3. "Urządzeniem" jest w tym przypadku obiekt klasy DummyConverter, który nie robi nic poza obliczaniem gabarytów przekazywanych potem do obiektu klasy TagInterpreter.

Umożliwia to przetworzenie podrzew przed eksportem przez obiekt klasy Columnizer, dlatego też metoda PDFMinerParser.extractHOCRFromPDF jest używana wtedy gdy używamy klasy Columnizer (również ten obiekt jest argumentem metody i jest przekazywany do konstruktora obiektu klasy PDFMinerConverter, jeżeli wartością tego argumentu jest None to PDFMinerConverter nie przetwarza poddrzew tylko od razu je eksportuje).

Zamiast funkcji process_page w metodzie PDFMinerParser.extractHOCRFromPDF używane są funkcje init_process_pdf i continue_process_pdf, dzięki czemu można uzyskać informacje o ilości stron w pliku i przekazać je do obiektu klasy XMLLib przed przekazaniem do obiektu klasy PDFMinerConverter (zob. 5.2.4) oraz wykorzystać wypisywanie numerów aktualnie przetwarzanych stron w continue_process_pdf.

Trzecią metodą jest PDFMinerParser.extractHOCRFromPDFDirect. Również eksportuje ona plik PDF do hOCR z użyciem układu strony, ale w przeciwieństwie do poprzedniej metody zamiast obiektu klasy PDFMinerConverter stosuje ona obieky klasy HOCRDirectConverter. Różnica między nimi polega na tym, że HOCRDirectConverter wypisuje zanalizowany układ strony bezpośrednio do pliku wyjściowego zamiast tworzyć drzewo układu strony lub jego poddrzewa dla poszczególnych stron.

W metodzie PDFMinerParser.extractHOCRFromPDFDirect są używane funkcje init_process_pdf i continue_process_pdf, analogicznie jak w PDFMinerParser.extractHOCRFromPDF, dzięki czemu można wykorzystać wypisywanie numerów aktualnie przetwarzanych stron w continue_process_pdf.

5.2.4. Pakiet *pdfminerconverter.py*

Pakiet *pdfminerconverter.py* zawiera trzy klasy: XMLLib, PDFMinerNode i PDFMinerConverter.

Podstawowym założeniem przyjętym przy tworzeniu klas XMLLib i PDFMinerNode było to, że ten sam kod obsługuje strukturę logiczną z dokumentu PDF przechowywaną w drzewie elementów Node jak i układ strony zanalizowany przez PDFMiner przechowywany w drzewie elementów PDFMinerNode. Założenie to nie zostało zrealizowane w całości (są miejsca gdzie sprawdzany jest typ przetwarzanej struktury i w zależności od wyniku wykonuje się inny kod), ale w większości przypadków nie ma rozrożnienia między XMLLib i PDFMinerNode a TagLib i Node.

W celu realizacji tego założenia powstała klasy XMLLib która jest zaślepką udającą obiekt klasy TagLib. Podobnie jak TagLib XMLLib udostępnia metody XMLLib.getPageNo, XMLLib.getPageBBox i metody mapujące identyfikatory stron na ich numery.

W przypadku użycia jej do obsługi wczytywanego pliku XML obiekt tej klasy jest inicjalizowany w metodzie PDFMinerParser.parser.

W przypadku użycia jej do obłsugi pliku PDF po utworzeniu obiektu klasy XMLLib wywoływana jest na nim metoda loadPTree, która wczytuje plik PDF z użyciem PDFMinera (w podobny sposób jak metoda TagLib.initialize i wywołuje metodę PDFMinerNode.__initializePTree tworzącą obiekt ze słownikami zasobów stron (XMLLib.__ptree), taki jak TagLib.__ptree. Następnie obiekt klasy XMLLib jest przekazywany do metod PDFMinerParser.extractFromPDF, PDFMinerParser.extractHOCRFromPDF i PDFMinerParser.extractHOCRFromPDFDirect. Niektóre pozostałe pola obiektu klasy XMLLib są inicjalizowane w metodach PDFMinerParser.extractFromPDF i PDFMinerParser.extractHOCRFromPDF.

W metodzie PDFMinerParser.extractFromPDF i PDFMinerParser.extractHOCRFromPDF obiekt klasy XMLLib jest przekazywany do obiektu klasy PDFMinerConverter, który wykorzystuje metodę XMLLib.findfont działającą analogicznie do TagLib.findfont (jedyną różnicą jest to, że fontu szukamy w XMLLib.__ptree po nazwie a nie po kluczu w słowniku zasobów) do

znaleznienia odpowiedniego obiektu klasy Font w obiekcie reprezentującym słowniki zasobów stron (XMLLib.__ptree).

Ponieważ w drzewie układu strony wczytywanym przez obiekt klasy PDFMinerParser dziećmi korzenia są elementy reprezentujące poszczególne strony, można łatwo zaimplemetnować metody XMLLib.getStructureToDrawById i XMLLib.getStructureToDraw - analogiczne do TagLib.getStructureToDrawById i TagLib.getStructureToDraw, ale w przeciwieństwie do nich zamiast stosowanego w nich złożonego algorytmu po prostu znajdujące dziecko korzenia na pozycji w jego liście dzieci odpowiadającej numerowi strony.

Podobnie do klasy XMLLib zgodnej z klasą TagLib klasa PDFMinerNode została stworzona z myślą o zgodości interfejsu z klasą Node.

Zasadnicza różnica między PDFMinerNode a Node to to, że w PDFMinerNode brak mechanizmu zarządzania pamięcią - obiekt klasy PDFMinerNode nie przechowuje obiektu który reprezentuje, tak jak to robi pole Node.__object, i brak w nim metody analogicznej do Node.__initialize. Całe drzewo układu strony jest od razu inicjalizowane przez obiekt klasy PDFMinerConverter lub PDFMinerHandler. Jedynym wyjątkiem jest gdy w przypadku użycia klasy PDFMinerConverter zanalizowana struktura jest od razu eksportowana do hOCR - wtedy pełne drzewo nie jest nigdy tworzone, a tylko poddrzewa dla poszczególnych stron, z których każde z osobna jest eksportowane do hOCR. Każde z tych poddrzew jest jednak także w całości zainicjalizowane.

Ponieważ elementy drzewa nie są poddane zarządzaniu pamięcią tak jak obiekty klasy Node, to metoda PDFMinerNode.uninitialize nic nie robi. Nie ma też potrzeby odzielnego zarządzania poddrzewem zwracanym przez XMLLib.getStructureToDrawById i XMLLib.getStructureToDraw od głównego drzewa, dlatego te dwie ostatnie metody właśnie z niego pobierają zwracane poddrzewa.

Struktura układu strony i implementacja konwertera na hOCR jest taka, że metoda PDFMinerNode.multipage nigdy nie zostanie wywołana dla elementu drzewa układu strony położonego na wielu stronach (jedynym takim elementem jest korzeń drzewa). Dlatego też metoda ta zwraca zawsze False. Z tego samego powodu metoda PDFMinerNode.split jest pusta.

Oprócz tych samych metod co Node klasa PDFMinerNode posiada dodatkowe metody używane przy tworzeniu drzewa układu strony przy ładowaniu pliku, w pakiecie *columnize.py* oraz w module podanym przez użytkownika.

Klasa PDFMinerConverter jest, podobnie jak DummyConverter, rozszerzeniem klasy PDFLayoutAnalyzer. W przeciwieństwie do niej przesłania nie wywołania metod obsługujących polecenia tekstowe i związane z zawartościami oznaczonymi a wywołania metody PDFLayoutAnalyzer.receive_layout. Do metody tej zostaje przekazany zanalizowany przez PDFMinera układ strony. Następnie na jego podstawie jest tworzone poddrzewo układu strony dla tej strony złożone z obiektów klasy PDFMinerNode.

Jeżeli PDFMinerConverter został użyty przez metodę PDFMinerParser.extractFromPDF to poddrzewa poszczególnych stron są dodawane jako dzieci do korzenia przechowywanego w polu PDFMinerConverter.__root, który jest potem zwracany do PDFMinerParser.extractFromPDF przez metodę PDFMinerConverter.getResult.

Natomiast jeżeli PDFMinerConverter został użyty przez metodę PDFMinerParser.extractHOCRFromPDF to każde z poddrzew jest przetwarzane przez przekazany jako parametr konstruktora PDFMinerConverter.__init__ obiekt klasy Columnizer a potem eksportowane do hOCR przez przekazany jako parametr konstruktora obiekt klasy HOCRExporter.

5.3. Zarządzanie pamięcią

Teoretycznie Python powinien sam zarządzać pamięcią. Okazało się jednak, że wraz z otwieraniem kolejnych plików w przeglądarce rośnie pamięć zajmowana przez program (pamięć po poprzednich plikach nie jest całkowicie zwalniana). Jedną z przyczyn może być naprzykład występowanie cyklicznych referencji do obiektów.

Dlatego wraz z otwieraniem kolejnych plików w przeglądarce może dojść do wystąpnienia błędu MemoryError. Jednym z rozwiązań tego problemu byłoby przechwytywanie MemoryError przy otwieraniu nowego pliku i w razie jego wystąpienia uruchomienie programu ponownie (wtedy zwolni się pamięć po poprzednich plikach). Niestety po wystąpieniu MemoryError zachowanie programu odbiega od standardowego, poza tym to nierozwiązuje problemu braku pamięci po wczytaniu pliku.

Na początku tworzenia pakietu PDFAUtilities założono, że głównym problemem jest pamięć zajmowana przez drzewo elementów Node. Dlatego są one inicjalizowane dopiero w razie potrzeby metodą Node.__initialize. Inicjalizowanie miało się odbywać przy rozwijaniu węzła w przeglądarce tekstowej i jego eksporcie w konwerterze. Następnie przy zwijaniu węzła (metoda LazyTree.__onCollapse) i po zakończeniu eksportu węzła w metodzie HOCRExporter.__processNode miał on powracać on do stanu sprzed inicjalizacji po wywołaniu metody Node.uninitialize (w szczególności powoduje ona, że element staje się liściem - odcinamy wszystkie jego dzieci). Okazuje się jednak, że wywołania różnych metod powodują ponowną lub wcześniejszą inicjalizację węzłów (np. metoda Node.getPageIds powoduje inicjalizację węzła i wszystkich jego potomków).

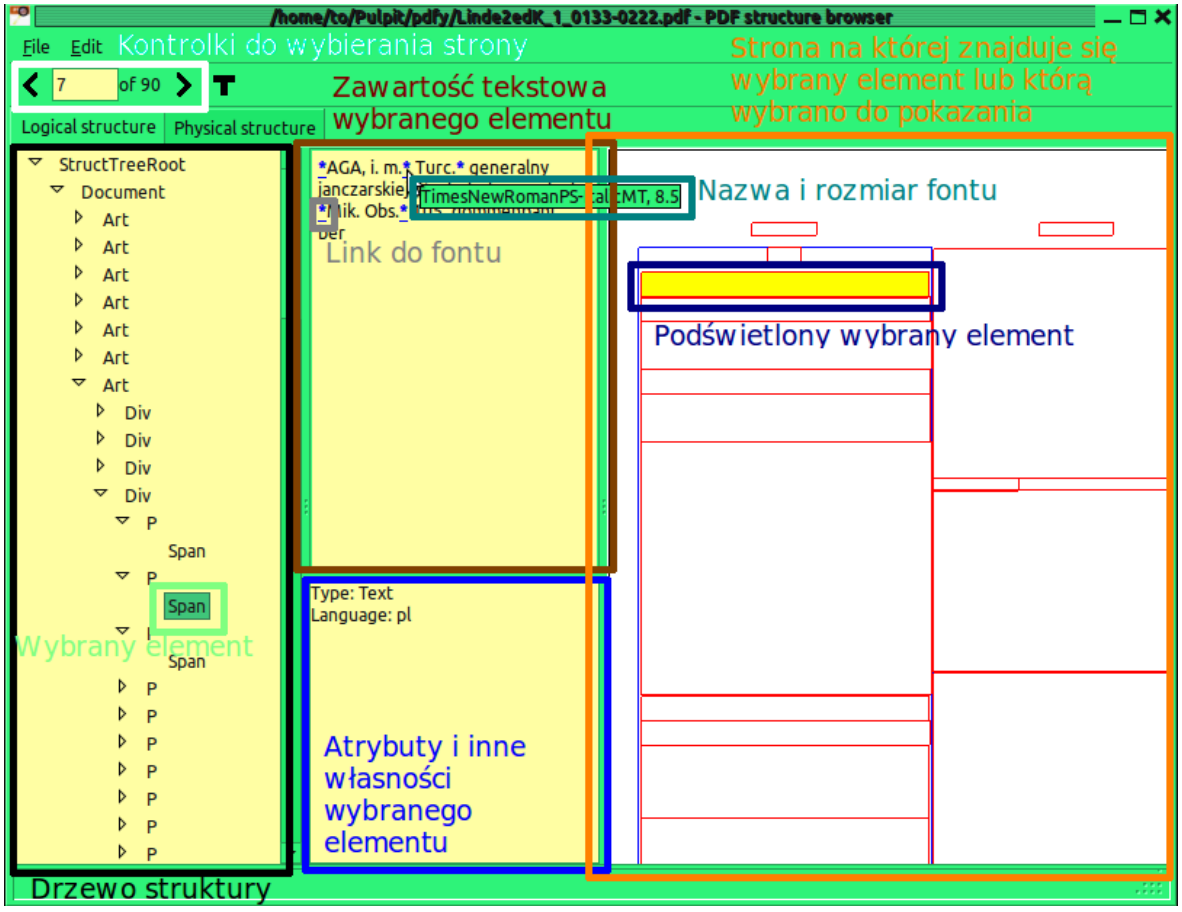
Ponadto okazało się, że większosc pamięci jest zajmowana przez obiekty PDFMinera, a drzewo Node w porównaniu z nimi zajmuje stosunkowo mało pamięci.

Rozdział 6

Przeglądarka struktury dokumentu

6.1. Opis funkcjonalności

Przeglądarka struktury dokumentu umożliwia otwarcie pliku PDF lub pliku XML z wynikami PDFMinera.



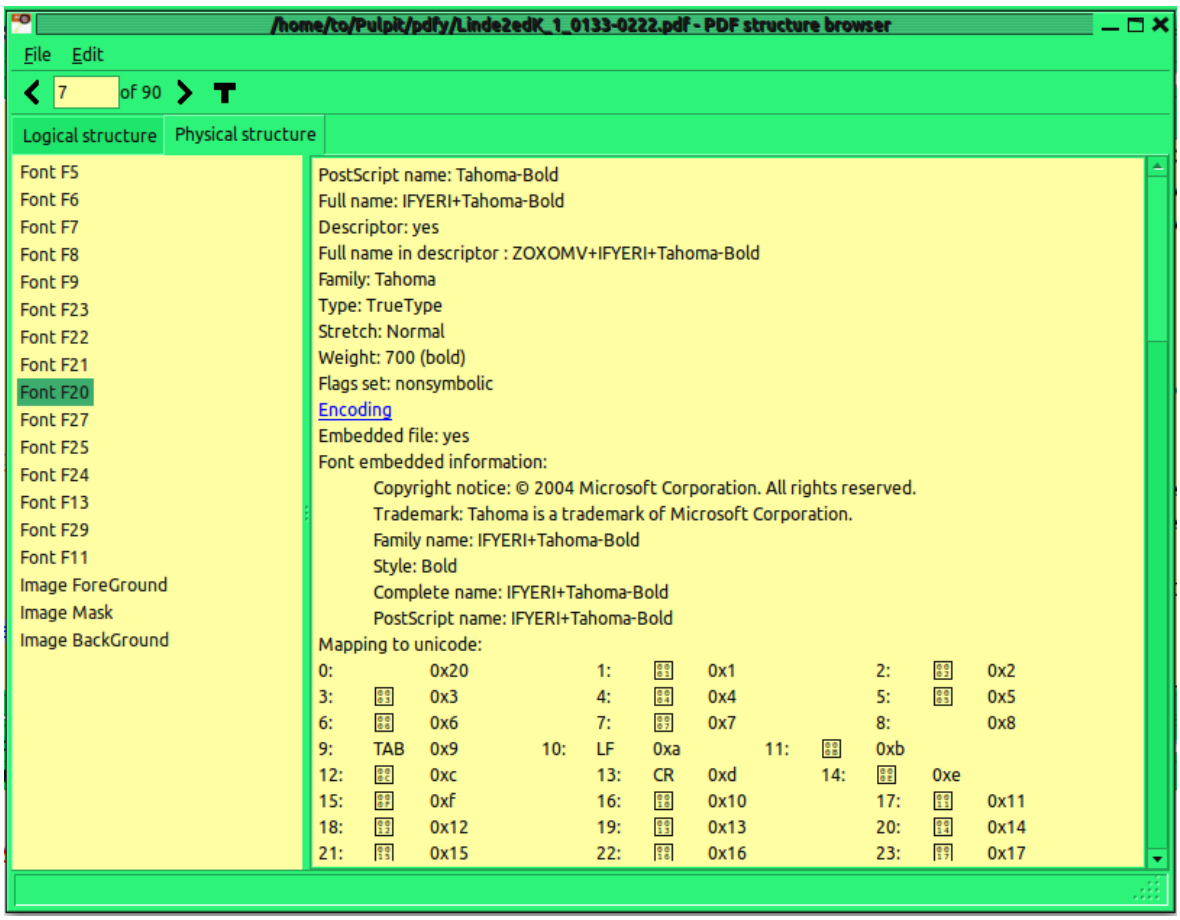
Rysunek 6.1: Przeglądarka struktury logicznej dokumentu

Przeglądarka ma dwie zakładki. W zakładce "Logical structure" z lewej strony (patrz rys. 6.1) pokazywane jest drzewo struktury. Jest to albo drzewo struktury logicznej zawarte w pliku PDF (jeżeli plik PDF otwarto poleceniem File->Open), albo układ strony zanalizowany

przez program PDFMiner (jeżeli plik PDF otwarto poleceniem File->Analize lub otwarto plik XML - w tym drugim przypadku tylko poleceniem File->Open). Po wybraniu jakiegoś elementu struktury w środkowym dolnym oknie pojawiają się atrybuty elementu i (w przypadku pokazywania struktury logicznej pliku PDF) właśności pod kluczami *Lang*, *ActualText* i *Alt* w słowniku elementu struktury logicznej elementu.

Jeżeli wybrany element jest liściem to w środkowym górnym oknie pojawia się zawartość tekstowa elementu. Widoczne są w niej też oznaczone gwiazdkami miejsca zmiany czcionki. Po kliknięciu na jeden z nich program zostanie przełączony do drugiej zakładki gdzie dana czcionka zostanie pokazana w słowniku zasobów strony. Jeżeli najedzie się myszą na tekst, to pojawi się podpowiedź w której będzie wyświetlona nazwa czcionki w danym miejscu i jej rozmiar. Funkcjonalność ta wymaga jeszcze pewnego dopracowania, bo nie zawsze podpowiedź się pokazuje.

W prawym oknie zostanie narysowana strona (gabaryty elementów struktury) na której znajduje się wybrany element i zostanie on zaznaczony na żółto (jeżeli element znajduje się na więcej niż jednej stronie to będzie pokazana jakaś strona na której się on znajduje). Oprócz tego na górze znajdują się kontrolki umożliwiające wybór dowolnej strony która zostanie pokazana w prawym oknie.



Rysunek 6.2: Przeglądarka słowników zasobów stron - font

Jeżeli przeglądamy układ strony zanalizowany przez PDFMiner a nie strukturę logiczną z pliku PDF to po kliknięciu na ikonkę "T" w oknie rysowania strony oprócz gabarytów zostanie pokazany tekst. Ta funkcjonalność nie pozwala na wierne odtworzenie wyglądu tekstu,

natomiast pokazuje rozmieszczeni poszczególnych słów na stronie a przez to np. analizę poprawności układu strony. Była użyteczna np. przy implementacji pakietu *columnize.py*. Jest to funkcjonalność bardzo nieefektywna, docelowo najlepszym rozwiązaniem byłoby zrenderowanie strony z użyciem biblioteki Poppler.

Przy ładowaniu dużych plików XML lub dużych plików PDF przy pomocy Open->Analize może zabraknąć pamięci. Pomóc mogłoby tutaj usunięcie elementów `text` z drzewa obiektów PDFMinerNode - liścmi byłyby wtedy elementy `textline`. Problem dotyczy też ładowania tych plików przez konwerter na hOCR.

Jeżeli PDF otwarto poleceniem File->Open i nie ma on struktury logicznej to po prostu w zakładce "Logical structure" nie zostanie nic pokazane. Natomiast słowniki zasobów stron wciąż będą widoczne w zakładce "Physical structure".

Po wybraniu polecenia Edit->Viewing parameters można ustalić gabaryty elementów o jakiej nazwie będą wyświetlane w oknie rysowania strony oraz kolor linii z użyciem którego będą narysowane. Jeżeli używamy struktury logicznej z pliku PDF to nazwy te uwzględniają już mapowanie z pod klucza *RoleMap* w słowniku korzenia drzewa struktury logicznej (tzn. jeżeli jakaś nazwa jest wybrana do pokazywania to będą pokazane gabaryty wszystkich elementów których nazwy mapują się bezpośrednio na tę nazwę).

Po wybraniu polecenia Edit->Columnize, jeżeli przeglądamy układ strony zanalizowany przez PDFMiner, możemy podzielić stronę na kolumny i ewentualnie przetworzyć potem własnym modułem. Polecenie Edit->Columnize all wykonuje tą operację od razu dla wszystkich stron.

Polecenie File->Export pozwala wyeksportować wczytany plik do formatu hOCR.

W zakładce "Physical structure" pokazywany jest słownik zasobów wybranej (przez kliknięcie na element lub przez kontrolki na górze) strony. W oknie z lewej strony (patrz rys. 6.2) pokazana jest lista czcionek i obrazów w słowniku (pokazywane są tylko te elementy słownika). Po kliknięciu na dany element w oknie z prawej strony pokazywane są różne informacje ze słownika fontu lub obrazka.

Pełne wsparcie dla PDF/A wymagałoby pokazywania także metadanych (m.in. o licencji fontu) zawartych pod kluczem *Metadata* w strumieniu zawierającym plik fontu ([PDFA05] *6.7.10 Font metadata*), które obecnie nie jest zaimplementowane.

Jeżeli wybrany element jest obrazkiem to dodatkowo w dolnej części prawego okna ten obrazek zostanie wyświetlony. Obrazek ten można zapisać do pliku poleceniem Edit->Export image.

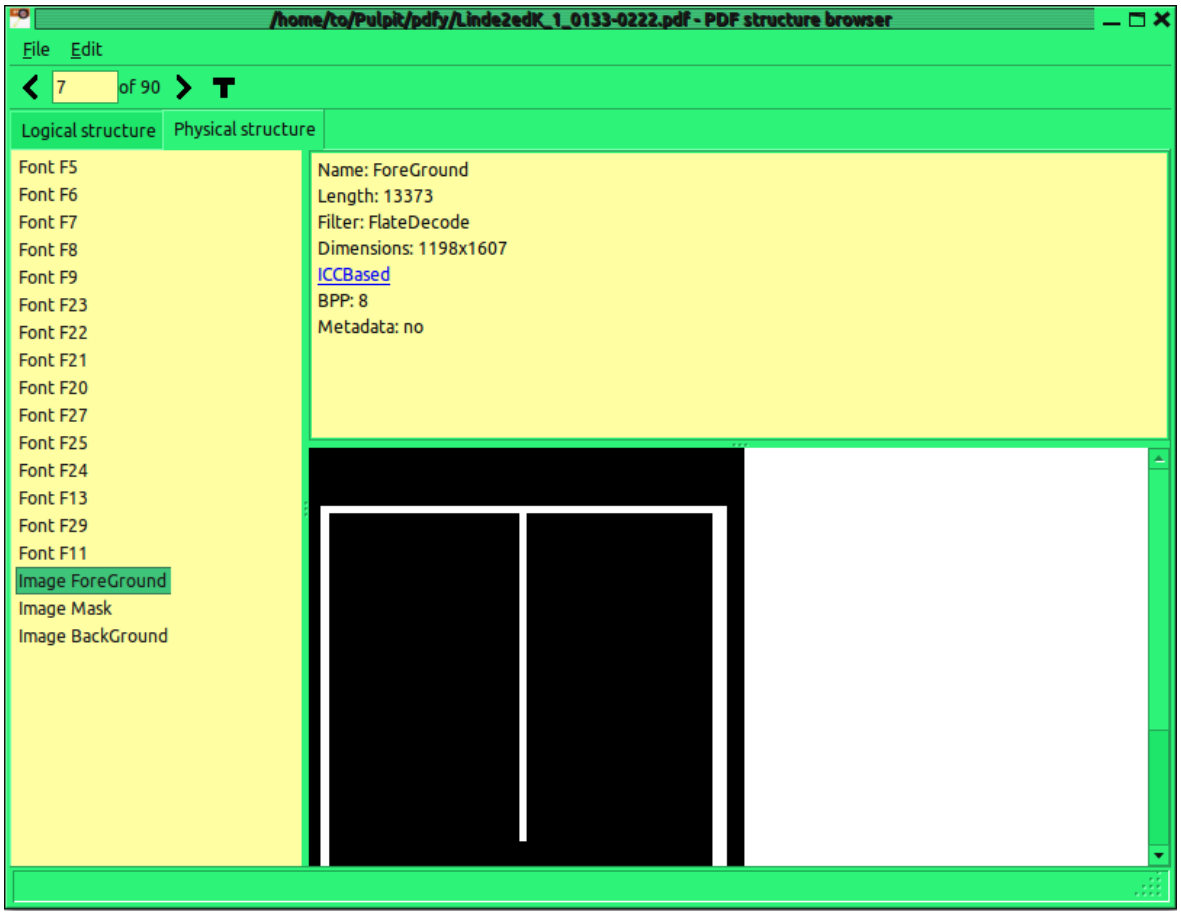
6.2. Opis implementacji

6.2.1. Pakiet *pdfastructurebrowser.py*

Pakiet ten zawiera klasy MainWindow, TagText, LazyTree i Preview. Implementują one interfejs graficzny przeglądarki roszerzając klasy z pakietu wxWidgets.

Po uruchomieniu przeglądarki tworzony jest obiekt klasy MainWindow, będący głównym oknem przeglądarki. Po wyborze polecenia File->Open (obsługiwanego przez metodę TagLib.__onOpen) i wybraniu pliku przez użytkownika program tworzy dla niego obiekt klasy TagLib i wywołuje na nim metodę TagLib.initialize. Metoda ta, poza wspomnianymi wcześniej operacjami, sprawdza czy plik jest plikiem PDF. Jeżeli tak, to za pomocą metody TagLib.getRoot pobierany jest obiekt klasy Node bedacy korzeniem drzewa struktury logicznej i przekazywany do obiektu klasy LazyTree zapisanego w polu MainWindow.__control.

Jeżeli otwierany plik nie jest plikiem PDF to traktowany jest jako plik XML. Sterowanie jest przekazywane do metody TagLib.__openXML, która wywołuje metodę PDFMinerPar-



Rysunek 6.3: Przeglądarka słowników zasobów stron - obraz

ser.parse i za pomocą metody PDFMinerParser.getResult pobiera korzeń drzewa układu strony (obiekt klasy PDFMinerNode) przekazany do MainWindow.__control. Przed wywołaniem PDFMinerParser.parser dla pliku tworzona jest zaślepka klasy XMLLib która udostępnia ten sam interfejs co TagLib (dzięki temu jeden kod obsługuje zarówno strukturę logiczną i układ strony, tylko w paru miejscach różni się w zależności od użytego trybu).

Jeżeli natomiast plik otwarto poleceniem File->Analize (obsługiwanym przez metode TagLib.__onAnalize), to sterowanie również jest przekazywane do TagLib.__openXML, ale wywołuje ona metodę PDFMinerParser.extractFromPDF. Metoda PDFMinerParser.getResult zwraca wtedy drzewo układu strony zanalizowane przez PDFMiner z pliku PDF. Przed wywołaniem metody PDFMinerParser.extractFromPDF tworzona jest zaślepka klasy XMLLib i jest wywoływana na niej metoda XMLLib.loadPTree, dzięki czemu zawiera ona informacje o słownikach zasobów stron z pliku PDF. Obiekt ze słownikami zasobów jest pobierany metodą XMLLib.getPhysicalTree (lub TagLib.getPhysicalTree jeśli otwarto plik PDF poleceniem File->Open) z obiektu MainWindow.__lib i przekazywany do kontrolki MainWindow.__tree2 klasy PhysList.

Utworzony obiekt klasy XMLLib lub TagLib jest zapamiętywany na zmiennej MainWindow.__lib.

We wszystkich przypadkach po wczytaniu pliku pokazywana strona jest ustawiana na pierwszą stronę w dokumencie - obiekt MainWindow.__preview klasy Preview rysuje jej gabaryty a obiekty odpowiedzialne za pokazywanie słowników zasobów stron (patrz opis pakietu

physbrowser.py) pokazują jej słownik zasobów. Wyjątkiem jest sytuacja gdy wczytujemy plik XML - wtedy nie pokazujemy słowników zasobów.

Klasa LazyTree implementuje kontrolkę wyświetlającą drzewo struktury logicznej. Po kliknięciu na węzeł w kontrolce wywołana zostaje metoda LazyTree.__onClick. Metoda ta pobiera z obiektu klasy Node reprezentującego węzeł informację o stronach na których on leży i przełącza pokazywaną stonę na jedną z nich. W tym celu pobiera metodą TagLib.getStrucutreToDrawById poddrzewo zawierające tylko elementy z danej strony (ewentualnie przycięte do jednej strony) i przekazuje do obiektu MainWindow.__preview wywołaniem metody Preview.setPageToView.

Pokazywaną stronę można też zmienić używając kontrolek na pasku narzędzi (MainWindow.__toolbar) obsługiwanych przez metody MainWindow.__onNext, MainWindow.__onPrev i MainWindow.__onEnter.

Klasa LazyTree analogicznie działa w przypadku otwarcia pliku XML lub użycia PDFMinera do zanalizowania układu strony. Wtedy zamiast drzewa struktury złożonego z obiektów klasy Node pokazuje ona drzewo układu strony złożone z PDFMinerNode. Z powodu różnic w implementacji metod XMLLib.getStrucutreToDrawById i TagLib.getStrucutreToDrawById istnieją drobne różnice w implementacji zaznaczania w zwróconym przez nie poddrzewie węzła dla którego została wywołana metoda LazyTree.__onClick. Element jest zaznaczany przez ustawienie flagi PDFMinerNode/Node.__selected na True. W przypadku przeglądania układu strony element jest zaznaczany bezpośrednio przez wywołanie metody PDFMinerNode.select, natomiast w przypadku przeglądania struktury logicznej PDF odbywa się to przez użycie metody TagLib.select która przekazuje do obiektu MainWindow.__lib[1] informację o zaznaczanym obiekcie, dzięki czemu jest on zaznaczany w metodzie TagLib.getStructureToDrawById.

Klasa Preview implementuje kontrolkę rysującą gabaryty elementów na stronie. Metoda Preview.setPageToView jest wywoływana przy wczytywaniu pliku i ustawianiu do pokazania pierwszej strony, w jednej z metod obsługujących kontrolki zmiany strony lub w metodzie LazyTree.__onClick. Przekazuje ona do obiektu podrzewo z elementami danej strony. Wszystkie elementy z poddrzewa są umieszczane na liście Preview.__els. Jeżeli użyty jest tryb pokazywania tekstu (włączany i wyłączany odpowiednią kontrolką na pasku narzędzi której naciśnięcie powoduje wywołanie metody MainWindow.__onText) to dodatkowo na liście Preview.__texts są umieszczane elementy reprezentujące znaki. Ta ostatnia funkcjonalność jest dostępna tylko przy przeglądaniu układu strony zanalizowanego przez PDFMiner.

Nazwy elementów do wyświetlenia są przechowywane w słowniku MainWindow.__tagmodel, które mapuje je na pary (*wartośc boolowska określająca czy gabaryty danego elementu są wyświetlane przez MainWindow.__preview*, *kolor w jakim będą wyświetlone brzegi gabarytów*). Słownik ten jest przy otwieraniu pliku PDF poleceniem File->Open inicjalizowany na standardowe nazwy elementów tagowanego PDF, z których mają być pokazywane *Div* i *P*. W pozostałych przypadkach słownik jest inicjalizowany na nazwy elementów układu strony, z których pokazane mają być `textline` i `textbox`. W przypadku użycia polecenia Edit->Viewing parameters wywoływana jest metoda MainWindow.__onViewing która tworzy okienko dialogowe klasy ModeDialog umożliwiające zmianę ustawień które elementy będą wyświetlane.

Za rysowanie strony odpowiada metoda Preview.__draw. Najpierws przechodzi ona listę Preview.__els i jeżeli znajdzie na niej element z flagą PDFMinerNode/Node.__selected ustawioną (w metodzie LazyTree.__onClick) na True to wypełnia jego gabaryt na żółto. Następnie ponownie przechodzi listę Preview.__els i rysuje gabaryty elementów które mają być wyświetlane zgodnie z ustawieniami w MainWindow.__tagmodel. Jeżeli włączony jest tryb rysowania znaków to przechodzi potem listę Preview.__texts i rysuje znaki (ignorując fonty i zwracając

[1]Obiekt klasy LazyTree ma do niego własną referencję w polu LazyTree.__lib

uwagę tylko na ich położenie na stronie).

gabaryty strony MainWindow.__preview pobiera z obiektu MainWindow.__lib za pomocą metody XMLLib/TagLib.getPageBBox.

Klasa TagText jest modyfikacją klasy RichTextControl z wxWidgets. Rozszerza ją ona o wyświetlanie podpowiedzi (po najechaniu myszą na tekst wyświetlają się nazwa i rozmiar fontu w tym miejscu tekstu). Ta funkcjonalność wymaga jeszcze dopracowania, ponieważ podpowiedź nie zawsze się pojawia. Oprócz tego przechwytuje ona kliknięcia na linki do fontu w metodzie TagTree.__onURL i przekazuje sterowanie do klasy LazyTree. Obiekt klasy LazyTree pamięta któremu linkowi odpowiada jaki font w polu LazyTree.__fontIdMap (fonty przechowywane w słowniku LazyTree.__fontIdMap to obiekty klasy Font będące dziećmi liści drzewa struktury). Po pobraniu fontu z tego słownika wywołuje ona metodę MainWindow.switchTabs, która przełącza widok na zakładkę słowników zasobów stron. Wywołana zostaje metoda PhysBrowser.show, która pokazuje odpowiedni font.

Klasa LazyTree udostępnia metodę LazyTree.getFontName, która przekazuje klasie TagText tekst z nazwą i rozmiarem fontu na podstawie identyfikatora linku, wykorzystując słownik LazyTree.__fontIdMap. Ponadto klasa TagText implementuje mechanizm znajdywania identyfikatora ostatniego linku przed daną pozycją w tekście. Dzięki temu klasa TagText wie jaką podpowiedź wyświetlić po najechaniu na dane miejsce tekstu.

6.2.2. Pakiet *physbrowser.py*

Pakiet zawiera dwie klasy. Pierwsza z nich, ImagePanel, rozszerza klasę wx.ScrolledWindow i obsługuje wyświetlanie obrazków z pliku PDF w przeglądarce słowników zasobów stron. Obiekt tej klasy znajduje się w polu MainWindow.__image.

Druga z klas, PhysList, rozszerza klasę wx.ListCtrl i implementuje wyświetlanie słowników zasobów stron oraz zawartości słowników posczególnych fontów i obrazków. Obiekt tej klasy jest pamiętany na zmiennej MainWindow.__tree2. Po załadowaniu pliku PDF obiekt PTree ze słownikami zasobów jest przekazywany do obiektu klasy PhysList metodą PhysList.setRoot i zapisywany w polu PhysList.__root.

Po ustawieniu pierwszej strony do pokazywania po wczytaniu pliku lub zmianie strony przez jeden z mechanizmów wspomnianych w opisie pakietu *pdfastructurebrowser.py* wywoływana jest metoda PhysList.show. Jej argumentem jest identyfikator obiektu strony której słownik zasobów chcemy pokazać. Metoda znajduje słownik tej strony w obiekcie PhysList.__root i wyświetla jego zawartość. Opcjonalny parametr seekedNode umożliwia podanie konkretnego fontu ze słownika (w postaci reprezentującego go obiektu klasy PTree), który zostanie wtedy pokazany. Dzięki temu po kliknięciu na link do czcionki w obiekcie MainWindow.__text można przejść do konkretnego fontu.

Po kliknięciu na element słownika zasobów uruchamiana jest metoda PhysList.__onClick. Znajduje ona odpowiedni element PTree i wyświetla informację o zawartym w nim obrazku lub foncie w kontrolce PhysList.__textCtrl klasy RichTextCtrl, będącej referencją do MainWindow.__text2. Jeżeli pokazywany obiekt jest obrazkiem, to dodatkowo jest on wyciągany z pliku PDF za pomocą metody getWxImage z pakietu *imagextract.py* i pokazywany za pośrednictwem metody MainWindow.showImage w kontrolce MainWindow.__image. Obrazek w tej kontrolce (w formie obiektu klasy wx.Bitmap) jest zapamiętywany w polu ImagePanel.__bmp, z którego pobiera go metoda MainWindow.__onExportImage obsługująca polecenie Edit->Export zapisujące aktualnie pokazywany obrazek do pliku.

Wśród informacji wyświetlanych w kontrolce PhysList.__textCtrl znajdują się m.in. kodowanie fontu i przestrzeń kolorów obrazka. Ponieważ czasem są one złożone, więc zaimplementowano ich pokazywanie w osobnym oknie dialogowym. W PhysList.__textCtrl umieszcza się w

takim wypadku link. Naciśnięcie na link powoduje wywołanie metody MainWindow.__onURL, która z kolei wywołuje metodę PhysTree.onURL. Następnie otwierane jest okno dialogowe w którym pokazywane jest kodowanie lub przestrzeń kolorów (które są wcześniej zapamiętywane w odpowiednich polach w metodzie PhysTree.__onClick).

6.2.3. Pakiet *dialogs.py*

Pakiet zawiera implementację okien dialogowych.

Okno dialogowe ColourSpaceDialog pokazuje przestrzeń kolorów lub kodowanie fontu. Po utworzeniu nowego obiektu tej klasy w metodzie PhysTree.__onURL w zależności od rodzaju informacji który mamy pokazać wywoływana jest metoda ColourSpaceDialog.onEncoding lub ColourSpaceDialog.onColourSpace za której pomocą przekazuje się odpowiedni obiekt do pokazania do okna. Dopiero potem okno jest pokazywane.

Ponieważ kodowanie lub przestrzeń kolorów mogą zawierać w sobie kolejne kodowanie lub przestrzeń kolorów, klasa ColourSpaceDialog implementuje linki prowadzące do nich z okna dialogowego. Po kliknięciu na nie metoda ColourSpaceDialog.__onURL otwiera nowe okno ColourSpaceDialog w którym pokazywane jest kodowanie lub przestrzeń kolorów do którego prowadzi link.

Okno dialogowe SimpleDialog pozwala na podanie przez użytkownika wartości tekstowej. Jest ono wykorzystywane przez metodę ModeDialog.__onAdd.

Okno dialogowe SimpleModuleDialog umożliwia załadowanie dowolnego modułu (pliku z kodem źródłowym Pythona), który następnie będzie wykorzystywany do analizy układu strony oraz podanie liczby kolumn na które należy podzielić stronę.

Okno dialogowe ModeDialog umożliwia zmianę ustawień rysowania strony przechowywanych w słowniku MainWindow.__tagmodel. Pokazuje ono listę nazw elementów (kluczy MainWindow.__tagmodel). Po zaznaczeniu nazwy wyświetla kontrolkę typu wx.CheckBox umożliwiającą zmianę ustawień czy element o danej nazwie będzie rysowany (obsługiwaną przez metodę ModeDialog.__onBox) oraz kontrolkę w którą można wpisać kolor w jakim będzie wyświetlony element (obsługiwaną metodą ModeDialog.__onType). Poza tym klikając w odpowiedni przycisk można w razie potrzeby dodać do listy nowy tag (metoda ModeDialog.__onAdd).

6.2.4. Pakiety *imagextract.py*, *pdfaimg.py* i część zaimplementowana w C++

Tytułowe pakiety odpowiadają za wyciąganie obrazków z plików PDF na potrzeby wyświetlania w przeglądarce i ewentualnego zapisywania na dysku.

Podstawą jest tutaj modyfikacja programu `pdfimages`, napisanego w C++ i stanowiącego część pakietu Poppler, który umożliwia wyciąganie obrazków z pliku. Modyfikacja polega na tym, że stworzony program, zamiast być uruchamiany jako samodzielna aplikacja, jest udostępniany jako funkcja w Pythonie. Funkcja ta to getImage z pakietu *pdfaimg.py*, który jest wygenerowanym przez narzędzie SWIG interfejsem między C++ a Pythonem.

Program `pdfimages` wyciąga wszystkie obrazki (można też podać numery stron z których mają być wyciągnięte). Modyfikacja umożliwia ich wyciągnięcie na podstawie numeru strony na której się znajdują i identyfikatora ich obiektu PDF. Ponieważ w przypadku obrazka będącego maską (tzn. takiego który jest pod kluczem Mask w słowniku jakiegoś innego obrazka) w programie pdfimages mamy dostęp nie do jego identyfikatora, tylko do identyfikatora obrazka przez nią maskowanego (tak przynajmniej było dla plików stworzonych przez FineReadera), to w takim przypadku podajemy metodzie getImages) identyfikator obiektu maskowanego i flagę informującą, że obrazek jest maską. Identyfikator ten i flaga są dodawane do obiektu PTree w metodach TagLib.__initializePTree i XMLLib.__initializePTree. Jeżeli maska maskuje

więcej niż jeden obrazek to program nie zadziała prawidłowo (najprawdopodobniej zakończy się z błędem).

Wybrane rozwiązanie umożliwiło szybkie rozwiązanie problemu wyciągania obrazków z plików PDF. Co prawda w słowniku obrazka przechowywanym w obiekcie PTree jest zawarty właściwy obrazek, który mógłby być rozkodowany i od razu wyświetlony, jednak w pakiecie PDFMiner (i wogóle w wolnym oprogramowaniu w Pythonie) brak funkcji umożliwiających rozkodowanie wszystkich możliwych sposobów kodowania obrazków. Być może warto byłoby zaimplementować w Pythonie brakujące funkcje dekodujące i dołączyć je do pakietu PDFMiner. Najprostszym rozwiązaniem byłoby stworzenie interfejsu do odpowiednich metod Popplera.

Pakiet imagextract.py udostępnia funkcję getWxImage która wywołuje getImage z odpowiednimi parametrami i na podstawie otrzymanych danych tworzy i zwraca obiekt klasy wx.Bitamp, który będzie użyty do wyświetlenia obrazka w kontrolce MainWindow.__image.

Rozdział 7

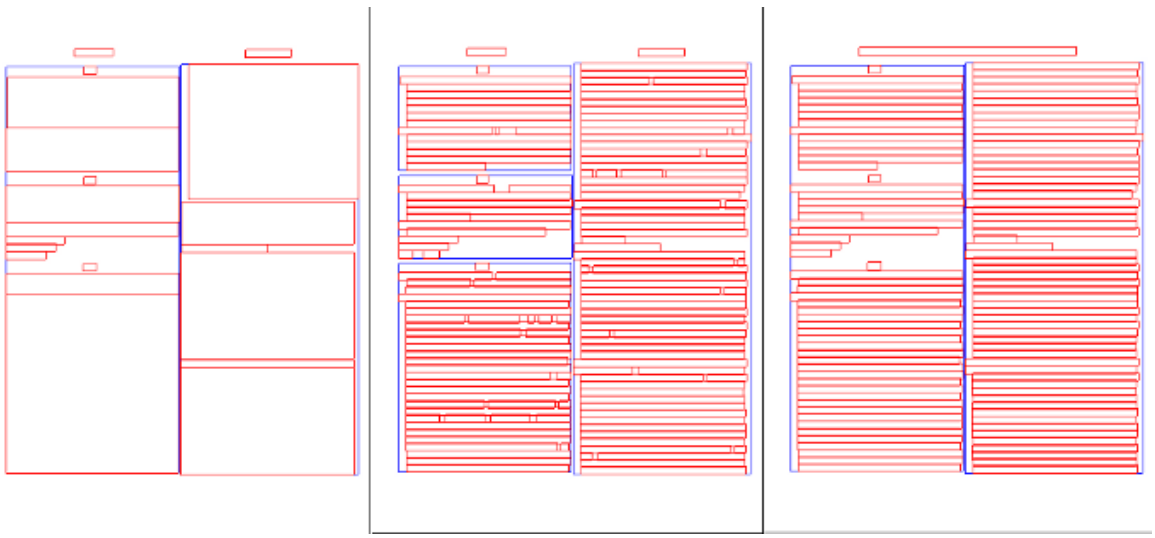
Konwerter PDF/A do formatu hOCR

7.1. Wprowadzenie

Konwerter umożliwia konwersję plików PDF/A i zwykłych plików PDF zawierających informacje o strukturze logicznej dokumentu do formatu hOCR. Elementy struktury logicznej formatu PDF są zamieniane na odpowiadające im elementy hOCR. Ponadto konwerter umożliwia przeprowadzenie analizy układu strony i zapis tej informacji w wynikowym pliku hOCR zamiast struktury logicznej.

Ideą przyświecająca stworzeniu konwertera była możliwość zapisywania wyników programu FineReader do postaci hOCR. FineReader zapisuje wyniki OCR w postaci plików PDF/A. Konwerter umożliwia ich przetworzenie na hOCR.

Do konwertera dodano możliwość zignorowania układu strony wygenerowanego przez FineReader i zapisanego w postaci struktury logicznej PDF. Zamiast niego można użyć układu strony będącego wynikiem działania algorytmu analizy układu strony z programu PDFMiner.



Rysunek 7.1: Porównanie układów strony wygnerowanych przez różne programy. Od lewej: FineReader, PDFMiner, PDFAUtilities z modulem użytkownika znajdującym paginę w konkretnym dokumencie.

Niektóre z przetwarzanych dokumentów mają przewidywalną postać - wszystkie strony (za wyjątkiem wstępu, itp.) były podzielone na taką samą ilość kolumn. Ponieważ analizator PDFMinera nie wykorzystuje tej informacji nie zawsze jest w stanie bezbłędnie podzielić cały dokument na kolumny, sklejając czasem niektóre kolumny lub dzieląc dokument na mniejsze bloki. Dlatego dodano do konwetera prosty algorytm, który wykorzystując tą informację dzieli każdą stronę na zadaną ilość kolumn.

Pojawił się również problem bezbłędnego rozpoznania żywej paginy. Ponieważ w każdym dokumencie postać paginy jest inna dodano do konwertera możliwość podania jako parametru skryptu napisanego pod konkretny dokument i bezbłędnie rozpoznającego paginę.

Doświadczenia nabyte podczas rozwiązywania problemu układu strony pozwoliły stwierdzić, że dobrym pomysłem byłby eksport układu strony zanalizowanego przed PDFMiner do hOCR a następnie ręczna edycja tego hOCR przy pomocy prostego edytora graficznego opartego np. na oknie układu strony z przeglądarki PDF/A.

7.2. Opis funkcjonalności

Konwerter jest uruchamiany poleceniem `pdfa2hocr`. Najprostszy przypadek użycia ma postać:

```
pdfa2hocr plik.pdf wynik.html
```

Polecenie to spowoduje eksport pliku *plik.pdf* do formatu hOCR z eksportem struktury logicznej pliku do struktury logicznej hOCR.

Polecenie:

```
pdfa2hocr -p plik.pdf wynik.html
```

zamiast eksportować strukturę logiczną z pliku PDF/A analizuje układ strony za pomocą programu PDFMiner i zapisuje go jako elementy struktury dokumentu.

Polecenie:

```
pdfa2hocr -p -i plik.pdf wynik.html
```

działa podobnie jak poprzednie, ale eksportuje tylko elementy typu `textbox` (odpowiedniki *ocr_carea* z hOCR), ignorując większe jednostki układu strony (`textgroup`) znalezione przez PDFMiner.

Podanie dodatkowej opcji `-u <LOCALE>` spowoduje podział tekstu w wynikowym dokumencie hOCR na wyrazy z użyciem biblioteki ICU dla lokalizacji *LOCALE*.

Podanie opcji `-m <MAPPING_FILE>` spowoduje użycie własnego mapowania elementów struktury logicznej PDF lub obiektów znalezionych przez analizator PDFMinera na elementy hOCR. Mapowanie powinno być zdefiniowane w pliku *MAPPING_FILE*.

Podanie opcji `-c <COL_NUM>` spowoduje podział każdej strony w dokumencie (nie wymienionej w argumencie opcji `-i`) na *COL_NUM* kolumn. Opcja może być użyta jedynie jeżeli jest jednocześnie użyta opcja `-p`.

Podanie opcji `-l <MODULE>` spowoduje przetworzenie każdej strony skryptem *MODULE* po podziale na kolumny (np. w celu znalezienia paginy). Opcja może być użyta jedynie jeżeli jest jednocześnie użyta opcja `-c`.

Podanie opcji `-g <PAGE_LIST>` spowoduje zignorowanie stron z listy stron oddzielonych przecinkami *PAGE_LIST* przez algorytm podziału na kolumny. Opcja może być użyta jedynie jeżeli jest jednocześnie użyta opcja `-c`.

Podanie opcji `-r <WIDTH>x<HEIGHT>` określa wymiary strony według których w wynikowym pliku będą określane gabaryty.

Oprócz tego istnieją długie wersje podanych powyżej opcji (np. `--module` dla `-l`). Szczegółowe informacje można uzyskać uruchamiając konwerter z opcja `-h`.

7.3. Opis implementacji

7.3.1. Pakiet *pdfa2hocr.py*

Pakiet ten zawiera funkcję main która obsługuje parametry konwertera na hOCR i w zależności od nich uruchamia różne sposoby konwersji.

Oprócz tego w pakiecie znajduje się metoda processModuleAndColumnsOptions, która po otrzymaniu z metody main parametrów wywołania programu:

- Jeżeli podano opcję `-c` tworzy obiekt klasy Columnizer.
- Jeżeli podano opcję `-g` to przekazuje do tego obiektu listę stron podanych jako jej argument metodą Columnizer.setPages.
- Jeżeli podano opcję `-l` to ładuje ona podany jako argument opcji moduł funkcją loadModule z pakietu *utils.py* i przekazuje do obiektu klasy Columnizer metodą Columnizer.setModule

Funkcja zwraca krotkę (*obiekt klasy Columnizer*, *liczba kolumn na które należy podzielić strony*) jeżeli podano opcję `-c` lub (*None*, *None*) w przeciwnym przypadku.

Jeżeli nie podano opcji `-p` to program tworzy obiekt klasy TagLib i uruchamia na nim metodę TagLib.initialize.

Konwersja PDF na hOCR z wykorzystaniem struktury logicznej zapisanej w pliku PDF

Jeżeli plik podany jako argument konwertera jest plikiem PDF to metoda TagLib.initialize otwiera plik. Za pomocą metody TagLib.getRoot z obiektu klasy TagLib jest pobierany korzeń drzewa struktury logicznej i przekazywany do konstruktora obiektu klasy HOCRExporter wraz ze ścieżką do pliku wynikowego oraz innymi parametrami. Plik wynikowy jest otwierany w konstruktorze. Następnie na utworzonym obiekcie wywoływana jest metoda HOCRExporter.export która eksportuje plik PDF do hOCR w trybie eksportu całego drzewa (patrz 7.3.2). Na koniec metoda HOCRExporter.save zamyka plik wynikowy.

Konwersja XML wygenerowanego przez PDFMiner na hOCR

Jeżeli natomiast podany jako argument konwertera plik jest plikiem XML z wynikiem analizy układu strony PDFMinera, to metoda TagLib.initialize informuje o tym zwracając wartość False. Zamiast obiektu klasy TagLib tworzony jest obiekt klasy XMLLib i przekazywany do konstruktora obiektu PDFMinerParser, na którym jest wywoływana metoda PDFMinerParser.parse, ładująca plik XML. Utworzone drzewo układu strony jest pobierane metodą PDFMinerParser.getResult.

Następnie uruchamiana jest funkcja processModuleAndColumnsOptions i jeżeli zwróci ona obiekt klasy Columnizer to jest na nim uruchamiana metoda Columnizer.columnize która przetwarza drzewo układu strony zgodnie z parametrami odczytanymi w funkcji processModuleAndColumnsOptions.

Następnie drzewo układu strony jest przekazywane do konstruktora obiektu klasy HOCRExporter który eksportuje je do hOCR tak jak w przypadku pliku PDF powyżej. Jedyną różnicą jest przekazanie do konstruktora parametru opcjonalnego xml ustawionego na True, ponieważ klasa HOCRExporter działa nieco inaczej dla eksportu układu strony niż dla eksportu struktury logicznej.

Konwersja PDF na hOCR z wykorzystaniem układu strony analizowanego przez PDFMiner

Jeżeli natomiast podano opcję `-p` to na początku tworzony jest obiekt klasy XMLLib który tworzy obiekt reprezentujący słowniki zasobów stron w metodzie XMLLib.loadPTree oraz obiekt klasy PDFMinerParser. Następnie działanie konwertera zależy od pozostałych opcji. Wywoływana jest funkcja processModuleAndColumnsOptions. Jeżeli zwraca ona krotkę (None, None), to znaczy że nie podano opcji `-c`. Jeżeli nie podano również opcji `-m` to wywoływana jest metoda PDFMinerParser.extractHOCRFromPDFDirect, która konwertuje plik na hOCR i zapisuje do pliku wyjściowego.

Konwersja PDF na hOCR z wykorzystaniem układu strony analizowanego przez pakiet *columnize.py* lub z wykorzystaniem układu strony analizowanego przez PDFMiner i mapowania na elementy hOCR podanego w pliku

Jeżeli natomiast podano opcję `-c` to znaczy, że funkcja processModuleAndColumnsOptions zwróciła obiekt klasy Columnizer i argument opcji `-c` (liczbę kolumn, która jest następnie przekazywana do wspomnianego obiektu metodą Columnizer.setCols).

W takim wypadku, lub jeżeli podano opcję `-m`, tworzony jest obiekt klasy HOCRExporter, do którego przekazywany jest argument opcji `-m` (ścieżka do pliku z mapowaniem elementów układu strony na elementy hOCR), jeżeli została podana. Parametr opcjonalny xml jest ustawiany na True. Następnie jest wywoływana metoda PDFMinerParser.extractHOCRFromPDF która która konwertuje plik na hOCR i zapisuje do pliku wyjściowego. Jako jej argumenty przekazywane są m.in. utworzone wcześniej obiekty klasy HOCRExporter i XMLLib oraz obiekt klasy Columnizer, który zostanie wykorzystany do analizy układu strony (jeżeli nie podano opcji `-c` to zamiast niego podawana jest wartość None oznaczająca, że nie należy przeprowadzać analizy układu strony).

7.3.2. Pakiet *hocrexport.py*

Pakiet implementuje klasę HOCRExporter która służy do eksportu pliku PDF (lub pliku XML z wynikami analizy układu strony przez PDFMiner) na hOCR.

W konstruktorze klasy (o miejscach w których tworzony jest obiekt wspomniano powyżej) tworzony jest obiekt klasy HOCRPDFMinerXMLMapping (jeżeli parametr opcjonalny xml jest ustawiony na True) lub HOCRStandardMapping (w przeciwnym przypadku) umieszczany w polu HOCRExporter.__mapping. Pierwsza z podanych klas implementuje domyślne mapowanie z elementów układu strony analizowanych przez PDFMiner na tagi HTML i elementy hOCR, natomiast druga domyślne mapowanie z elementów struktury logicznej PDF. W konstruktorze otwierany jest plik do którego będzie zapisywany dokument hOCR.

Jeżeli do konstruktora przekazano parametr opcjonalny mapping z nazwą pliku z mapowaniem, to jest on przekazywany do konstruktora obiektu HOCRStandardMapping lub HOCRPDFMinerXMLMapping, który w tworzonym obiekcie używa mapowania pliku a nie domyślnego. Klasa HOCRPDFMinerXMLMapping jest podklasą HOCRStandardMapping.

Klasa HOCRExporter działa w dwóch trybach.

Pierwszy to tryb eksportu strona po stronie. Jest on używany wyłącznie do eksportu układu strony zanalizowanego przez PDFMiner.

W metodzie PDFMinerParser.extractHOCRFromPDF wywoływana jest metoda HOCRExporter.beginExportByPages, która wypisuje tag otwierający <html>, nagłówek pliku hOCR oraz tag otwierający <body>. Następnie układ strony jest analizowany przez obiekt klasy PDFMinerConverter. Tworzy on poddrzewa układu strony dla każdej z analizowanych stron i przetwarza obiektem klasy Columnizer (jeżeli została podana opcja -c). Poddrzewo jest następnie przekazywane do metody HOCRExporter.exportPage.

Metoda najpierw wypisuje tag otwierający <div class="ocr_page" ... > dla strony po czym w pętli dla każdego z dzieci strony (tzn. elementu drzewa układu strony reprezentującego stronę) wywołuje metodę HOCRExporter.__getElement, która metodą HOCRStandardMapping.hOCRElement pobiera z HOCRExporter.__mapping obiekt klasy HOCRElement reprezentujący tag HTML i element hOCR odpowiadający danemu dziecku.

Następnie wywoływana jest metoda HOCRExport.__exportNode której argumentami są obiekt klasy PDFMinerNode będący dzieckiem strony, utworzony obiekt klasy HOCRElement oraz parametr opcjonalny pagesInTree ustawiony na True oznaczający że działamy w trybie eksportu strona po stronie.

Metoda HOCRExport.__exportNode wypisuje najpierw tag otwierający elementu hOCR odpowiadającego eksportowanemu węzłowi PDFMinerNode wywołując metodę o nazwie HOCRElement.start. Następnie, jeżeli eksportowany element nie jest liściem, to dla każdego dziecka elementu pobiera obiekt klasy HOCRElement z HOCRExporter.__mapping i wywołuje dla niego HOCRExport.__exportNode. Jeżeli eksportowany element jest typu `textline` i konwerter był wywołany z opcją -u, to dodatkowo jego zawartość tekstowa jest pobierana metodą PDFMinerNode.getTextContent po czym jest dzielona na wyrazy funkcją divideIntoWords z pakietu *utils.py*. W takim wypadku ustawiana jest flaga HOCRExporter.__useICU.

Jeżeli eksportowany element jest liściem, to dla jego dzieci będących fontami patrzymy, czy został wypisany tag otwierający opisujący styl fontu (flaga HOCRExporter.__tagHasFont) - jeżeli nie to wypisujemy początek nowego fontu, jeżeli tak to patrzymy czy font się zmienił i jeśli tak to wypisujemy tag zamykający starego fontu i wypisujemy nowy.

Dla dzieci będących napisami wypisujemy je do pliku. Jeżeli ustawiona jest flaga HOCRExporter.__useICU to wykorzystując informacje o podziale na wyrazy wypisujemy do pliku w odpowiednich miejscach tagi <span class="ocrx_word" ... > oznaczające słowa[1].

Na koniec metoda HOCRExporter.__exportNode wypisuje tag zamykający wywołując metodę HOCRElement.stop.

Na końcu metody PDFMinerParser.extractHOCRFromPDF wywoływana jest metoda HOCRExporter.endExportByPages która wypisuje tagi zamykające </body> i </html> oraz zamyka plik wynikowy.

Drugi z trybów to tryb eksportu całego drzewa. Używany jest w przypadku eksportu pliku XML z wynikami analizy układu strony PDFMinera i w przypadku eksportu struktury logicznej pliku PDF.

W takim przypadku uruchamiana jest metoda HOCRExporter.export. Wywołuje ona metodę HOCRExporter.__processNode na korzeniu drzewa.

Jeżeli węzeł drzewa dla którego została uruchomiona metoda HOCRExporter.__processNode jest korzeniem to uruchamia ona metodę HOCRExporter.__intializeRoot. Wypisuje ona początek pliku (tag <html>, nagłówek i tag <body>), potem wywołuje HOCRExporter.__pro-

[1]Szczegółowy opis implementacji w komentarzach do pliku *hocrexport.py*. Warto zwrócić uwagę, że jeżeli wewnątrz słowa zmieniana jest czcionka, to w miejscu zmiany czcionki słowo zostanie dodatkowo podzielone. Można by to ominąć stosując właściwość `groupid` dla tak podzielonego słowa.

cessNode dla dzieci i na koniec wypisuje tagi </body> oraz </html>.

Jeżeli zaś węzeł nie jest korzeniem, to metodą PDFMinerNode/Node.multipage[2] sprawdzamy, czy dany element leży na wielu stronach. Jeżeli tak, to dzielimy go na części metodą Node.split. Następnie wywołujemy HOCRExporter.__processNode jeszcze raz dla danego elementu (który po wywołaniu metody Node.split ogranicza się do jednej strony) i dla jego części ograniczonych do pozostałych stron na których znajdował się oryginalny element (udostępnianych przez Node.getCopies). W tym przypadku nie jest wywoływana metoda HOCRExporter.__exportNode.

Jeżeli zaś dany element leży tylko na jednej stronie, to sprawdzamy, czy leży on na stronie dla której ostatnio wypisano tag otwierający <div class="ocr_page" ... > (zmienna HOCRExporter.__page) i czy nie wypisano jeszcze tagu otwierającego dla żadnej strony. Jeżeli strona się zmieniła albo jeżeli nie wypisano jeszcze początku strony, to wypisujemy tag otwierający elementu hOCR reprezentującego stronę (w tym pierwszym przypadku zamykamy także tag poprzedniej strony).

Następnie wywołujemy metodę HOCRExporter.__exportNode dla danego elementu z parametrem pagesInTree ustawionym na False, co informuje metodę, że jesteśmy w trybie eksportu całego drzewa.

Wyjątkiem jest tutaj element struktury logicznej o nazwie *Document*, jeżeli jest on jedynym dzieckiem korzenia. Nie jest on dzielony między strony ani nie jest dla niego sprawdzane czy strona się zmieniła - od razy wywołujemy metodę HOCRExporter.__exportNode.

Metoda HOCRExporter.__exportNode działa w przypadku trybu eksportu całego drzewa nieco inaczej. Po pierwsze na słowa nie jest dzielona zawartość elementu `textline`, tylko liści. Po drugie, jeżeli przetwarzany element nie jest liściem to dla jego dzieci nie wywołujemy metody PDFMiner.__exportNode tylko PDFMiner.__processNode.

7.3.3. Pakiet *hocrdirectconverter.py*

Pakiet zawiera klasę HOCRDirectConverter będącą rozszerzeniem klasy PDFLayoutAnalyzer. Działa ona analogicznie do klasy PDFMinerConverter, tylko zamiast tworzyć drzewo struktury na podstawie informacji otrzymanych w metodzie HOCRDirectConverter.receive_layout wypisuje je od razu do pliku w postaci hOCR. Mapowanie z elementów układu strony zanalizowanych przez PDFMiner na elementy hOCR jest tutaj wbudowane w kod źródłowy (takie jak w metodzie HOCRPDFMinerXMLMapping.__init__), dlatego jeśli chcemy użyć mapowania z pliku nie możemy skorzystać z tej klasy i musimy ekportować z użyciem klas PDFMinerConverter i HOCRExporter.

7.3.4. Eksport do hOCR z przeglądarki

Oprócz tego można wyeksportować do hOCR aktualnie przeglądany dokument w przeglądarce. Dzieje się tak po wybraniu polecenia File->Export, co powoduje wywołanie metody MainWindow.__onExport. Eksportowane jest drzewo pokazywane w obiekcie MainWindow.__control w analogiczny sposób jak eksport struktury logicznej PDF lub pliku XML w konwerterze.

[2]Chociaż właściwości struktury układu strony analizowanego przez PDFMiner powodują, że PDFMinerNode.multipage zawsze daje False. Elementy układu strony typu page są domyślnie ignorowane, ponieważ wypisywanie elementów hOCR *ocr_page* zapewnia algorytm opisany poniżej wspólny dla układu strony i struktury logicznej PDF.

Rozdział 8

Analiza układu strony

8.1. Użyty algorytm

Przy implementacji własnego modułu analizy układu strony przyjęto założenie, że z góry znamy liczbę kolumn na które podzielone są strony i na każdej stronie jest tyle samo kolumn.

1. PDFMiner analizuje układ strony.
2. Z układu strony wybieramy wszystkie linie (ponieważ PDFMiner w większości przypadków znajduje kolumny lub kawałki kolumn większość z tych linii to kawałki linii ograniczone do jednej kolumny).
3. Sortujemy linie po górnym brzegu gabarytu w porządku z góry strony na dół.
4. Łączymy dwie linie jeśli jedna występuje bezpośrednio po drugiej po powyższym posortowaniu i jeżeli wysokość jednej jest w całości zawarta w drugiej lub jeżeli część wspólna ich wysokości wynosi ponad 60% wysokości pierwszej linii. Po połączeniu dwóch linii tak połączona linia może być łączona z kolejnymi liniami jeżeli spełnione są powyższe warunki.
5. Dzielimy linie na trzy grupy (staramy się podzielić stronę na trzy równe części w poprzek).
6. W każdej linii znajdujemy $k-1$ największych odstępów między dwoma znakami (gdzie k - liczba kolumn).
7. Jeżeli w zbyt dużej liczbie linii jest za mało znaków by podzielić je na k kolumn podział strony zakończy się z błędem.
8. Dla każdego ze znalezionych odstępów obliczamy wartość, która jest sumą "części wspólnych" danego odstępu i każdego odstępu z linii (w obrębie tej samej grupy) innych niż ta do której należy dany odstęp. "Część wspólna" dwóch odstępów to stosunek długości ich przecięcia do długości jednego z nich.
9. Dla każdej z grup wybieramy $k-1$ odstępów dla których ta wartość jest największa. W obrębie grupy dwa odstępy nie mogą na siebie nachodzić.
10. Następnie bierzemy oryginalne (sprzed łączenia) linie i sprawdzamy, czy są przed (zaczynają się przed początkiem i kończą co najwyżej[1] na końcu odstępu) czy po (zaczynają

[1] "co najwyżej w miejscu X" - dokładnie w X lub na lewo od X

się co najmniej[2] na początku odstępu) odstępie. W zależności od tego przydziela je do kolumn - ponieważ PDFMiner analizuje strukturę więc istnieje duże prawdopodobieństwo, że dana linia jest w całości przed lub po.

11. Jeżeli natomiast linia nie spełnia żadnego z powyższych warunków to przydzielamy do kolumn pojedyncze znaki. Znak jest przydzielany do kolumny jeżeli znajduje się przed kończącym ją odstępem. Uważamy, że znak jest przed odstępem, jeżeli:

    - Znak kończy się co najwyżej na początku odstępu.
    - Znak zaczyna się co najwyżej na początkku odstępu i kończy przed końcem odstępu.
    - Jeżeli znak zaczyna się wewnątrz odstępu i kończy wewnątrz odstępu to żeby ustalić, czy znak jest przed odstępem stosujemy następujący algorytm:
        - Obliczamy wartość $affinityBefore$ oznaczającą największy odstęp między dwoma znakami w ciągu kolejnych znaków w którym ostatni to dany znak a pierwszy to pierwszy znak idąc w lewo od danego znaku który zaczyna się przed początkiem odstępu (oznaczmy go X)[3].
        - Obliczamy wartość $affinityAfter$ oznaczającą największy odstęp między dwoma znakami w ciągu kolejnych znaków w którym pierwszy to dany znak a ostatni to pierwszy znak idąc w prawo od danego znaku który kończy się po końcu odstępu (oznaczmy go Y)[4].
        - Jeżeli nie ma znaku Y to znaczy, że znak jest przed odstępem.
        - W przeciwnym przypadku jeżeli nie ma znaku X to znaczy, że znak nie jest przed odstępem.
        - W przeciwnym przypadku jeżeli wartość $affinityBefore$ jest mniejsza niż 0.1 to znaczy, że znak jest przed odstępem.
        - Jeżeli natomiast wartość $affinityAfter$ jest różna od zera a stosunek $affinityBefore$ do $affinityAfter$ wynosi mniej niż 0.25 to także znak jest przed odstępem.
        - Jeżeli żaden z powyższych warunków nie został spełniony, to znaczy, że znak jest za odstępem.
        - Znaki następne i poprzednie pobieramy tylko z oryginalnej linii (zwróconej przez analizator PDFMinera) w której znajduje się znak.

12. W obrębie każdej tak utworzonej kolumny sortujemy linie względem górnego brzegu ich gabarytów.

13. Następnie sklejamy linie. Dwie linie sklejamy gdy jedna następuje bezpośrednio po drugiej po posortowaniu (i linia powstała ze sklejenia dwóch innych może być sklejona z następną) oraz spełniony jest jeden z warunków:

    - Wysokość jednej jest w całości zawarta w wysokości drugiej i część wspólna ich wysokości wynosi ponad 90% wysokości wyższej linii.
    - Część wspólna ich wysokości wynosi ponad 90% wysokości każdej z linii.

[2] "co najmniej w miejscu X" - dokladnie w X lub na prawo od X
[3] Wartość ta może być ujemna jeśli znaki na siebie nachodzą.
[4] j.w.

8.2. Pakiet *columnize.py*

Algorytm jest zaimplementowany w pakiecie *columnize.py*, a konkretnie w klasie Columnizer.

Oprócz wspomnianych powyżej sytuacji gdy obiekt tej klasy jest tworzony w funkcji processModuleAndColumnsOptions w pakiecie *utils.py* jest on tworzony w metodach MainWindow.__onColumnize i MainWindow.__onColumnizeAll dzielących na kolumny odpowiednio aktualnie widoczną w przeglądarce stronę i wszystkie strony. Metody MainWindow.__onColumnizeAll i MainWindow.__onColumnize obsługują wybranie poleceń Edit->Columnize all i Edit->Columnize w przeglądarce. Pokazują one okno dialogowe SimpleModuleDialog, które umożliwia podanie liczby kolumn na które mamy podzielić strony i modułu użytkownika dodatkowo przetwarzającego strony. Moduł jest przekazywany do obiektu klasy Columnizer poleceniem Columnizer.setModule.

Żeby podzielić na kolumny całe drzewo należy wywołać na nim metodę Columnizer.columnize przekazując jako parametr korzeń drzewa struktury i liczbę kolumn na które należy podzielić strony. Metoda ta wywołuje następnie dla poszczególnych stron dzielącą je na kolumny metodę Columnizer.columnizePageUsingGroups.

Można też do utworzonego obiektu klasy Columnizer przekazać docelową liczbę kolumn poleceniem Columnizer.setCols. Następnie można go użyć do podzielenia pojedynczej strony wywołując metodę Columnizer.columnizePageUsingGroups i przekazując jej jako parametr korzeń poddrzewa struktury reprezentującego stronę.

Metoda Columnizer.columnize jest używana przy eksporcie drzewa struktury logicznej PDF w metodzie main pakietu *pdfa2hocr.py* oraz w metodzie MainWindow.__onColumnizeAll, natomiast Columnizer.columnizePageUsingGroups w klasie PDFMinerConverter i metodzie MainWindow.__onColumnize.

Metoda Columnizer.columnizePageUsingGroups wykonuje przedstawiony wcześniej algorytm i ewentualnie przetwarza dodatkowo strony modułem podanym przez użytkownika. Szczegóły implementacyjne algorytmu opisano w komentarzach do pliku.

Być może warto byłoby zintegrować powyższy algorytm z analizatorem układu strony PDFMinera (metoda LTLayoutContainer.analyze), który przekazywałby następnie zanalizowany układ strony do obiektu klasy HOCRDirectConverter bez pośrednictwa drzewa węzłów klasy PDFMinerNode.

Różne parametry algorytmu są w tej chwili umieszczone w kodzie źródłowym. W razie potrzeby można umożliwić podanie ich jako parametrów programu `pdfa2hocr`.

8.3. Moduł użytkownika

Powyższy algorytm dzieli dokument na kolumny, ale do kolumn włącza paginę. Do oddzielania paginy służy moduł użytkownika, który powinien być napisany pod konkretny dokument.

8.3.1. API

Implementując moduł należy na początku umieścić polecenie `from pdfminerconverter import PDFMinerNode`. W module powinna znaleźć się funkcja customProcessing, której jedyny argument jest elementem drzewa układu strony reprezentującym pojedynczą stronę. Po zakończeniu działania metody w obiekcie tym powinien się znaleźć przetworzony przez moduł układ strony.

Działając na poddrzewie z korzeniem w przekazanym elemencie możemy korzystać z publicznych metod klasy PDFMinerNode. Ważne jest, by po jakichkolwiek działaniach na dzieciach jakiegoś węzła (dodajemy coś lub usuwamy z listy PDFMinerNode.getChildren, ustawia-

my listę dzieci metodą PDFMinerNode.setChildren) wywołać metodę PDFMinderNode.resetBbox która ponownie przeliczy gabaryty.

Przydatne mogą być np. metody PDFMinerNode.getPageId, która zwraca numer strony na której znajduje się węzeł zmniejszony o jeden, PDFMinerNode.join, która łączy swój argument z węzełem na którym wywołano metodę w jeden i zapisuje go na węźle na którym wykonano metodę.

Jeżeli tworzymy nowy obiekt klasy PDFMinerNode, do którego potem dodajemy dzieci, musimy wywołać metodę PDFMinerNode.resetBbox, która obliczy gabaryt elementu (dostępny przez metodę PDFMinerNode.__getBbox) ustawiając go na sumę gabarytów dzieci.

8.3.2. Przykład modułu - znajdowanie paginy w słowniku Lindego

Na załączonej płycie CD znajduje się przykładowy moduł użytkownika *pagina.py* służący do znajdowania paginy w słowniku Lindego. Słownik Lindego ma dwie kolumny. Górna pagina jest zawsze pierwszą linią tekstu w dokumencie, dlatego jest ona po prostu wyjmowana z kolumn i dodawana do osobnego elementu `textbox`.

Na pierwszej stronie w miejscu gdzie normalnie powinna być pagina znajduje się informacja o literze na jaką zaczynają się kolejne hasła ("A, a."). Dlatego pagina jest w tym jednym miejscu rozpoznawana niepoprawnie. Można ten problem rozwiązać np. sprawdzając numer strony w module i dla stron na których pagina jest nietypowa użyć innej reguły.

Znajdując paginę dolną bierzemy ostatnią linię z lewej kolumny - jeżeli jest tam napis "Słownik Lindego wyd. 2. Tom I." (sprawdzamy czy odległość redaktorska jest odpowiednio mała, bo błędy OCR powodują, że napis tam umieszczony nie jest zupełnie identyczny ze "Słownik Lindego wyd. 2. Tom I.") to znaczy, że jest to dolna pagina. Jeżeli jest tam taki napis to jednocześnie ostatnia linia z prawej kolumny także jest dolną paginą.

Może się też zdarzyć, że tego napisu nie ma. Wtedy patrzymy na ostatnią linię w prawej kolumnie i jeżeli jest krótsza niż średnia długość linii w kolumnie to uznajemy ją za paginę. Jeżeli zaś nie jest to sprawdzamy jeszcze, czy jej górny kraniec znajduje się poniżej dolnego krańca ostatniej linii w lewej kolumnie. Jeżeli tak jest, to również uznajemy ją za paginę.

Linie które uznamy za dolną paginę usuwamy z kolumn i dodajemy do nowego elementu `textbox`.

Wraz z analizatorem układu strony moduł ten umożliwia bezbłędny (poza wspomnianym powyżej problemem z "A, a.") podział pierwszych 50 stron słownika Lindego na kolumny i paginę.

Rozdział 9

Walidator plików w formacie PDF/A

9.1. Opis funkcjonalności

Walidator sprawdza w tej chwili wybrane wytyczne określonych w specyfikacji PDF/A, głównie te dotyczące wiernego odtwarzania wyglądu dokumentu. Są to głównie ograniczenia nakładane na obrazki, intencje renderingu ([PDF01] *4.5.5 CIE-Based Color Spaces*, sekcja *Rendering Intents*), stany graficzne ([PDF01] *4.3.4 Graphics State Parameter Dictionaries*), fonty, adnotacje i akcje oraz zakaz używania przezroczystości, określone w rozdziałach *6.2*, *6.3*, *6.4*, *6.5* i *6.6* specyfikacji.

Funkcjonalność programu jest bardzo prosta - należy wywołać go z jednym argumentem którym jest plik do zwalidowania. Następnie program wypisze błędy które wystąpiły globalnie w pliku i w poszczególnych stronach. Jeżeli żaden błąd się nie pojawi to program wypisuje komunikat `File OK`.

9.2. Opis implementacji

Program implementuje metoda main z pakietu *pdfavalidate.py*. Metoda ta tworzy obiekt klasy PDFAValidator z tego samego pakietu, który zajmuje się właściwą walidacją. Wywołana zostaje na nim metoda PDFAValidator.validate, która otwiera najpierw dokument przy pomocy metody PDFAValidator.__loadDocument. Metoda PDFAValidator.__loadDocument działaja podobnie jak funkcja process_pdf z pakietu PDFMiner. "Urządzeniem" jest w tym przypadku obiekt klasy DummyConverter, służący jako zaślepka. Zamiast obiektu klasy PDFPageInterpreter zostaje użyty obiekt klasy PDFAPageInterpreter z pakietu *pdfareader.py*. Klasa ta jest podklasą PDFPageInterpreter. W zamierzeniu miały się w niej znaleść metody przesłaniające metody klasy PDFPageInterpreter sprawdzające zgodnośc poleceń i ich argumentów z formatem PDF/A. W chwili obecnej sprawdzany jest jedynie argument polecenia ri. Różnicą wobec funkcji proces__pdf jest to, że wywołanie metody PDFAPageInterpreter.__process_page, która spowoduje sprawdzenie poprawności poleceń dla poszczególnych stron, nie jest wykonywane w metodzie PDFAValidator.__loadDocument tylko w opisanej poniżej pętli.

Po wykonaniu metody PDFAValidator.__loadDocument metoda PDFAValidator.validate wywołuje metodę PDFAValidator__validateDocumentCatalog, która sprawdza zgodność z formatem PDF/A katalogu dokumentu, a następnie w pętli przechodzi wszystkie strony i sprawdza ich słowniki zasobów oraz adnotacje.

Rozdział 10

Podsumowanie

Stworzony zestaw narzędzi, mimo pewnych niedociągnięć (warto byłoby np. poprawić efektywność pakietu), okazuje się użyteczny. Narzędzia umieszono w postaci jednego pakietu pod adresem: http://students.mimuw.edu.pl/ to236111/PDFAUtilities/pdfautilities.tar.gz.

Przeglądarka (`pdfastructurebrowser.py`) umożliwia przeglądanie struktury logicznej dokumentów PDF, a co za tym idzie układu strony wygenerowanego przez program FineReader. Co prawda nie obsługuje on w pełni formatu PDF, jednak obsługuje dokumenty tworzone przez FineReader i niektóre inne dokumenty tekstowe. Głównym problemem jest obecnie brak obsługi obrazów w strukturze logicznej i zawartości w elementach nie będących liśćmi.

Poza tym stworzony konwerter (`pdfa2hocr.py`) umożliwia generowanie plików hOCR które mogą być dalej przetwarzane przez inne programy wykorzystywane w Katedrze Lingwistyki Formalnej UW. Dzięki użyciu programu PDFMiner potrafi przeprowadzać analizę układu strony której wyniki są zapisywane w tworzonych plikach. W konwerterze brakuje możliwości określenia sposobu wypisywania informacji o fontach lub zupełnego jego wyłączenia oraz jego zgodności ze specyfikacją. Pozostałe problemy nie wpływają na użyteczność programu.

Walidator (`pdfavalidate.py`) PDF/A sprawdza w chwili obecnej jedynie zgodność z najprostszymi wymogami określonymi w specyfikacji PDF/A.

Do PDFAUtilities zostały włączone fragmenty kodu z pakietów PDFMiner i FontTools zmodyfikowane na jego potrzeby. Ponadto do pakietu włączono zmodyfikowany program `pdfaimages` z biblioteki Poppler.

Dodatek A

Opis zawartości płyty CD

Na załączonej płycie CD znajdują się:

- katalog z kodem źródłowym stworzonego pakietu (`src`),
- katalog z dokumentacją (`doc`), zawierający:
  - elektroniczną kopię niniejszej pracy (`pdfa-praca.pdf`),
  - źródła niniejszej pracy (katalog `pdfa-praca-src`),
  - instrukcję instalacji (`instalacja.txt`),
  - instrukcję użytkownika (`instrukcja.txt`),
  - informację o licencji (`LICENSE`).

Dodatek B

GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication

that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Bibliografia

[Breu] Thomas Breuel, *The hOCR Embedded OCR Workflow and Output Format*, https://docs.google.com/View?docid=dfxcv4vc_67g844kf

[PDF08] *Document management - Portable document format - Part 1: PDF 1.7*, Adobe Systems Incorporated, 2008

[Hara07] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007

[PDFA05] *ISO-19005-1 - Document management - Electronic document file format for long-term preservation - Part 1: Use of PDF 1.4 (PDF/A-1)*, International Standards Organization, 2005

[PDFA07] *ISO-19005-1 - Document management - Electronic document file format for long-term preservation - Part 1: Use of PDF 1.4 (PDF/A-1), Technical Corrigendum 1*, International Standards Organization, 2007

[PDF01] *PDF Reference, Third Edition, Adobe Portable Document Format Version 1.4*, Addison-Wesley, 2001

[ICU] http://userguide.icu-project.org/boundaryanalysis

[PDFMin] http://www.unixuser.org/~euske/python/pdfminer/index.html

[PIL] http://www.pythonware.com/products/pil/

[Pop] http://poppler.freedesktop.org/

[PyICU] http://pyicu.osafoundation.org/

[SWI] http://www.swig.org/

[TTX] http://sourceforge.net/projects/fonttools/

[WX] http://www.wxpython.org/