VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

IMPLEMENTACE ALGORITMU PRO VIZUÁLNÍ SEGMENTACI WWW STRÁNEK

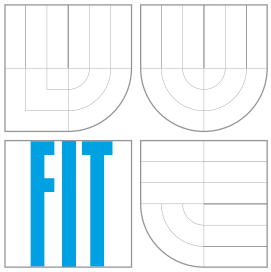
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ POPELA

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

IMPLEMENTACE ALGORITMU PRO VIZUÁLNÍ SEGMENTACI WWW STRÁNEK

IMPLEMENTATION OF ALGORITHM FOR VISUAL WEB PAGE SEGMENTATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ POPELA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN ZELENÝ

BRNO 2012

Abstrakt

Segmentace WWW stránek, neboli dělení stránky na různé sémantické bloky, je jedna z disciplín techniky extrakce informací. Diplomová práce se zabývá metodou Vision-based Page Segmentation - VIPS, která spočívá v dělení stránky na základě vizuálních vlastností prvků stránky. Metoda je uvedena v kontextu dalších význačných segmentačních postupů. V práci jsou popsány a na příkladech ukázány nejdůležitější kroky, ze kterých se tato metodika skládá. Pro metodu VIPS je nezbytná spolupráce s vykreslovacím jádrem WWW stránek, z důvodu získání DOM stromu stránky. V práci jsou představeny a popsány čtyři nejvýznačnější enginy pro programovací jazyk Java. Výstupem této práce je implementace algoritmu VIPS právě v jazyce Java s využitím jádra CSSBox. Dále je představena původní implementace algoritmu z laboratoří firmy Microsoft. Popsány jsou jednotlivé etapy vývoje knihovny realizující metodu VIPS a vlastního přístupu k jejímu řešení. Výsledek práce je v závěru demonstrován při segmentaci několika internetových stránek.

Abstract

Segmentation of WWW pages or page division on different semantics blocks is one of the disciplines of information extraction. Master's thesis deals with Vision-based Page Segmentation - VIPS method, which consist in division based on visual properties of page's elements. The method is given in context of other prominent segmentation procedures. In this work, the key steps, that this method consist of are shown and described on examples. For VIPS method it is necessary to cooperate with WWW pages rendering engine in order to obtain Document Object Model of page. The paper presents and describes four most important engines for Java programming language. The output of this work is implementation of VIPS algorithm just in Java language with usage of CSSBox core. The original algorithm implementation from Microsoft's labs is presented. The different development stages of library implementing VIPS method and my approach to it's solution are described. In the end of this work the work's outcome is demonstrated on several pages segmentation.

Klíčová slova

Vision-based Page Segmentation, Java, Linux, WWW, Segmentace, CSSBox, Document Object Model

Keywords

Vision-based Page Segmentation, Java, Linux, WWW, Segmentation, CSSBox, Document Object Model

Citace

Tomáš Popela: Implementace algoritmu pro vizuální
segmentaci www stránek, diplomová práce, Brno, FIT VUT v Brně, 2012

Implementace algoritmu pro vizuální segmentaci www stránek

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jana Zeleného

.....
Tomáš Popela
23. května 2012

Poděkování

Tímto bych chtěl rád poděkovat panu Ing. Janu Zelenému za jeho pomoc a vedení při realizaci této práce.

Obsah

1 Úvod **3**

2 Segmentace WWW stránek **4**
2.1 Text-based Page Segmentation - TextPS 4
2.2 DOM-based Page Segmentation - DomPS 5
2.3 Combined Page Segmentation - CombPS 6

3 Vision-based Page Segmentation - VIPS **8**
3.1 Struktura webové stránky v rámci VIPS 8
3.2 Popis algoritmu VIPS 12
3.2.1 Extrakce vizuálních bloků 14
3.2.2 Detekce vizuálních separátorů 19
3.2.3 Konstrukce struktury obsahu WWW stránky 21
3.3 Využití algoritmu VIPS 21

4 Vykreslovací jádra WWW stránek **22**
4.1 WebKit 22
4.2 Gecko 23
4.3 SWT toolkit 23
4.4 CSSBox 24
4.5 Cobra 25
4.6 Renderovací schopnosti 25

5 Implementace - algoritmus VIPS **27**
5.1 Knihovna VIPS 27
5.1.1 Použití 27
5.1.2 Výstup 28
5.2 CSSBox 30
5.2.1 Proces vykreslování 30
5.3 Návrh 31
5.4 Implementace 33
5.4.1 Detekce a extrakce vizuálních bloků 33
5.4.2 Detekce vizuálních separátorů 34
5.4.3 Konstrukce vizuální struktury 36
5.4.4 Výstup 38
5.4.5 API 39

6 Dosažené výsledky **42**
6.1 Demonstrace řešení . . . 42
6.1.1 Ukázka č. 1 . . . 42
6.1.2 Ukázka č. 2 . . . 43
6.1.3 Ukázka č. 3 . . . 45
6.2 Výkonnost . . . 46

7 Závěr **48**

Literatura **49**

Seznam příloh **53**

Přílohy **53**
Obsah DVD . . . 54

Kapitola 1

Úvod

Segmentace internetových stránek je pojem, který většině uživatelů sítě Internet bude jistě vzdálený, ačkoliv jsou s ním prakticky denně v kontaktu. Provází nás při hledání menu naší oblíbené restaurace nebo při hledání hodnocení nebo recenze na právě běžící film v kině. Tato technologie je totiž součástí vyhledávacích enginů internetových vyhledávačů jako například Yahoo[28]. S jejím využitím nám mohou vyhledávací portály navracet relevantnější výsledky, filtrovat je a lépe nás směrovat k výsledkům na jednotlivých stránkách.

Segmentace stránek je jednou z disciplín vědního oboru extrakce informací. Extrakce informací se zabývá získáváním strukturovaných informací z nestrukturovaných nebo částečně strukturovaných dokumentů. Jelikož se jedná o disciplínu počítačových věd, tak tyto dokumenty musí být strojově čitelné. Typickým příkladem této metodiky je zpracování vyfocených či jinak získaných dokumentů napsaných v přirozeném jazyce. Následně můžeme získaná data interpretovat nebo například uložit do databáze. Z uvedeného příkladu vidíme praktickou aplikaci těchto segmentačních technik.

V druhé kapitole jsou popsány tři techniky segmentace webových stránek z vědního oboru extrakce informací. Patří mezi ně Text-based Page Segmentation, která segmentuje stránku na základě textu, který se na stránce nachází. Dále je to DOM-based Page Segmentation, která analyzuje Document Object Model stránky. Nejvýznačnější metodou je segmentace založená na vizuálních vlastnostech a analýze DOM stromu na vizuální bloky. Tato technika se nazývá Vision-based Page Segmentation (VIPS) a je popsána v druhé kapitole. Posledním zástupcem je metoda Combined Page Segmentation, která se skládá z více metod, postupně aplikovaných na stránku.

Třetí kapitola se dopodrobna zabývá metodou Vision-based Page Segmentation. Jsou zde popsány a demonstrovány hlavní principy a postupy vizuální segmentace při dělení internetové stránky na význačné vizuální bloky.

Následující čtvrtá kapitola představuje nejvýznamnější dostupná vykreslovací jádra webových stránek pro programovací jazyk Java. Jsou zde popsány jejich vlastnosti, ukázáno použití v jazyce Java a také možnosti přístupu k Document Object Modelu stránky. V závěru jsou porovnány jejich vykreslovací schopnosti.

V pořadí pátá kapitola popisuje vlastní implementaci metody segmentace Vision-based Page Segmentation v programovacím jazyce Java. Je zde popsáná originální implementace z laboratoří firmy Microsoft a dále samotný návrh a realizace v jazyce Java.

Předposlední kapitola číslo šest obsahuje shrnutí výsledků a porovnání výstupů a výkonnostní stránky s originální implementací algoritmu VIPS.

Kapitola 2

Segmentace WWW stránek

Pod pojmem segmentace WWW stránek si můžeme představit dělení stránky dle určitých pravidel či postupů na vícero sémanticky různých bloků, jejichž obsah můžeme dále zkoumat nebo tyto bloky filtrovat.

Segmentace internetových stránek nachází uplatnění především v enginech internetových vyhledávačů jako například Yahoo. Zde slouží ke zpřesnění nalezených výsledků pro vyhledávanou slovní sekvenci a identifikaci její polohy v rámci stránky, na které se nachází. S touto technologií může engine rozpoznat a vynechat výskyty vyhledávaného sousloví například v komentářích pod článkem nebo v odkazech sloužících k navigaci po internetovém portálu a tímto může navracet relevantnější výsledky.[28]

Metodiky segmentace WWW stránek se také částečně prolínají s technikami analýzy dokumentů a jsou využívány při aplikaci oboru získávání znalostí.

V jednotlivých podkapitolách si přiblížíme tři nejvýznamnější postupy segmentace webových stránek dle [23]. Poslední metoda VIPS - Vision-based Page Segmentation bude popsána v samostatné kapitole číslo 3.

2.1 Text-based Page Segmentation - TextPS

Text-based Page Segmentation neboli metody segmentace stránky na základě textu, který se na stránce nachází. Mohou být tedy použity i k analýze textových dokumentů. Nejvíce se tyto metody používaly pro segmentaci webových stránek v době před 15 lety, kdy se internetové stránky skládaly především z bloků textu. Jejich základem je práce s plošným modelem stránky, kdy jsou ze segmentované WWW stránky odstraněny všechny HTML[1] značky a atributy. Tyto metody již dnes nejsou aplikovány na webové stránky samostatně, ale našly svoji roli jako dílčí části komplexnějších metod, jak je například uvedeno v sekci 2.3.[23]

Zástupcem těchto metod je metoda Fixed-length Page Segmentation, která je založena na extrakci tzv. pasáží (bloky dokumentu) pomocí pevného počtu sousedících slov. Tento proces si můžeme představit jako použití okna daných rozměrů (počet slov či znaků, velikost okna je rovna zpravidla od 50 až po 400 znaků), které se pohybuje celým dokumentem. Pokaždé, když okno zastaví, tak zobrazí nebo extrahuje daný počet sousedních slov. Každé prostřední slovo v daném okně je zároveň konečným slovem předchozího okna a začátkem následujícího. Pro každou takto získanou pasáž je spočteno ohodnocení, na jehož základě jsou poté určeny bloky v dokumentu. Navzdory jednoduchosti je tato metoda velmi robustní

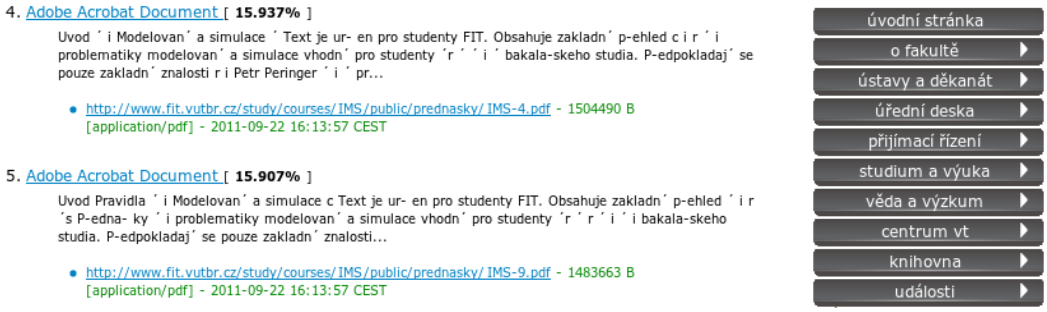
[1] HyperText Markup Language - http://www.w3.org/MarkUp

a výkonná. Naproti tomu její největší nevýhodou je, že do segmentačního procesu nejsou zakomponovány sémantické informace ze segmentované stránky či bloku. V [29] jsou dále rozebrány různé přístupy k velikosti okna a ohodnocení získané pasáže.[21, 26, 29]

2.2 DOM-based Page Segmentation - DomPS

DomPS je metoda realizující segmentaci stránky na základě jejího DOM[2] modelu se značkami jazyka HTML. Při procházení DOM modelu se jako segmentované bloky uvažují části uvozené v `HTML` značkách `<TITLE>` (titulek), `<P>` (odstavec), `<H1>` až `<H6>` (nadpis), `<META>` (popis obsahu), `<TABLE>` (tabulka) nebo jiných. Problémem této metody je přesnost rozkladu stránky na jednotlivé bloky, jelikož značky `<TABLE>` je využíváno kromě jiného také k rozvržení prvků v rámci stránky. Z tohoto důvodu nedojde k odpovídajícímu rozdělení na bloky. Dalším problémem je fakt, že mnoho webových stránek nedodržuje specifikace HTML, a tak ztěžuje či úplně znemožňuje správnou segmentaci.[23]

Příkladem algoritmu metody DomPS je například algoritmus WISH, jehož úkolem je zpracovat DOM strom stránky a extrahovat požadovaný obsah. Algoritmus je zaměřen na speciální struktury nazvané *datové záznamy - data records*, což jsou části stránky, které se opakují, ale pokaždé s jiným obsahem. Datové záznamy jsou definovány jako elementy, jenž se nachází na stejné úrovni DOM stromu, dále musí obsahovat sekvence opakujících se potomků a musejí mít podobného předka (uzel v DOM stromě). Předka označujeme jako *datovou oblast - data region*. Příkladem datového záznamu mohou být například položky po vyhledávání nebo položky menu (rozcestník) na stránce, jak je ukázáno na obrázku 2.1.



(a) Výsledky po vyhledávání (b) Menu

Obrázek 2.1: Ukázky struktury datový záznam v rámci algoritmu WISH

Prvním krokem algoritmu WISH je extrakce takových uzlů DOM stromu, jenž mohou být potencionálními datovými záznamy. Ty jsou detekovány postupným procházením DOM stromu po jednotlivých úrovních. Pokud na dané úrovni není nalezen žádný kandidát na datový záznam, tak se algoritmus přesune o úroveň níže. Po detekci všech potencionálních datových záznamů je zahájena fáze jejich filtrování, která využívá následující heuristiky s jejichž pomocí poznáme, zda se jedná o datový záznam či nikoliv.

- Datové záznamy (a k nim odpovídající datové oblasti) mají velké rozměry v porovnání s rozměry celé stránky.

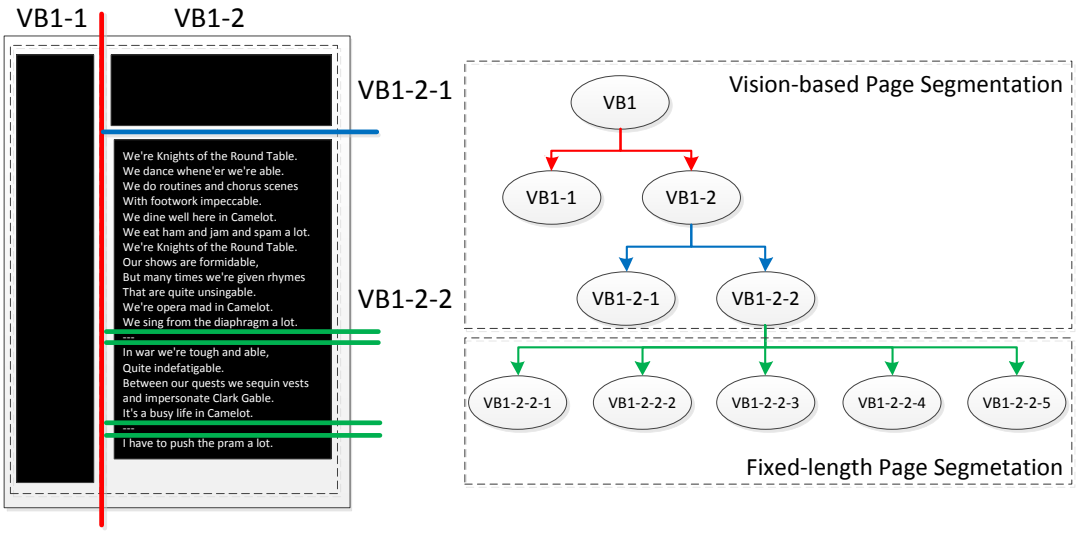
[2]Document Object Model - http://www.w3.org/DOM

- Datové záznamy se zpravidla opakují více než třikrát na celé stránce.
- Z popisu datového záznamu můžeme utvořit vzor, který můžeme aplikovat na ostatní záznamy.
- Obvykle skládají z nízkého počtu HTML značek.

Po filtraci všech potencionálních datových záznamů je všem datovým oblastem přiřazena číselná hodnota, která je vypočtena na základě obsahu datových záznamů v nich zapouzdřených. Do výpočtu jsou zahrnuty znaky textu, obrázky, ale také i elementy reprezentující prázdné místo. Datová oblast s největší takto vypočtenou hodnotou je označena za hlavní obsah stránky.[31]

2.3 Combined Page Segmentation - CombPS

Metoda skládající se z postupné aplikace různých segmentačních metod. Nejčastěji je používána kombinace metod VIPS a TextPS. V prvním kroku je aplikována níže popsaná metoda VIPS, kterou získáme jednotlivé vizuální bloky stránky. Na tyto bloky poté aplikujeme metodu Fixed-length Page Segmentation popsanou v sekci 2.1. Při využití CombPS dochází v porovnání s předešlými metodami k získání nejlepšího dělení stránek na jednotlivé bloky.[23]



Obrázek 2.2: Kombinace segmentačních metod Vision-based Page Segmentation a Fixed-length Page Segmentation

Na výše uvedeném obrázku můžeme vidět aplikaci metody CombPS na stránku uvedenou vlevo. První je aplikována metoda VIPS a stránka je segmentována na bloky VB1_1, VB1_2 a dále VB1_2_1 a VB1_2_2. Poté je proces segmentace metodou VIPS ukončen. Následně je na blok VB1_2_2 aplikována metoda Fixed-length Page Segmentation. Text v bloku bude postupně načítán do okna o zvolené velikosti. Pokud je k dispozici menší počet

slov než velikost okna, tak je text v bloku označen jako finální blok. Blok VB1_2_2 bude tedy rozdělen na pět bloků VB1_2_2_1 až VB1_2_2_5.

Kapitola 3

Vision-based Page Segmentation - VIPS

Vision-based Page Segmentation Algorithm neboli algoritmus pro vizuální segmentaci stránek je postup dělení stránky na sémanticky různé bloky tak, jak stránku člověk vnímá. Mezi hlavní vlastnosti této metody patří průchod webovou stránkou od shora dolů a nezávislost na stromu značek (například HTML značek) použitého programovacího jazyka.

Pokud není v této kapitole uvedeno jinak, tak níže uvedené informace v rámci této kapitoly pochází z práce VIPS: a Vision-based Page Segmentation Algorithm od autorů Deng Cai, Shipeng Yu, Ji-Rong Wen a Wei-Ying Ma[24].

3.1 Struktura webové stránky v rámci VIPS

Pro práci s vizuální segmentací webové stránky si musíme nadefinovat některé základní pojmy.

Definice 1. *Základním objektem nazveme koncový uzel DOM stromu, který nejde již dále segmentovat na menší části. V rámci VIPS pracujeme se strukturami, kde každý prvek zvaný blok, je buďto základním objektem nebo množinou základních objektů. Prvek v rámci metody VIPS nemusí odpovídat prvku v DOM stromu.*

Definice 2. *Webová stránka či podstránka* Ω *je trojice* $\Omega = (O, \Phi, \delta)$*, kde*
$O = \{\Omega^1, \Omega^2, \ldots, \Omega^N\}$ *je konečná množina bloků, jenž se vzájemně nepřekrývají,*
$\Phi = \{\varphi^1, \varphi^2, \ldots, \varphi^T\}$ *je konečná množina horizontálních a vertikálních separátorů, kde každý separátor má přiřazenu váhu, která je vypočtena na základě jeho vizuálních vlastností,*
$\delta : O \times O \to \Phi \cup \{NULL\}$ *je zobrazení, které popisuje vztah každých dvou bloků z množiny* O*.*

Definice 3. *Nechť* Ω_i *a* Ω_j *jsou dva objekty v* O*, pak* $\delta(\Omega_i, \Omega_j) \neq NULL$ *říká, že objekty* Ω_i *a* Ω_j *jsou od sebe odděleny separátorem* $\delta(\Omega_i, \Omega_j)$*. Můžeme také říci, že dva objekty jsou sousedními, pokud se mezi bloky* Ω_i *a* Ω_j *nenachází žádné jiné objekty.*

Dle definice číslo 2 je webová stránka definována jako trojice $\Omega = (O, \Phi, \delta)$, kde množina O obsahuje všechny vizuální bloky, které se na dané stránce nachází. Jsou to například bloky VB1_2 nebo VB1_3_3_2 z obrázku 3.2. Některé z těchto bloků jsou od sebe vizuálně odděleny. Tato skutečnost je reflektována v množině Φ , která obsahuje všechny vizuální separátory. Separátory dělíme na horizontální či vertikální a zpravidla to jsou různě široké linie na

webové stránce. Příkladem separátoru je například φ_2 nebo φ_3^1 na obrázku 3.2. Separátory mají přiřazenu určitou váhu, která odpovídá jejich vizuálním vlastnostem a vizuálním vlastnostem bloků, které oddělují. Všechny separátory v jedné množině Φ musí mít stejnou hodnotu. Posledním parametrem je zobrazení δ, které popisuje vztah mezi bloky z množiny všech vizuálních bloků O. Definice číslo 3 říká, kdy jsou dva bloky sousedními. Příklad sousedních bloků můžeme vidět na obrázku 3.2, kde právě bloky VB1_3_2_1 a VB1_3_2_2 jsou sousedními.

Obrázek 3.2 ukazuje vizuální strukturu pro webovou stránku 3.1. Celou, nerozdělenou stránku nazýváme VB1. Poznamenejme, že separátor φ_1^1 je zde zobrazen dvakrát, kdy první jeho výskyt označuje počátek separátoru a druhý konec separátoru. V hierarchickém stromě vizuálních bloků 3.3 si můžeme povšimnout, že vrchní část stránky se skládá ze dvou objektů neboli vizuálních bloků VB1_1 a VB1_2, které jsou odděleny separátorem φ_1. Vizuální struktura této části bude vypadat následovně:

$$\begin{aligned} O &= (\text{VB1\_1},\text{VB1\_2}) \\ \Phi &= \{\varphi_1\} \\ \delta &\begin{pmatrix} (\text{VB1\_1},\text{VB1\_2}) \\ jinak \end{pmatrix} = \begin{pmatrix} \varphi_1 \\ \text{NULL} \end{pmatrix} \end{aligned} \tag{3.1}$$

Dále můžeme specifikovat vizuální strukturu pro každý zanořený objekt stránky. Blok VB1_3_2 má tři potomky, které vymezují dva separátory:

$$\begin{aligned} \text{VB1\_3\_2} &= (\text{VB1\_3\_2\_1},\ \text{VB1\_3\_2\_2},\ \text{VB1\_3\_2\_3}) \\ \Phi^2 &= \{\varphi_3^1, \varphi_3^2\} \\ \delta^2 &\begin{pmatrix} (\text{VB1\_3\_2\_1},\text{VB1\_3\_2\_2}) \\ (\text{VB1\_3\_2\_2},\text{VB1\_3\_2\_3}) \\ jinak \end{pmatrix} = \begin{pmatrix} \varphi_3^1 \\ \varphi_3^2 \\ \text{NULL} \end{pmatrix} \end{aligned} \tag{3.2}$$

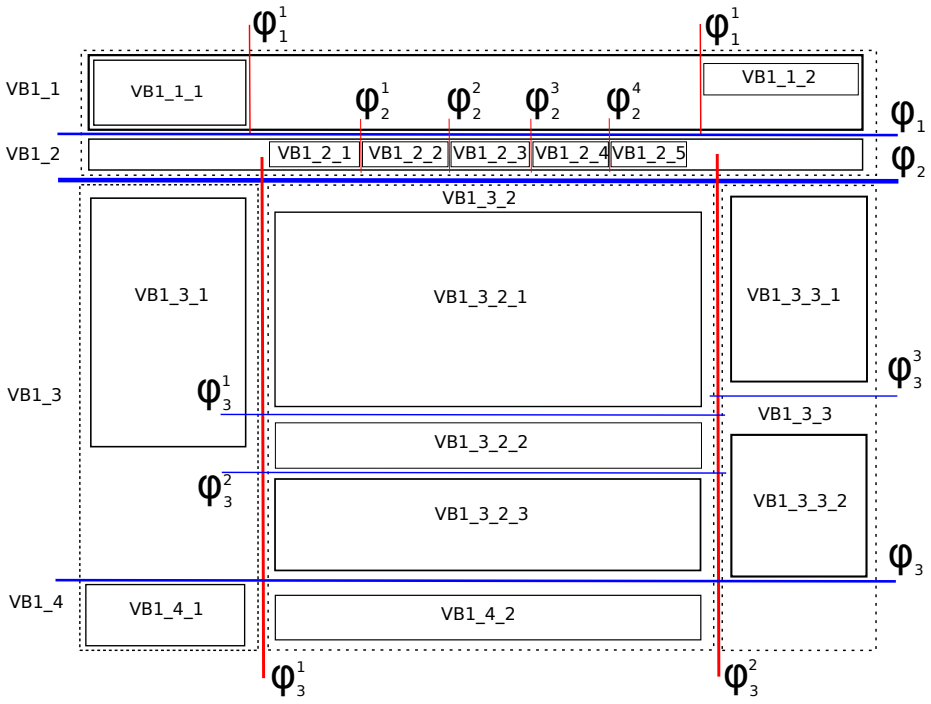
Pro každý vizuální blok je definován *stupeň konzistence* (Degree of Coherence - DoC) v rozmezí od 1 do 10, který určuje, jak konzistentní daný blok je. DoC má tyto dvě vlastnosti:

- Čím větší je hodnota DoC, tím více je blok konzistentní.
- V hierarchickém stromě není stupeň konzistence syna menší než jeho předka.

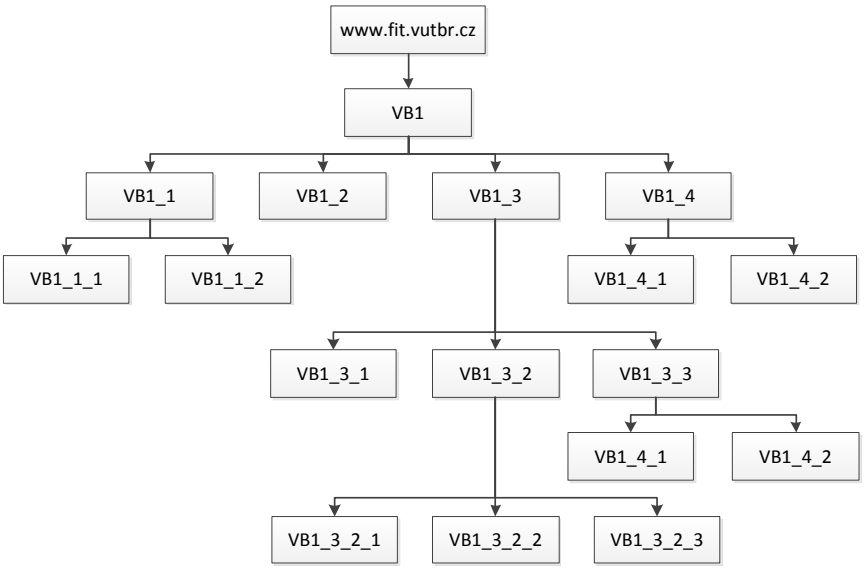
Můžeme také předdefinovat *povolený stupeň konzistence* (Permitted Degree of Coherence - PDoC) k dosažení různé jemnosti segmentace stránky na vizuální bloky. Čím je menší hodnota PDoC, tím více bude výsledek hrubší. Například v obrázku 3.2 nemusí být vizuální blok VB1_3_3 dále rozdělen na bloky VB1_3_3_1 a VB1_3_3_2, pokud mu bude přiřazeno dostatečně nízké ohodnocení PDoC.

Vizuální struktury slouží především k sémantickému dělení stránky. Každý prvek této struktury představuje určitou sémantiku. V obrázku 3.2 vidíme že blok VB1_3_1 obsahuje odkazy pro orientaci na internetové stránce nebo blok VB1_3_3 zobrazuje informace o naší fakultě.

Obrázek 3.1: Webová stránka `www.fit.vutbr.cz` před extrakcí vizuálních bloků

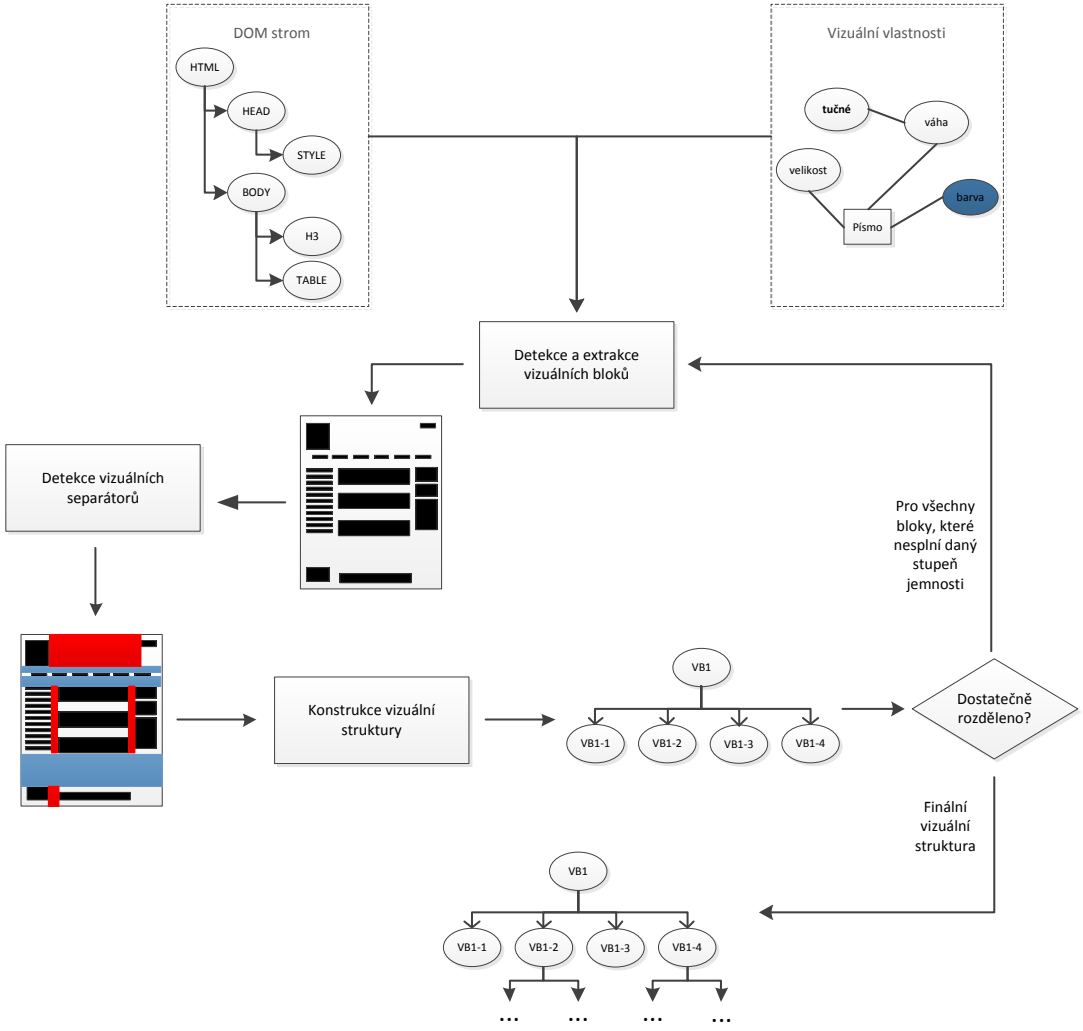


Obrázek 3.2: Extrahované vizuální bloky a separátory



Obrázek 3.3: Hierarchický strom vizuálních bloků stránky (pro zjednodušení není rozvinut blok VB1_2)

3.2 Popis algoritmu VIPS

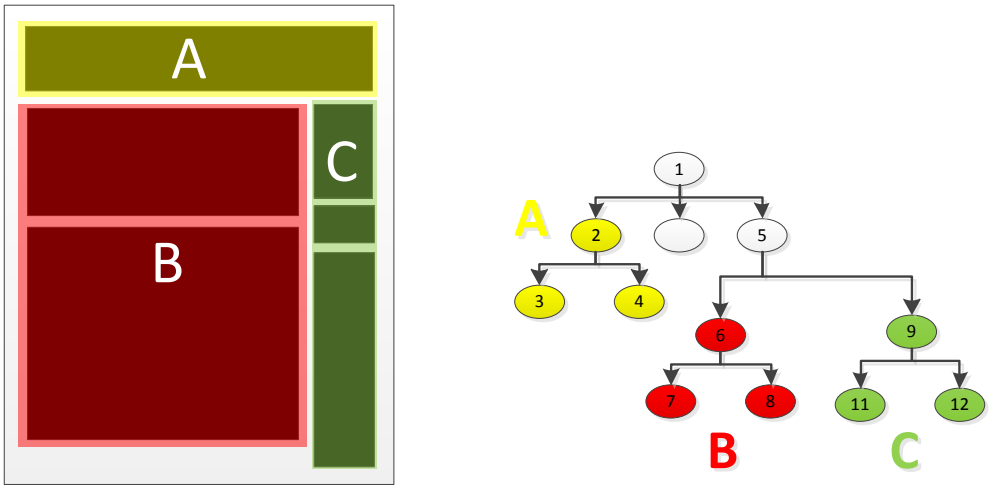


Obrázek 3.4: Postup vizuální segmentace

Proces segmentace, tak jak je uveden na obrázku 3.4, sestává ze 3 částí: detekce a extrakce vizuálních bloků, detekce vizuálních separátorů a konstrukce vizuální struktury obsahu. Webová stránka je v první iteraci rozdělena na několik velkých bloků. Poté na každý z těchto bloků opakovaně aplikujeme stejný segmentační proces, dokud nezískáme takové bloky, jejichž stupeň konzistence je větší, než povolený stupeň konzistence PDoC.

Pro každou aplikaci segmentačního procesu je z webového prohlížeče získán DOM strom s vizuálními informacemi, které odpovídají danému zpracovávanému bloku (pro první iteraci získáme informace pro celou internetovou stránku). Poté je od kořenových uzlů DOM stromu (například DOM uzel 1 v obrázku 3.5b) zahájen proces extrakce bloků. Pro každý uzel DOM stromu (uzly 1,2,5,6,9) je rozhodnuto, zda tvoří samostatný blok (nelze je dále rozdělit) nebo nikoliv. Pokud netvoří vizuální blok (uzly 1,5), tak bude dále segmentován na menší. Každému extrahovanému bloku (uzly 2,6,9) přiřadíme stupeň konzistence DoC,

který určíme na základě jeho vizuálních vlastností a HTML značky uzlu, který danému bloku odpovídá.



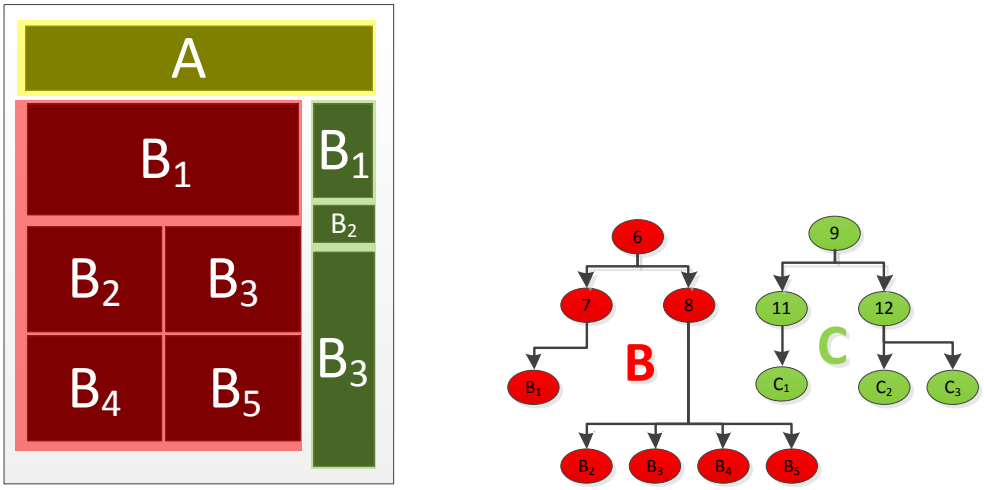
(a) Rozložení webové stránky

(b) Odpovídající část DOM stromu

Obrázek 3.5: Ukázka bloků na stránce spolu s odpovídajícím DOM stromem

Pokud jsou všechny bloky dané stránky či podstránky v dané iteraci extrahovány, tak se přechází do fáze detekce vizuálních separátorů. V této etapě jsou detekovány separátory oddělující získané vizuální bloky v předchozí fázi. Po detekci jsou jim nastaveny váhy na základě vlastností bloků, které oddělují. S pomocí těchto separátorů můžeme zkonstruovat hierarchii rozložení bloků na stránce (viz. diagram 3.3). Po konstrukci této hierarchie v dané iteraci algoritmu VIPS je pro každý koncový uzel ze získané struktury ověřeno, zda splňuje danou jemnost segmentace (PDoC). Pokud určenou jemnost nesplňuje, tak tento koncový uzel není označen jako vizuální blok a bude dále rozdělen. Například, pokud bloky B či C v obrázku 3.5 nesplní povolený stupeň konzistence, tak je označíme jako podstránky a budou dále segmentovány tak, jak je ukázáno na obrázku 3.6.

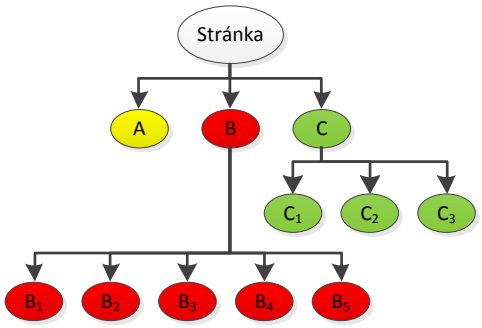
Po zpracování všech bloků je vytvořena výsledná vizuální struktura pro danou webovou stránku. Pro výše zmíněné příklady obdržíme strukturu uvedenou na obrázku 3.7.



(a) Rozložení webové stránky

(b) DOM strom odpovídající blokům B a C

Obrázek 3.6: Rozdělení bloků B a C



Obrázek 3.7: Vizuální struktura webové stránky z obrázku 3.5a

3.2.1 Extrakce vizuálních bloků

Nyní si blíže popíšeme nalezení a extrakci vizuálních bloků na internetové stránce. Obecně každý uzel v DOM stromě může reprezentovat vizuální blok. Avšak některé uzly, jako `<TABLE>` a `<P>` jsou použity spíše pro organizační účely a nejsou vhodné k reprezentaci samostatného vizuálního bloku. Při použití těchto HTML značek je daný uzel dále rozdělen na menší části (například tabulka na jednotlivé řádky či sloupce). Navíc, z důvodu velké flexibility jazyka HTML, mnoho stránek plně nedodržuje specifikaci W3C[1] HTML, a proto DOM strom nemůže vždy plně reflektovat pravý vztah různých DOM uzlů.

Pro každý extrahovaný uzel, který reprezentuje vizuální blok, je nastavena hodnota stupně konzistence DoC, která odpovídá jeho vizuálním vlastnostem. Tento proces je opako-

[1]World Wide Web Consortium - http://www.w3.org

ván, dokud nejsou nalezeny všechny příslušné uzly, které reprezentují všechny vizuální bloky na dané stránce.

Posouzení, zda může být DOM uzel segmentován, probíhá podle následujících pravidel:

- *Vlastnosti DOM uzlu.*
  Například HTML značka uzlu, barva pozadí uzlu, velikost a tvar bloku, který odpovídá danému DOM uzlu.

- *Vlastnosti potomků DOM uzlu.*
  Jako příklad můžeme uvést HTML značky synovských uzlů, barvu pozadí a velikost potomků. Dále můžeme například použít počet navzájem různých potomků daného uzlu.

Pokud budeme uvažovat specifikaci HTML 4.01[2], tak můžeme DOM uzly rozdělit do dvou kategorií, *řádkový uzel* a *blokový uzel* takto:

- *Řádkový uzel*
  Takový DOM uzel s jednořádkovými textovými HTML značkami, které ovlivňují vzhled textu a mohou být aplikovány na posloupnost znaků bez toho, aniž by způsobily vznik nového řádku. Mezi tyto značky patří například `<B>` (tučný text), `<BIG>` (velký text), `<FONT>` (specifikace písma), `<I>` (kurzíva), `<U>` (podtržený text) a další.

- *Blokový uzel*
  Všechny ostatní HTML značky mimo těch, které specifikují *jednořádkový uzel.*

Výše uvedené dělení ale není pevně dané, jelikož pomocí vlastnosti `display`[3] kaskádových stylů CSS můžeme měnit vlastnosti zobrazení HTML elementu. Můžeme tedy například prvku `<B>` přiřadit chování blokových uzlů.

Na základě vzhledu uzlu v prohlížeči a vlastností potomků uzlu definujeme následující typy uzlů:

- *Validní uzel*
  Takový uzel, který můžeme vidět v prohlížeči. To znamená, že šířka a výška uzlu jsou nenulové.

- *Textový uzel*
  Uzel DOM stromu, který odpovídá volnému textu a nemá přiřazenu žádnou HTML značku.

- *Virtuální textový uzel*

  - Řádkový uzel s potomky, kteří jsou textovými uzly, se nazývá virtuální textový uzel.
  - Řádkový uzel s potomky, kteří jsou textovými uzly nebo virtuálními textovými uzly nazveme také virtuálním textovým uzlem.

Vizuální bloky při extrakci dělíme dle následujících vlastností DOM objektů:

- HTML značky:

[2] Specifikace HTML 4.01 - http://www.w3.org/TR/html4
[3] CSS vlastnost `display` - http://www.w3schools.com/cssref/pr_class_display.asp

1. Značky jako <HR> (horizontální linie) jsou často používány k oddělení různých témat z vizuální perspektivy. Proto preferujeme dělení DOM uzlu pokud obsahuje jednu z těchto značek.
2. Pokud *řádkový uzel* má syna, který je *blokovým uzlem*, tak tento jednořádkový uzel dále dělíme.

- Barva: Preferujeme dělení DOM uzlu, pokud barva jeho pozadí je různá od barvy pozadí jednoho ze synů. Zároveň nebude tento potomek s odlišnou barvou pozadí v této iteraci dále dělen.
- Text: Pokud je většina potomků daného DOM uzlu textovými uzly nebo virtuálními textovými uzly, tak preferujeme dále nedělit tento uzel.
- Rozměry: Předdefinujeme si práh relativní velikosti (velikost uzlu vzhledem k velikosti celé stránky nebo podstránky) pro různé HTML značky (práh se liší pro DOM uzly, které mají různé HTML značky). Pokud jsou relativní rozměry uzlu menší než daný práh, tak preferujeme uzel dále nedělit.

Na základě těchto pravidel, můžeme určit heuristická pravidla, která jsou použita při rozhodování, zda by měl být uzel dále rozdělen či nikoliv. Pokud by neměl být dále dělen, tak je blok extrahován a je mu následně přiřazena odpovídající hodnota stupně konzistence DoC. Seznam heuristických pravidel seřazených podle jejich priority nalezneme v tabulce 3.2.1.

Pravidlo 1	Pokud DOM uzel není textovým uzlem a nemá žádné validní potomky, tak tento uzel nemůže být dále rozdělen a bude vyřazen.
Pravidlo 2	Pokud DOM uzel má pouze jednoho validního potomka a tento potomek není textovým uzlem, tak jej rozdělíme.
Pravidlo 3	Pokud DOM uzel je kořenovým uzlem DOM podstromu (který odpovídá danému bloku) a je jediným podstromem, který odpovídá danému bloku, tak jej dále dělíme.
Pravidlo 4	Pokud všechny potomci DOM uzlu jsou textovými nebo virtuálními textovými uzly, tak daný uzel nedělíme. Pokud je velikost a váha písma všech těchto potomků stejná, tak nastavíme stupeň konzistence extrahovaného bloku na maximum, tedy hodnotu 10. Jinak nastavíme DoC získaného bloku na 9.
Pravidlo 5	Pokud je jeden z potomků aktuálního DOM uzlu víceřádkovým uzlem, tak aktuální uzel dále dělíme.
Pravidlo 6	Pokud má jeden z potomků aktuálního DOM uzlu HTML značku `<HR>`, tak aktuální uzel rozdělíme.
Pravidlo 7	Pokud je barva pozadí aktuálního uzlu různá od jednoho z jeho synů, tak rozdělíme aktuální uzel a zároveň potomek s různou barvou pozadí nebude v této iteraci dále dělen. Nastavíme hodnotu DoC na 6-8 podle HTML značky uzlu potomka a jeho rozměrů.
Pravidlo 8	Pokud uzel má alespoň jednoho textového nebo virtuálního textového potomka a relativní rozměry uzlu jsou menší než práh, tak tento uzel nebudeme dále dělit. Nastavíme hodnotu DoC na 5-8 dle HTML značky uzlu.
Pravidlo 9	Pokud velikost potomka s největšími rozměry je menší než práh (relativní velikost), tak tento uzel dále nedělíme. Nastavíme hodnotu DoC odpovídající HTML značce uzlu.
Pravidlo 10	Pokud předchozí sourozenecký uzel nebyl rozdělen, tak aktuální uzel nedělíme.
Pravidlo 11	Rozděl aktuální uzel.
Pravidlo 12	Dále nerozděluj aktuální uzel. Nastavíme hodnotu DoC podle jeho HTML značky a velikosti.

Tabulka 3.1: Heuristická pravidla použitá ve fázi extrakce vizuálních bloků[24]

Pokud porovnáme dvě verze technických zpráv o algoritmu VIPS z let 2003[22] a 2004[24], tak zjistíme, že ve starší zprávě je uvedeno ještě jedno pravidlo vedle výše jmenovaných dvanácti.

Pravidlo	Pokud je suma velikostí všech potomků uzlu vetší než velikosti tohoto DOM uzlu, tak tento uzel bude dále rozdělen.

Tabulka 3.2: Dodatečné pravidlo ze starší technické zprávy algoritmu VIPS[22]

Toto pravidlo je umístěno mezi šestým a sedmým pravidlem v tabulce . Při dodatečné implementaci tohoto pravidla se však detekce vizuálních bloků nezměnila, a proto bylo nejspíše v originální zprávě také odstraněno. Při implementaci algoritmu VIPS v jazyce Java bude také toto pravidlo vynecháno.

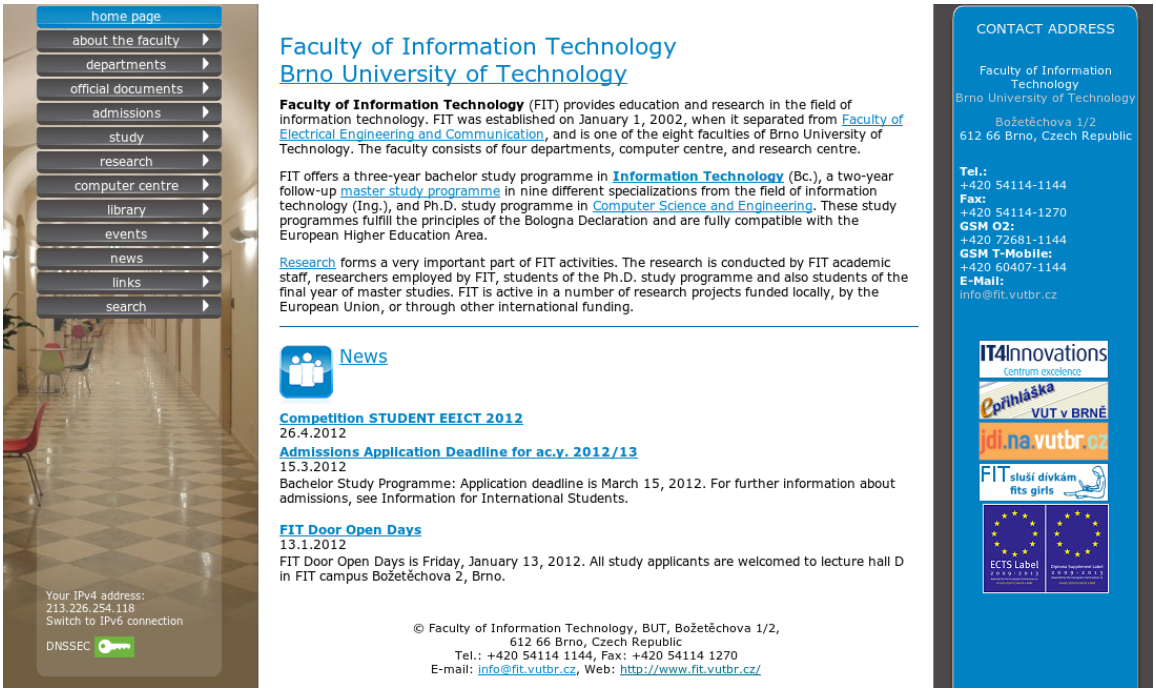
Pro různé DOM uzly s různými HTML značkami aplikujeme jiná pravidla, která jsou

uvedena v tabulce 3.3.

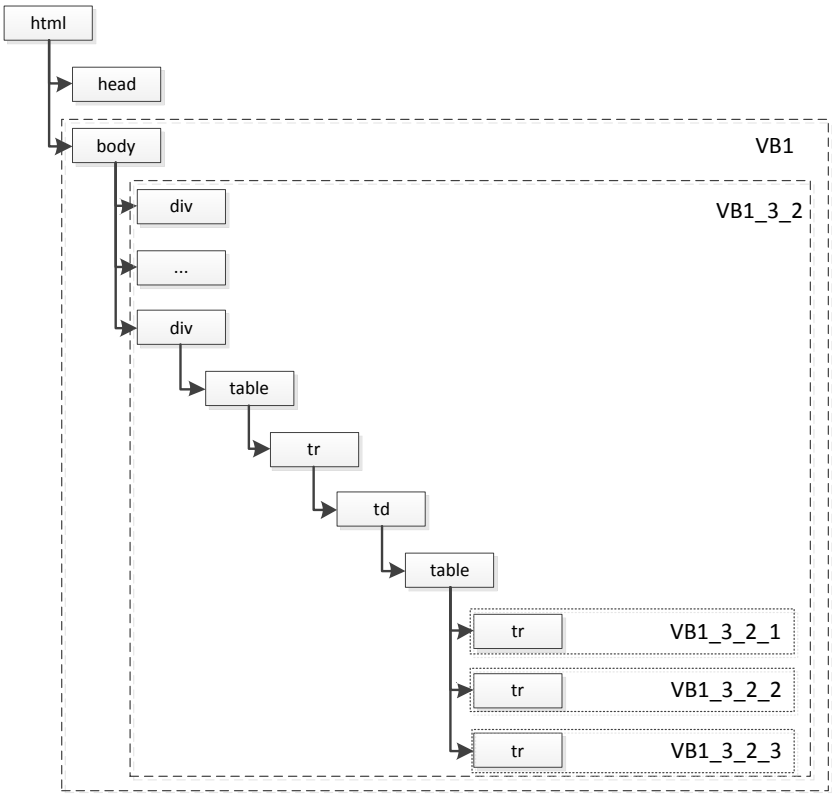
	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12
Jednořádkový textový uzel	x	x	x	x	x	x		x	x		x	
<TABLE>	x	x	x				x		x			x
<TR>	x	x	x				x		x			x
<TD>	x	x	x	x				x	x	x		x
<P>	x	x	x	x	x	x		x	x		x	
Ostatní značky	x	x	x	x		x		x	x		x	

Tabulka 3.3: Pravidla pro různé DOM uzly (odkazující se na pravidla v tabulce 3.2.1)[24]

Pokud budeme aplikovat extrakci vizuálních bloků na stránku, která je ukázána na obrázku 3.1 a její struktura na 3.2, tak budeme postupovat následovně. V první iteraci budou extrahovány vizuální bloky VB1_1 až VB1_4. Obrázek 3.8 zobrazuje vizuální bloky VB1_3, VB1_4 a obrázek 3.9 zobrazuje část odpovídajícího DOM stromu bloku VB1_3. Ten je dále rozdělen na tři vertikální bloky VB1_3_1, VB1_3_2 a VB1_3_3. Blok VB1_3_2 v sobě obsahuje tabulku, a proto jej budeme dále dělit. Při procházení tabulky po řádcích jsou extrahovány bloky VB1_3_2_1, VB1_3_2_2 a VB1_3_2_3. Pro danou podstránku jsme tedy získali tři vizuální bloky.



Obrázek 3.8: Část stránky s vizuálními bloky VB1_3 a VB1_4



Obrázek 3.9: Část DOM stromu výše uvedené podstránky

3.2.2 Detekce vizuálních separátorů

Poté, co jsou všechny vizuální bloky extrahovány a uloženy, nastává fáze detekce separátorů. Separátory jsou horizontální nebo vertikální linie ve webové stránce, které se nekříží s žádnými dříve extrahovanými vizuálními bloky. Oddělovače jsou reprezentovány dvojicí (P_s, P_e), kde P_s je počáteční a P_e koncový pixel[4]. Délku separátoru zjistíme spočítáním rozdílu mezi těmito dvěma hodnotami.

Algoritmus VIPS dokáže najednou pracovat pouze s horizontálními nebo vertikálními separátory, ne s oběma typy najednou. Musíme tedy aplikovat detekci separátorů v jednom směru a v takto vzniklých vizuálních blocích detektujeme separátory opačné orientace, čímž dosáhneme správného dělení vizuálních bloků.

Detekce vizuálních separátorů se skládá z následujících kroků:

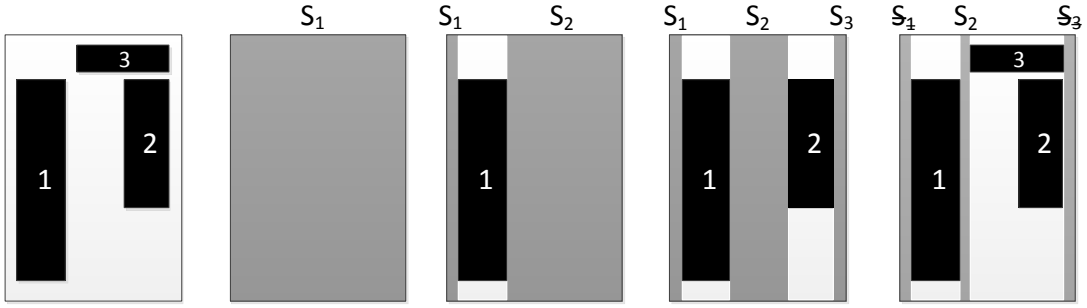
1. Inicializace seznamu separátorů. Seznam obsahuje pouze jeden separátor (P_{be}, P_{ee}), jehož počáteční a koncový pixel odpovídá rozměrům stránky.
2. Pro každý získaný vizuální blok je vyhodnocen vztah mezi ním a každým separátorem:
   (a) Pokud se blok nachází v separátoru, tak jej rozdělíme.
   (b) Pokud se blok kříží se separátorem, upravíme parametry separátorů.

[4]Pixel - https://en.wikipedia.org/wiki/Pixel

(c) Pokud blok zakrývá separátor, tak separátor odstraníme.

3. Odstraníme čtyři separátory, které slouží jako ohraničení stránky (například separátory S_1 a S_3 v obrázku 3.10).

Mějme obrázek 3.10 jako příklad, kde černé bloky reprezentují vizuální bloky, které se nacházejí na stránce. Ukážeme si postup detekce vertikálních separátorů. Na počátku máme pouze jeden separátor S_1, který pokrývá celou stránku. Jak je ukázáno na třetím obrázku, po vložení bloku číslo 1 je jediný separátor rozdělen na dva a to na S_1 a S_2. Obdobně vložíme blok číslo 2. Při vkládání třetího bloku se tento blok překříží se separátorem S_2, jehož parametry budou upraveny. Následně jsou odstraněny separátory S_1 a S_3, které tvořily ohraničení stránky a získáme separátor S_2, který je jediným detekovaným vertikálním separátorem na této stránce.



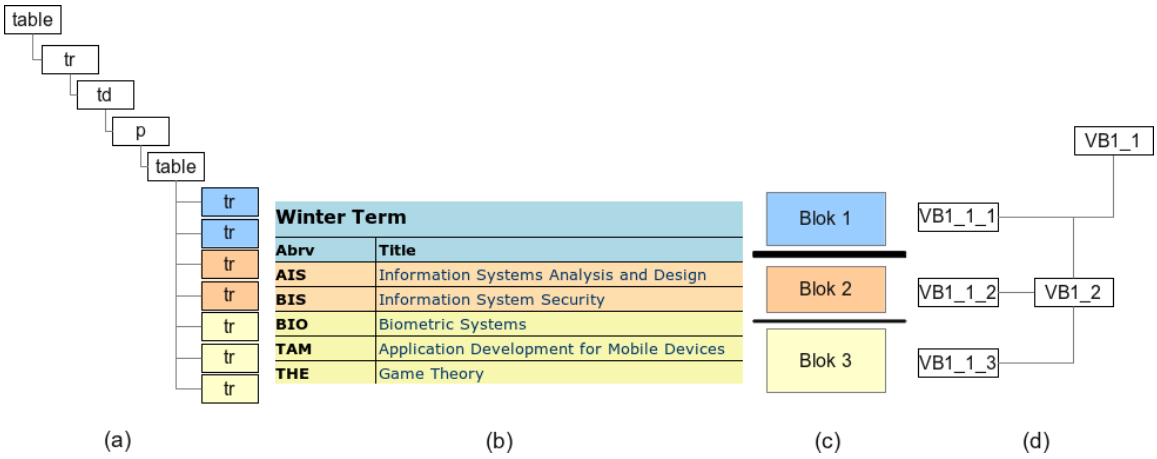
Obrázek 3.10: Detekce vertikálních separátorů stránky

Separátory jsou určeny k oddělení bloků s různou sémantikou, z čehož plyne možnost nastavit váhu separátoru, která je určena na základě vizuálního rozdílu těch bloků, které separátor odděluje. Následující pravidla jsou použita při určování váhy:

1. Čím větší je vzdálenost mezi bloky na opačných stranách separátoru, tím větší je hodnota váhy.
2. Pokud se separátor překrývá s určitými HTML značkami (například `<HR>` (horizontální linie)), tak jeho váha bude vyšší.
3. Pokud je barva pozadí vizuálního bloku různá na obou stranách separátoru, tak váha tohoto separátoru bude zvýšena.
4. Pokud jsou větší rozdíly ve vlastnostech písma (velikost či váha) na obou stranách horizontálních separátorů, tak váhu separátoru zvýšíme. Navíc bude váha zvýšena i v tom případě, že velikost písma v bloku nad separátorem je menší, než velikost písma v bloku pod separátorem.
5. Pro horizontální separátory dále platí, že pokud je struktura na obou stranách separátorů velmi podobná (například oba bloky obsahují text), tak váha tohoto separátoru bude snížena.

Na obrázku 3.11b je ukázána část stránky, na které budeme provádět proces extrakce separátorů. Odpovídající část DOM stromu je na obrázku 3.11a. Po procesu extrakce vizuálních bloků jsou získány a uloženy tři bloky a následně jsou detekovány dva horizontální separátory. Následně nastavíme váhy separátorů dle výše uvedených pěti pravidel. Separátor,

který se nachází mezi blokem číslo 1 a 3 získá větší váhu, než separátor mezi bloky číslo 2 a 3 a to z důvodu různých vah písma (pravidlo 4). Výsledné separátory jsou ukázány na obrázku 3.11c, kdy platí, že čím silnější je čára, tím větší je její váha.



Obrázek 3.11: Extrakce vizuálních bloků stránky, detekce separátorů, konstrukce struktury obsahu.

3.2.3 Konstrukce struktury obsahu WWW stránky

Po úspěšné detekci separátorů a přiřazení jejich vah může být sestavena odpovídající struktura obsahu. Proces konstrukce začíná od separátoru s nejnižší vahou. Bloky, které tento separátor odděluje jsou sloučeny, čímž vznikne nový blok. Tento proces slučování probíhá do té chvíle, dokud nejsou zpracovány i separátory s nejvyššími vahami. Stupeň konzistence (DoC) těchto nově vzniklých bloků je nastaven na základě maximální váhy separátorů nacházejících se v oblasti bloku.

Poté je každý koncový uzel zkontrolován, zda splňuje předem danou jemnost (DoC > PDoC). Pro každý blok, který daný požadavek nesplní, aplikujeme znovu fázi extrakce vizuálních bloků za účelem dalšího dělení daného uzlu. Pokud všechny uzly splní danou jemnost, tak je proces konstrukce struktury obsahu zastaven a vizuální struktura obsahu pro celou stránku je sestavena.

V první iteraci algoritmu pro obrázek 3.11 na vstupu je vybrán druhý separátor a bloky číslo 2 a 3 jsou spojeny do nového bloku VB1_2. Takto získaný nový blok je spojen přes první separátor s prvním blokem, čímž vznikne výsledný blok VB1_1. Každý koncový uzel jako VB1_1_1, VB1_1_2 a VB1_1_3 bude navíc zkontrolován, zda splňuje danou jemnost.

3.3 Využití algoritmu VIPS

Algoritmus VIPS se stal velmi dobrým základem pro mnohé další segmentační metody a algoritmy. Můžeme zde zmínit algoritmus od autorů Burget a Rudolfová, který vychází z VIPS, ale dále jej vylepšuje. Je nezávislý na DOM stromě, ale vychází z konstrukce tzv. stromu boxů, jenž zastávají vizuální bloky na stránce. Dále je tento strom zpracováván a analyzován, až je vytvořena finální vizuální segmentace stránky.[31]

Další využití nalezla knihovna VIPS například při modelování webových aplikací a jejich následné prezentaci v jazyce UML.[25]

Kapitola 4

Vykreslovací jádra WWW stránek

Úlohou vykreslovacích jader webových stránek (layout engine nebo také rendering engine) je po předložení obsahu a informací o rozložení a formátu tohoto obsahu dané informace vykreslit na obrazovku. Obsah může být ve formě jazyka HTML, XML[1] či obrázků a jiných. Formátovací informace jsou poskytovány v jazyce CSS[2] nebo určitými HTML značkami. Zpravidla jsou tyto informace definovány v rámci jedné webové stránky, jejíž adresu stačí vykreslovacímu jádru předat. Vykreslovací jádro je typicky použito v internetových prohlížečích, emailových klientech či všude tam, kde je třeba zobrazit webové stránky nebo obsah v jazyce HTML.[17]

Mezi nejznámější vykreslovací jádra patří WebKit a Gecko, jenž si v následujících kapitolách popíšeme spolu s některými, ne tak rozšířenými jádry CSSBox a Cobra, která jsou implementována v programovacím jazyce Java.

4.1 WebKit

Mezi v poslední době nejvíce rozšířená renderovací jádra webových stránek patří WebKit. Tento engine je open source a je používán napříč všemi platformami. Je založen na jádře KHTML z projektu desktopového prostředí KDE[3]. Za jeho vývojem nyní stojí takové firmy jako Google, který jej používá ve svém internetovém prohlížeči Google Chrome, Apple (prohlížeč Safari) či Nokia. Oblíben je také v mobilních operačních systémech, kde je použit ve výchozích prohlížečích platforem iOS a Android. V době psaní této práce byl prohlížeč Google Chrome 16, jenž WebKit využívá, nejrozšířenějším internetovým prohlížečem na trhu.[13, 18, 19]

Vykreslovací jádro WebKit je napsáno v jazyce C++. Je multiplatformní a podporuje řadu moderních technologií jako HTML 5[4] a CSS3. WebKit je distribuován pod licencemi LGPL [9] (části WebCore, JavaScriptCore) a licencí BSD verze 2.0 [1] (všechny jeho ostatní části).

Pokud chceme tento engine využít v jazyce Java, tak nejjednodušší použití nám nabízí sada grafických prvků SWT pro jazyk Java, jak je ukázáno v části 4.3.

[1] Extensible Markup Language - http://www.w3.org/XML
[2] Cascading Style Sheets - http://www.w3.org/Style/CSS
[3] KDE - http://www.kde.org/
[4] HTML 5 - http://www.w3.org/TR/html5/

4.2 Gecko

Gecko (dříve NGLayout či Raptor) je zdarma dostupný a otevřený vykreslovací engine vyvinutý společností Mozilla Corporation. Je navržen tak, aby podporoval otevřené internetové standardy a je používán nejen k vykreslování internetový stránek, ale také může být použit jako aplikační uživatelské rozhraní. Toho je dosaženo tak, že rozhraní je popsáno ve speciálním jazyce XUL[5] a následně vykresleno. Gecko je úspěšně používáno v mnoha známých aplikacích jako Firefox (internetový prohlížeč) či Thunderbird (správce elektronické pošty).[6, 7]

Renderovací engine Gecko je napsán v jazyce C++ a je multiplatformní. Dále podporuje moderní technologie jako CSS3 nebo HTML verze 5. Gecko je licencováno pod licencemi MPL [11], GPL [8] a LGPL [9].

Pro použití v programech napsaných v jazyce Java se dá Gecko aplikovat dvěma způsoby. Prvním je použití knihoven z Gecko SDK (technologie JavaXPCOM[6]) a druhým způsobem je využití prvku Browser ze sady grafických prvků SWT tak, jak je popsáno v kapitole 4.3.

Technologie JavaXPCOM, zprostředkovává komunikaci mezi naší aplikací a instancí jádra Gecko. Při použití této metody se DOM strom získá následujícím příkazem[5]:

```
nsIDOMDocument doc = browser.getDocument();
```

Zdrojový kód 4.1: Získání DOM stromu pomocí JavaXPCOM

Bohužel JavaXPCOM byl v nejnovější verzi GeckoSDK odstraněn a nebude dále podporován[10].

4.3 SWT toolkit

Výše uvedená renderovací jádra internetových stránek můžeme použít v programovacím jazyce Java v rámci prvků uživatelského rozhraní SWT.

SWT je sada grafických prvků pro platformu Java. Byla vyvinuta společností IBM a nyní je udržována společenstvím Eclipse Foundation společně s IDE Eclipse[7]. Cílem bylo vytvořit takovou sadu grafických prvků, která bude vykreslena stejně, nezávisle na platformě. Standard Widget Toolkit je distribuován pod licencí Eclipse Public License.[30, 16, 4]

Pro naše použití je nejdůležitější prvek uživatelského rozhraní *Browser*, který slouží pro vykreslování webových stránek. Tento prvek dokáže využít renderovací jádra Gecko a WebKit pro jejich vyrenderování.

Pokud chceme využít jádro WebKit v rámci prvku Browser, tak musíme mít v systému nainstalován balíček poskytující WebKitGTK (minimálně verze 1.2.0) a při vytváření instance prvku Browser zvolíme styl `SWT.WEBKIT`. Naopak, pokud chceme využít Gecko tak musí být nainstalován balíček poskytující XULRunner a instanci vytvoříme se stylem `SWT.MOZILLA`. Nabízela by se tedy možnost využití obou jader a uživateli nechat možnost jejich volby. Tato možnost, ale není doporučena, jelikož dochází ke konfliktům a neočekávanému chování při běhu aplikace.[15, 14]

[5] XML User Interface Language - https://developer.mozilla.org/En/XUL
[6] JavaXPCOM - https://developer.mozilla.org/en/JavaXPCOM
[7] Integrated Developement Environment Eclipse - http://www.eclipse.org

```
1  Display display = new Display();
2  Shell shell = new Shell(display);
3  shell.setLayout(new FillLayout(SWT.HORIZONTAL));
4  Browser browser = new Browser(shell, SWT.WEBKIT);
5  browser.setUrl("www.fit.vutbr.cz");
6  shell.open();
7  while (!shell.isDisposed())
8  {
9      if (!display.readAndDispatch()) display.sleep();
10 }
11 display.dispose();
```

Zdrojový kód 4.2: Aplikace vyžívající prvek Browser z SWT

Na výše uvedeném programu v jazyce Java je ukázán výsek zdrojového kódu aplikace, která vykreslí danou webovou stránku (v tomto případě `www.fit.vutbr.cz`). Na čtvrtém řádku je ukázána volba vykreslovacího jádra. Jak již bylo zmíněno, pokud nahradíme `SWT.WEBKIT` za `SWT.MOZILLA`, tak bude použito jádro Gecko. Ukázka vykreslené stránky je uvedena na obrázku 4.1a.

Pokud bychom chtěli z vykreslené stránky získat reprezentaci DOM stromu, tak tuto možnost implementace v SWT nenabízí. Jedinou možností je tedy využít funkce napsané v programovacím jazyce JavaScript a její spuštění nad danou stránkou[12]:

```
browser.evaluate("return document.getElementById('id_prvku}')
                                                    .childNodes[0].nodeValue;");
```

Zdrojový kód 4.3: Dotazování v jazyce JavaScript nad stránkou v prvku Browser

4.4 CSSBox

CSSBox je HTML/CSS vykreslovací engine napsaný v jazyce Java. Jeho hlavním účelem je poskytnout kompletní informace o obsahu renderované stránky a jejím rozvržení tak, aby bylo možné tyto informace dále zpracovávat. Výstupem enginu je objektově orientovaný model webové stránky, který můžeme vykreslit a získat tak obraz dané stránky. Model je dále vhodný na následné zpracování pomocí analytických algoritmů a to jak segmentačních algoritmů, tak analytických, jako například algoritmy sloužící pro extrakci informací. CSSBox je distribuován pod licencí GNU LGPL v3.0[9].[20]

Ve zdrojovém kódu 4.4 je ukázán výsek kódu aplikace, která otevře webovou stránku `www.fit.vutbr.cz` a pomocí jádra CSSBox ji vykreslí. Na prvním až třetím řádku se připojíme na danou adresu a získáme její obsah, který dále na řádku číslo 4 předáme parseru, jenž následně vytvoří reprezentaci DOM stromu. Dále jsou od řádku sedm přidávány informace z kaskádových stylů stránky. Takto rozšířená reprezentace DOM stromu, tvoří spolu s webovou adresou stránky a kořenovým prvek DOM stromu všechny nutné informace k následnému vykreslení stránky pomocí pomocí objektu `SimpleBrowser`. Kód je převzat z ukázkového příkladu `SimpleBrowser` projektu CSSBox. Vykreslenou stránku můžeme vidět na obrázku 4.1c.

Pokud chceme získat DOM strom vykreslené stránky, tak tuto možnost nám také CSSBox umožňuje. Ukázka je zakomponována do níže uvedeného kódu.

```
URL url = new URL("http://www.fit.vutbr.cz");
URLConnection con = url.openConnection();
InputStream is = con.getInputStream();
DOMSource parser = new DOMSource(is);
Document doc = parser.parse();
DOMAnalyzer dom_an = new DOMAnalyzer(doc, url);
dom_an.attributesToStyles();
dom_an.addStyleSheet(null, CSSNorm.stdStyleSheet());
dom_an.addStyleSheet(null, CSSNorm.userStyleSheet());
dom_an.getStyleSheets();
SimpleBrowser browser = new SimpleBrowser(dom_an.getRoot(), url, dom_an);
```

Zdrojový kód 4.4: Část zdrojového kódu aplikace, která vykreslí stránku pomocí jádra CSSBox

Zajímavostí je, že toto vykreslovací jádro je vytvářeno na naší fakultě v rámci Výzkumu informačních technologií z hlediska bezpečnosti a jeho autorem je Ing. Radek Burget, Ph.D..

4.5 Cobra

Cobra engine je součástí svobodného internetového prohlížeče Lobo Browser, který je kompletně napsán v jazyce Java. Podporuje HTML 4, JavaScript (využívá Mozilla Rhino JavaScript engine) a CSS 2. Vykreslovací jádro Cobra je vydáno pod licencí GNU LGPL.[3][9]

Renderovací engine Cobra není od roku 2009 dále vyvíjen. Bohužel se v poslední uvolněné verzi nachází chyba, která má za následek neschopnost vykreslit jakoukoliv webovou stránku na novějších distribucích Linuxu (nefunguje ve Fedoře 16, Ubuntu 11.10, ale funkční v Ubuntu 10.04). V operačních systémech firmy Microsoft se tato chyba neprojevuje.

Ukázka získaní vykreslení stránky a získání DOM stromu stránky je uvedena v následujícím zdrojovém kódu. Na prvním řádku se nachází deklarace prvku, do které budeme stránku renderovat. V druhém řádku vytvoříme instanci objektu, jenž renderuje webové stránky a tomuto objektu na dalším řádku předáme URL stránky. Od řádku číslo 4 je pak uvedeno přístup k DOM stromu.[2]

```
HtmlPanel panel = new HtmlPanel();
new SimpleHtmlRendererContext(panel, new SimpleUserAgentContext())
    .navigate("http://www.fit.vutbr.cz");
URL url = new URL("http://www.fit.vutbr.cz");
InputStream in = url.openConnection().getInputStream();
Reader reader = new InputStreamReader(in, "ISO-8859-2");
InputSourceImpl inputSource = new InputSourceImpl(reader,
    "http://www.fit.vutbr.cz");
Document doc = builder.parse(inputSource);
```

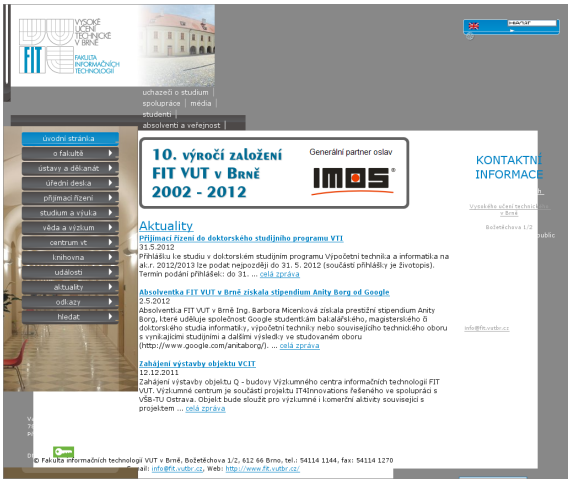
Zdrojový kód 4.5: Výsek aplikace využívající vykreslovací jádro Cobra

4.6 Renderovací schopnosti

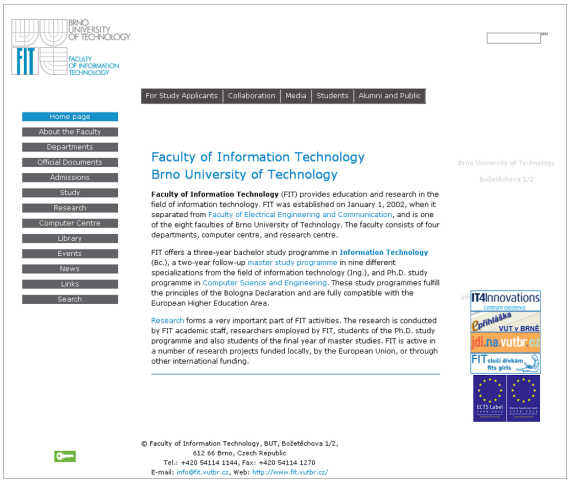
Na obrázku 4.1 může navzájem porovnat vykreslovací schopnosti jednotlivých jader. Zvětšený obrázek vykreslené stránky nalezneme na obrázku 3.1. Víme, že jádra WebKit a Gecko jsou využívány v nejrozšířenějších prohlížečích, a proto, jak je z obrázků patrné, tak nemají s vykreslením stránky sebemenší problémy. Pokud bychom dále pokračovali v porovnávání

kvality vykreslení, tak bude následovat CSSBox, který zvládl tuto stránku uspokojivě vykreslit. Chyby se vyskytly pouze u vyhledávacího pole vpravo nahoře, při vykreslování textu v části „Kontaktní informace" a dále nejsou vykresleny obrázky na pozadí jednotlivých prvků. V otázce obrázků na pozadí mi bylo od pana Burgeta sděleno, že podpora pro vykreslování obrázků na pozadí nemá v nynějším vývoji vysokou prioritu. Na poslední pozici by se umístilo jádro Cobra, které špatně vykreslilo horizontální menu, pozadí a spodní pruh s kontaktními informacemi.

(a) WebKit



(b) Cobra



(c) CSSBox

(d) Gecko

Obrázek 4.1: Porovnání stránky `www.fit.vutbr.cz` vykreslené jednotlivými renderovacími enginy

Kapitola 5

Implementace - algoritmus VIPS

V této kapitole si popíšeme originální knihovnu VIPS a dále návrh a implementaci Vision-based Page Segmentation algoritmu v jazyce Java. Vývoj probíhal na platformě Linux v distribuci Fedora 16 (64-bit) a v IDE Eclipse (3.7.2). Při implementaci bylo využito programovacího jazyka Java verze 1.7 (OpenJDK) a renderovacího jádra webových stránek CSSBox. Pro správu zdrojových kódů aplikace byl použit systém správy verzí Git[1].

5.1 Knihovna VIPS

Originální implementace algoritmu VIPS z laboratoří firmy Microsoft je dodávána jako dynamická knihovna ve formátu DLL[2]. Její poslední sestavení je ze dne 5.11.2008 pro platformu i386 a má velikost 496 kb. Knihovna využívá pro vykreslení internetové stránky a následné získání jejího DOM stromu vykreslovací jádro Trident[3], které je součástí internetového prohlížeče Internet Explorer. V operačním systému Windows XP (nainstalovány všechny dostupné aktualizace) s nainstalovaným Internet Explorer 8 je knihovnou využíváno jádro z Internet Explorer 7.

5.1.1 Použití

K demonstraci použití knihovny VIPS ve zdrojovém kódu jsou uveřejněny dva demo programy UsingVIPS a PageAnalysis, které provádějí vizuální segmentaci internetové stránky za pomoci této knihovny. První z nich je napsán v jazyce Visual C++ (dodáván ve formě zdrojových kódů) a umožňuje výslednou vizuální strukturu exportovat ve formátu XML či na obrazovku ve formátu plaintext. Druhý program je napsán také v jazyce C++ (dodáván v binární formě) a umožňuje prohlížení vizuální struktury, ale neumožňuje její následný export. Využití dostupného API knihovny VIPS si můžeme prohlédnout v ukázce 5.1. Upozorňuji, že se jedná pouze o ukázku a tento zdrojový kód není uveden v kontextu.

Oba uvedené demo programy jsou funkční na OS Windows XP. K přeložení prvního z nich je třeba mít nainstalováno Microsoft Visual C++ 6.0 a pomocí něj zdrojové kódy přeložit. Druhý stačí pouze spustit. Ke správné funkčnosti musíme knihovnu VIPS - PageAnalyzer.dll nakopírovat do úložiště systémových knihoven a následně ji zaregistrovat. Programy se mi z neznámých důvodů nepodařilo zprovoznit na OS Windows 7 (32-bit a 64-

[1] Git - http://git-scm.com/
[2] Dynamic-link library - https://en.wikipedia.org/wiki/Dynamic-link_library
[3] Engine Trident - https://en.wikipedia.org/wiki/Trident_(layout_engine)

bit). Dle mého názoru je na vině zastaralost jednotlivých částí těchto demo programů a přiložených knihoven.

Musím zde také zdůraznit jistou nestabilitu těchto programů, kdy docházelo dle mého názoru velmi často k jejich pádu a to jak při segmentaci stránky, tak při prohlížení výsledné vizuální struktury segmentované stránky. První chyba se týkala obou nástrojů, takže k pádu muselo docházet pravděpodobně někde uvnitř knihovny VIPS. K druhé chybě docházelo pouze v nástroji PageAnalysis například na stránce www.seznam.cz.

```
1  PAGEANALYZERLib::ILayoutAnalyzer2Ptr m_pLayoutAnalyzer;
2  m_pLayoutAnalyzer.CreateInstance(CLSID_LayoutAnalyzer2);
3  m_pLayoutAnalyzer->Initialize(0);
4  MSHTML::IHTMLDocument2Ptr pHTMLDoc = m_webBrowser.GetDocument();
5  m_iPDOC = 5;
6
7  if (xml_Output)
8  {
9      m_pLayoutAnalyzer->Analyze4(pHTMLDoc,_variant_t((long)m_iPDOC));
10     MSXML2::IXMLDOMDocumentPtr pFOMPage = m_pLayoutAnalyzer->GetFOMPage();
11     pFOMPage->save("VIPSResult.xml");
12 }
13 else
14 {
15     m_pLayoutAnalyzer->AnalyzeOutputAll_Text(pHTMLDoc,
16                                              _variant_t((long)m_iPDOC));
17     _bstr_t strResult = m_pLayoutAnalyzer->getResult();
18     m_strOut = (char*)strResult;
19 }
```

Zdrojový kód 5.1: Použití knihovny VIPS

Na prvních dvou řádcích se nachází deklarace a instanciace objektu, který slouží jako analyzátor využívající knihovnu VIPS. Na třetím řádku je prováděna inicializace objektu číselnou hodnotou 0. Při experimentování se změnou této hodnoty na libovolnou kladnou se chování knihovny neměnilo. Na dalším řádku je získána reprezentace segmentované stránky ve formě HTML dokumentu a poté je nastavena hodnota povoleného stupně konzistence (PDoC). Pokud bude výsledná struktura exportována do XML souboru, tak se provede segmentace pomocí metody `Analyze4`, která dostane jako parametry HTML dokument a PDoC. Následně je získána finální vizuální struktura stránky a uložena do XML souboru. Na řádcích 15 až 18 je uveden postup, vedoucí k segmentaci metodou `AnalyzeOutputAll_Text` a navrácení struktury v textové formě. API knihovny VIPS obsahuje ještě několik dalších metod, které ale nejsou bohužel nikde zdokumentovány.

5.1.2 Výstup

Knihovna VIPS umožňuje export výsledné vizuální struktury do formátu XML (viz. ukázka v 5.1). Výstupní soubor má jistou strukturu, která bude nyní popsána.

Pro každou segmentovanou stránku je vytvořen kořenový element `VIPSPage`. Ukázka pro stránku www.fit.vutbr.cz je uvedena v 5.2.

```
<VIPSPage PageRectHeight="1070" PageRectTop="1" PageRectWidth="1002"
    PageRectLeft="1" PageTitle="Fakulta informacnich technologii VUT v Brne"
    Url="http://www.fit.vutbr.cz" WindowHeight="1070" WindowWidth="1002"
    neworder="0" order="0">
```

```
    ...
</VIPSPage>
```

Zdrojový kód 5.2: Kořenový element `VIPSPage` ve výstupním XML knihovny VIPS

Jak vidíme v ukázce, tak element `VIPSPage` má řadu atributů, jejichž význam je uveden v následující tabulce 5.1.

Název atributu	Popis
PageRectHeight	Výška stránky v pixelech
PageRectTop	Zarovnání stránky z vrchní strany v pixelech
PageRectLeft	Zarovnání stránky z levé strany v pixelech
PageRectWidth	Výška stránky v pixelech
PageTitle	Titulek stránky (z HTML elementu `<TITLE>`)
Url	Internetová adresa stránky
WindowHeight	Výška okna v pixelech, ve kterém byla stránka vykreslena
WindowWidth	Šířka okna v pixelech, ve kterém byla stránka vykreslena
neworder	Není zdokumentováno
order	Pořadí elementu v rámci v výstupního XML

Tabulka 5.1: Atributy elementu `VIPSPage`

Pro každý detekovaný finální vizuální blok je poté do XML elementu `VIPSPage` generován element `LayoutNode`. Tyto elementy dodržují stejnou hierarchii v jaké se nachází na segmentované stránce. Element je ukázán v 5.3 a jeho atributy popsány v tabulce 5.2.

```
<LayoutNode BgColor="#ffffff" ContainImg="8" ContainP="3"
    ContainTable="false" DOMCldNum="4" DoC="1" FontSize="8"
    FontWeight="normal" FrameSourceIndex="0" ID="1-2-1" IsImg="false"
    LinkTextLen="503" ObjectRectHeight="1072" ObjectRectLeft="0"
    ObjectRectTop="0" ObjectRectWidth="1004" SourceIndex="57"
    TextLen="1898" order="5"/>
```

Zdrojový kód 5.3: Element reprezentující vizuální blok ve výstupním XML knihovny VIPS

Název atributu	Popis
BgColor	Barva pozadí stránky v hexa kódu či slovní reprezentaci
ContainImg	Počet obrázků (HTML značka `<IMG>`), které se nachází ve vizuálním bloku
ContainP	Počet odstavců (HTML značka `<P>`), které vizuální blok obsahuje
ContainTable	`true` pokud blok obsahuje tabulku (HTML značka `TABLE`), jinak `false`
IsImg	`true` pokud je vizuální blok obrázkem (má HTML značku `IMG`)
DOMCldNum	Počet potomků DOM uzlu reprezentujícího tento blok
DoC	Stupeň konzistence daného vizuálního bloku
FontSize	Velikost písma bloku
FontWeight	Váha písma použitého v bloku
ID	Identifikátor bloku
TextLen	Délka textu ve vizuálním bloku
LinkTextLen	Délka texty odkazů (HTML značka `A`)
SourceIndex	Číslo řádku ve zdrojovém kódu stránky, na kterém se nachází deklarace daného elementu
FrameSourceIndex	Číslo řádku, na kterém se nachází deklarace elementu v rámci HTML značky `<FRAME>`
ObjectRectTop	Zarovnání bloku z vrchní strany v pixelech
ObjectRectLeft	Zarovnání bloku z levé strany v pixelech
ObjectRectWidth	Výška bloku v pixelech
ObjectRectHeight	Šířka bloku v pixelech
order	Pořadí elementu v rámci výstupního XML

Tabulka 5.2: Atributy elementu `LayoutNode`

5.2 CSSBox

Jako vykreslovací jádro pro implementaci algoritmu VIPS jsem zvolil engine CSSBox. Mezi jeho přednosti, kvůli kterým jsem jej zvolil patří bezproblémová práce s DOM stromem stránky a tzv. stromem boxů, který v sobě uchovává informace o vykreslených prvcích dané stránky. Dále to byla také dostupnost zdrojových kódů a možnost přímé konzultace a komunikace s panem Burgetem, autorem tohoto jádra. Při vývoji byla použita poslední verze z SVN[4] ke dni 20. 4. 2012.

5.2.1 Proces vykreslování

Proces zpracování a vykreslení stránky probíhá v několika krocích. Zde si nejdůležitější z nich stručně popíšeme. Prvním je navázání spojení na danou stránku a získání jejího DOM stromu přes parser typu `DOMSource`. Takto získaná reprezentace DOM stromu je předána objektu typu `DOMAnalyzer`, který si ji uloží a poté postupným voláním jeho metod jsou elementy DOM stromu rozšiřovány o informace, které jsou uvedeny v kaskádových stylech dané stránky a také výchozích vlastností elementů. Takto rozšířená reprezentace DOM stromu je spolu s rozměry okna, do kterého chceme vykreslovat a adresou segmentované stránky předána objektu `BrowserCanvas`, jenž stránku vyrenderuje. Renderování probíhá

[4]Subversion - https://subversion.apache.org/

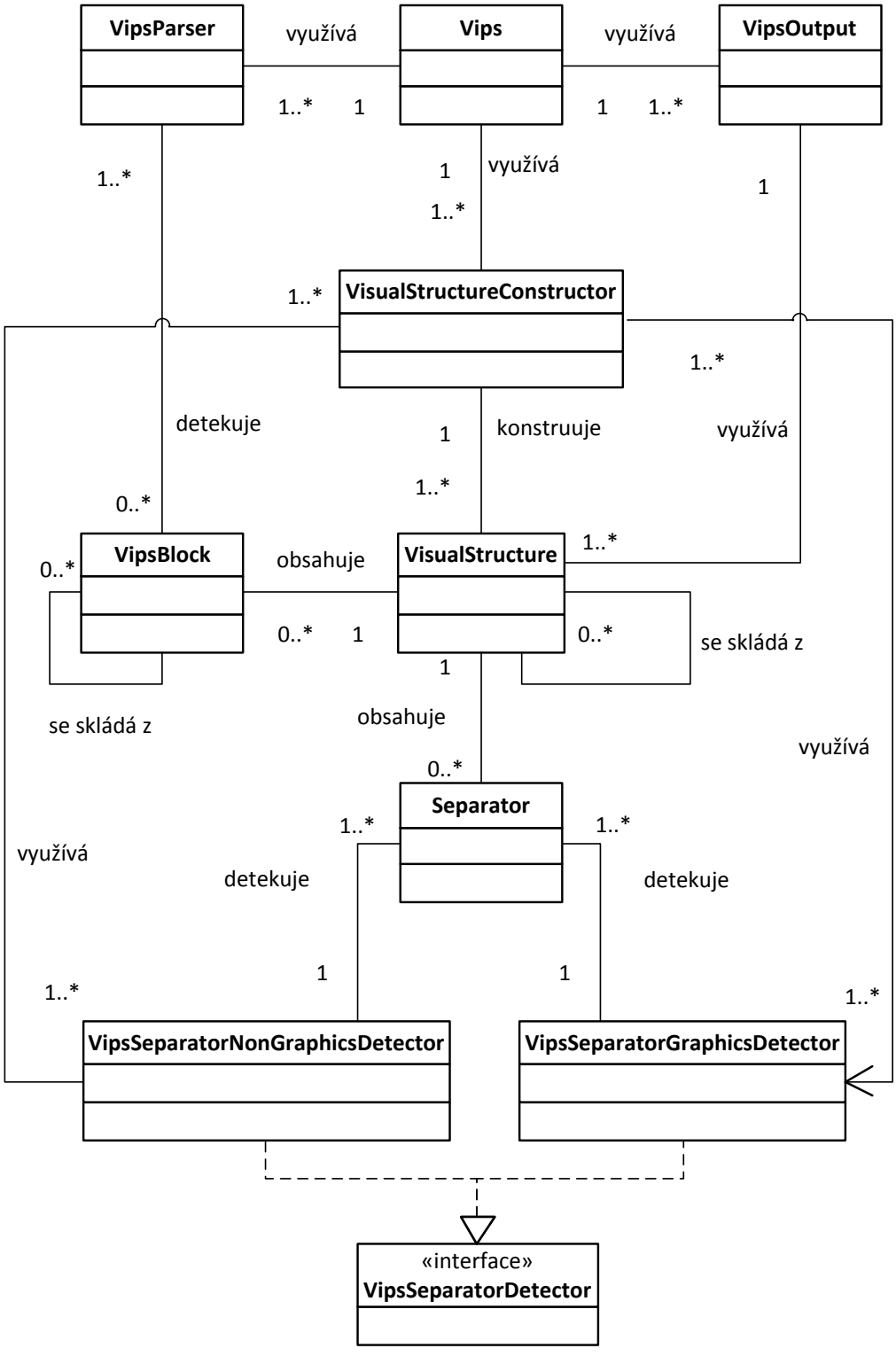
v několika krocích. Prvně objektem typu `BoxFactory`, kterému předáme `DOMAnalyzer` a adresu stránky získáme všechny grafické prvky neboli boxy, jenž se nachází na stránce. Z těchto prvků je poté vytvořena stromová struktura typu `Viewport`. Poté jsou upraveny rozměry okna předané před objekt `BrowserCanvas` tak, aby v něm byly všechny boxy obsaženy. Z takto získané reprezentace prvků stránky je poté v objektu `BrowserCanvas` vytvořena grafika stránky, kterou můžeme zobrazit uživateli.

Z pohledu implementace algoritmu VIPS jsou nejdůležitější objekty typu `DOMAnalyzer` a `Viewport`.

5.3 Návrh

Při návrhu skladby modulů implementace algoritmu VIPS v jazyce Java jsem vycházel z diagramu 3.4. Algoritmus jako celek byl rozdělen na tři hlavní moduly a to extraktor vizuálních bloků `VipsParser`, detektor vizuálních separátorů `VipsSeparatorDetector` a konstruktor vizuální struktury stránky `VisualStructureConstructor`. Dále byl přidán modul realizující výstup do XML souboru. Mezi těmito moduly, které odpovídají fázím vizuální segmentace algoritmem VIPS probíhala výměna informací pomocí modulů, jenž reprezentují vizuální blok `VipsBlock`, separátor `Separator` a vizuální strukturu `VisualStructure`. Při návrhu bylo rozhodnuto o tvorbě dodatečného grafického výstupu, který bude zobrazovat průběh segmentace. Ten je realizován ve třídě `VipsSeparatorGraphicsDetector`. Naproti tomu třída `VipsSeparatorNonGraphicsDetector` nemá závislost na potřebných grafických komponentách. Obě tyto třídy implementují společné rozhraní `VipsSeparatorDetector`, kterým je zaručena stejná skladba povinných funkcí pro realizaci detekce separátorů. Vhodnější by bylo modelovat daný problém pomocí abstraktní třídy, ale jelikož jazyk Java přímo neumožňuje (nepřímo lze provést přes prostředníka) dědičnost od dvou tříd zároveň (třída VipsSeparatorGraphicsDetector by rozšiřovala třídu `JPanel` a danou abstraktní třídu), tak bylo zvoleno rozhraní.

Vztahy mezi jednotlivými moduly jsou ukázány v diagramu tříd na obrázku 5.1.



Obrázek 5.1: Diagram tříd

5.4 Implementace

V následující sekci je popsána implementace algoritmu VIPS v programovacím jazyce Java. Budou zde nastíněny struktury jednotlivých tříd a popsána jejich funkce a chování. Celá implementace je zapouzdřena v balíčku `org.fit.vips` a obsahuje také zdrojové kódy testovacího programu `VIPSTester`.

5.4.1 Detekce a extrakce vizuálních bloků

Detekce vizuálních bloků začíná vykreslením dané stránky pomocí jádra CSSBox a následného získání DOM stromu, tak jak je ukázáno na 4.4. Po vykreslení stránky je nezbytné získat informace o vykreslené stránce v objektu typu `ViewPort`, který je popsán v 5.2.1. Samotná detekce probíhá ve třídě `VipsParser`, kde na základě objektu `Viewport` zkonstruujeme stromovou strukturu typu `VipsBlock`, která obsahuje všechny dostupné bloky na stránce a zachovává jejich hierarchii. K nim přidává informace o vlastnostech daných bloků jako jsou například barva pozadí, váha písma či například délka textu obsaženého uvnitř boku. Tato struktura je poté procházena od kořenového prvku a jsou detekovány vizuální bloky. K tomuto účelu byla použita pravidla z tabulky 3.2.1, kdy se podle názvu uzlu daného HTML elementu aplikují určitá pravidla dle tabulky 3.3. V rámci jejich implementace došlo pouze k mírným úpravám z důvodu přizpůsobení na renderovací jádro CSSBox.

Po praktických testech bylo dále zjištěno, že jsou jako vizuální bloky označovány i takové bloky, které nemají žádný obsah či nejsou např. viditelné. Proto, pokud je blok označen jako vizuální, tak jsou na něj dodatečné aplikována pravidla založena na následujících vlastnostech, která zaručí, že je opravdu vizuálním blokem.

- *Pozice prvku na stránce*
  Pokud se souřadnice krajních bodů vizuálního bloku nachází mimo stránku, tak blok nemůže být vizuálním blokem a bude odstraněn. Tato technika (posunutí elementu mimo viditelnou oblast) se často používala ke skrytí elementu uživateli.

- *Rozměry prvku*
  Pouze blok s výškou a šířkou větší než nula můžeme označit jako vizuální.

- *Viditelnost prvku*
  Prvek je viditelný pokud splňuje předešlé podmínky a zároveň nemá vlastnost `display` nastavenu na hodnotu `none` nebo `hidden`.

- *Obsah bloku*
  Blok označíme jako vizuální pokud délka v něm obsaženého textu není nulová. Pokud je nulová, tak procházíme dostupné potomky bloku a podle jejich HTML značek určíme, zda je blok vizuální. V této fázi můžeme již vynechat HTML značky týkající se textu, které jsme eliminovali zjištěním délky obsaženého textu. Proto uvažujeme značky jako `<IMG>` či `<INPUT>`, u kterých také dodatečně ověřujeme jejich viditelnost.

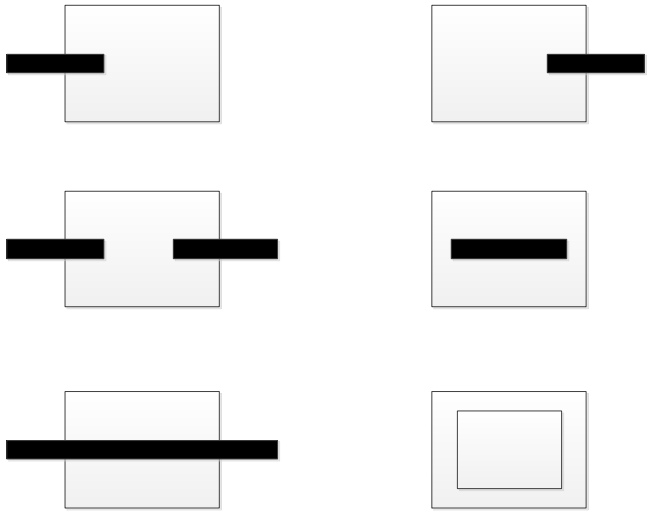
V původní implementaci algoritmu VIPS je tento proces opakován vždy, pokud segmentovaný blok splní či nesplní podmínku další segmentace ($PDoC \leq DoC$). Jak je ukázáno v části 5.4.3, tak v implementaci v jazyce Java nebyl nalezen postup přesného generovaní stupně konzistence pro nově vzniklé bloky v rámci fáze konstrukce vizuální struktury. Z tohoto důvodu nemůže dojít k dodržení stejné podmínky na ukončování a musí vždy dojít

k maximální segmentaci. V první iteraci se detekují bloky větší než 350 pixelů na šířku a 400 pixelů na výšku, čímž získáme hrubou strukturu stránky, která je velmi vhodná pro získání globálních separátorů stránky. V dalších iteraci je vždy z každého rozměru odečteno 50 pixelů a spuštěn proces segmentace znovu. Po dosažení hodnoty 100 px na 150px jsou tyto rozměry nahrazeny hodnotami 100 px na 100 px a poté je dále pokračováno s rozměry 80 na 80, 40 na 10 a na závěr 1 pixel na 1 pixel pro dosažení maximální segmentace.

Výstupem této fáze je stromová struktura `VipsBlock` s informacemi o všech blocích (nejen vizuálních) a také seznam detekovaných vizuálních bloků. Objekt třídy `VipsBlock` v sobě uchovává informace například o barvě pozadí bloku, velikosti použitého písma či počtu obrázků, které se v bloku nacházejí.

5.4.2 Detekce vizuálních separátorů

Z předchozí fáze extrakce vizuálních bloků získáme seznam vizuálních bloků s nímž budeme dále pracovat. Jednotlivě přidáváme vizuální bloky do nového seznamu bloků a aplikujeme pravidla uvedena v 3.2.2. Po detekci separátorů odstraníme ty, které tvoří hranice stránky.



Obrázek 5.2: Vzájemné vztahy separátorů a vizuálních bloků ve fázi detekce vertikálních separátorů

Po praktických pokusech bylo dále přikročeno k pročišťování separátorů na základě jejich šířky. Při první iteraci v rámci algoritmu VIPS je důležité najít takové separátory, které můžeme nazvat globálními a jsou významné pro správnou počáteční segmentaci. Při nesprávné detekci těchto separátorů je vždy výsledkem nesprávná vizuální struktura stránky. Příklad této situace vidíme na obrázcích 5.4a a 5.4d. Na obrázku 5.4a vidíme, že jsou nalezeny horizontální separátory (modrá barva) v pořadí číslo 3 a 4 (z vrchní strany). Proto by v první fázi konstrukce byla stránka rozdělena na šest vizuálních bloků. Pokud jsou však tyto globální separátory pročištěny, tak je stránka správně rozdělena na čtyři horizontální bloky, jak je ukázáno na obrázku 5.4d. Proto v první iteraci je po detekci horizontálních a vertikálních separátorů přistoupeno k jejich pročištění, kdy separátory se šířkou menší nebo rovnou 3 pixelům jsou odstraněny. V dalších iteracích jsou vždy automaticky odstraňovány separátory se šířkou menší než 4 pixely. Tyto separátory malých

rozměrů zpravidla neodděluji vizuálně význačné bloky a mohou být tedy odstraněny. Je také možné, že tyto separátory vznikly nepřesným vykreslením stránky jádrem CSSBox.

Dalším krokem po detekci horizontálních či vertikálních separátorů je přiřazení váhy jednotlivým separátorům podle pravidel uvedených v 5.4.2. Po experimentech bylo zvoleno následující ohodnocení pro jednotlivá pravidla:

1. Pravidlo 1 - šířka separátoru
   Váha byla separátoru zvýšena na základě jeho šířky podle následující stupnice:
   - šířka $\leq$ 8 px $\implies$ váha zvýšena o 1 bod
   - 8 px $<$ šířka $\leq$ 15 px $\implies$ váha zvýšena o 2 body
   - 15 px $<$ šířka $\leq$ 25 px $\implies$ váha zvýšena o 4 body
   - 25 px $<$ šířka $\leq$ 35 px $\implies$ váha zvýšena o 6 bodů
   - 35 px $<$ šířka $\leq$ 45 px $\implies$ váha zvýšena o 8 bodů
   - 45 px $<$ šířka $\leq$ 55 px $\implies$ váha zvýšena o 10 bodů
   - šířka $>$ 55 px $\implies$ váha zvýšena o 12 bodů
2. Pravidlo 2 - překrytí
   Pokud se separátor překrývá s prvkem `<hr>` - horizontální linie, tak je jeho váha zvýšena o 2 body.
3. Pravidlo 3 - barva
   Pokud se barva bloků, které separátor odděluji liší, tak je jeho váha zvýšena o 2 body. Pokud je na obou stranách separátoru více bloků, tak se aplikuje pouze jednou.
4. Pravidlo 4 - písmo
   Pokud se liší váha či velikost písma bloků, které separátor odděluje, tak je jeho váha zvýšena o 2 body. Navíc pokud je výška písma bloků na vrchní straně separátoru, tak je váha znovu navýšena o další 2 body.
5. Pravidlo 5 - struktura
   Pokud jsou bloky na obou stranách separátoru velmi podobné (například se jedná o text), tak je váha separátoru snížena o 2 body.

Každý nově vytvořený separátor má nastavenou váhu na hodnotu tří bodů. Maximální možná váha, kterou může horizontální separátor získat je tedy 21 bodů a vertikální 19 bodů. Naopak minimální váha může být 1 bod pro horizontální separátor a 3 body pro vertikální separátor a to z toho důvodu, že pravidlo číslo 5 se vztahuje pouze na horizontální separátory.

Musíme zde zdůraznit, že výše uvedený počet bodů v jednotlivých pravidlech, o který se má váha separátoru zvětšit je založený na vlastních experimentech. V technické zprávě [24] není uvedena jediná zmínka o počtu těchto bodů. Proto není jisté, zda výše zvolené hodnoty jsou zvoleny správně. Dalším problémem s tím spojeným je otázka, zda u pravidla číslo 3 inkrementovat hodnotu pro každou dvojici bloků ze všech bloků přiléhajících k separátoru či nikoliv.

5.4.3 Konstrukce vizuální struktury

Fáze konstrukce vizuální struktury slouží ke sloučení vizuálních bloků stránky spolu se separátory, které k nim byly detekovány. Výstupem této fáze je finální vizuální struktura stránky, jenž obsahuje informace o rozložení prvků na stránce a jejich uskupení do vizuálních bloků.

Konstrukce vizuální struktury se skládá z několika dílčích kroků. Prvním z nich je získání bloků z fáze detekce a extrakce vizuálních bloků, pro které je poté spuštěna detekce horizontálních separátorů. Nalezené separátory jsou seřazeny podle získané váhy od nejnižší k nejvyšší. Pokud se jedná o první iteraci, tak je vytvořena nová vizuální struktura, která reprezentuje celou stránku a je jí přiřazen identifikátor 1. Poté přichází samotná konstrukce vizuální struktury, kdy zpracováváme separátory v pořadí od toho s nejnižší vahou. Po vyjmutí prvního separátoru je daná vizuální struktura rozdělena na dvě, které tento separátor odděluje. Poté se vezmou všechny vizuální bloky, jenž se nachází v dané struktuře a jsou podle svých souřadnic rozděleny do struktur kam svou polohou náleží. Po vyjmutí následujícího separátoru je podle souřadnic separátoru nalezena taková struktura, uvnitř které se separátor nachází. Tato struktura je tedy dále rozdělena na dvě nové a tento proces probíhá dokud nevyjmeme separátor s nejvyšší vahou. Poté je proces dělení zastaven a nově vzniklým strukturám je přiřazen identifikátor např. 1-3-2, který odráží jejich hierarchické zanoření. V dalším kroku jsou všechny horizontální separátory, které byly použity ke konstrukci vizuální struktury vloženy do původní segmentované struktury.

Po ukončení konstrukce vizuální struktury s horizontálními separátory nastává fáze přiřazení stupně konzistence DoC nově vzniklým blokům v rámci vizuální struktury. Stupeň konzistence je přiřazen bloku na základě váhy separátoru s nejvyšší vahou, který se nachází v oblasti vizuálního bloku. Proto je třeba správně převést hodnoty vah na odpovídající stupeň konzistence. Tato transformace je realizována po ukončení poslední fáze extrakce vizuální bloků a je provedena nad všemi detekovanými separátory na celé stránce. Prvním krokem k získání hodnoty DoC je lineární transformace hodnot vah separátorů z uvedeného seznamu pomocí normalizace metodou Max-Min[27]. Rovnice lineární normalizace je definována následovně:

$$v' = \frac{v - min_S}{max_S - min_S}(new\_max_S - new\_min_S) + new\_min_S \text{ , kde} \tag{5.1}$$

- v' je normalizovaná váha separátoru
- v je váha separátoru před normalizací
- max_S je váha separátoru s největší vahou
- min_S je nejmenší váha mezi separátory
- new_max_S je nejvyšší hodnota v rozsahu, na který chceme váhy normalizovat (v našem případě 11)
- new_min_S je minimální hodnota v v rozsahu, na který chceme váhy normalizovat (v našem případě 1).

Po získání normalizované váhy je nutný její převod na odpovídající stupeň konzistence. Musíme si zde uvědomit, jak je DoC definováno. Stupeň konzistence nám říká jak je daný

blok konzistentní, a proto pokud máme separátor s nízkou vahou, tak velmi těsně odděluje dva bloky a tudíž jeho stupeň konzistence musí být u horní hranice a to 11 bodů (maxDoC). Naopak pokud máme dva od sebe velmi vzdálené bloky, tak musí být výsledné DoC u dolní hranice tj. 1 bodu. Funkce pro získání DoC má předpis:

$$f : DoC = (maxDoC + 1) - normalizovaná\ váha \wedge f(0) = maxDoC \tag{5.2}$$

Takto získané stupně konzistence byly přiřazeny jednotlivým vizuálním blokům. Porovnáním s originální implementací algoritmu VIPS bylo zjištěno, že hodnoty vypočteného DoC neodpovídají požadovaným hodnotám. Proto byl k separátorům přiřazen jeden nový s vahou dvojnásobku maximální dosažitelné váhy (tudíž 42 bodů), který reprezentoval celou stránku. Vychází se zde z předpokladu, že stránka jako celek má stupeň konzistence roven jedné. Dále byla také vyzkoušena metoda Soft-Max[27] (nelineární transformace), která ale také nevedla k uspokojivým výsledkům. Proto byla ponechána lineární transformace pomocí metody Max-Min.

Nyní jsme již získali stupně konzistence, které zhruba odpovídají hodnotám originální implementace, jenž se liší v průměru o 2 body na obě strany. Zde je vhodné zmínit, že v převážné většině případů nezískáme shodné výsledky s originální implementací. Jelikož je stupeň konzistence novému bloku přiřazován na základě nejvyšší váhy separátoru ze všech separátorů dostupných v oblasti bloku, tak už v tomto kroku může dojít k chybě způsobené úpravou váhy separátorů popsanou v části 5.4.2. Dále mohlo dojít k chybě při volbě metody, jak převést váhu separátoru na odpovídající stupeň konzistence. V technické zprávě není nikde zmíněna technika převodu mezi těmito dvěma hodnotami. Může to tedy být mnou zvolená metoda lineární transformace Max-Min či úplně jiná transformace. Jestliže se opodstatnění předchozích domněnek dá jen těžko ověřit, tak poslední problém se dá zcela fakticky ukázat. Tímto problémem je rozdíl ve vykreslení stránky jádrem CSSBox a jádrem Trident, se kterým pracuje originální implementace algoritmu VIPS. Můžeme zde ukázat takový příklad na obrázku 5.3.

Jak vidíme na obrázku 5.3a, tak jádro Trident nemá s vykreslením žádné problémy. Naproti tomu jádro CSSBox nesprávně vykreslilo horizontální mezery mezi jednotlivými obrázky a dále emailovou adresu vykreslilo za tyto obrázky. Poznamenejme, že text „E-Mail:" není viditelný, protože barva pozadí - bílá je shodná s barvou použitého písma.

Po přiřazení stupně konzistence novým vizuálním blokům je na jejich základě rozhodnuto, zda dojde k dalšímu dělení či nikoliv. Všechny vizuální bloky, které mají stupeň konzistence DoC menší nebo roven povolenému stupni konzistence PDoC, budou dále segmentovány. Ostatní nikoliv.

Po ukončení konstrukce vizuální struktury s horizontálními separátory jsou vyhledány všechny listové struktury ve stromové vizuální struktuře. Pro tyto uzly je opakován výše uvedený proces, ale s vertikálními separátory.

(a) jádro Trident (b) jádro CSSBox

Obrázek 5.3: Ukázka rozdílu ve vykreslení jádry Trident a CSSBox

5.4.4 Výstup

Výstupem algoritmu VIPS je XML soubor jehož struktura je popsána v části 5.1.2. Drobnou změnou oproti originálnímu výstupu jsou hodnoty atributů uzlu `VIPSPage` a to `WindowHeight` a `WindowWidth`. Hodnoty těchto atributů jsou v mojí implementaci shodné s hodnotami `PageRectHeight` a `PageRectWidth`, protože aplikace nevyužívá žádný grafický prvek, do kterého je stránka vykreslena, a proto zde nemohou být uvedeny jeho rozměry.

Ve třídě `VipsOutput`, která řeší převod finální vizuální struktury stránky na výstupní XML soubor je implementována logika zastavení segmentace při nesplnění podmínky (PDoC $\leq$ DoC). Použití právě zde plyne ze způsobu vlastní implementace, kdy je stránka segmentována tak, jak nejvíce to lze a poté jsou jednotlivým vizuálním strukturám přiděleny vypočtené stupně konzistence. Proto je zde při procházení stromové struktury, která reprezentuje finální vizuální dělení stránky implementována tato podmínka zastavení, kdy pokud daná vizuální struktura má stupeň konzistence DoC větší než povolený stupeň konzistence PDoC, tak je spolu se všemi jejími potomky spojena do jednoho elementu `LayoutNode`. Naopak pokud má být dále segmentována, tak je pro ni vytvořen samostatný element a pro všechny její synovské vizuální struktury jsou aplikována ty samá pravidla, až je vyexportována celá vizuální struktura.

Dále byla implementována volba pro zapnutí grafického výstupu knihovny. Pokud je tato možnost zapnuta, tak jsou na výstup generovány obrázky ve formátu PNG[5], jejichž přehled najdeme v tabulce 5.3. Detekované vizuální bloky mají černou barvu, horizontální separátory jsou znázorněny modrou barvou a vertikální jsou vyplněny červeně, jak můžeme vidět na obrázku 5.4.

[5]Portable Network Graphics - http://www.libpng.org/pub/png/

Název souboru + ".png"	Obsah
"page"	Snímek stránky vyrenderovaný jádrem CSSBox.
"blocks" + iterace	Detekované vizuální bloky v dané iteraci.
"horizontalSeparators"	Globální horizontální separátory (separátory detekované v první iteraci). Může obsahovat i separátory, jejichž šířka je menší než 10 pixelů, které jsou poté ignorovány.
"horizontalSeparators" + iterace	Horizontální separátory detekované v dané iteraci algoritmu VIPS.
"verticalSeparators"	Globální vertikální separátory (separátory detekované v první iteraci). Může obsahovat i separátory, jejichž šířka je menší než 10 pixelů, které jsou poté ignorovány.
"verticalSeparators" + iterace	Vertikální separátory detekované v dané iteraci algoritmu VIPS.
"iteration" + iterace	Separátory a vizuální bloky použité v dané iteraci. Skládá se z kombinace blocksX.png + horizontalSeparatorsX + verticalSeparatorsX, kde X udává číslo iterace.
"all"	Separátory a vizuální bloky detekované před první iterací. Skládá se z kombinace blocks1.png + horizontalSeparators + verticalSeparators.

Tabulka 5.3: Názvy grafických výstupů implementace VIPS v jazyce Java

Musíme zde však upozornit, že povolení grafického výstupu navýší časovou náročnost algoritmu VIPS, jak můžeme vidět na grafu 6.7.

5.4.5 API

Application Programming Interface (API) označuje rozhraní pro programování aplikací. Jedná se o množinu funkcí přístupných programátorovi s jejichž pomocí, může ovlivňovat chování a vlastnosti objektů či je využívat ve vlastních aplikacích.

Přístup k implementaci algoritmu VIPS v jazyce Java probíhá přes objekt třídy `Vips`, který obsahuje následující veřejné metody, které tvoří API implementace algoritmu VIPS.

- `void setUrl(String url)`
  Nastaví internetovou adresu stránky, která má být segmentována. Pokud adresa neobsahuje prefix "`http://`"nebo "`https://`", tak je automaticky doplněn první jmenovaný.
- `String getUrl()`
  Navrátí nastavenou internetovou adresu.
- `void setPermittedDoC(int pDoC)`
  Nastaví hodnotu povoleného stupně konzistence na hodnotu danou parametrem `pDoC`. Výchozí hodnota je nastavena na 5.
- `int getPredefinedDoC()`
  Vrátí povolený stupeň konzistence pDoC.

- `void startSegmentation(String url)`
  Spustí segmentační proces nad danou internetovou stránkou danou hodnotou parametru `url`.

- `void startSegmentation()`
  Spustí segmentační proces nad stránkou jejíž adresa musí být nastavena pomocí metody `setUrl`. Pokud není nadefinována, tak je použita adresa poslední segmentované stránky.

- `void enableGraphicsOutput(boolean enable)`
  Povolí či zakáže grafický výstup aplikace. Viz. 5.4.4.

- `void enableOutputToFolder(boolean enable)`
  Zapne či vypne tvorbu samostatné složky pro výstupní soubory pro každý běh algoritmu VIPS. Jméno nové složky je vždy v následujícím tvaru `den_měsíc_rok_hodina_minuta_adresa` ,kde adresa je host dané stránky v němž je znak „.” nahrazen znakem „_”. Příklad: `15_05_2012_12_57_51_www_fit_vutbr_cz`.

- `void enableOutputEscaping(boolean enable)`
  Povolí nebo zakáže kódování znaků[6] ve výstupním XML.

- `void setOutputFileName(String filename)`
  Nastaví jméno výstupního XML souboru. Zadává se bez koncovky ”.xml”. Výchozí název výstupního souboru je ”VIPSResult.xml”.

Základní použití API můžeme vidět ve třídě `VipsTester`, které je jako parametr předána adresa internetové stránky, jenž chceme segmentovat. Níže je uvedena část kódu, ze které si můžeme udělat představu o posloupnosti volání funkcí z API implementace VIPS v jazyce Java vedoucí k segmentaci dané stránky a uložení výstupního XML souboru.

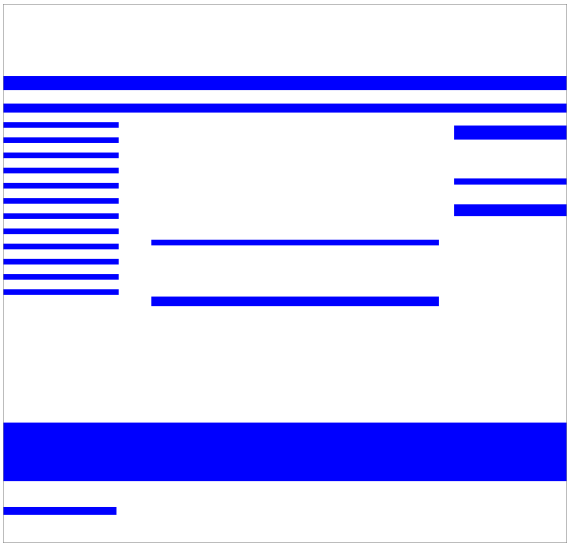
```
1 String url = args[0];
2 Vips vips = new Vips();
3 vips.enableGraphicsOutput(true);
4 vips.enableOutputToFolder(true);
5 vips.setPredefinedDoC(5);
6 vips.startSegmentation(url);
```

Zdrojový kód 5.4: Ukázka použití API v testovací aplikaci `VipsTester`

[6] Kódování znaků - https://en.wikipedia.org/wiki/Escape_character

(a) all.png

(b) page.png



(c) horizontalSeparator3.png

(d) iteration3.png

Obrázek 5.4: Ukázky grafického výstupu pro stránku `www.fit.vutbr.cz`

Kapitola 6

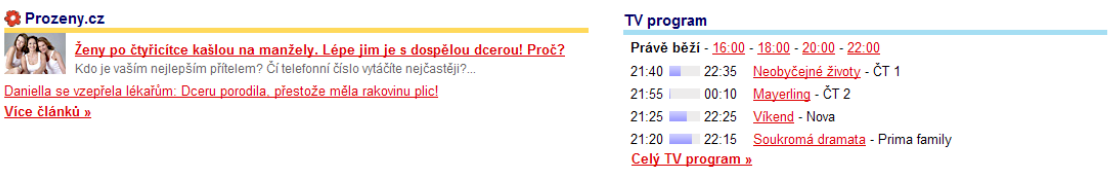
Dosažené výsledky

6.1 Demonstrace řešení

V následující sekci je provedeno srovnání výstupu původní implementace algoritmu Visio-based Page Segmentation a jeho implementace v jazyce Java, která je výstupem této práce. Pro získání XML výstupu původní implementace byla použita aplikace UsingVIPS. Finální struktura byla poté přenesena na stránku vykreslenou jádrem CSSBox. Všechny zde uvedené obrázky jsou v originální velikosti přiloženy na DVD spolu s vygenerovanými XML výstupy.

6.1.1 Ukázka č. 1

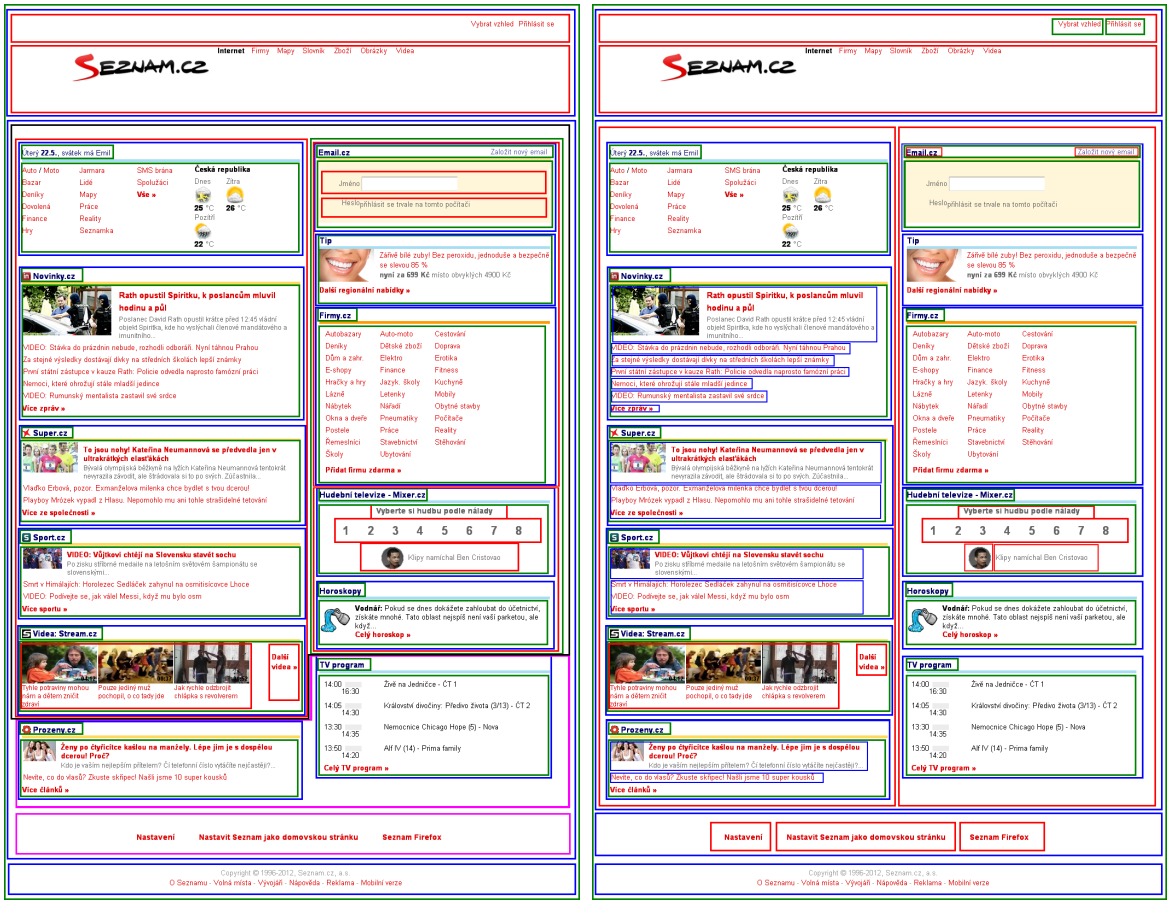
Jako první ukázku jsem zvolil vizuální segmentaci internetového portálu Seznam - `www.seznam.cz`. Při segmentaci byl oběma implementacím nastaven povolený stupeň konzistence na hodnotu 8. Jak vidíme na obrázku 6.2 tak oba výstupy jsou si velmi podobné. Při bližším zkoumání na několik odlišností narazíme. Pokud půjdeme od horního okraje stránky, tak zjistíme, že v první blok byl oproti originální implementaci dále segmentován na dva bloky s obsahem „Vybrat vzhled" a „Přihlásit se". Pokud přejdeme k bloku s přihlášením, tak i zde vidíme, že vrchní blok byl opět segmentován. Byl dělen z toho důvodu, že mezi bloky obsahujícími „Email.cz" a „Založit nový email" byl detekován vertikální separátor, na jehož základě bylo nově vzniklému bloku přiřazeno nižší DoC než 8. Proto byl při exportu do XML dále segmentován. Další změnou bylo nerozdělení přihlašovacího formuláře, kdy tomuto bloku bylo přiřazeno vyšší DoC než 8, a proto nebyl dále segmentován. Další dělení oproti výstupu originální implementace proběhlo v levém prostředním sloupci. Jednotlivé bloky (Novinky, Sport) byly dále děleny na jednotlivé zprávy, jenž jsou označeny modrým hraničením. Tyto bloky byly segmentovány z důvodu přiřazení menšího DoC než PDoC.



Obrázek 6.1: Vykreslení bloků „Prozeny.cz" a „TV program"

V originální implementaci byly od ostatních bloků separovány bloky „TV program" a „Prozeny.cz", které přešly do vlastního samostatného bloku. Pro pochopení tohoto oddělení

od zbytku si musíme ukázat, jak danou část vykreslilo jádro Trident, což je ukázáno v obrázku 6.1. Jak vidíme tak tyto dva bloky byly vykresleny v jedné rovině na rozdíl od enginu CSSBox, a proto mohl být v první iteraci detekován horizontální separátor přes celou šířku stránky. Poté by došlo ke stejné správné segmentaci jako v případě původní implementace využívající jádro Trident. Posledním rozdílem bylo zařazení druhého bloku od dolního okraje (s obsahem „Nastavení … Seznam Firefox") jako samostatného bloku stránky (nadřazený blok je pouze celá stránka). V původní implementaci je tento blok součástí jiného bloku.



(a) Originální implementace

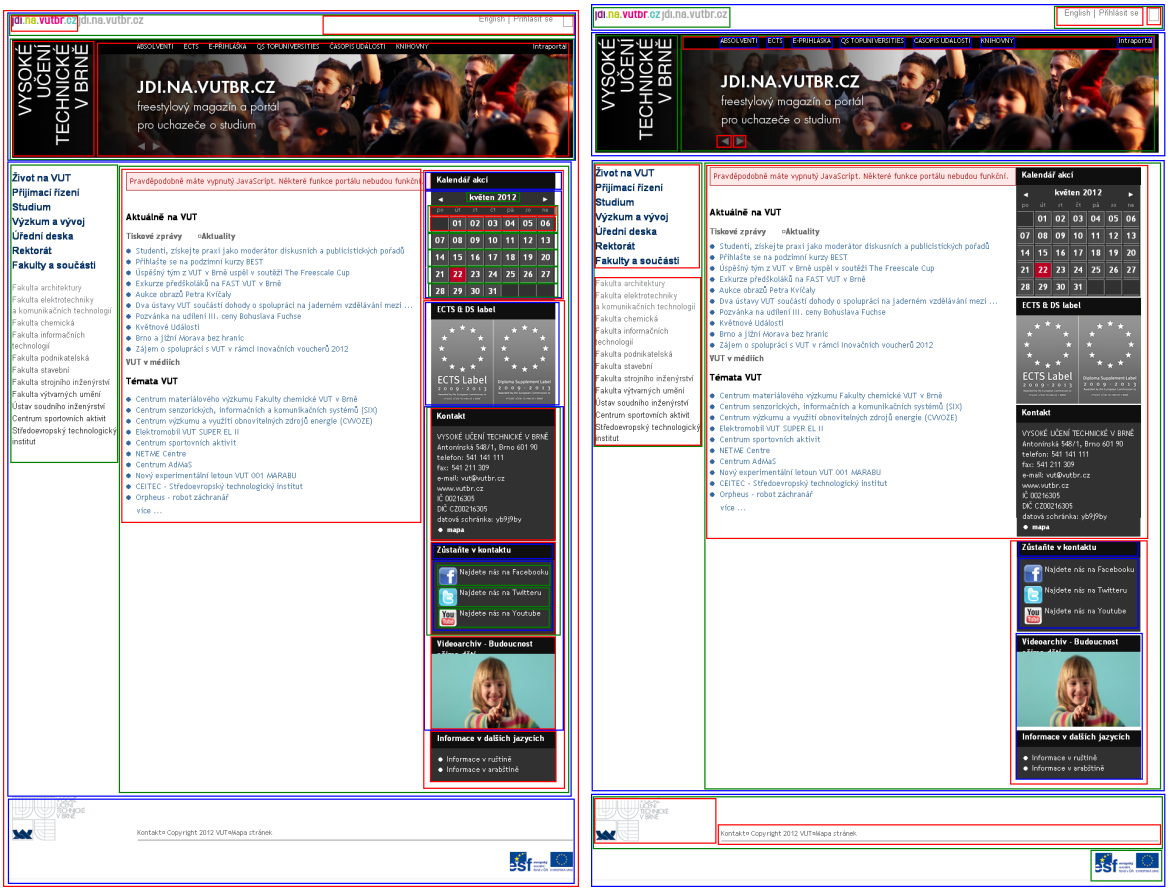
(b) Implementace v jazyce Java

Obrázek 6.2: Výsledná vizuální struktura pro stránku `www.seznam.cz`

6.1.2 Ukázka č. 2

Na obrázku 6.3 je ukázána výsledná segmentace stránky `www.vutbr.cz` pro PDoC rovno 8. V prvním bloku od vrchního okraje stránky vidíme dvakrát nápis „jdi.na.vutbr.cz". Barevný text je součástí stránky, ale šedý nápis byl do stránky přidán při renderování jádrem CSSBox. Ve stejném bloku je oproti originální implementaci rozdělen levý blok na dvě části. První obsahuje odkazy a druhý pole pro vyhledávání, které ale není korektně vykresleno. V pořadí druhý blok je v obou implementacích rozdělen správně, ale jen v implementaci v jazyce Java je horní menu odděleno a dále segmentováno na jednotlivé položky. Pokud se

dále podíváme na menu na levé straně, tak je dle mého názoru správné dělení na dvě části jako v implementaci VIPS v jazyce Java. Velkým rozdílem mezi oběma výstupy je dělení bloků vedle výše zmíněného menu. V originální implementaci je tento velký blok rozdělen na dva vertikální, z nichž levý je dále segmentován. Na výstupu z implementace v jazyce Java není tento blok správně rozdělen a to z důvodu chybného vykreslení stránky jádrem CSSBox, jak vidíme na obrázku 6.4.



(a) Originální implementace

(b) Implementace v jazyce Java

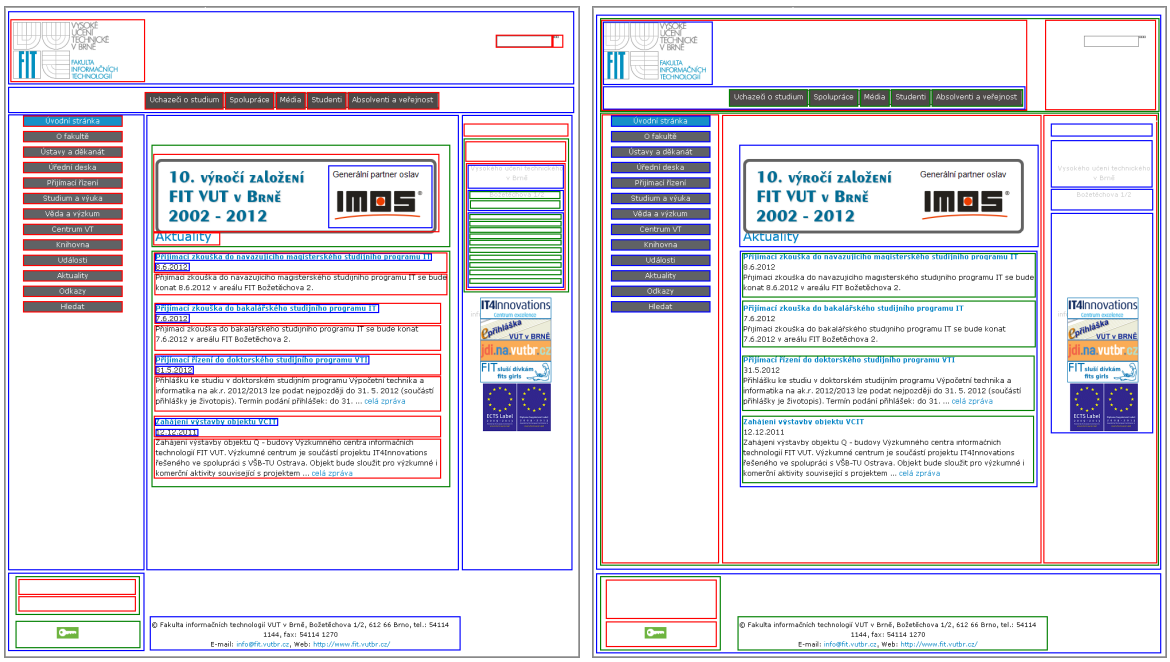
Obrázek 6.3: Výsledná vizuální struktura pro stránku `www.vutbr.cz`

Jak je z obrázku vidět, tak při vykreslení jádrem CSSBox není pravý okraj kalendáře zarovnán s pravým okrajem obrázku nad ním. Z toho důvodu není správně vykreslena mezera mezi kalendářem a prvkem vlevo. Proto nedokáže má implementace detekovat vertikální separátor, jako v případě správně nalezeného separátoru mezi blokem s menu a obsahem s aktualitami a tématy. Pokud by tato část stránky byla vykreslena správně, pak by se segmentace těchto částí shodovala pro obě implementace. Posledním rozdílem v rámci srovnání jednotlivých výstupů je segmentace nejspodnějšího bloku, tak jak je ukázáno na obrázku vpravo. Dle mého názoru by měl tento blok být segmentován i v původní implementaci.

(a) Jádro Trident (b) Jádro CSSBox

Obrázek 6.4: Porovnání vykreslené části stránky `www.vutbr.cz`

6.1.3 Ukázka č. 3



(a) Originální implementace (b) Implementace v jazyce Java

Obrázek 6.5: Výsledná vizuální struktura pro stránku `www.fit.vutbr.cz`

Jako poslední ukázku jsem zvolil segmentaci fakultního portálu `www.fit.vutbr.cz` s nastaveným povoleným stupněm konzistence na 11 bodů. Zde si na úrovni celé stránky můžeme všimnout neshody v dělení. Zatímco v původní implementaci je stránka segmentována odděleně na jednotlivé bloky, tak v mé implementaci je stránka v dělena na tři hlavní horizontální bloky, které jsou pak dále segmentovány. Vidíme zde také, že pravý vertikální blok s kontaktními informacemi je v původní implementaci rozdělen řádek po řádku na rozdíl od implementace v jazyce Java, který jí dělí po skupinách. Zde také dochází k chybě, která byla popsána v 5.3, jejíž vinou zde není správně detekován horizontální separátor a nedochází k oddělení skupinky obrázků do samostatného bloku. Prostřední blok je v původní implementaci velmi jemně segmentován, ale naproti tomu v mé implementaci nedochází k tak

podrobnému dělení. V horním bloku je oproti výstupu originální implementace detekován prvně vertikální separátor (kvůli zvolenému prahu velikosti bloku tvoří ve chvíli detekce separátoru menu a logo jeden blok) a proto dochází k jiné segmentaci bloku.

6.2 Výkonnost

Důležitým měřítkem kvality implementace algoritmu VIPS jakožto výstupu této práce je porovnání s původní implementací. Měření probíhalo v čisté instalaci systému Microsoft Windows XP, která byla virtualizována programem VirtualBox. Virtualizace byla zvolena ze dvou důvodů a to z nedostupnosti instalace systému Windows XP na fyzickém stroji a za druhé se na méně výkonném hardware velmi dobře projevuje kvalita implementace. Virtualizovaný operační systém byl hostován na notebooku Lenovo T61 s procesorem Intel Core2Duo T9300 o frekvenci 2500 MHz, ze kterého bylo virtuálnímu stroji přiřazeno jedno výpočetní jádro. Dále systému byl přidělen 1 GB operační paměti. Na fyzickém počítači byla nainstalována 64-bitová distribuce Fedora 16 s linuxovým jádrem 3.3 a Java SE verze 1.6.0_31.

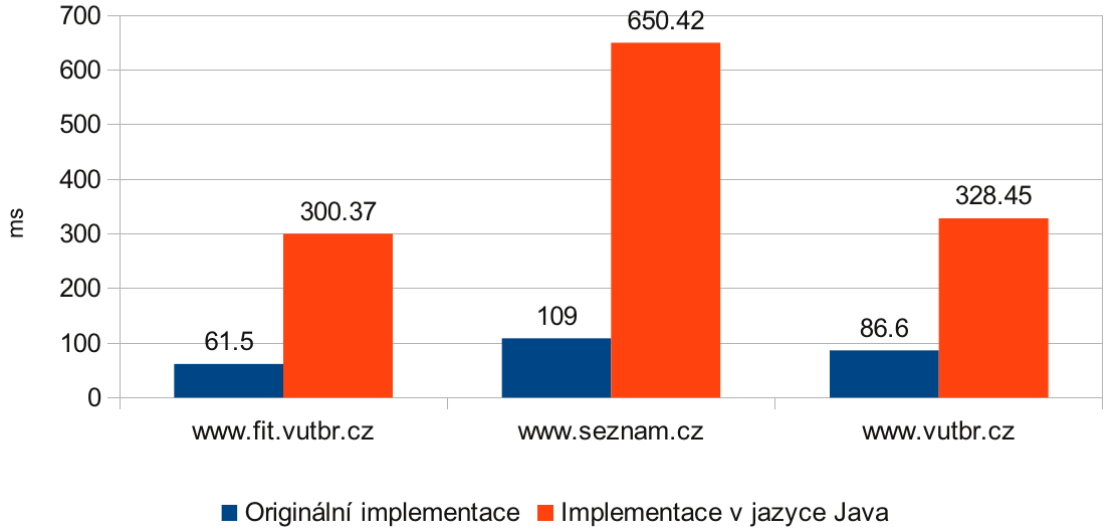
Vstupním bodem měření pro implementaci v jazyce Java bylo zpracování stránky pomocí objektu třídy `DOMAnalyzer`, před získáním struktury `Viewport`. Měření bylo ukončeno po uložení výstupního XML. Čas je měřen pomocí funkce `nanoTime()`[1] ze třídy `System`, která navrátí hodnotu nejpřesnějšího časovače v systému v nanosekundách. Odečtením těchto naměřených hodnot získáme čas potřebný k vykonání segmentace s PDoC 10. Segmentace byla spouštěna přes spustitelný JAR soubor, kterému se předávala adresa stránky. V aplikaci PageAnalysis, jenž je určena k demonstraci originální implementace byl časovač aktivován po načtení segmentované stránky v prohlížeči a stisknutí tlačítka pro zahájení procesu segmentace. Časovač je také ukončen po uložení výstupního XML. Pro umožnění měření času v nástroji PageAnalysis bylo třeba přidat do zdrojových kódů aplikace volání funkce `GetTickCount()`[2] zdrojové kódy aplikace a znovu ji přeložit. Tato funkce vrací čas od spuštění operačního systému v milisekundách.

Měření probíhalo na stránkách jejichž segmentace byla ukázána v předchozí sekci. Vzorky v rámci měření byly získány desetkrát a poté z nich byl spočten průměr.

Jak vidíme v prvním grafu, tak originální implementace je v průměru pětkrát rychlejší, než implementace v jazyce Java. Rychlost segmentace vlastního řešení je ovlivněna především chybami spojenými s určením hodnoty stupně konzistence popsanými v 5.4.1 a 5.4.3, kdy k jejich potlačení je zvolen proces segmentace v deseti krocích oproti velmi malému počtu v původní implementaci, která při přiřazení správné hodnoty DoC dokáže segmentační proces ukončit například již v první iteraci. Tímto došlo k významnému nárostu času potřebného k dosažení obdobné finální vizuální struktury. Dalším aspektem by mohla být výkonnost Javy v porovnání s jazykem Visual C++, kdy druhý jmenovaný je výkonnější a využívá také toho, že běží na platformě Windows nativně a ne přes dodatečný runtime jako Java.

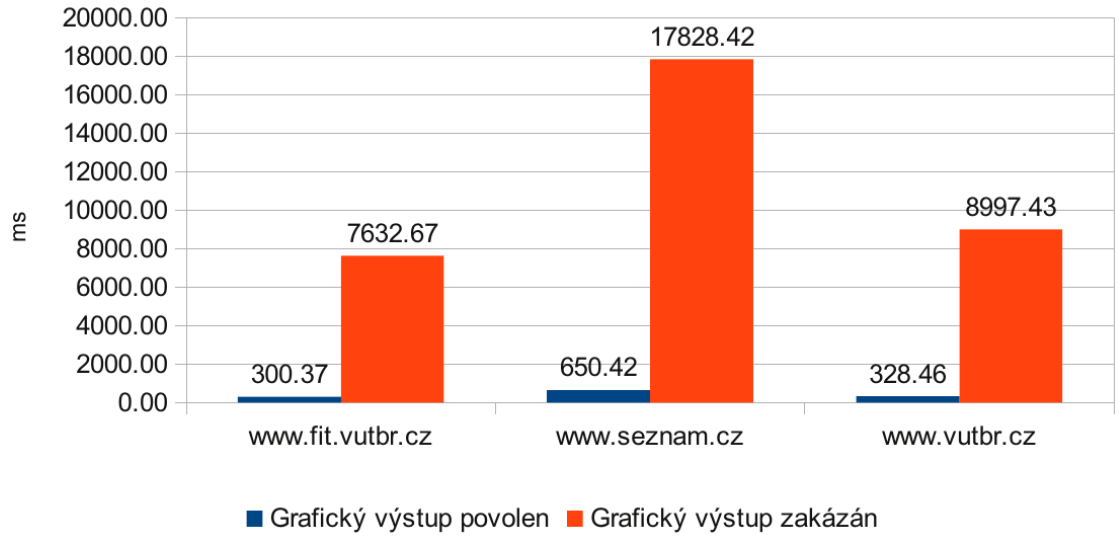
[1] Funkce nanoTime() - http://docs.oracle.com/javase/6/docs/api/java/lang/System.html#nanoTime()

[2] Funkce GetTickCount() - http://msdn.microsoft.com/en-us/library/windows/desktop/ms724408(v=vs.85).aspx



Obrázek 6.6: Porovnání časové náročnosti implementací algoritmu VIPS

Na obrázku 6.7 vidíme porovnání časové náročnosti při zapnuté volbě dodatečného grafického výstupu. Vidíme, že dojde k mnohonásobnému zdržení, které roste se složitosti dané stránky (počtu vizuálních bloků, separátorů). Při jednom běhu aplikace je generováno až 44 různých grafických výstupů.



Obrázek 6.7: Porovnání časové náročnosti grafického výstupu

Kapitola 7

Závěr

V rámci diplomové práce byla představena metoda vizuální segmentace internetových stránek Vision-based Page Segmentation - VIPS. Cílem této techniky je dělení stránky na základě vizuálních vlastností tak, jak by danou stránkou vnímal člověk. Vymezeny byly základní pojmy důležité pro porozumění rozebírané tématiky jako (předdefinovaný) stupeň konzistence, vizuální blok či separátor. Proces dělení touto metodou byl ukázán ve srovnání s dalšími třemi segmentačními metodami. Spolu s metodou Combined Page Segmentation - CombPS, jenž se skládá i z aplikace metody VIPS, dosahuje nejlepších výsledků.

Technika vizuálního dělení stránky pomocí metody VIPS musí znát ke svému provedení Document Object Model stránky. Proto je nutná spolupráce s vykreslovacím jádrem internetových stránek. V této práci jsme si představili čtyři nejznámější dostupná jádra pro použití v programovacím jazyce Java.

Cílem této diplomové práce byla implementace algoritmu Vision-based Page Segmentation v programovacím jazyce Java. Tento algoritmus byl vyvinut v laboratořích firmy Microsoft v roce 2003. Pro implementaci bylo zvoleno jako renderovací jádro WWW stránek engine CSSBox, jenž sice nepodporuje moderní technologie jako HTML 5 či CSS 3, ale na rozdíl od jiných, je již uzpůsoben na aplikaci segmentačních či analytických technik. Návrh a postup implementace vycházel z dostupné technické zprávy.

Při implementaci se vyskytly problémy při vývoji fáze realizující konstrukci vizuální struktury. Přesněji se jedná o krok přidělování stupně konzistence nově vzniklým blokům. Technické zpráva zmiňuje, že dochází k určení stupně konzistence na základě váhy separátorů, které slučované bloky oddělují. Technika této transformace ale není ve zprávě zmíněna. Proto byly provedeny experimenty s lineárními - normalizace Max-Min a nelineárními transformacemi - normalizace Soft-Max či ohodnocením na základě určitého intervalu hodnot. Nejlepších výsledků dosahovala metoda Max-Min s dodatečnou modifikací vstupních dat. Při aplikování získaných hodnot se ohodnocení stupňů konzistence lišilo oběma směry od originální implementace. Důsledkem je v některých případech odlišný stupeň segmentace ve srovnání s původní implementací. Proto byl zvolen jiný postup segmentace, kdy je stránka rozdělena na co nejmenší bloky a poté je nad všemi dostupnými separátory aplikována normalizace Max-Min. Tato modifikace snížila výkonnost implementace v jazyce Java, ale zvýšila úspěšnost správného dělení. Původní implementace dokáže zastavit segmentační proces, pokud je splněna podmínka týkající se konzistence dané bloku a tím snížit časovou náročnost algoritmu. Jako důsledek zavedené modifikace není možné segmentační proces dříve ukončit.

K zabránění ve správné segmentaci stránky také docházelo ze strany jádra CSSBox, kdy některé části různých stránek vykreslovalo odlišně od jádra Trident, jenž používá původní

knihovna. Na obhajobu jádra CSSBox musím říci, že o jeho vývoj se stará pouze jeden vývojář, ale i přesto si udržuje vysokou kvalitu výstupu.

Pro testování byla využita originální implementace. Testování probíhalo v systému Microsoft Windows XP. Implementace v jazyce Java dosahovala dle mého názoru uspokojivých a srovnatelných výsledků s původní knihovnou. Pokud se podíváme na výkonnost daného řešení, tak je čtyřikrát až šestkrát pomalejší. Nutno zmínit, že potřebný čas k segmentaci dané stránky je v průměru kolem 110 milisekund pro původní implementaci.

Literatura

[1] BSD License. [online], [cit. 2011-12-30].
URL http://www.opensource.org/licenses/bsd-license.php

[2] Cobra: Java HTML Parser. [online], [cit. 2011-1-10].
URL http://lobobrowser.org/cobra/java-html-parser.jsp

[3] Cobra: Pure Java HTML Renderer & Parser (Open Source). [online], [cit. 2011-12-30].
URL http://lobobrowser.org/cobra.jsp

[4] Eclipse Public License. [online], [cit. 2011-12-30].
URL http://eclipse.org/legal/epl-v10.html

[5] Embedding FAQ - MDN. [online], [cit. 2011-1-9].
URL https://developer.mozilla.org/en/Embedding_FAQ

[6] Gecko - Mozilla Developer Network. [online], [cit. 2011-1-7].
URL https://developer.mozilla.org/en/Gecko

[7] Gecko (layout engine) - Wikipedia, the free encyclopedia. [online], [cit. 2011-1-7].
URL http://en.wikipedia.org/wiki/Gecko_(layout_engine)

[8] GNU General Public License. [online], [cit. 2011-12-30].
URL http://www.gnu.org/licenses/gpl.html

[9] GNU Lesser General Public License. [online], [cit. 2011-12-30].
URL http://www.gnu.org/licenses/lgpl.html

[10] JavaXPCOM - MDN. [online], [cit. 2011-1-9].
URL https://developer.mozilla.org/en/JavaXPCOM

[11] Mozilla Public License. [online], [cit. 2011-12-30].
URL http://www.mozilla.org/MPL

[12] Snippet308.java. [online], [cit. 2011-1-9].
URL http://dev.eclipse.org/viewcvs/viewvc.cgi/org.eclipse.swt.snippets/src/org/eclipse/swt/snippets/Snippet308.java?view=co

[13] StatCounter Global Stats. [online], [cit. 2011-1-7].
URL http://gs.statcounter.com/#browser_version-ww-monthly-201201-201201-bar

[14] The SWT FAQ - How do I explicitly use Mozilla as the Browser's underlying renderer? [online], [cit. 2011-12-30].
URL http://www.eclipse.org/swt/faq.php#howusemozilla

[15] The SWT FAQ - How do I explicitly use WebKit as the Browser's underlying renderer? [online], [cit. 2011-12-30].
URL http://www.eclipse.org/swt/faq.php#howusewebkit

[16] SWT: The Standard Widget Toolkit. [online], [cit. 2011-12-30].
URL http://eclipse.org/swt

[17] Web browser engine - Wikipedia, the free encyclopedia. [online], [cit. 2011-1-7].
URL http://en.wikipedia.org/wiki/Web_browser_engine

[18] WebKit - Wikipedia, the free encyclopedia. [online], [cit. 2011-1-7].
URL http://en.wikipedia.org/wiki/WebKit

[19] The WebKit Open Source Project. [online], [cit. 2011-1-7].
URL http://www.webkit.org/

[20] Burget, R.: CSSBox - Java HTML rendering engine. 2011, [online], [cit. 2011-12-30].
URL http://cssbox.sourceforge.net/

[21] Callan, J. P.: Passage-Level Evidence in Document Retrieval. 1994, [online], [cit. 2012-1-8].
URL www.cs.cmu.edu/~callan/Papers/callan794.ps.gz

[22] Deng, C.; Shipeng, Y.; Ji-Rong, W.; aj.: VIPS: a Vision-based Page Segmentation Algorithm. 2003, [online], [cit. 2012-4-26].
URL https://research.microsoft.com/pubs/70027/tr-2003-79.pdf

[23] Deng, C.; Shipeng, Y.; Ji-Rong, W.; aj.: Block-based Web Search. 2004, [online], [cit. 2011-12-27].
URL http://research.microsoft.com/pubs/69113/21.pdf

[24] Deng, C.; Shipeng, Y.; Ji-Rong, W.; aj.: VIPS: a Vision-based Page Segmentation Algorithm. 2004, [online], [cit. 2011-12-26].
URL www.zjucadcg.cn/dengcai/VIPS/VIPS_July-2004.pdf

[25] Khasawneh, N.; Samarah, O.; Al-Omari, S.; aj.: Vision-based Presentation Modeling of Web Applications: A Reverse Engineering Approach. *Journal of Emerging Technologies in Web Intelligence*, ročník 4, č. 2, 2012, [online], [cit. 2012-5-13].
URL http://ojs.academypublisher.com/index.php/jetwi/article/view/jetwi0402134141

[26] Marcin Kaszkiel, J. Z.: Passage Retrieval Revisited. 1997, [online], [cit. 2012-1-8].
URL http://nlp.korea.ac.kr/new/seminar/2001spring/research/[Kaszkiel(SIGIR97)]PassageRetrievalRevisited.pdf

[27] Meško, D.: Normalizace dat pro neuronovou síť GAME. 2008, [online], [cit. 2012-5-18].
URL http://fakegame.sourceforge.net/lib/exe/fetch.php?media=meskod1_2008bach.pdf

[28] Slawinski, B.: Yahoo Web Page Segmentation: Distinguishing Noise from Information. 2009, [online], [cit. 2011-01-10].
URL http://www.seobythesea.com/2009/10/yahoo-web-page-segmentation-distinguishing-noise-from-information

[29] Wensi Xi, C. S. K., Richard Xu-Rong: Incorporating Window-Based Passage-Level Evidence in Document Retrieval. 2001, [online], [cit. 2012-5-18].
URL http://www.mariapinto.es/ciberabstracts/Articulos/wensi.pdf

[30] Wikipedia: Standard Widget Toolkit - Wikipedia, the free encyclopedia. [online], [cit. 2011-12-30].
URL http://en.wikipedia.org/wiki/Standard_Widget_Toolkit

[31] Zelený, J.: Web page segmentation and classification. 2011, [online], [cit. 2012-5-17].
URL http://www.feec.vutbr.cz/EEICT/2011/sbornik/03-Doktorske%20projekty/08-Informacni%20systemy/10-xzelen11.pdf

Seznam příloh

- Příloha A – Obsah DVD

Obsah DVD

Přiložené DVD obsahuje zdrojové kódy implementace VIPS v jazyce Java, dále zabalenou implementaci ve formátu JAR a potřebné knihovny ke správné funkčnosti. Přiloženy jsou také demonstrační aplikace dodávané k originální implementaci VIPS. Na DVD se také nachází kopie této diplomové práce spolu s obrázky a XML soubory z šesté kapitoly.

Adresářová struktura:

`/implementation`	implementace algoritmu VIPS
`doc/`	dokumentace ve formátu JavaDoc
`jar/`	knihovna VIPS
`vips_java.jar`	obsahuje pouze zdrojové kódy
`vips_java_libraries.jar`	obsahuje zdrojové kódy a všechny potřebné knihovny
`vips_java_runnable.jar`	spustitelná verze
`library/`	potřebné knihovny k běhu
`original/`	originální nástroje firmy Microsoft
`demoTools/`	demonstrační nástroje
`MFCbrowser/`	nástroj MFCBrowser
`source/`	zdrojové kódy
`PageAnalyzer/`	nástroj PageAnalyzer
`UsingVips/`	nástroj UsingVips
`vipsLibrary/`	originální knihovna VIPS
`source/`	zdrojové kódy
`source/`	zdrojové kódy implementace VIPS
`/tests`	obrázky a XML soubory z testů implementace
`/thesis`	diplomová práce
`source/`	zdrojové kódy práce ve formátu LaTeX
`thesis.pdf`	text diplomové práce
`thesis_print.pdf`	text diplomové práce – verze pro tisk
`install.txt`	postup pro spuštění
`readme.txt`	obsah DVD