VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

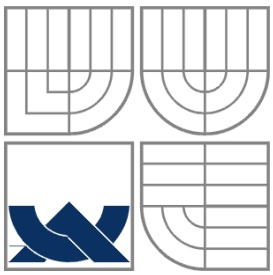
NÁSTROJ PRO TRANSFORMACE DOKUMENTŮ PDF

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

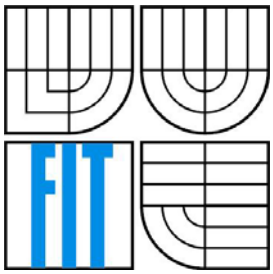
AUTOR PRÁCE
AUTHOR

Bc. LADISLAV ŠÚSTEK

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁSTROJ PRO TRANSFORMACE DOKUMENTŮ PDF

TOOL FOR PDF DOCUMENT TRANSFORMATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. LADISLAV ŠÚSTEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2008

Abstrakt

Výsledkem mé diplomové práce je knihovna pro programovací jazyk Java, která transformuje PDF soubory na XHTML za pomocí knihoven PDFBox a FONTBox. Výsledný XHTML soubor je vytvářen pomocí CSS stylů, které umožňují jeho snadnou modifikaci. Knihovna extrahuje z PDF souboru text, obrázky, ale i grafické prvky.

Klíčová slova

Java, XHTML, CSS, PDFBox, FONTBox, PDF, transformace PDF na XHTML

Abstract

Result of my diploma work is library for Java programming language. Which transform PDF to XHTML files using by open source library PDFBox and FONTBox. Final XHTML file is made by CSS style which allows its easier modification. Library extracts text, images and graphic objects from PDF.

Keywords

Java, XHTML, CSS, PDFBox, FONTBox, PDF, transformation from PDF to XHTML

Citace

Šústek Ladislav: Nástroj pro transformaci dokumentů PDF. Brno, 2008, diplomová práce, FIT VUT v Brně.

NÁSTROJ PRO TRANSFORMACE DOKUMENTŮ PDF

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením
Ing. Radka Burgeta, Ph.D.

.....
Ladislav Šústek
3.1.2008

Poděkování

Děkuji panu Ing. Radku Burgetovi, Ph.D. za jeho odborné vedení.

Obsah

Seznam tabulek a obrázků 3
1 Úvod 4
2 Použité technologie 5
2.1 PDF 5
2.1.1 Historie 5
2.1.2 Co je to PDF 5
2.1.3 Rozdíly mezi PDF a Postskript 6
2.1.4 Struktura PDF souboru 7
2.1.4.1 Header 7
2.1.4.2 Body 8
2.1.4.3 Cross-Reference Table 8
2.1.4.4 Trailer 9
2.1.5 Jak se PDF aktualizuje 9
2.1.6 Datové typy PDF 11
2.1.6.1 Nepřímé objekty 11
2.1.6.2 Streams 12
2.1.6.3 Katalogový strom 12
2.1.6.4 Stránková stromová struktura 13
2.1.7 Grafické operátory 14
2.1.7.1 Grafické objekty 15
2.1.7.2 Souřadný systém 18
2.1.7.3 Transformací matice 20
2.2 PDFBox 21
2.3 FontBox 22
2.4 Java SE 22
2.4.1 Základní vlastnosti jazyka Javy 22
2.4.2 Nevýhody Javy 23
2.5 XHTML 23
2.5.1 Vztah k HTML 24
2.6 CSS 24
2.6.1 Použité CSS vlastnosti 25
3 Cíle 26
3.1 Extrakce textu 26
3.1.1 Pozice textu na stránce 26

3.1.2 Velikost a barva písma 27
3.2 Grafické prvky 27
3.3 Obrázky 28
3.4 Další možná rozšíření 28
4 Návrh a Implementace 29
4.1 Extrakce textu 29
4.2 Tvorba vzhledu dokumentu 32
4.2.1 Vlastnosti a zobrazení písma 32
4.2.2 Grafické prvky 34
4.2.3 Obrázky 34
4.3 Kódování češtiny 35
4.4 Problémy s grafickou skladbou dokumentu 36
4.5 Ukázka datového toku v PDF souboru 38
5 Souhrn operátorů 40
6 Použití 42
7 Testování 43
8 Závěr 44
9 Literatura 45

Seznam tabulek a obrázků

Tabulka 1 - Kategorie operátorů 17
Tabulka 2 - Souhrn Operátorů 40

Obrázek 1 - Čtyři typy struktury dokumentu PDF 6
Obrázek 2 - Struktura PDF 7
Obrázek 3 - Struktura aktualizování PDF souboru 10
Obrázek 4 - Zobrazovací prostor 18
Obrázek 5 - Uživatelský prostor 19
Obrázek 6 - Efekty transformační matice 20
Obrázek 7 - Pořadí aplikování transformací 21

1 Úvod

Diplomová práce se zabývá zpracováním souborů PDF (zkratka pochází z anglického názvu **P**ortable **D**okument **F**ormat) a následnou transformací do XHTML (e**x**tensible **h**yper**t**ext **m**arkup **l**anguage*)*. PDF nám umožňuje přenášet dokumenty mezi různými operačními systémy a prohlížeči v nezměněné formě. Soubor může obsahovat bohatou škálu grafických prvků, různých druhů fontů, videí a mnoho dalších zapouzdřených objektů.

Výhodou tohoto formátu oproti jiným je jak snadný přenos v nezměněné formě, což je vždy největší problém u dokumentů, tak i možnost zvětšení celého dokumentu. Pokud přesáhne rozměry okna, ve kterém je zobrazován, je možné jej posouvat pomocí rolovacích lišt na jeho okrajích. Nezvětšuje se pouze určitá část dokumentu, ale mění se jeho měřítko. I při takové změně je zaručen stejný vzhled dokumentu. Tuto vlastnost zejména oceňují lidé se zrakovou vadou.

Mohlo by se zdát, že PDF má pouze klady a je výborným formátem pro naše dokumenty. Toto by byla pravda v případě, že není nutné již s dokumentem v budoucnu nijak pracovat. Pokud ovšem potřebujeme získat nějaké informace z PDF již jde se potýkáme s problémy. Těmi se budu zabývat v jedné z dalších částí diplomové práce.

XHTML jedná se o jazyk pro tvorbu webových aplikací, vyvinutý konsorciem W3C. Vyvíjel se z existujícího HTML a je mnohem silnější, flexibilnější, mocnější a má blíže k tomu, být využíván v budoucnu. Více o něm v jedné z následujících kapitol.

Postupně se v diplomové práci budu zabývat specifikací a základními rysy souboru PDF a technikami a postupy, které byly použity při extrakci textu a převodu do formátu XHTML.

Problémy, které se vyskytly během řešení a možným dalším vývojem softwaru a samozřejmě jeho testováním.

2 Použité technologie

2.1 PDF

2.1.1 Historie

PDF „přichází na svět“ v 90. letech, ale jeho adaptace je pomalá. Nástroje na jeho tvorbu i prohlížení byly komerční a byly proto zpoplatněny. První verze PDF nepodporovaly hyperlink, což hodně snižovalo jeho použití na internetu. PDF soubory byly větší než prostý text a to znamenalo delší dobu stahování pomocí modemů, které se v té době používaly k přihlášení na internet. Také otevírání trvalo tehdejším počítačům delší dobu něž běžné načítání dokumentu. Navíc se vyskytovaly konkurenční formáty jako například: Envoy, Common Ground Digital Paper a také firma Adobe vlastnila PostSkript.

Adobe brzy začala šířit Acrobat Reader bez poplatků a věnovat více pozornosti podpoře formátu PDF, který se nakonec stal standardem pro tisknutelné dokumenty.

PDF formát byl několikrát měněn. Vzniklo několik verzí tohoto formátu, ale všechny jsou navzájem zpětně kompatibilní. Každou verzí byly do PDF přidány nové rysy. Například zabezpečení dokumentu heslem, oddělení barvy od objektu, interaktivní elementy na stránce, formuláře, digitální podpis, obrázky, videa a mnoho dalších užitečných součástí.

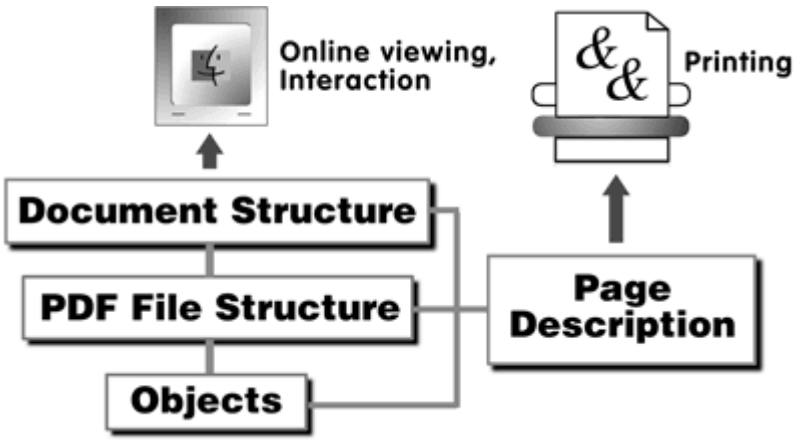
2.1.2 Co je to PDF

Portable Dokument Format je rozšiřitelný popisující stránkový protokol, který implementuje implicitní formát souboru Adobe Acrobat vhodný pro komerční využití. Cílem formátu je vytvořit hardwarově nezávislý vyměnitelný dokument s vysokou rozlišovací schopností.

Může obsahovat text, grafické nebo multimediální prvky, různé datové typy, odkazy na jiné soubory nebo hyperlink. Formát podporuje textové vyhledávání, záložky, odkazy, poznámky, interaktivní stránkové elementy (checkboxes, textová editovatelná pole atd.), kódování, kompresi, JavaScript a mnohem více.

PDF implementuje dokumenty jako hierarchii označených objektů organizovaných do stromové struktury nebo zřetězeného seznamu. Objekt může nabývat jednoho ze sedmi základních typů, může být čistě strukturální nebo zapouzdřovat různé druhy obsahů, atributů nebo ukazatelů na externí zdroje. Je zde několik pevných pravidel, jak musí být dokument strukturován, neboť logická a fyzická struktura dokumentu se může lišit. Na PDF se můžeme dívat tak, že se skládá ze čtyř typů struktur. Jak je ukázáno na obrázku 1.

Na nejnižší úrovni se PDF skládá z objektů jako jsou : jména, čísla, pole atd. O úroveň výše se nachází struktura PDF dokumentu, která určuje, jak jsou objekty uloženy a jak jsou zpřístupněny. Nad touto úrovní se nachází struktura dokumentu. Určuje, jak jsou objekty organizovány, řazeny na stránkách a jak jsou přiřazeny atributy, aby byl zaručen daný vzhled dokumentu při jeho prohlížení.



Obrázek 1 - Čtyři typy struktury dokumentu PDF (převzato z PDF Reference)

Na úrovni stránek se PDF dokument skládá z objektů a stránkových značících operátorů potřebných k fyzickému zobrazení stránky.

Na úrovni stránkového popisu je brán PDF jako jazyk PostScriptu, zatímco při popisu dokumentové struktury se PDF dostává na mnohem vyšší úroveň.

PDF nebylo stvořeno pouze k zobrazování a tisknutí textu a jeho grafických částí, ale má schopnosti prohledávat dokument, vkládat do něj poznámky, chránit ho heslem. Může obsahovat multimediální prvky, provádět JavaScriptové operace a mnoho dalšího.

2.1.3 Rozdíly mezi PDF a Postskript

Nezkušený uživatel by mohl říci, že mnoho PDF vypadá stejně jako PostScript. Ale opak je pravdou. Rozdíly mezi oběma formáty jsou výrazné. Zatímco PostScript je skutečný jazyk, PDF není. PDF postrádá procedury, proměnné, struktury toku řízení, které by byly potřeba k syntaktické síle jazyka. V tomto pohledu je PDF skutečně jen stránkový popisovací protokol.

Jazykové prvky byly odebrány hlavně kvůli jednoduchému parsování dokumentu a redukci pravděpodobnosti výskytu chyb. Bylo by obtížné zaručit náhodný přístup k datům jiným způsobem. Prohlížecí program, který může extrahovat text a zobrazit vybranou stránku z velkého množství PostScriptových stránek, by neměl jinou možnost než procházet soubor od začátku do konce, aby vybranou stránku našel a rovněž všechny její prvky. Navíc čas potřebný k nalezení a zobrazení stránky by nezávisel pouze na složitosti stránek, ale i na délce dokumentu, což je velmi nevhodné pro dnešní dobu, kdy je preferován mít rychlý a náhodný přístup k vybraným stránkám.

Každé PDF má „cross-reference table“, sloužící k rychlému nalezení a zpřístupnění stránky a jiných důležitých zdrojů v souboru. Xref tabulka je vždy uložena na konci dokumentu, takže programy, které vytváří PDF soubory, ji jednoduše ukládají na konec dokumentu a naopak programy které pracují s PDF, snáze naleznou vybrané odkazy. Z toho vyplývá, že čas potřebný k nalezení stránky je nezávislý na velikosti dokumentu.

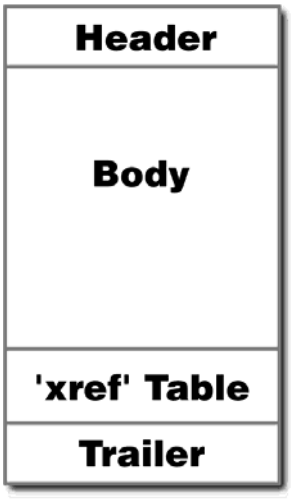
Postupné rozšiřování a editování dokumentu je další charakteristickou vlastností, jež by byla obtížně realizovatelná v PostScriptu. Při vytváření, editování nebo přidání nové části, nemusíme čekat na přepsání celého dokumentu, ale PDF nám dovoluje přidat nová nebo editovaná data do dokumentu, aniž by se původní data nějak změnila, pokud nebyla editována. Změny mohou být provedeny v čase úměrném velikosti změn raději, než v čase úměrném velikosti dokumentu (PostScript).

Další rozdíly mezi PDF a Postscript:

- PDF soubory obsahují informace o použitém fontu v dokumentu, aby zabezpečily stejné zobrazení v různých prohlížečích a platformách.
- PDF soubory obsahují hyperlink a jiné objekty
- PDF jsou rozšiřitelné, ale i přes tuto vlastnost definují způsob, jakým mají prohlížeče pracovat s neznámými objekty, pokud je formát souboru vyšší verze než je prohlížeč.

2.1.4 Struktura PDF souboru

PDF se skládá ze čtyř hlavních částí: hlavička, tělo, „cross-reference“ tabulka (xref) a trailer.



Obrázek 2 - Struktura PDF (převzato z PDF Reference)

2.1.4.1 Header

První řádek PDF souboru, specifikuje verzi dokumentu, ta je napsána v notaci PostScriptu. Například:

Příklad 1- Header

```
%PDF – 1.3
```

Toto by znamenalo, že PDF dokument je verze 1.3. jako v PostScriptu **%** předchází všem komentářům. Ty se mohou vyskytovat během celého dokumentu.

Druhý řádek PDF je také zakomentován a obvykle obsahuje jeden nebo více vyšších bitů ASCII znaků (v rozsahu 0x80 až 0xFF). Ty sdělují klientům nebo programům, že soubor obsahuje binární data a neměl by být brán jako 7-bitový ASCII text.

2.1.4.2 Body

Tělo PDF se skládá z objektů, které vytváří obsah dokumentu. Objekty mohou zahrnovat text, obrázky, fonty, poznámky a mnohé další.

Tělo může také obsahovat velké množství neviditelných objektů, které pomáhají implementovat interaktivitu dokumentu, bezpečnostní rysy a logickou strukturu.

Více se s konkrétními objekty seznámíme později.

2.1.4.3 Cross-Reference Table

Obsahuje ofsety ke všem objektům v souboru. Proto není nezbytné prohledávat velkou část souboru aby byl nalezen hledaný prvek. Pokud dokument nebyl aktualizován, cross-reference table bude souvislá, skládající se z jedné sekce. Nové sekce se přidávají při každé modifikaci souboru.

V každé jednoduché sekci cross-ref table, jsou podsekce odpovídající postupně očíslovaným objektům. Zápis každého objektu je přesně 20 bytů dlouhý, zahrnuje i ukončovací znak(y). Prvních 10 bytů specifikuje ofset objektu – deseti místné číslo, následuje mezera a pěti místné číslo udávající v jakém pořadí byly objekty vygenerovány. Další mezerou je odděleno písmeno „ n “ nebo „ f “, které značí, jestli je objekt volný, nebo je obsazený. Dále následuje jen ukončovací značka, která může nabývat jednu z následujících podob: 0x200A, 0x200D nebo 0x0D0A.

Ukázka cross-ref table:

Příklad 2 - Cross Reference table

```
Xref
0 1
0000000023 65535 f
3 1
0000025324 00000 n
21 4
0000025518 00002 n
0000025632 00000 n
0000000024 00001 f
0000000000 00001 f
```

```
36 1
0000026900 00000 n
```

První položka v cross-ref table je vždy volná. Obsahuje číslo vytvoření 65535. Je první položkou ve zřetězeném seznamu volných objektů. Poslední objekt v seznamu požívá 0 jako číslo dalšího volného objektu.

Druhá podsekce obsahuje jeden objekt. Jeho ofset od začátku PDF souboru až po samotný objekt je 25324.

Třetí podsekce obsahuje čtyři objekty. První z nich má číslo 21, ostatní jsou postupně očíslovány 22, 23 a 24.

Čtvrtá podsekce má jen jeden objekt - číslo 36.

Všechny objekty jsou označeny **f** - volný nebo **n** – obsazený. Lepší značení by možná bylo platný / nepatný nebo aktuální / zastaralý.

Volný znamená, že ačkoliv objekt stále může být fyzicky v souboru, je neplatný a neměl by být používán.

Obsazený znamená, že objekt je platný a použitelný (to ovšem neznamená, že objekt je vybrán a používán).

Objekty označené **n** mají ofset následovaný číslem, určujícím v jakém pořadí byly vygenerovány. Zatímco objekty označené **f** obsahují čísla následujících volných objektů a generační číslo, které bude použito, když je aktuální objekt obnoven.

Generační číslo objektu je zvětšeno v době uvolnění. Proto objekty 23 a 24, mají stejné generační číslo 1. Pokud tyto objekty budou znovu obnoveny, což znamená, že přejdou ze stavu **f** do stavu **n** bude jim přiřazena hodnota ofsetu a generační číslo bude stále 1. Následným znovu uvolněním přejdou zpět do stavu **f** , ale již s generačním číslem 2.

2.1.4.4 Trailer

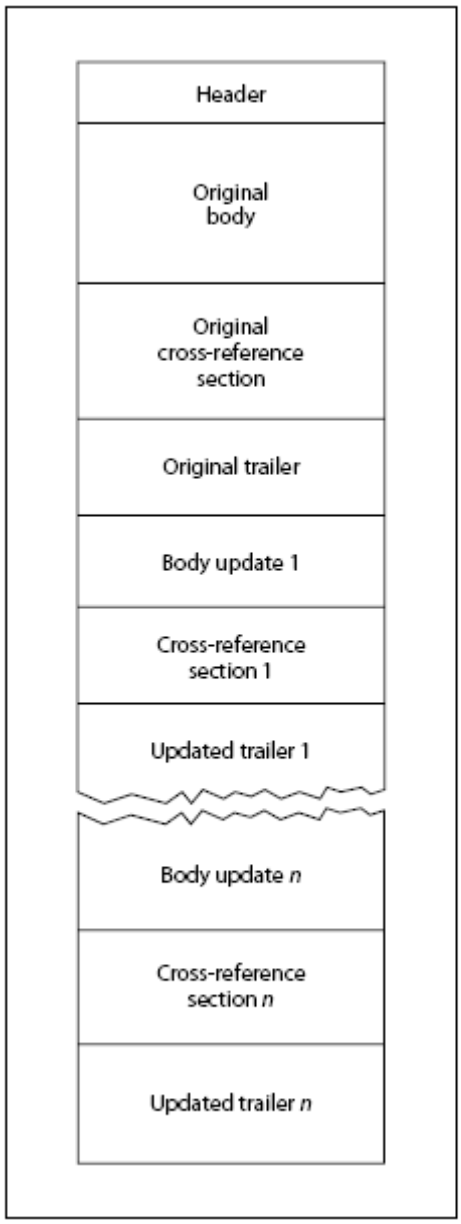
Umožňuje aplikacím rychle najít cross-reference table a některé další speciální objekty. Od aplikací se očekává čtení souboru od konce. Poslední řádek v souboru obsahuje ukončující znak a nad ním se nachází **startxref** a ofset od začátku souboru k xref-table. Tomu předchází adresář trailer, který začíná totožným názvem.

2.1.5 Jak se PDF aktualizuje

Trailer hraje důležitou roli v aktualizaci PDF souboru. Ten není nikdy přepsán, pouze jsou přidány nové informace. Toto obecně platí pro všechny části souboru, dokonce i traileru a znaku konce

souboru. Vícenásobné aktualizování PDF může vést k vícenásobným trailerům a vícenásobným znakům konce souboru (může ze vyskytnout více %%EOF).

Při každé editaci je objekt, který se změnil, uložen na konec souboru. Nová x-ref sekce je přidána a nový trailer obsahuje všechny informace předchozího, stejně jako **/Prev** klíč specifikuje ofset předchozí xref sekce. Cross-reference informace budou rozděleny na více než jednu xref sekci. K získání všech cross-references musíme procházet seznam **/Prev** klíčů ve všech trailerech od konce.



Obrázek 3 - Struktura aktualizování PDF souboru (převzato z PDF Reference)

2.1.6 Datové typy PDF

Má sedm základních typů objektů: Booleans, čísla, řetězce, pole, dictionaries, streams a nepřímé objekty. V této části práce se budu zabývat jen objekty, které byly použity při tvorbě programu a jsou tedy pro nás nepostradatelné.

2.1.6.1 Nepřímé objekty

Nepřímý objekt má číslo a obsahem může být jakýkoliv objekt PDF souboru, uzavřený mezi klíčovými slovy:

- **obj** – vyskytuje se za identifikací objektu, která obsahuje číslo objektu a jeho generační číslo.
- **endobj** – je zapsáno na novém řádku

Řádek ID objektu tedy obsahuje: jeho číslo, číslo generování a klíčové slovo **obj**. Příklad:

Příklad 3 - Nepřímý objekt

```
9 2 obj
39
Endobj
```

Tento nepřímý objekt zapouzdřuje číselný objekt – **integer 39.** (Obecně vzato by mohl zapouzdřovat jakýkoliv objekt – string, name, dictionary atd., nemůže však obsahovat další nepřímé objekty.)

Výhodou deklarování nepřímých objektů je jejich snadný přístup – každý nepřímý objekt má záznam v xref tabulce, což znamená možnost rychlého nalezení nebo využití na více stránkách dokumentu.

Příklad 4 - Nepřímý objekt + řetězec

```
2 0 obj
<<
/Length 9 2 R
>>
stream
BT
/F1 12 Tf
72 712 Td (A short text stream.) Tj
ET
endstream
endobj

9 2 obj
39
endobj
```

Nyní existují dva nepřímé objekty, objekt číslo 2 (text stream) a 9 (integer). Pole **/Length** má hodnoty **9 2 R**. Ukazuje na objekt číslo 9, který obsahuje skutečnou délku řetězce. Této nesporné

výhody systému využívají aplikace, které vytváří PDF soubory. Dovolí jim ukládat text do řetězců, aniž by předem znaly jejich délku. Ta se pak následně zapíše do nově vzniklého objektu. Tato vlastnost umožňuje vytvářet PDF dokumenty během jediného průchodu.

2.1.6.2 Streams

Stream je posloupnost osmibitových bytů uzavřená klíčovými slovy **stream** a **endstream**. Jakýkoliv druh obsahu, skládající se pouze z binárních dat je reprezentován streamem. Na stream se můžeme dívat jako na obrovský string, s tím rozdílem, že jej můžeme zpracovávat postupně.

Streams jsou zabaleny do nepřímých objektů, aby mohly být rychle přístupné. Tudíž budou uzavřeny v **obj** a **endobj** značkách. Za obj následuje atribut **dictionary,** udávající informaci o následujících datech. Minimálně musí obsahovat klíč **/Length,** ale může mít i jiné atributy jako například **/Filter** udávající v jaké kompresi budou data uložena.

Příklad 5 - Streams

```
2 0 obj
<<
/Length 39
>>
stream
BT
/F1 12 Tf
72 712 Td (A short text stream.) Tj
ET
endstream
endobj
```

První řádek začíná číslem objektu a následuje generační číslo. Dictionary obsahuje jen jeden atribut **/Lenght** délku řetězce. Pokud stream obsahuje zobrazitelný text, je uzavřen mezi značky **BT** a **ET** jako begin / end text. Řádek začínající **/F1** značí natažení a použití fontu číslo 1 a velikosti **12pt.** Další řádek zakončený operátorem **Td** určuje pozici textu na stránce. Ta je dána absolutním pozicováním (x, y), tedy jeden palec od levého okraje stránky a zhruba deset palců od spodního. Text je dán jako string následovaný operátorem **Tj**, který určuje, že daný text bude zobrazen na stránce dokumentu.

2.1.6.3 Katalogový strom

Je kořen dokumentových objektů, dosažitelný přes kořenovou položku v traileru PDF souboru. Katalog obsahuje odkazy na jiné objekty definující dokumentový obsah. Navíc lze najít informace jak má být dokument zobrazen. Například zda osnova nebo náhledy stránek mají být zobrazeny automaticky, popřípadě zda má být zobrazena jiná než první stránka při otevření dokumentu.

Příklad položek:

- **/Pages** – kořen stromové struktury dokumentu
- **/Outlines** – kořen stromu osnovy, pokud existuje
- /**PageMode** – určuje, jak má být dokument zobrazen při jeho otevření

Příklad 6 - Katalogový strom

```
1 0 obj
<<
/Type /Catalog
/Pages 2 0 R
/Outlines 3 0 R
/PageMode /UseOutlines
>>
endobj
```

Jediná nutná položka v katalogovém stromu je odkaz na kořen stromové struktury dokumentu. Pokud ovšem dokument používá osnovu, logické odkazy nebo další objekty reference na kořeny stromů těchto objektů objeví se v katalogu.

2.1.6.4 Stránková stromová struktura

Stránky dokumentu jsou přístupné díky stromové struktuře. Uzly stromu jsou adresáře obsahující odkazy na zobrazitelné stránky v dokumentu nebo na jiné objekty. Acrobat Distiller vytváří vyvážený strom, ve kterém uchovává všechny informace, aby tak minimalizoval vyhledávací čas. Není však vždycky nutné vytvářet vyvážený strom a někdy ani stromovou strukturu. Vše může být nahrazeno jediným uzlem, který obsahuje odkazy na všechny objekty stránek v souboru.

Stránkové objekty jsou většinou uloženy v listech stromu. Uzly jsou adresáře, obsahující čtyři základní položky:

- **/Type** – obsahuje vždy položku /Pages
- **/Count** – udává počet stránek, které se nacházejí pod tímto uzlem
- **/Kids –** položka typu pole, kde se nacházejí čísla objektů všech dostupných stránek
- **/Parent –** odkaz na nadřazený uzel

Příklad 7 - Formát stránkové stromové struktury

```
2 0 obj
<<
/Type /Pages
/Kids [6 0 R 10 0 R 18 0 R]
/Count 3
>>
endobj
```

V příkladě č.7 je ukázka uzlu, který ukazuje na tři listy: 6, 10 a 18. Jak všechny listy, tak i uzel samotný jsou realizovány jako nepřímý objekt, aby mohly být odkazovány jinými objekty.

Stránkové objekty jsou typu **/Page,** který popisuje různé objekty a atributy, vytvářející zobrazitelnou stránku. Typické přídavné položky jsou **/Parent, /MediaBox, /Resources, /Content** a mnoho dalších. Stránkový obsah je obvykle dán streamem nebo polem stream, odkazovaný tagem **/Content.**

Příklad 8 - Stránková stromová struktura

```
8 0 obj
<<
/Type /Page
/Parent 4 0 R
/MediaBox [0 0 612 792]
/Resources <<
        /Font << /F3 7 0 R /F5 9 0 R /F7 11 0 R >>
        /ProcSet [/PDF] >>
/Thumb 12 0 R
/Contents 14 0 R
/Annots [23 0 R 24 0 R]
>>
endobj
```

/Content obsahuje data potřebná pro zobrazení stránky v objektu 14. Pokud by nebyl definován žádný objekt s daty, stránka by byla prázdná. **/MediaBox** definuje maximální zobrazitelnou plochu, na níž může být vykreslena stránka. V tomto příkladu je to 612 na 792 typografických bodů. Pokud jsou k dispozici náhledy stránek, Lze je použít pomocí tagu **/Thumb,** který určuje v jakém objektu se budou nacházet. Poznámky se nacházejí v objektu číslo 23 a 24 a jsou zpřístupněny pomocí **/Annots.** **/Font** určuje, jaké druhy fontů se budou v dokumentu používat a **/ProSet** je množina PostScript procedur definujících, že PDF stránka bude popsána PostScript operátory.

2.1.7 Grafické operátory

Grafické operátory popisující vzhled stránek rozdělujeme do šesti skupin:

- **Grafické stavové operátory** – pracují s datovou strukturou, vytváří globální kostru, uvnitř které jsou ostatní grafické operátory prováděny. Je možno zde najít aktuální transformační matici **CTM** (current transformation matrix), která převádí souřadný systém používaný uvnitř PDF na souřadný systém výstupních zařízení, taktéž použitou **barvu**, způsob **ořezu** a mnoho dalších parametrů, které jsou implicitní operandy zobrazovacích operátorů.

- **Blokové konstrukční operátory** – definují tvary, trajektorie čar a oblasti různých druhů. Definují začátky a konce bloků pro vykreslování grafických útvarů, přidávání částí křivek a čar.
- **Blokové kreslící operátory** – vyplňují blok barvou, vytvoří jeho obrys nebo je lze použít k ořezaní hranic.
- **Ostatní kreslící operátory** – vykreslují některé grafické objekty.
- **Textové operátory –** vykreslují textové řetězce, neboť PDF pracuje s písmeny jako s grafickými prvky, proto mnoho textových operátorů může být zařazeno do grafických stavových nebo kreslících operátorů.
- **Obsahové operátory -** nemají žádný efekt na vzhled stránky, ale používají se v aplikacích, které používají formát PDF k výměně dokumentů.

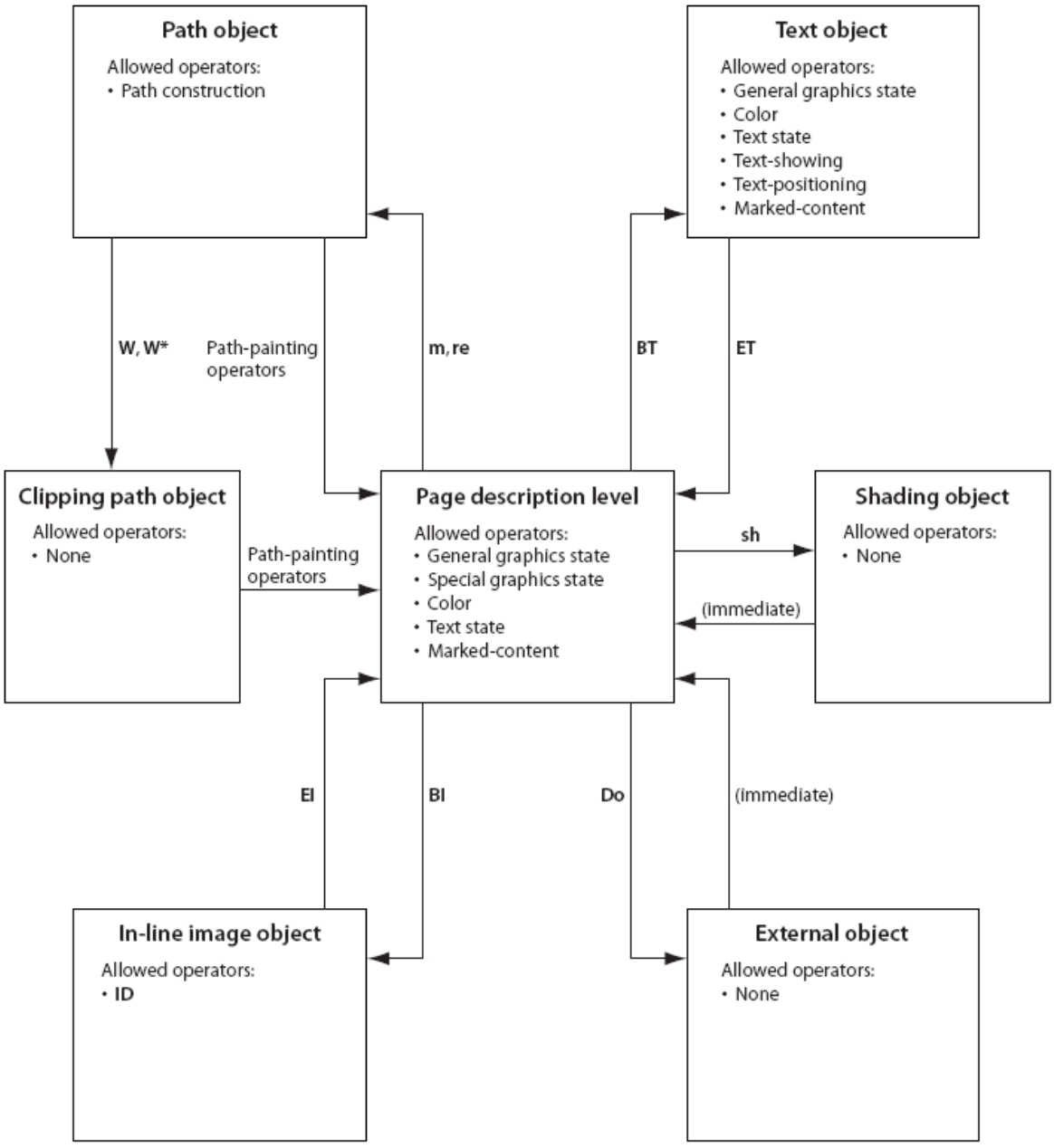
2.1.7.1 Grafické objekty

Data v datovém toku jsou interpretována jako posloupnost operátorů a operandů, vyjádřená jako základní datové objekty podle standardů syntaxe PDF. Mohou tak vytvářet vzhled stránky nebo v jiném případě být považovány za grafické elementy. I když datový tok není přeložitelný program, ale pouze sekvence grafických objektů, existují zde jistá pravidla pro operátory a operandy.

Popis pšti typů grafických objektů:

- **blokové objekty** – jsou libovolné objekty tvořené přímými čarami, čtyřúhelníky a Bezierovými křivkami. Mohou se protínat, mít mezi sebou mezery nebo díry. Blokové objekty končí jedním nebo více kreslícími operátory, které určují jestli je blok vybarven, ohraničen nebo použit k ořezu. Operátory se mohou kombinovat.
- **Textové objekty** – skládají se z jednoho nebo více znaků, tvořící řetězce, které se pak vykreslují na stránku.
- **Externí objekty** – jedná se o objekt definovaný mimo datový tok odkazovaným jménem.
- **Inline objekty** – ukrývají data pro malé obrázky přímo v datovém toku, používají speciální syntaxi.
- **Stínové objekty** – popisuje geometrický tvar, jehož barva je závislá na pozici v daném tvaru.

Na obrázku ukazují šipky operátory, značící začátek nebo konec každého grafického objektu. V následující tabulce jsou shrnuty a rozřazeny nejdůležitější operátory. Některé z nich bude také používat v mém programu.



Příklad 9 - Grafické objekty (převzato z PDF Reference)

Tabulka 1 - Kategorie operátorů (převzato z PDF Reference)

CATEGORY	OPERATORS	TABLE	PAGE
General graphics state	w, J, j, M, d, ri, i, gs	4.7	142
Special graphics state	q, Q, cm	4.7	142
Path construction	m, l, c, v, y, h, re	4.9	149
Path painting	S, s, f, F, f*, B, B*, b, b*, n	4.10	152
Clipping paths	W, W*	4.11	156
Text objects	BT, ET	5.4	286
Text state	Tc, Tw, Tz, TL, Tf, Tr, Ts	5.2	280
Text positioning	Td, TD, Tm, T*	5.5	287
Text showing	Tj, TJ, ', "	5.6	289
Type 3 fonts	d0, d1	5.10	303
Color	cs, CS, sc, scn, SC, SCN, g, G, rg, RG, k, K	4.21	198
Shading patterns	sh	4.24	214
In-line images	BI, ID, EI	4.38	260
XObjects	Do	4.34	243
Marked content	BMC, BDC, EMC, MP, DP	8.5	480
Compatibility	BX, EX	3.19	84

Grafické objekty implicitně obsahují všechny grafické stavové parametry, které mají vliv na jejich funkce. Například blokové objekty závisejí na aktuální barvě v době kdy jsou zobrazovány. Blokové objekty si mohou definovat vlastní parametry, a nebo mohou také využívat již dříve definovaných. Pak záleží pouze na nás, jestli je chceme použít, nebo si nadefinujeme nové, lépe vyhovující pro máš objekt.

PDF nepoužívá žádný předem daný postup jak mají být grafické operátory uspořádány. Aplikace, které čtou nebo vytváří PDF dokumenty tak nemají předem dán algoritmus, kterým budou řadit jednotlivé grafické objekty, ale každá aplikace si vytváří své vlastní uspořádání za předpokladu, že parametry grafických objektů zůstanou bezezněny.

2.1.7.2 Souřadný systém

Souřadný systém definuje plochu, na kterou se vykreslují objekty, určuje jejich pozici a orientaci, velikost písma, rozměry grafických objektů a obrázků na stránce.

Bloky a pozice jsou definovány pomocí souřadnic x a y kartézského souřadného systému. Souřadnice jsou čísla typu real a určují horizontální a vertikální souřadnice objektů. PDF používá několik souřadných systémů. Jsou mezi nimi definovány vztahy a metody převodů mezi jednotlivými systémy. Souřadnicový prostor je s ohledem na aktuální stránku charakterizován následujícími vlastnostmi:

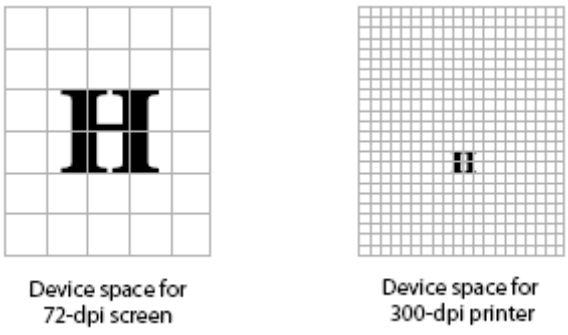
- Původní umístění
- Orientace os x a y
- Typ jednotek jednotlivých os

Převod mezi jednotlivými souřadnicovými systémy zajišťuje transformační matice, která definuje jakékoliv lineární mapování do dvourozměrného systému. Zahrnuje posuny, rotace, zvětšení a zkosení.

Zobrazovací prostor různých zařízení

Každá PDF stránka je nakonec zobrazena na display nebo je vytisknuta. Různá zobrazovací zařízení disponují svými vlastními souřadnými systémy, které adresují pixely na jejich zobrazitelnou plochu. Jednotlivé souřadné systémy výstupních zařízení se nazývají zobrazovací prostory. Ty mohou být na zobrazované stránce v různých operačních systémech nebo aplikacích různě umístěny. Protože papír nebo jiné výstupní médium prochází přes různé výstupní zařízení v různých směrech, osy jejich zobrazitelného prostoru mohou být orientovány obráceně. Například vertikální souřadnice se mohou zvětšovat shora dolů nebo zhola nahoru. Navíc různá výstupní zařízení mají odlišné rozlišení. Některá z nich mají dokonce různé rozlišení v horizontálním a vertikálním směru.

Pokud by souřadný systém byl specifikován zobrazovacím prostorem zařízení, pak by byl výstupní soubor na něm závislý a mohl by se lišit i jeho vzhled. Příkladem může být různé rozlišení, kdy můžeme mít 72 pixelů na palec nebo také 300, což je podstatný rozdíl.



Obrázek 4 - Zobrazovací prostor (převzato z PDF Reference)

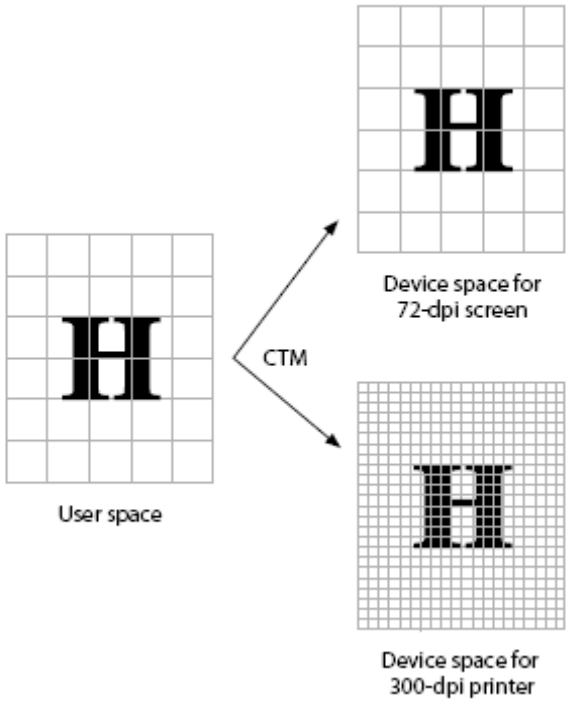
Uživatelský prostor

Abychom se vyhnuli zobrazovacímu prostoru závisejícímu na zařízení, definujeme souřadný systém, který zachovává stejný vzhled stránky, nezávisle na výstupním zařízení.

Uživatelský prostor souřadného systému je inicializován jako výchozí pro každou stránku dokumentu. Počátek je umístěn v levém dolním rohu výstupní stránky. Osa x je umístěna horizontálně a zvětšuje se směrem k pravému okraji, osa y je ve vertikálním směru a nabývá kladných hodnot směrem k hornímu okraji daného dokumentu. Jednotky obou os jsou udávány v typografických bodech. Její rozměr je asi 1/72 palce. Jedná se o výchozí zobrazovací prostor, kde všechny souřadnice bodů stránky nabývají kladných hodnot.

Jelikož souřadný systém může být definován jako integer nebo real, nevytváří tak žádnou předem dohodnutou mřížku, do které by se zanášely body. Rozlišení souřadného systému neodpovídá žádnému rozlišení zobrazovacích prostorů výstupních zařízení, jež se udávají v pixelech.

Existuje však mechanizmus, který transformuje uživatelský prostor do zobrazovacího prostoru výstupních zařízení, nebo naopak. Jedná se **transformační matici CTM** (current transformation matrix). Této budu věnovat více pozornosti v následující kapitole..



Obrázek 5 - Uživatelský prostor (převzato z PDF Reference)

Výchozí uživatelský prostor je stálý a spolehlivý základ pro tvorbu PDF stránek, bez ohledu na použité výstupní zařízení. Pokud je to nezbytné, datový tok může měnit uživatelský prostor tak aby byl vhodnější k jeho potřebám, použitím souřadného transformačního operátoru **cm**. Tudíž, to co se může jevit jako absolutní souřadnice v datovém toku není absolutní pro aktuální stránku, protože ta je

také vyjádřena v souřadném systému, jenž může být zvětšován nebo zmenšován. Transformační souřadný systém pouze nezvětšuje nezávislost na výstupním zařízení, ale je to i užitečný nástroj. Například datový tok přizpůsobený k zobrazení na jednu stránku může být začleněný beze změny jako element jiné stránky pouze zmenšením souřadného systému, ve kterém je tato vykreslena.

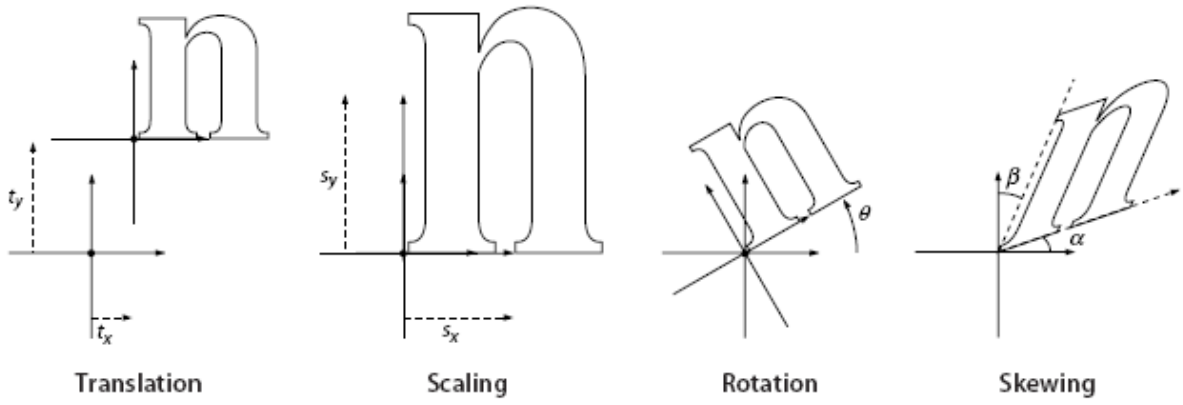
2.1.7.3 Transformací matice

Transformační matice specifikuje vztah mezi dvěma souřadnými systémy. Změnou transformační matice můžeme dosáhnout změny objektu – velikost, rotace, posunutí nebo jiné transformace.

Transformační matice v PDF je dána 6 hodnotami obvykle ve formě pole [a b c d e f], majících šest položek. Může provádět jakoukoli lineární transformaci z jednoho souřadného systému do druhého.

Nejvíce používané transformace:

- Posunutí (translations) – dané [1 0 0 1 t_x t_y], kde t_x a t_y udávají velikost posunutí v horizontálním a vertikálním směru od původních souřadnic.
- Zvětšení je získáno z [s_x 0 0 s_y 0 0]. Pokud je s_x a s_y rovno 1 zůstávají původní hodnoty nezměněny. Jestliže ale s_x a s_y nabývají jiných hodnot než 1 měníme původní velikost.
- Rotace (rotations) jsou tvořeny [cosΦ sinΦ -sinΦ cosΦ 0 0], záleží pouze na velikosti úhlu Φ, úhel se počítá protisměru hodinových ručiček.
- Zkosení (skew) je specifikováno [1 tanα tanβ 1 0 0] je dáno zkosením os x a y.

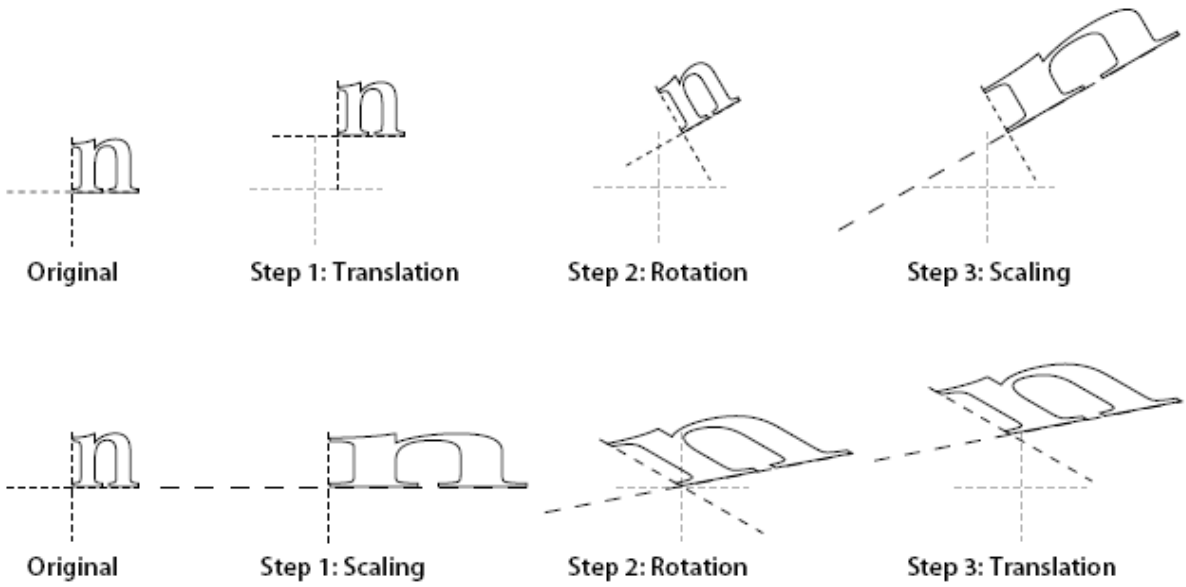


Obrázek 6 - Efekty transformační matice (převzato z PDF Reference)

Pokud se uplatní několik transformací za sebou, je třeba je provádět podle určitého postupu. Nelze provést nejprve zvětšení a poté posunutí osy x nebo nejdříve posunutí a následně zvětšení. Pokud je nutné provádět více transformací najednou, je třeba zachovávat následující pořadí.

1. Posunutí
2. Rotace
3. Zvětšení nebo zkosení

Na obrázku vidíme v jakém pořadí jsou transformace prováděny. Jde o dva různé postupy aplikování transformací. Můžeme si všimnout různých výsledků. Z toho plyne, že vskutku záleží na pořadí jejich aplikování.



Obrázek 7 - Pořadí aplikování transformací (převzato z PDF Reference)

Podrobnější informace lze získat z PDF Reference, second edition, Verze 1.3, 2000

2.2 PDFBox

PDFBOX je open source Java knihovna pro práci s PDF dokumenty. Tato knihovna dovoluje vytvářet PDF dokumenty, pracovat s již existujícími a umožňuje extrahovat text z dokumentů. PDFBox obsahuje také pár utilit, jež jdou spustit z příkazové řádky.

Obsahuje širokou škálu tříd pro práci s PDF soubory. Umožňuje nám členit dokument na tok objektů, které následují za sebou. A tak máme možnost získávat více informací o daném souboru. Po rozčlenění je možné použití tříd pro zpracování textového obsahu, získání obrázků nebo grafických prvků.

PDFBox je také silným nástrojem pro vytváření a upravování PDF souborů.

2.3 FontBox

Je knihovna pro parsování fontů a získání informací z nich. Připojuje také navíc k Javě nové fonty, které využijeme pro naše převody textu mezi PDF soubory a XHTM.

2.4 Java SE

Java je objektově orientovaný programovací jazyk, vyvinutý firmou Sun Microsystem v roce 1995. Je jedním z nejpoužívanějších programovacích jazyků na světě. Díky své přenositelnosti je používána pro programy pracující v různých systémech. Používá se v čipových kartách, mobilních telefonech a různých zabudovaných zařízeních (platforma Java ME). V aplikacích pro desktopové počítače můžeme najít platformu Javy SE a konečně v rozsáhlých distribuovaných systémech je platforma Javy EE. V roce 2007 Sun uvolnil zdrojové kódy Javy, a tak se od této chvíle Java vyvíjí jako open source.

2.4.1 Základní vlastnosti jazyka Javy

- **Jednoduchý** – jeho syntaxe je zjednodušenou verzí jazyka C a C++.
- **Objektově orientovaný** – s výjimkou osmi primitivních typů jsou všechny ostatní datové typy objektové.
- **Distribuovaný** – je navržený pro podporu aplikací v síti.
- **Interpretovaný** – místo skutečného kódu se vytváří pouze tzv. mezikód (bajtkód). Tento formát je nezávislý na operačním systému počítače nebo zařízení. Program pak může pracovat všude tam, kde má k dispozici interpreta Javy = virtuální stroj Javy (**J**ava **V**irtual **M**achine) JVM.
- **Robustní** – je určen pro psaní velice spolehlivého softwaru, proto neumožňuje některé programátorské konstrukce, které bývají častou příčinou chyb. Používá **silnou typovou kontrolu** - každá proměnná musí mít definovaný svůj datový typ. Správa paměti je realizována pomocí **Garbage collectoru**, který automaticky vyhledává nepoužívané části paměti a uvolňuje je pro další použití.
- **Bezpečný** – vlastnost, která chrání počítač v síťovém prostředí, v němž je program zpracován, před nebezpečnými operacemi nebo napadením vlastního operačního systému nepřátelským kódem.
- **Nezávislý na architektuře** – pokud je k dispozici správný virtuální stroj, pak program poběží v libovolném operačním systému.

- **Přenositelný** – jazyk je nezávislý i co do vlastností základních datových typů.
- **Víceúlohový** – podporuje zpracování vícevláknových aplikací.
- **Dynamický** – knihovna může být za chodu dynamicky rozšiřována o nové třídy a funkce, a to jak z externích zdrojů, tak vlastním programem.

2.4.2 Nevýhody Javy

Proti programovacím jazykům, které provádějí tzv. statickou kompilaci (např. C++), je start programů psaných v Javě pomalejší, protože prostředí musí program nejprve přeložit, a potom teprve spustit. Je však možno využít mechanismů JIT a HotSpot, kdy se často prováděné nebo neefektivní části kódu přeloží do strojového kódu a program se tak zrychlí. Na zrychlení se také podílí nové přístupy ke správě paměti.

Další nevýhodou, projevující se hlavně u jednodušších programů, je větší paměťová náročnost při běhu způsobená nutností mít v paměti celé běhové prostředí.

V návrhu Javy je vidět snaha znemožnit programátorovi psát problematické konstrukce známé především z jazyka C. Součástí jazyka proto nejsou například bezznaménková čísla, příkaz goto nebo preprocesor, ačkoli se v odůvodněných příkladech jedná o užitečné nástroje a ani rozšířené možnosti Javy je plně nenahrazují.

2.5 XHTML

Zkratka pochází z anglického e**x**tensible **h**yper**t**ext **m**arkup **l**anguage. Jedná se o rozšiřitelný značkovací jazyk pro tvorbu hypertextových dokumentů. Byl vyvinutý konsorciem W3C, aby se stal nástupcem jazyka HTML, jehož vývoj byl ukončen verzí 4.01. V roce 2007 vznikla nová skupina, která měla vytvořit novou verzi HTML s označením HTML 5.0. XHTML je vyvíjeno paralelně.

Zatímco HTML je jazykem třídy SGML (Standard Generalized Markup Language) , jedná se o velmi přizpůsobivý jazyk, tak XHTML je aplikací XML, což je mnohem striktnější podmnožina třídy SGML. Musí být mnohem lépe formulovány. Skutečné XHTML dokumenty povolují automatické zpracování pomocí standardních XML nástrojů, narozdíl od HTML, které vyžaduje relativně komplexní, méně striktní a obecně používaný parser. Na XHTML lze pohlížet jako na průnik HTML a XML v mnoha ohledech, obzvláště, když je vytvořeno z HTML při použití rysů XML.

2.5.1 Vztah k HTML

HTML je předchozí technologie k XHTML. Změny z HTML k první generaci XHTML 1.0 jsou minimální. Jedná se hlavně o změny, které zavádějí rysy XML. Nejdůležitější změna je požadavek na mnohem důkladnější formulování dokumentů. To vyžaduje, aby všechny elementy byly uzavřeny jako je tomu u XML. V XML jsou všechny elementy a atributy case-sensitive, proto také v XHTML jsou všechna jména tagů psána malými písmeny.

Toto je velkou změnou oproti dříve používané tradici, kdy se všechny tagy psaly velkými písmeny. Navíc všechny atributy v XHTML musí být uzavřeny v uvozovkách jednoduchých nebo složitých. Na rozdíl od SGML, kde tohle bylo pouze volitelné a nebylo zapotřebí používat uvozovek. Jedinou výjimkou byly řetězce, které se vkládaly do uvozovek. Všechny elementy musí být uzavřené a to i prázdné jako jsou např. **img** nebo **br**. To můžeme udělat pomocí zpětného lomítka ve startovním tagu: <img /> nebo
. Dále je zakázaná minimalizace atributů jako například "selected," neobsahuje žádnou explicitní hodnotu, místo toho musíme používat <option selected="selected">.

Protože XHTML a HTML jsou velice podobné technologie, jsou někdy popisovány a dokumentovány paralelně. V takovém případě někteří autoři používají speciální notaci (X)HTML. To znamená, že dokumentace a principy mohou být používány obecně pro oba standardy.

2.6 CSS

CSS je zkratka z anglických slov Cascading Style Sheets, česky tabulky kaskádových stylů. Jedná se o jazyk pro popis zobrazení stránek v jazycích HTML, XHTML nebo XML.

Jazyk byl navržen standardizační organizací W3C. Byly vydány zatím dvě verze CSS1 a CSS2 a nyní se pracuje na verzi CSS3.

Hlavním smyslem je umožnit návrhářům oddělit vzhled dokumentu od jeho struktury a obsahu. Původně to měl umožnit už jazyk HTML, ale v důsledku nedostatečných standardů a konkurenčního boje výrobců prohlížečů se vyvinul jinak. Starší verze HTML obsahují celou řadu elementů, které nepopisují jen obsah a strukturu dokumentu, ale i způsob jeho zobrazení.

Výhodou CSS oproti starému formátování v HTML je, že kód a obsah webu je uložen v souboru .html a veškerý design a formátování se načítá z jednoho souboru .css, který je většinou společný pro celý web. To znamená, že pokud chceme provést změnu designu webu, stačí změnit jeden soubor .css a změna se provede v celém webu. CSS soubor je uložen v mezipaměti prohlížeče a pokud není změněn, tak se načítá pouze jednou a tím se celkové načítání výrazně urychlí.

Lze vytvořit více stylů pro různá výstupní zařízení a tím také určovat, jak v nich má dokument vypadat. Specifikace CSS nezapomínají ani na zrakově postižené. Je možno napsat styly pro hlasový syntetizátor nebo hmatovou čtečku Braillova písma.

2.6.1 Použité CSS vlastnosti

K formátování textu je užíváno CSS, které je umístěno v externím souboru, jež je vygenerován při vytváření webové stránky. Každý prvek zde má své vlastnosti, udávající pozici prvku, jeho barvu, velikost písma atd..

Celý dokument je stylizován a tudíž je možné změnit jeho vzhled velmi jednoduše. Stačí si jen najít patřičný prvek, který je třeba editovat, a jeho ID. V souboru kaskádových stylů vyhledat patřičné ID prvku a změnit tak jeho vlastnosti. Námi provedené změny se projeví již při dalším načtení stránky.

Každý prvek má na stránce absolutní umístění, čímž je možno dosáhnout podobného vzhledu se souborem PDF. Označení „podobného“ zde má své opodstatnění. Nikdy nemůžeme dosáhnout úplně totožného vzhledu, neboť PDF nepoužívá jednotky jako jsou pixely, ale všechno se udává v typografických bodech. Z mých vlastních zkušeností s použitím stejných jednotek jsem vyvodil závěr, že dosažené výsledky neodpovídají očekávání, a proto upřednostňuji použití pixelů s rozlišením 96 pixelů na palec. Tímto lze dosáhnout takřka shodného vzhledu dokumentů. 96 pixelů na palec není pevně stanovená hodnota. Lze ji v programu změnit a tím dosáhnout jiného rozlišení.

```
public static int pixel = 96;
```

Jak jsem již zmínil dříve, každý prvek je absolutně umístěn na stránce, proto musíme rozlišovat, zda se jedná o text, obrázek nebo grafické prvky. Text umisťujeme od levého horního rohu, kde osa x je horizontální a lineárně roste k pravému okraji stránky. Osa y je umístěna vertikálně a nabývá rostoucích kladných hodnot směrem ke spodnímu okraji stránky. U obrázků a grafických prvků začínají osy v levém dolním rohu. Osa x nabývá stejných hodnot jako dříve jen osa y roste do kladných hodnot směrem k hornímu okraji.

Využíval jsem mnoho vlastností kaskádových stylů, z nich uvedu jen ty nejdůležitější. Za všechny zmíním **z-index**. Tato vlastnost povoluje překrývat jednotlivé prvky na stránce, a tedy kombinovat grafické prvky, text nebo obrázky, mezi sebou a libovolně nastavovat jejich pořadí překrývání. Např. grafické prvky jsou umístěné až za textem, tedy mají menší **z-index**.

O dalších vlastnostech blíže v příštích kapitolách.

3 Cíle

Cílem této práce bylo vytvořit program na transformaci dokumentů PDF na XHTML za pomoci open source knihovny PDFBOX a knihovny FONTBOX. Vzniklý XHTML soubor má být stylizován pomocí kaskádových stylů v externím souboru a program má být funkční pro jakýkoliv PDF dokument.

3.1 Extrakce textu

Extrakce textu byla primárním úkolem. Již v předešlých kapitolách jsem se zmiňoval o tom, že všechna data jsou v PDF souboru umístěna v objektech, které na sebe nemusí navazovat, ale pořadí je dáno reference table na konci dokumentu. Proto je třeba nejprve rozdělit data na objekty a následně s nimi dále pracovat.

Pomocí funkcí knihovny PDFBOX, podrobněji se o nich zmíním v kapitole Extrakce textu, dokážeme vytvořit tok objektů v takovém pořadí, aby na sebe navazovaly. Dále je třeba rozlišovat, jedná-li se o známý, či neznámý druh písma. Pokud PDFBOX dané písmo zná, dokáže s ním lépe pracovat. Má přímo v sobě zapouzdřenou třídu **showCharacter**, která extrahuje text přímo z PDF dokumentu, ale bez formátování. Výsledkem operace je pouze jeden dlouhý řetězec, který obsahuje celý textový obsah PDF dokumentu. Následným dalším zpracováním text upravíme.

Problém nastává, když dané písmo nezná. Pokud tato situace nastane je nutné použít třídu **processOperators**, která pracuje s tokem objektů. Ta vybere jen ty, které obsahují textové řetězce, protože zde se již nacházejí všechna data: obrázky, části grafických prvků, a jiné druhy objektů. V kapitole grafické objekty jsem se zmínil o operátorech, pomocí kterých můžeme rozlišovat objekty.

Při práci s textem budeme používat operátory **TJ** a **Tj.** Ty vyberou objekty obsahující textové řetězce, které bude nutné ještě dále upravovat.

Dalším problémem, který je třeba vyřešit, je dělení slov. Slovo není vždy umístěno v jednom objektu, ale může být, a většinou je, rozděleno na více částí. To není příliš ideální, a proto je třeba části slov nejprve spojit dohromady, aby se nám s nimi lépe pracovalo.

3.1.1 Pozice textu na stránce

Nejprve si je nutno vytvořit stránku, do které jsou vloženy textové elementy. Rozměry stránky udává přímo PDF dokument. Pokud hodnoty nebyly zjištěny, jsou nastaveny implicitně zadané.

Po této základní přípravě stránky do ní lze umísťovat textové elementy. Každé slovo bude uzavřeno do tagů a , kde otevírací tag ponese **id** nějakého stylu umístěného v CSS souboru.

Pozice každého prvku je přesně dána. Pomocí funkcí **.getX()** a **.getY()** v třídě **TextPosition** lze zjistit přesné souřadnice elementu na stránce. Za pomocí CSS a jejich absolutního pozicování je prvek na stránku umístěn.

3.1.2 Velikost a barva písma

Jedná se o velmi problematickou oblast, ve které se vyskytuje mnoho obtíží. PDFBOX zná jen asi 14 druhů písma a z toho vyplývá, že u většiny neznámých fontů bude problém s určováním přesné velikosti. Zná-li PDFBOX písmo, pak je možno jednoduše získat velikost pomocí operátorů **Tm** a **Tf.**

Problém nastane, nezná-li daný typ fontu, nejprve tedy budeme muset zjistit velikost písma orientačně. Pro tento účel jsem vytvořil pole velikosti písem, kde je uvedena velikost písma v PDF dokumentu a k ní odpovídající hodnota skutečné velikosti. Velikost písma lze zjistit pomocí funkce **.getHeight()** ve třídě **TextPosition** a pak už jen vyhledat odpovídající skutečnou velikost písma, kterou chceme použít.

Jako výchozí barva je nastavena černá, ta se ale v průběhu vytváření dokumentu bude měnit. Na zjišťování barvy je možno používat následující operátory v třídě **processOperators**:

- **g –** pokud je nastaveno na 1 bude aktuální používaná barva bílá, při hodnotě 0 přejdeme na původní černou barvu
- **scn –** nastavuje barvu v RGB hodnotách

3.2 Grafické prvky

PDF řadí mezi grafické prvky téměř všechny objekty. Dokonce i text je brán jako grafický prvek. Pro potřeby této práce budu za grafické prvky považovat pouze části tabulek, rámů, různých čar nebo úseček v dokumentu. Podtržený text je brán jako grafický prvek. Některé jednodušší obrázky jsou složeny z čar nebo křivek.

Všechny grafické prvky budeme získávat z toku objektů pomocí třídy **processOperators.** Zde je k dispozici operátor **re**, který informuje o tom, že daný objekt obsahuje grafický prvek. Každý grafický objekt je pole, které obsahuje 4 hodnoty. První dvě jsou jeho souřadnice a poslední dvě jeho rozměry.

Není výjimka, že rozměry nabývají záporných hodnot. Pokud k tomu dojde, je třeba od počátečních souřadnic odečíst jeho rozměry, tedy od osy x odečteme šířku a od osy y výšku, a tím získat nové souřadnice objektu. Pak jen rozměry vynásobit -1 a získáme tak kladné hodnoty.

Dalším operátorem, který je k dispozice je **f**. Tento operátor signalizuje, kdy je možné daný objekt vyplnit barvou. Pokud je tok objektů prohlížen a je nalezen operátor **f** je jasné, že objekt, jež mu předcházel, musí být vyplněn aktuální barvou.

3.3 Obrázky

Obrázky lze extrahovat pomocí operátoru **Do**. Tento operátor extrahuje současně s obrázky i některé prvky, které obrázky nejsou, a proto je třeba rozlišit, zda se jedná o obrázek, či nikoli. K tomu slouží třída PDXObjectImage z PDFBOXu.

Dále jsou k dispozici další pomocné funkce, které daný obrázek např. uloží do externího souboru, pomohou určit pozici obrázku na stránce a jeho rozměry atd. V kapitole Obrázky zmíněné funkce podrobně popíši. S rozměry obrázku mohou nastat menší problémy, neboť někdy PDF používá CTM (current transformation matrix), která dokáže upravit rozměry daného objektu.

Pokud se v toku objektů nachází operátor **cm**, tak je CTM uplatněna a lze použít její rozměry pro daný objekt. Jak je již známo z předchozí kapitoly transformační matice CTM obsahuje šest hodnot. Pro potřeby této práce využívám první a čtvrtou hodnotu.

3.4 Další možná rozšíření

Další vývoj by se mohl vydávat cestou lepšího propracování grafických prvků a jeho vlastností. Mezi největší problém zatím patří správné velikosti písma a pozicování. Ačkoliv program dosahuje poměrně dobrých výsledků, bylo by ještě třeba více času věnovat zjišťování druhů fontů a dalších vlastností spjatých s nimi.

Přestože úkolem této práce bylo více se věnovat extrahování textu než práci s grafikou, rozhodně by stálo za povšimnutí i větší propracování grafických objektů, popřípadě obrázků.

Další velkou výhodou by mohlo být grafické rozhraní, ve kterém by bylo možno měnit například rozměry obrázků, velikost a umístění písma, barvu písma, ale i výplně grafických objektů. Určité zjednodušení by přineslo i editovací okno, ve kterém by byl zobrazen text a tím by byla dána lepší možnost jej dodatečně editovat. Tato funkce by se dobře uplatnila obzvláště při korektuře českých textů, se kterými má program velké problémy.

Další vývoj programu by se rozhodně měl vydat cestou správné české diakritiky, což je asi největším dosavadním nedostatkem.

4 Návrh a Implementace

4.1 Extrakce textu

V kapitole Extrakce textu byl popsán postup, s jehož pomocí je možno získat text. V této kapitole mé práce se dostávám k hlubší analýze dané problematiky z důvodu lepší práce s PDF dokumentem jej nejprve převedu na objekty a ty následně seřadím.

K tomu slouží následující funkce.

```
file = new FileInputStream( args[0]); //Načteme vstupní soubor
PDFParser parser = new PDFParser( file );
parser.parse(); // rozdělíme dokument na objekty
document = parser.getPDDocument(); // obsahuje tok objektů
List allPages = document.getDocumentCatalog().getAllPages();
```

V této části zdrojového kódu si nejprve načtu vstupní soubor, který následně převedu na tok objektů, jež na sebe navazují. Do proměnné **allPages** si uložím všechny stránky dokumentu. Pomocí této proměnné s nimi pak mohu dále pracovat.

```
for( int i=0; i<allPages.size(); i++ ) {
PDPage page = (PDPage)allPages.get( i );
printer.processStream(page,page.findResources(),page.getContents().getStre
am());
}
```

V následujícím cyklu for procházím jednotlivými stránkami dokumentu a pomocí **`printer.processStream`** volám třídy **showCharacter(TextPosition text)** nebo **processOperators(PDFOperator operator, List arguments)**. Pomocí první třídy lehce získám textové řetězce, které bude nutné ještě dále upravovat. Druhou funkci používám jen tehdy, nezná-li PDFBOX dané písmo, a tudíž nemohu třídu showCharacter použít.

Nyní podrobně popíši nejdůležitější část celého mého projektu. Jedná se o třídu **showCharacter**, která je jádrem celého programu.

Funkce showCharacter slouží pro extrakci textového obsahu z objektů. Slova mohou být v objektech rozdělena na více částí. Nejprve jsem tedy musel zjistit jestli je dané slovo celé nebo se skládá z více částí. Abych mohl správně rozlišit, jedná li se o dvě různá slova nebo jen o jeho části, ukládal jsem si vždy začátek následujícího slova, který jsem vypočetl ze slova předchozího. Tuto hodnotu zjistím sečtením funkcí **text.getX() + text.getWidth()**. V proměnné text jsou uloženy všechny informace o řetězci. Pomocí getX() lze zjistit jeho x-ovou souřadnici a přičíst k ní jeho šířku.

Tu lze dostat díky funkci getWidth(). Tak jsem získal souřadnice následujícího slova. Bylo-li slovo rozděleno na více částí, tak jeho následující část ,by měla začínat na souřadnicích, jež byly získány výpočtem. Pokud slovo na těchto souřadnicích nezačíná, je nutné vložit mezi slova mezeru, aby se oddělily.

Díky předchozímu postupu jsou rozlišena slova, která k sobě patří. To velmi usnadní umístění daných slov na výstupní stránku dokumentu. Každé slovo je uzavřeno do tagů ** / ** a za pomocí CSS stylů je zobrazeno.

K dispozici je metoda **druh**, která zjišťuje CSS vlastnosti každého slova. Tato metoda bude podrobně popsána v následující kapitole. Nyní bude stačit, že vrátí **id** CSS stylu, který bude použit pro aktuální slovo a jeho vlastnosti budou uloženy v externím souboru. Metoda rovněž umožňuje měnit CSS vlastnosti uvnitř vět. V některých případech nejsou ohraničeny mezi tagy div jen jednotlivá slova, ale může se jednat přímo i celý řádek, záleží na tom jak jsou velké mezery mezi slovy a rovněž na druhu použitého fontu. Z pohledu stylizování je tento způsob jednodušší, neboť není nutné vytvářet velké množství CSS stylů pro každé slovo.

To však s sebou přináší i jisté nepříjemnosti. Pokud je do divů uzavřen celý řádek, bere se první slovo jako vzor pro vytvoření CSS stylu. Což ovšem může být problém. Proto je nutné i za těchto podmínek kontrolovat každé slovo, nemění-li své vlastnosti. Pokud je zjištěno, že došlo ke změně, odlišné části do tagů se uzavřou dá se jim nové id, čímž vytvoříme jejich nové CSS vlastnosti.

Všechny data jsou postupně připojována do **StringBuffer.** To nám usnadňuje jak práci při zapisování do souboru, tak i jejich modifikovatelnost, pokud chceme dále programy upravovat nebo s nimi jinak pracovat.

Popsal jsem, jak se dají slova složit, pokud jsou z více částí, získávání jejich CSS vlastností a způsobu uzavírání do tagů. Ještě jsem se však nezmínil o jejich získání. Text z PDF souboru je možné získat dvěma způsoby. První, lehčí z nich, je pomocí funkce **text.getCharacter** ve třídě showCharacter, která nám vrátí textový obsah objektu. Tento způsob je sice velmi jednoduchý, ale ne vždy funkční. Dá se použít pouze za předpokladu, že PDFBox zná druh použitého písma nebo s písmem dokáže pracovat. Ke zjištění fontu je dobré použít funkci **text.getFont().getBaseFont(),** jestliže je její návratová hodnota rovna null PDFBox písmo nezná a je nutné použít jiný způsob pro získání textového obsahu.

Jak již bylo řečeno PDF zapouzdřuje všechna data do objektů. Ta mají své operátory, pomocí nichž je možné rozlišit o jaký typ objektu se jedná a jaká data nese. Díky metodě processOperators je možné získat přístup k těmto objektům a tudíž i datům, jež zapouzdřují. A to za pomoci dvou operátorů **Tj** a **TJ.** Díky nim lze získat textový obsah z objektů. Ten se ale musí ještě dále zpracovávat. K tomu slouží funkce **zavorky_slova,** starající se o odstranění přebytečných znaků v získaném řetězci. Jejím výstupem je čistý text, se kterým lze dále pracovat.

Tedy na začátku programu je nezbytné zjistit, o jaký typ fontu se jedná a z toho usoudit, jakým způsobem se bude získávat textový obsah dokumentu. Způsoby se mohou kombinovat a to se také ve

velké míře děje. Vychází to z vlastností slov, které mohou být napsány různými fonty a proto je nutné používat oba způsoby získávání textu.

Při převodu českých souborů lze počítat s problémy, které se týkají české diakritiky. PDFBox pracuje pouze s určitými typy fontů a obzvláště problémové jsou fontu z Latexu, který používá speciální české fonty. Ty si kódují české znaky různými způsoby, proto je problém s jejich převodem a následnou úpravou.

V programu je řešena částečná podpora kódovaní českých znaků. Některé fonty, speciálně z latexu, rozdělují české znaky na dva. Podle fontu se nejprve ukládají diakritická znaménka a pak až písmeno, jemuž náleží nebo obráceně. V mém programu jsem použil první případ. Pomocí metody **znaky** se provádí kódování češtiny. Funkci je dán řetězec, ve kterém jsou nahrazeny všechny české znaky a následně je poslán na výstup. Podrobně postup kódování popíši v kapitole Kódování češtiny.

Po podrobném prostudování principů a způsobu možného řešení předkládám část zdrojového kódu metody **showCharacter.**

```
protected void showCharacter( TextPosition text )
    {

      if ( pre_word == null) {
        id = druh(text,0);
        pre_word = text.getCharacter(); // uklada predchozi prvek
        str.append("<div "+id+">");
        div_open =1;
        poc_div++;
      }
      int sirka_slova = (int) (text.getX()+ text.getWidth());
      if ( pre_word_size == 0) pre_word_size = sirka_slova;

     mezera =(int) (text.getX() - pre_word_size) ;
     if(pre_word != text.getCharacter())
       if (mezera > 1 || mezera < -6 ){
             id = druh(text,0);
             if (poc_div!=0){
                   str.append("</div>\n<div "+id+">");
                   div_open =1;
                   poc_div++;
             }
             else {
                   str.append("\n<div "+id+">");
                   div_open =1;
                   poc_div++;
             }
       }
       else {
             String css = druh(text,1);
              int zmena  = css.length();
             if (zmena != 0)  str.append("</div><div "+css+">");
       }
     pre_word_size = sirka_slova;
   if (unknow_font == 1){
         if (words !=null)
         str.append(zavorky_slova(words));
         else
```

```
                str.append("");
    }
        else
      str.append(znaky(text));
     pre_word = text.getCharacter();
}
```

4.2 Tvorba vzhledu dokumentu

4.2.1 Vlastnosti a zobrazení písma

Vzhled výstupní XHTML stránky je dán kaskádovými styly, uloženými v externím souboru. Styly se generují přímo s textem v našem programu.O jejich vygenerování se starají dvě funkce:

- **Druh**(TextPosition text, int uvnitř) – vrací id použitého CSS stylu a získává informace z aktuálního slova, které se pak používají pro tvorbu CSS vlastností
- **Styly**(TextPosition text, String styl, String [] písmo) – zapisuje CSS vlastnosti do externího souboru

V kapitole Extrakce textu jsem se již o funkci druh hovořil. Zde jsem popisoval třídu ShowCharacter. Nyní se zaměřím na její hlubší analýzu.

Hlavním jejím úkolem je zjistit:

- **Typ písma** – jestliže je typ písma znám, lze jej získat pomocí funkce PDFBoxu **getFont().getBaseFont()**. Funkce vrátí řetězec, který obsahuje typ fontu a také jeho druh. Pak je již velmi jednoduché tyto informace z řetězce získat. Zde je ukázka některých druhů písma, jež PDF zná: Times New Roman, Arial, Arial Narrow, Verdana, Courier New, MS Sans Serif.
- **Druh použitého písma** – získám obdobným způsobem jako typ, jen nebudu z řetězce extrahovat typ písma, ale jeho druh. Například: Bold, Italic, BoldItalic pokud není nastaven ani jeden druh nastaví se normal
- **Barva písma** – s ní se zde pouze pracuje, ale je získána ve třídě processOperators, jak jsem si již dříve uvedl v kapitole Velikost a barva písma.

Každé slovo, které prochází funkcí showCharacter, aby mohlo být zobrazeno, musí projít také funkcí **druh**. V ní se z daného slova získávají potřebné informace uvedené výše. Výstupem funkce je id CSS stylu, které je přiřazeno procházejícímu slovu. A předchozí CSS vlastnosti slova jsou uloženy.

Druh a typ použitého písma se podaří získat za předpokladu, že PDF používá jeden ze známých fontů. Pokud ne, je třeba nastavit hodnoty „napevno“. Druh písma bude nastaven na „normal“, to znamená, že písmo nebude nijak zvýrazněné, a jeho typ na „Verdana“. Samozřejmě je jen volbou

uživatele jaké vlastnosti nastaví. Já jsem zvolil toto nastavení, ale není problém ho změnit. Tyto informace jsou pak předány funkci **styly,** která se nimi dále pracuje.

Funkce **druh** má ale ještě jeden důležitý úkol. Tím jsou změny těchto vlastností. Změní-li se ve větě druh, typ nebo barvu písma, musí funkce tuto změnu zaregistrovat a provést patřičné kroky. Funkce si pamatuje CSS vlastnosti předchozího slova, a pokud se aktuální slovo v nějaké vlastnosti liší, vytvoří se nový CSS styl a je vráceno nové id toho stylu. Pokud se oproti předchozímu slovu nic nemění, zůstává pořád aktivní předchozí styl.

Pokud se ale styl změnil, je o tom informována funkce **showCharacter.** Ta se postará o to, aby slovo bylo uzavřeno do div tagů, a tak odděleno od předchozího, aby bylo možno použít nové CSS vlastnosti.

Pak přichází na řadu funkce **styly,** která vytváří konkrétní CSS vlastnosti zobrazovaných slov. Id stylu dostává jako parametr současně s již zjištěnými vlastnostmi písma, které se získají ve funkci **druh**. Další vlastnosti - jako pozice - jsou získány z funkcí **getX()** a **get(),** které se musí ještě upravit a pak je přidal ke konkrétní CSS vlastnosti.

O velikosti písma, jsem se částečně zmínil v kapitole Velikost a barva písma, ale nyní se jí budu zabývat o něco podrobněji. K jejímu zjištění je zapotřebí dvou operátorů a to Tm a Tf. Pokud PDFBox zná typ použitého písma, aplikujeme operátor **Tf**. Jestliže je typ neznámý, používáme **Tm**. Tyto operátory používáme ve funkci processOperators, kde z nich dokážeme získat potřebné informace.

Po získání velikosti písma a pomocí metody **font_size**, která podle získané velikosti vybere odpovídající velikost písma. Ta dopovídá skutečné velikosti zobrazovaného písma. Může ale nastat možnost, že velikost písma nejde zjistit. V takovém případě použijeme funkci **getHeight()**, která nám určí výšku daného písma a následující postup je stejný. Opět je zavolána funkce font_size.

Do externího CSS souboru se pak zapisují následující vlastnosti:

```
#pozice_0{
        position:absolute;
        left:96px;
        top:146px;
        font-weight:800;
        font-family:Verdana;
        font-size: 35px;
        z-index:10;
        word-spacing:0.0px;
        color:rgb(100.0%,0.0%,0.0%);
}
```

4.2.2 Grafické prvky

Jak jsem se již dříve zmínil, mezi grafické prvky se počítá téměř všechno, co je tvořeno čarami, křivkami, také některé druhy obrázků a mnoho dalších objektů. Já jsem se zabýval pouze extrakcí tabulek různého druhu, ohraničením a podtržením písma.

Všechny tyto objekty je možné získat pomocí operátoru **re** ve třídě processOperators. Jsou-li známa požadovaná data, jedná se o čtveřici hodnot. První dvě udávají pozici prvku na stránce a další dvě jeho rozměr. Objekty jsou vykreslovány pomocí CSS stylů, což zajišťuje funkce **styly_grafika**, která do externího CSS souboru zapisuje získané informace o objektu tak, aby mohl být věrohodně vykreslen. V některých případech jsem se potýkal se zápornými rozměry objektů, tento problém jsem popisoval v kapitole Grafické prvky, kde je také uvedeno jeho řešení problému. Příklad zápisu jednoho CSS stylu:

```
#Grafika_3{
    position:absolute;
    left:783.27997px;
    bottom:934.27997px;
    width:0.0px;
    height:54.720013px;
    border:1px solid black;
    z-index:1;
    font-size:0px;
    background-color:black;
}
```

Barva daného objektu je zjištěna pomocí operátoru **f** ve třídě processOperator, který, pokud je nastaven udává, že se má objekt vykreslit aktuální barvou.

Grafické objekty nejsou ihned zapisovány do nového souboru XHTML, ale postupně se zapisují do buffru, který je vypsán do souboru až při načítání nové stránky, čímž se také buffer smaže. Tento postup jsem zvolil kvůli lepší a snadnější práci s formátováním dokumentu. Jelikož byly vždy problémy s ukončovacími divy textu, zvolil jsem zapisovaní grafických objektů až na konec souboru.

4.2.3 Obrázky

Jak již jsem nastínil dříve, ne všechny obrázky se dají z dokumentu extrahovat do externího souboru daného typu, který je přímo zadaný v souboru PDF. Pro získání obrázků slouží operátor **Do** ve třídě processOperators. Pokud se v dokumentu vyskytuje nějaký obrázek tato funkce ho zachytí a umožní s ním dále pracovat.

Obrázky, které jsou získány pomocí operátoru **Do** je třeba ještě dále specifikovat. Ne všechny se dají uložit do externího souboru. Některé jsou složeny pouze z křivek a čar a jsou uloženy jako inline objekty v PDF souboru.

Pomocí třídy PDXObjectImage knihovny PDFBox lze zjistit, zda se jedná o obrázek, který je složitelný do souboru. A díky funkci **write2file()** je možno obrázek do souboru uložit. Dále jsou získány souřadnice obrázku, a to pomocí aktuální transformační matice a také jeho velikost. To jsou velmi důležitá data, která jsou předána funkci **styly_obrazek**. Ta se postará o zobrazení obrázku na stránce výstupního XHTM dokumentu a následném správném umístění a zajištění skutečné velikosti obrázku na stránce.

Občasným problémem s nímž jsem se setkal byla velikost obrázků, které byly větší než ve skutečném PDF dokumentu. Tento problém jsem vyřešil možností nastavení procentuální velikosti obrázků. Pokud je tato přidána při spouštění jako další parametr (jde o procentuální velikost od 0 do 1), budou všechny velikosti obrázků násobeny touto konstantou. A tím je umožněno měnit jejich velikost ve výsledném dokumentu.

Nyní následuje příklad jednoho CSS stylu pro konkrétní obrázek:

```
#image_1{
    position:absolute;
    left:65px;
    bottom:896px;
    width:204px;
    height:66px;
    z-index:1;
}
```

4.3 Kódování češtiny

Při testování mého programu jsem narazil na problémy s českými znaky. Nejprve jsem se domníval, že princip, který jsem objevil je stejný pro všechny dokumenty, ale bohužel tomu tak není. Zdálo se, že české znaky se kódují způsobem oddělením háčků a čárek od písmen a jsou zobrazeny vždy před daný znak. Při pokusech o vyřešení tohoto problému se ukázalo, že některá písma pořadí přehodí. Nakonec jsem zjistil, že každé písmo si kóduje české znaky svým způsobem, a bylo tedy pro mě nemožné, tuto vadu programu zcela odstranit.

V programu je vyřešeno kódování pro české znaky, které si ukládají diakritická znaménka před dané písmenko. Tento způsob funguje bez problému, ale neřeší celý problém, ale jen jeho část.

K dispozici je metoda **znaky,** která se zabývá tímto nedostatkem a provádí kódování českých znaků. V následujícím postupu ukáži a vysvětlím princip, na kterém je metoda postavena.

Postup řešení:

1) Nejprve v toku dat zjistit, jestli se v něm vyskytuje nějaké diakritické znaménko.
2) Pokud ano, zjistit jeho pozici v řetězci a podle toho vykonat příslušné kroky.
3) Nachází-li se na konci, je třeba jej odstranit a především zapamatovat o jaké znaménko šlo.
4) V následujícím řetězci je nutno nahradit první písmenko adekvátním českým znakem, podle toho o jaké diakritické znaménko šlo.
5) Nachází-li se ale znaménko uprostřed řetězce, je nezbytné jej odstranit a následující znak nahradit českým.

Takto je nastíněna jen základní myšlenka celého problému, který je ve skutečnosti daleko složitější, ale princip jeho řešení zůstává zachován.

Dlouho jsem zkoušel najít vhodný způsob řešení této problematiky, ale bohužel jsem nebyl v hledání řešení úspěšný. Podle specifikace PDF, by měl každý PDF soubor obsahovat použitý font pokud je neznámý a současně způsob kódování znaků. Snažil jsem se přijít na způsob získání těchto velmi důležitých údajů, avšak bohužel neúspěšně.

Díky problémům s diakritikou jsem však vyřešil problém s převodem některých PDF souborů. Občas se stávalo, že při převodu některé PDF soubory vykazovaly chyby a nebylo tak možné převést je na XHTML. Problém spočíval v tom, že metoda showCharacter nedokázala pomocí funkce **getCharacter()** získat textový obsah. Tento nedostatek jsem vyřešil až za pomocí metody **processOperators.** Z toho plyne že, v metodě showCharacter je třeba způsob získávání textového obsahu podle toho, zda dané písmo známe nebo ne.

4.4 Problémy s grafickou skladbou dokumentu

Při extrahování textového obsahu a jeho následném zobrazení na stránce pomocí CSS stylů nenastávají závažnější problémy. Ty se začnou vyskytovat až při vkládání grafických prvků.

Při práci s textem „bojujeme“ pouze s velikostí písma, mezerami mezi slovy a znaky. Tyto vlastnosti formátování textu jsou velmi důležité kvůli správnému umístění obsahu na stránce dokumentu. Zkoušel jsem používat velikosti písma přímo ze souboru PDF, ale u každého dokumentu se vyskytovaly specifické problémy. Ne vždy daná velikost odpovídala skutečné velikosti. Proto jsem se rozhodl, že si vytvořím tabulku velikostí písma, ve které bude velikost písma v pixelech a jemu odpovídající velikost zobrazovaného písma.

Pomocí funkce **processOperators** a operátoru **Tw** lze zjistit velikost mezery mezi slovy. V některých případech se vyskytovaly velké hodnoty, proto jsem musel vybrat pouze takové, které byly přípustné pro mé formátování.

Nejprve jsem umísťoval na stránku celý řádek, který byl uzavřen mezi tagy , a pomocí absolutního pozicování v CSS, kdy za pomocí vlastností **top** a **left**, jež tvoří souřadný systém stránky, mu byla dána konkrétní poloha. Zde se ale vyskytovala řada problémů. Pokud některá slova měla mezi sebou větší mezery nebo se jednalo například o data v tabulkách. Tento způsob umístění zrušil veškeré mezery mezi slovy a ta se tak nacházela hned za sebou mezerami neoddělena.

Proto jsem zvolil obtížnější způsob umístění slov na stránku. Tento ale zachování pozice slov tak, jak byly skutečně umístěny v původním dokumentu. Každé slovo je uzavřeno mezi otevírací a zavírací **div** tag a je mu dána jeho přesná poloha na stránce. Tento způsob tedy zachovává všechny velikosti mezer mezi slovy, a tak dokáže vytvářet i tabulky, či sloupce. Složitější je v tom ohledu, že je problematické uzavírat slova do tagů. Nejprve musíme dané slovo složit a teprve pak ho vložit mezi tagy. Tato problematika už byla popsána dříve v kapitole Extrakce textu.

Když jsem začal přidávat grafické prvky a obrázky, začalo problémů přibývat. Nejvíce jich bylo způsobeno tím, že grafické prvky a obrázky mají počátek svého souřadného systému v levém dolním rohu, zatím co text v pravém horním. To tvořilo největší problém, neboť vždy záleží na velikosti stránky, podle níž se odvíjí umístění grafických prvků a obrázků.

Velikost stránky lze zjistit pomocí funkcí v PDFBoxu následujícím způsobem:

```
PDRectangle layout = page.getMediaBox();
s_width = (int) (layout.getUpperRightX()*(float)pixel/72);
s_height = (int) (layout.getUpperRightY()*(float)pixel/72);
```

Pokud je správně nastavena velikost stránky, vyhneme se velkým problémům.

Umístění grafických prvků na správné místo není tak obtížné, jako zajistit, zda se odpovídající textové a grafické prvky shodují. Jelikož velikost textu neodpovídá naprosto přesně původní velikosti, vyskytují se zde problémy. Původně odpovídající si textové a grafické prvky jsou vůči sobě někdy posunuty nebo není zachována jejich skutečná velikost.

Občas se vyskytly problémy s vybarvením grafických prvků. Především se tyto vybarvily aniž by měly. Snažil jsem se tento problém řešit, ale nepodařilo se mi najít způsob, kterým bych tomu mohl zabránit.

Vkládání obrázků se obchází bez větších obtíží. Zpočátku jsem s nimi měl značné problémy, ale když jsem objevil transformační matici a dokázal z ním získat skutečné rozměry, které jsou použity k určení správné velikosti obrázku na stránce dokumentu, byla práce s nimi dále už velmi jednoduchá. V programu je také možnost zadat parametr, jež procentuálně zmenšuje všechny obrázky v textu, pokud by transformační matice tyto informace neobsahovala.

4.5 Ukázka datového toku v PDF souboru

Pomocí operátorů, které uvedu později je možno z datového toku získat veškeré informace o PDF souboru a také veškerý jeho obsah. Ve funkci **processOperators** pak jen pomocí podmínek lze daný operátor zachytit a získat z něho potřebné informace. Zde uvádím příklad části jedno datového toku PDF souboru. Upozorňuji zvlášť na skutečnost, že vše je ukončeno nějakým druhem operátoru. Nebudu na tomto místě popisovat všechny, o některých jsem se již dříve zmínil a popsal. Zbylé operátory se stručnou charakteristikou uvedu v další kapitole.

Pomocí PDFBox, můžeme daný tok dat získat a dále ve funkci **processOperators** dále zpracovávat. Příklad získání datového toku:

```
String struktura = page.getContents().getInputStreamAsString();
System.out.println(struktura);
```

Zde uvedu příklad takto získaného datového toku a popíši některé jeho prvky, se kterými jsem se setkal v mé práci.

```
/Shape <</MCID 0 >>BDC
q
/GS0 gs
Jedná se o transformační matici,kde 1. a 4. hodnota udávají velikost objektu v typografických bodech,
5. a 6. hodnota jsou souřadnice obrázku ve stejných jednotkách
153.71999 0 0 49.5 49.07999 679.32001 cm
Zde je uložen obrázek
/Im0 Do
Q
EMC
/P <</MCID 1 >>BDC
scn nastavuje barvu, která se bude používat až do chvíle, než bude změněna, barva je typu RGB
/CS0 cs 1 0 0  scn
1 i
/GS0 gs

BT – označuje začátek textu
/TT0 1 Tf – nastavuje velikost textu na 1 typografický bod
Tc – velikosti mezer mezi písmeny, Tw – mezery mezi slovy, Ts – sklon písma,Tz – horizontální
zvětšení písma, Tr – způsob vykreslení písma, následujících 6 hodnot je transformační matice písma,
kde první a čtvrtá hodnot udává velikost písma a poslední dvě udávají jeho pozici neboli jsou to jeho
souřadnice
0 Tc 0 Tw 0  Ts 100  Tz 0 Tr 28.02 0 0 28.02 72 716.40002 Tm
( )Tj
ET – konec textu
EMC
Nastaví aktuální barvu na černou 0 – černá, 1 - bílá
0 g
re značí práci  s grafickým prvkem, pokud je získán ve funkci processOperators obsahuje čtyři
hodnoty, které určují jeho souřadnice x,y a dále jeho rozměry šířku a výšku.
29.82001 751.5 558.35999 4.5 re
Operátor f, pokud je nastaven, znamená, že daný objekt bude vykreslen aktuální nastavenou barvou
f
29.82001 710.45996 0.72 41.04001 re
```

```
f
587.45996 710.45996 0.71997 41.04001 re
f
/P <</MCID 2 >>BDC
/CS0 cs 1 0 0  scn
BT
/TT0 1 Tf
1.55971 Tc 28.02 0 0 28.02 72 682.32001 Tm
(  )Tj
ET
EMC
0 g
29.82001 676.44 0.72 34.02 re
f
587.45996 676.44 0.71997 34.02 re
f
/P <</MCID 3 >>BDC
/CS0 cs 1 0 0  scn
BT
/TT0 1 Tf
0.00011 Tc 0.00101 Tw 21 0 0 21 147.66 655.32007 Tm
Tj obsahuje textové informace, v programu je použita funkce showcharacter k jejich získání, ale
v některých případech jsem musel použít i tento způsob k získání daného textu. Vše závisí na typu
použitého fontu.Pokud je neznámý, mohou vzniknout komplikace a je nutná větší  práce s operátory.
(YMCA Training Program News )Tj
ET
```

Datový tok je velmi důležitým zdrojem informací o PDF dokumentu. Za pomoci operátorů z něho lze získat veškeré potřebné informace k vytvoření XHTML dokumentu. PDFBox sice dává k dispozici řadu nástrojů k získání všech různých údajů nebo informací, ale někdy je příliš složité je používat a nebo dokonce najít vhodný nástroj. Proto jsem nejprve nastudoval všechny potřebné operátory a pak již je snadné je v metodě **processOperators** používat.

Informace je většinou ve formě pole a obsahuje navíc nějaká data, která je nutné odstranit a pokud se toto podaří, všechny informace k sestavení požadovaného XHTML souboru jsou shromážděny.

5 Souhrn operátorů

Zde uvádím nejdůležitější operátory a současně číslo strany v PDF specifikaci, kde je možné se dozvědět více podrobnějších informací. Tento přehled obsahuje téměř všechny operátory. Kompletní a podrobnější souhrn operátorů je možno nalézt ve Specifikaci k PDF souboru na straně 539.

Jak již jsem uvedl dříve, operátory jsou nezbytnou částí mého projektu a pomocí nich lze získat veškeré potřebné informace o PDF souboru nebo extrahovat konkrétní data.

Tabulka 2 - Souhrn Operátorů

Operátor	Popis operátoru	Strana v PDF dokumentaci
b	Uzavření bloku, vyplnění a ohraničení bloku	152
B	Vyplnění a ohraničení bloku	152
b*	Uzavření, vyplnění a ohraničení bloku	152
B*	Vyplnění a ohraničení bloku	152
BI	Začátek obrázku	260
BMC	Začátek značeného obsahu	480
BT	Začátek textu	286
BX	Začátek sekce povolující nedefinované operátory	84
c	Připojení křivky do bloku	149
cm	Aktuální transformační matice	143
cs	Nastavení barvy pro vyplnění	198
CS	Nastavení barvy pro ohraničení	198
d	Nastavení stylu čáry	143
Do	Používá XObjekty	243
DP	Vyznačuje místo v datovém toku, používá slovník	480
EI	Konec obrázku	260
EMC	Konec značeného obsahu	480
ET	Konec textového obsahu	286
EX	Konec sekce povolující nedefinované operátory	84
f	Vyplní blok	152
g	Nastavení barvy – vyplnění	199
G	Nastavení barvy – obarvení	199
gs	Nastavení parametrů pro grafické prvky	143
h	Uzavření bloku	149
i	Nastavení tolerance plochy	143

ID	Začátek dat obrázku	260
j	Nastaví typ propojení čar	143
J	Nastaví typ konce čar	143
k	Nastaví CMYK barvu pro neohraničující operace	199
K	Nastaví CMYK barvu pro ohraničující operace	199
l	Přidává čáry do bloku	149
m	Začátek nového podbloku	149
n	Ukončení bloku bez vyplnění nebo ohraničení	152
q	Uložení grafických prvků	142
Q	Obnovení grafických prvků	142
re	Obdélník	149
rg	Nastaví RGB barvu pro neohraničující operace	199
RG	Nastaví RGB barvu pro ohraničující operace	199
s	Zavře blok a vybarví ho	152
S	Ohraničí blok	152
sc	Nastaví barvu pro neohraničující operace	198
SC	Nastaví barvu pro ohraničující operace	199
scn	Nastaví barvu pro neohraničující operace	199
SCN	Nastaví barvu pro ohraničující operace	199
sh	Vykreslí blok daným šrafováním	214
Tc	Nastavení mezer mezi znaky	280
Td	Přesune textovou pozici	287
TD	Přesun textovou pozici a nastav začátek	287
Tf	Nastav font a jeho velikost	280
Tj	Zobraz text	289
TJ	Zobraz text a umožni pozicování jednotlivých znaků	289
TL	Nastav začátek	280
Tm	Nastav textovou matici	288
Tr	Nastav způsob vykreslení textu	280
Tw	Nastav mezery mezi slovy	280
Tz	Nastav horizontální zvětšení	280
v	Připoj křivku do bloku	149
w	Nastav šířku čar	143
W	Nastav ořezávací blok	156

6 Použití

Vytvořil jsem program pro transformaci PDF dokumentů na XHTML. Tento program dokáže z daného PDF souboru extrahovat text a další objekty jako jsou například obrázky nebo grafické prvky. Nejvíce ho oceníme tehdy, pokud potřebujeme umístit některé PDF dokumenty na internet tak, aby bylo možné si je online prohlížet a nemuseli jsme je stahovat.

Program dokáže vytvořit téměř stejnou kopii PDF souboru, ale uložit ji jako XHTML stránku, tudíž další práce s ní je velmi jednoduchá. Navíc je tato stránka celá formátována pomocí CSS stylů, z čehož plyne její snadná změna vzhledu, a její jednoduché začlení do webových stránek nebo aplikací, kde se s ní dá mnohem lépe pracovat, než za použití PDF souboru. Velmi se tím usnadní práce s obsahem stránky, který je možné jednoduchou cestou změnit, a to kteroukoliv jeho část, bez potřeby dalších nástrojů.

Další možné použití je získání obrázků z PDF souborů, které program ukládá jako externí soubory, tudíž jsou nám dále k dispozici. A je možné je použít v dalších aplikacích, programech nebo dokumentech. Nelze tak ale získat všechny obrázky. Některé jsou složeny pouze z čar, křivek a jsou umístěny v toku dat. Tudíž je nejde uložit do externího souboru. Jedná se ale spíše o menší a jednodušší obrázky.

Při nenáročných úpravách by se dal program využít i k získání čistého textu z PDF souboru a k jeho dalšímu použití.

7 Testování

Program jsem testoval na několika různých PDF souborech, které jsem stáhnul z internetu. Tato práce přinesla řadu zajímavých výsledků.

Nejprve jsem pracoval s anglickými soubory. U těchto jsem dosáhl velmi věrohodných kopií. Jednalo se však pouze o čistě textové obsahy. Později se ukázalo, že je třeba do programu připojit grafické prvky a postupně začaly vyvstávat problémy s jejich umístěním tak, aby korespondovaly s textovým obsahem stránky. Nakonec se mi však podařilo vzniklé problémy z velké části odstranit a dosáhnout dobrých výsledků v tom, že PDF soubory si vzájemně odpovídají.

Grafické prvky jsou sice v některých částech mírně posunuty na rozdíl od textu, ale to je dáno tím, že se mi úplně nepodařilo vyřešit správné umístění a velikosti daného písma.

Při extrakci text z PDF souborů a za použití jednoduchých grafických prvků funguje program spolehlivě na téměř libovolný počet stránek. Zkoušel jsem přeložit studijní materiál do Analýzy informačních systémů, který má 178 stránek, a výsledek byl vynikající. Původní a nově vzniklý XHTML soubor jsou téměř identické.

Problémy nastávají až s příchodem českých PDF souborů, které používají speciální české fonty. Zde narážíme na problém s kódováním češtiny. Zkoušel jsem tento problém řešit, ne však s velkou úspěšností. Některé fonty rozkládají české znaky na dva, rozlišují háčky, čárky, kroužek a pak zvlášť znak, nad kterým se tyto symboly vyskytují. Některá písma je ukládají před, jiné za znak. V programu je řešena verze, v níž se symboly vyskytují před znakem a funguje spolehlivě.

Problémem zůstává že, každý český font si kóduje své znaky různým způsobem, a tak se mi nepodařilo tento problém plně vyřešit. Při srovnání běžně dostupných programů podobných tomuto na internetu jsem dospěl k velmi zajímavým závěrům. Většina z nich se totiž českými znaky vůbec nezaobírá a některé dokonce nezachovávají ani stejný vzhled dokumentů. Pouze extrahují text a zobrazí ho na HTML nebo XHTML stránku.

Další problém, který se vyskytl je vybarvování grafických útvarů. Zde se občas stává, že jsou vybarveny nesprávné objekty. Dlouho jsem se tímto problémem zabýval, ale nenašel jsem žádné řešení, které by daný nedostatek odstranilo. Jediné možné a velmi rychlé řešení je najít si ID objektu a v externím CSS souboru změnit jeho barvu.

Myslím si, že ač můj program vykazuje jisté chyby v české diakritice, dá se plně použít pro transformaci dokumentů a vzniklé chyby lze velmi jednoduše odstranit ručně. I přesto program práci velmi ulehčí a usnadní přístup k textovému obsahu PDF souboru, jež je jinak bez dalších nástrojů téměř nedostupný.

8 Závěr

Domnívám se, že při tvorbě tohoto programu jsem dosáhl velmi dobrých výsledků a tak splnil všechny požadavky, jež byly na tento program kladeny. Program funguje spolehlivě, snad až na drobné odchylky od originálu, které se dají lehce odstranit změnou CSS vlastností.

Při programovaní jsem se seznámil s řadou nových a zajímavých věcí. Naučil jsem se pracovat s programovacím jazykem Javou ve spolupráci s externími knihovnami, což bylo pro mě asi největším přínosem. Při získávání informací jsem procházel specifikaci PDF souboru, která je celá pouze v angličtině a při následném překladu jsem se naučil pracovat s odbornými výrazy, správně je definovat a používat.

Program je dále možné rozšiřovat. Toto téma jsem již dříve podrobně popisoval v kapitole Další možná rozšíření.

Z mých dosažených výsledků se dá usuzovat, že tento program je použitelný pro tvorbu XHTML stránek a jejich následném použití v různých druzích aplikací.

9 Literatura

[1] Thomas, K.: Portable Document Format: An Introduction for Programmers, MacTech vol. 15, issue 9, 1999

[2] PDF Reference, second edition, Verze 1.3, 2000
www.adobe.com/devnet/pdf/pdfs/PDFReference13.pdf

Webové stránky

[3] Kaskádové styly – wikipedie, otevřená encyklopedie
http://en.wikipedia.org/wiki/Cascading_Style_Sheets

[4] Java - wikipedie, otevřená encyklopedie
http://en.wikipedia.org/wiki/Java_programming_language
http://cs.wikipedia.org/wiki/Java

[5] XHTML - wikipedie, otevřená encyklopedie
http://en.wikipedia.org/wiki/Xhtml