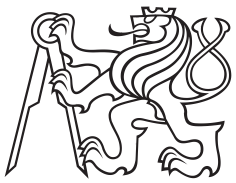
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Uživatelské rozhraní pro čtení vědeckých článků

Bc. Tomáš Novotný
Otevřená informatika

Leden 2017
Vedoucí práce: Ing. Jan Šedivý CSc.

Poděkování / Prohlášení

Chtěl bych poděkovat všem, kteří mě v posledním roce provázeli a podporovali ve studiu i mimo něj. Rodině a přátelům za morální podporu a útěchu. Velký dík patří také Ing. Janu Šedivému CSc. a Ing. Tomáši Gogárovi za trpělivost, pomoc a zápal pro věc.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 18. 5. 2017

.....

Abstrakt / Abstract

Hlavním cílem této práce je usnadnění čtení vědeckých článků publikovaných formátu PDF. Řešením je vytvoření aplikace pro čtení článků, která bude zobrazovat informace o referencích v místě odkazu na referenci v těle článku. K odkazu je možné dotáhnou dodatečné informace z Google Scholar.Celá aplikace je složena ze svou částí, čtečky a serveru. Čtečka je realizována pomocí zásuvného modulo Google Chrome a ke své funkci potřebuje server. Server je Java EE aplikace zajišťující zobrazovaná data pro čtečku.

Klíčová slova: Parsování PDF dokumentů; vědecké články; reference na vědecké články;

Goal of this thesis is to enhance experience of reading scientific article in PDF format by displaying reference information on reference marker in text body of the article. Solution is application, which provides information about references in point of the link to reference. The application is made of two parts. Reader on client side and server. Reader is plugin for Google Chrome and Server is Java EE application.

Keywords: Parsing PDF document; scientific articles reading; reference on scientific articles;

Title translation: Scientific PDF reader

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

Knihovna je dodnes symbolem vědění. Každý, kdo toužil po vědění, tedy zamířil na místo, kde jsou knihy s jím hledaným obsahem dostupné, tedy pravděpodobně do archivu nebo knihovny. Zde hledal v katalozích a policích dlouhé hodiny, jen aby našel knihu nebo sborník prací, ve kterém by dané informace mohl najít.

Dnes je naprostá většina vědeckých článků a publikací dnes distribuována pomocí internetu v digitální formě – většinou ve formátu PDF. Místo v knihovnách a archivech hledáme v databázích knihoven a univerzit a práce si stahujeme nebo čteme přímo online. Málokdo si články tiskne a většinou je čteme na počítači nebo tabletu v elektronické formě. Při hledání relevantních informací k nějakému odbornému tématu procházíme mnohdy desítky článků ve snaze získat co nejvíce podkladů pro naši vlastní práci. Už nás nezpomaluje listování v katalogu a chození mezi regály, ale často je to právě procházení obsahu samotného, co nás zpomaluje při hledání informací.

Když začínáme pracovat na nějakém problému, tak se nejdříve snažíme zjistit, zda tento nebo podobný problém již někdo řešil před námi. Snažíme se vybudovat si povědomí o daném problému. Při tomto procesu často nestudujeme článek důkladně, ale pouze povrchně, a v tomto procesu přecházíme na další a další články. Každý vědecký článek obsahuje citace souvisejících prací, a nezřídka se stává, že k citované informaci bychom rádi znali původ. Tedy je třeba zapamatovat si číslo odkazu na referenci, a vyhledat tuto informaci na konci práce v sekci referencí. U několikastránkových prací, které odkazují na několik málo dalších prací to není tak markantní problém, jako u prací obsahujících desítky stran a desítky referencí. Pak jsme bohužel odkázáni na neustálé skákání z dokumentu, které je velice nepohodlné a často vede ke ztrátě původního místa v textu.

Výše zmíněný problém se čtením referencí se dá vyřešit zobrazením reference, případně dalších informací, přímo v místě odkazu na danou referenci. Informace budou zobrazeny po najetí kurzoru na odkaz, aby nerušily při čtení, ale byly okamžitě dostupné v momentě potřeby. Tímto by čtenář ušetřil čas i nervy při čtení vědeckých publikací.

Kapitola 2
PDF format

2.1 Historie

V roce 1993 firma Adobe Systems představila vlastní standard pro přenos dokumentů mezi platformami Windows, Mac OS a UNIX. Tímto formátem byl *Portable Document Format* známý hlavně pod zkratkou *PDF*. PDF je založeno na jazyce PostScript, který stál u zrodu Adobe Systems v roce 1982. PostScript byl vyvinut jako jazyk popisující grafiku hlavně pro tiskárny. Do příchodu PostScriptu neexitoval jazyk popisující tisknutelnou grafiku pro klasické tiskárny. Specifikace PDF byla od začátku volně přístupná, avšak formát samotný byl majetkem společnosti Adobe Systems. PDF se stalo velice populární pro publikování elektronických dokumentů v době, kdy World Wide Web a HTML dokumenty byly stále v plenkách. V roce 2008 Adobe Systems publikovalo formát PDF jako ISO 32000-1 standard[1]. Adobe Systems si ovšem ponechalo práva pro některé přidružené technologie, jako XML Forms *Architecture*, což je technologie pro práci s formuláři nejen ve formátu PDF. Tato technologie je odkazována v samotném ISO standardu PDF.

2.2 Obecná struktura PDF

PDF dokumenty jsou členěny na následující 4 sekce:

- Hlavička – Obsahuje verzi PDF a komentář k souboru
- Tělo – Obsahuje jednotlivé objekty
- Tabulka křížových referencí – Obsahuje reference na jednotlivé objekty v souboru
- Přívěs – Obsahuje informace o informace o pozici tabulky křížových referencí a dalších specifických objektů

PDF začíná vždy hlavičkou, která obsahuje pouze verzi PDF standardu a komentář souboru. Dále následuje tělo, které obsahuje veškeré viditelné informace. Tedy grafiku a text. Může, ale nemusí, následovat tabulka křížových referencí. V této tabulce je soupis všech objektů z těla dokumentu a informace o jejich umístění v těle dokumentu. Tato informace však neříká nic o jejich skutečné pozici ve zobrazeném PDF. Dokument vždy ukončuje přívěs v anglickém originále *trailer*. Přívěs obsahuje informaci o umístění tabulky křížových referencí, metadata a odkaz na kořenový objekt PDF dokument. Takto vypadá „klasické“ PDF.

Existují ještě dva další módy zápisu PDF dokumentů, a to inkrementální a linearizovaný. V inkrementálním módu začínáme s obyčejným PDF se základní strukturou. V momentě, kdy přidáváme další obsah do dokumentu, nemodifikujeme stávající dokument, ale přidáváme další tělo, tabulku křížových referencí a přívěs na konec souboru. Tímto způsobem je zajištěno určité verzování dokumentu a nehrozí přepsání původních částí dokumentu. Dále také touto úpravou není znehodnocen podpis původního dokumentu. Tato forma se také používá pro vyplňování editovatelných formulářů. Linearizovaný zápis se používá pro velké dokumenty, aby mohly být zobrazovány a načítany

postupně po jednotlivých stranách. Linearizovaný zápis je velice podobný inkrementálnímu s tím rozdílem, že tělo, tabulka křížových referencí a přívěs je vytvářen pro každou stranu zvlášť. Tedy je možné zobrazovat dokument po stranách a tím získat čas pro načtení dalších stran. Dalším podstatným rozdílem je to, že linearizovaný dokument se nečte od konce jako normální PDF, ale po jednotlivých stranách.

2.3 Primitivní objekty PDF

PDF používá pro zápis objektů *Carousel Object Structure* zkráceně *COS*, která obsahuje 8 níže uvedených primitivních typů.

- Boolean – binární hodnta
- Numeric object – číselná hodnota
- String – textový řetězec
- Name – unikátní identifikátor složený z více znaků
- Array – jedno–dimenzionální pole
- Dictionary – datová struktura mapa, klič je vždy *Name* a hodnota může být jakýkoliv jiný objekt nebo reference na jiný objekt.
- Stream – bajtový zápis používaný pro větší části textu a grafiku.
- Null object – prázdný objekt

Pomocí těchto primitivních objektů se skládají komplexnější objekty uvnitř dokumentů. *COS* je tedy zápis objektů pro PDF, podobně jako *JSON* pro *JavaScript*.

2.4 Textové objekty

Textový objekt je základním stavebním kamenem dokumentu, obsahuje totiž veškerý viditelný text v dokumentu. PDF textový objekt začíná operátorem BT a končí operátorem BT. Textový objekt zpravidla obsahuje kromě textu, i operátory pracující s pozici nebo podobou textu. Ve většině případů je textový řetězec uvnitř objektu komprimovaný. Textový objekt má tři parametry unikátní pro svou třídu. Jsou to následujícíc matice:

- Tm – matice textu
- Tlm – matice řádky textu
- Trm – matice textového renderu. Tato matice je vlastně mezivýpočet mezi parametry stavu textu, textovou maticí Tm a stávající transformační maticí

Podoba textu v textovém objektu může být řízena operátory, které specifikují dodatečné informace grafickému procesoru.

2.4.1 Textový stav

Operátory textového stavu mohou být použity v objektu, ale i mimo něj, a ovlivňují tedy všechny textové objekty na dané straně dokumentu. Každá strana dokumentu má svůj vlastní nezávislý textový stav. Mezi nejdůležitější z mého pohledu patří:

- Tf – operátor fontu
- Tfs – operátor velikosti fontu
- Tc – operátor odsazení textu jednotlivých znaků.
- Tw – operátor odsazení jednotlivých slov.
- Tl – operátor řádkování

2.4.2 Operátory pozice textu

Operátory pozice textu jsou Td a TD. Tyto operátory jsou velice podobné svou funkcí jen s drobnými rozdíly. Oba pracují s maticemi Tm a Tlm a jsou používány pro odsazení řádků textu.

2.4.3 Operátory zobrazení textu

Mezi operátory zobrazení textu vyniká svou četností použití operátor Tj. Tento operátor umožňuje vkládat libovolně dlouhé mezery mezi jednotlivé znaky. Hojně se využivá pro odsazování prního velkého písmene a pro vytváření mezer mezi slovy, místo znaku mezery.

2.4.4 Příklad textového objektu

Ukázka zdrojového kódu textového streamu z PDF generováného programem *MiKTeX*:

```
stream
BT
/F58 23.9103 Tf 70.509 728.738 Td [(Scheduling)-350(in)-350(W)40(ireless)
-350(Sensor)-350(Netw)10(orks)-350(WSNs)]TJ
ET
endstream
```

Tato ukázka je nadpis článku s názvem *Scheduling in Wireless Sensor Networks WSNs*, tedy nejedná se o uměle vytvořený příklad. Je použit operátor Tf pro nastavení fontu a jeho velikosti, operátor Td pro nastavení polohy začátku textu a operátor TJ pro přidání rozestupu mezi písmeny a slovy.

2.5 Fonty

Aby bylo možné jednotlivé znaky tvořící text tisknou nebo zobrazit, je nutné znát jejich přesnou grafickou reprezentaci. Protože znak jako takový nemá grafickou reprezentaci, je jeho přesná grafická podoba zapsána ve fontu. Funkcí fontu je tedy poskytnout grafickou podobu jednotlivým znakům. V této práci jsou fonty popsány z hlediska specifikace PDF.

2.5.1 Simple font

Fonty řadící se do skupiny simple fontů spojuje několik společných znaků: Znaky ve fontu jsou mapovány na literály jedním bajtem Font se skládá ze Slovníku fontu, ve kterém se nachází popis fontu a znaků, a programu fontu, který obstarává grafickou reprezentaci. Každý znak je popsán sadou metrik popisující jeho rozměry a prostorové usazení. Protože součástí popisu je i šířka nebo odsazení, podporuje simple font jen psaní horizontálně. Část slovníku fontu popisující vlastnosti fontu se nazývá font descriptor, Font descriptor může obsahovat samotný program fontu zapsaný jako proud.

2.5.2 Fonty Typu 1

Fonty Typu 1 vycházejí z PostScriptu a byly vytvořeny společností Adobe. Obsahují komprimovaný popis znaku a další informace pro jeho vykreslování a scaling. Fonty typu 1 Jsou vhodné i pro vykreslování znaků v malém měřítku.Existuje 14 fontů typu 1, které jsou považovány za implicitní a tedy u nich není povinnost uvádět některé parametry.

2.5.3 Multiple Master Font

Nadtyp fontů typu 1. Agreguje více fontů typu 1 do fontové rodiny za účelem rozšíření dimenzí fontu. Například rozšiřuje standardní font o bold a italic.

2.5.4 TrueType Font

Podobný fontům Typu 1. Vytvořený Apple inc. a později adoptovaný i Microsoft Windows. Původně umožňoval větší kontrolu nad zpusobem rozbrazení znaků, než Type 1. Díky rozšířenosti a rozmanitosti užití fontu, není již tato kontrola možná.

2.5.5 Type 3 Font

Ve Fontu Typu 3 jsou, na rozdíl od zbylých fontů ze skupiny simple fontů, glyfy definovány proudem PDF grafických operátorů. Tyto proudy jsou asociovány s názvy znaků. Dále font obsahuje mapu převádějící kódování znaků na správné názvy znaků.

2.5.6 Composit Font

Také známé jako fonty Typu 0. Jsou to fonty, které získávají glyfy z objektů znavaných CIDFont. Font Typu 0 je root font a CIDFont je nazývan potomkem. Hlavním rozdílem kompozitních fontů užitých v PDF oproti simple fontům je možnost mapování jednoho a více bajtů na glyf z daného CIDFontu. Díky tomu je možné kódovat i velké množství znaků např. čínštinu, japonštinu a další. Kompozitní fonty používají CMapu, která mapuje znaky na kódy, pod kterými jsou uloženy glyfy v CIDFontech.

2.6 Grafické objekty

Tato práce se zaměřuje převážně na textový obsah PDF dokumentů. Z tohoto důvodu bude tato kapitola méně detailní něž kapitola o textovém obsahu.

Podobně jako textové objekty, i grafické objekty mají operátory, které pracují s grafickým stavem. Tyto operátory nebudou z výše zmíněného důvodu popisovány. Grafické objekty mají krom tvarů také barvu. Formát PDF poskytuje velký výběr způsobů zápisu barevnosti pomocí takzvaných *barevných prostorů*. Od různých stupnic šedi přes RBG až po CMYK a různé způsoby převodů mezi těmito barevnými prostory. Dále je třeba zmínit, že formát PDF je při zobrazování grafiky nezávislý na cílovém zařízení. Grafika je uložena v rozlišení dokumentu a těsně před tiskem nebo zobrazením je grafika transformována do potřebného rozlišení.

V PDF se rozlišují grafické objekty na následujících pět druhů:

2.6.1 Path object

Path object nebo také česky trasový objekt. Je to objekt vektorové grafiky, kam kromě základních tvarů, lomených čar patří také bézierovy křivky.

PDF dokáže vykreslit téměř libovolnou vektorovou grafiku. U těchto objektů je možné definovat typ čáry, tedy její tloušťku, barvu, zda bude konec křivky zakulacený nebo plochý, nebo zda a jakou bude mít tvar či objekt barevnou výplň, či zda a kde bude čára přerušovaná.

2.6.2 Text object

K textu může být také přistupováno jako ke grafickému objektu. Font textu pak definuje obrys a dále je možné s objektem pracovat obdobně jako s *trasovým objektem*. Toto pojetí umožňuje text natáčet, odsazovat, disproporcionálně roztahovat, obarvovat a aplikovat další úpravy.

2.6.3 External object

Externí objekty umožňují využívat referencí na jiné objekty. Odkazované objekty můžou být shlukovány do skupin a následně modifikovány, třeba změnou barvy. Nemusí být ani přímo ve struktuře samotného PDF. Můžou stát i samostatně, mimo strukturu, ale ve stejném souboru, nebo dokonce v jiném souboru.

2.6.4 Inline image object

Inline obrázky jsou rastrové obrázky vložené přímo do těla PDF jako stream. Tento typ grafiky je hojně používaný pro vkládání menších rastrových obrázků. Z pohledu formátování a kódování jsou obrázky vcelku „nudné“. Inline obrázek má uvedený počet sloupců a řádků, barevnost jednotlivých buněk, umístění na ose x a y, a případně náklon. Protože bitmapa i rozlišení dokumentu je dáno jako dvourozměrné pole pixelů, je obrázek při náklonu přetransformován pomocí transformační matice do souřadnicového systému dokumentu. Děje se tak proto, aby nebyl obrázek deformován.

2.6.5 Shading object

Tento objekt obstarává barevný přechod na grafických objektech, může být také použit jako barva. PDF podporuje 6 různých druhů přechodů. Od přechodů podle funkcí, os, gradientů, po techniky využívající polygonů a sítí na výpočet barevného přechodu.

Kapitola 3
Související práce

Tato kapitola se skládá ze dvou hlavních částí. Úkolem první části je analyzovat stávající řešení pro práci s vědeckými články. Úkolem druhé části je seznámit čtenáře s odbornými články zabývajícími se problematikou parsování dokumentů ve formátu PDF a parsováním citací jako takových.

3.1 Aplikace pro organizaci a čtení vědeckých článků

Na trhu v současné době existují čtyři hlavní produkty zabývající se hledáním, psaním, organizací a citacemi vědeckých prací. V této sekci je popsán každý z nich.

3.1.1 Mendeley

Mendeley [2] je cloudová platforma pro čtení a organizaci vědeckých publikací. Umožňuje uživateli pracovat s publikacemi jak pomocí nativního klienta, tak webové stránky. Publikace je možné třídit podle metadat, která Mendeley doplní sám, nebo je může vyplnit uživatel. Při čtení publikace dovoluje Mendeley vytvářet poznámky k jednotlivým částem nebo k publikaci jako celku, dále umožňuje zvýrazňovat text a zobrazit pouze zvýrazněný text. Mendeley obsahuje i možnost sdružovat uživatele do skupin a sdílet s nimi příspěvky. Mendeley také obsahuje funkci pro ukládání a kategorizaci „[“dat] souvisejících s vědeckou prací. Mendeley rovněž obsahuje funkci, jež má za úkol vyhledat podobné články a navrhnout je k přečtení. Platforma obsahuje také pluginy do kancelářských balíků pro jednoduché citování článků z knihovny uživatele. Mendeley jako takový se nesnaží usnadnit čtení publikace samotné, spíše se snaží pomoci s organizací práce s větším množstvím publikací. Mendeley je komerční produkt. Každý má možnost zřídit si účet s omezeným 2GB prostorem pro ukládání publikací. Navýšení velikosti prostoru je zpoplatněno.

Mendely umí extrahovat z článku metadata o samotném článku a vytvořit referenci na článek samotný. Rozpoznávání metadat článků není stoprocentně automatizované. Když se Mendeley nepodaří rozpoznat článek, spoléhá na opravu od uživatele. Tyto informace jsou pak odeslány na server, kde jsou uloženy pro příští import. Reference na další články v rámci článku samotného Mendeley nezpracovává.

3.1.2 ReadCube

Readcube [3] je komerční produkt pro čtení vědeckých publikací. ReadCube umožňuje uživateli naimportovat dokumenty z adresáře nebo vyhledat a zapůjčit online. ReadCube spoléhá na silné propojení se službou Google Scholar a vědeckými žurnály, kde články vyhledává a nabízí ke stažení nebo zapůjčení. ReadCube umí pro některé importované články dohledat metadata. Úspěšnost plně automatizovaného dohledání je okolo 60% (testováno na 40 článcích), ve zbylých případech je potřeba označit v textu článku např. nadpis a autora. Poté ReadCube nabídne seznam článků, které podle vyplněných údajů dohledal na Google Scholar nebo v jiné databázi a uživatel zvolí, o který článek se

opravdu jedná. Po dotažení metadat je schopný ReadCube, opět jen pro některé články, zobrazit další články z refencí daného článku, nebo zdroje, ve kterých byl tento článek citován. ReadCube krom stahování a organizace článků umožňuje také články číst a dělat si do nich poznámky a nabízí augmentaci PDF pro rozpoznané články. Bohužel augmentace se ani na jednom z článků, pro které na základě metadat byla dotažena, vůbec neprojevila. Navíc nebylo možné používat ani linky v PDF samotném.

ReadCube má velice minimalistické grafické rozhraní, které má i drobné nedostatky z uživatelského hlediska. Základní účet je zdarma, avšak některé bonusové funkce a více diskového prostoru na cloudovém úložišti jsou zpoplatněny. Zpoplatněno je také stahování a půjčování článků.

3.1.3 EndNote

EndNote je komerční software na správu referencí. Aplikace je ve formě webové stránky s možností instalace doplňku do Microsoft Wordu. Aplikace je propojená s digitálními knihovnami, ve kterých je možné hledat jednotlivé články. Reference je také možné vkládat ze souborů s bibliografickými informacemi v různých formátech, nebo ručně vyplnit formulář. Články je možné třídit do jednotlivých skupin vytvořených uživatelem a v těchto skupinách hledat duplikáty. Dále aplikace podporuje podle bibliografických informací export jednotlivých článků, nebo celých skupin v různých formátech. Formátů je celá řada od *BibTexu* po *RichText*. Aplikace rovněž nabízí speciální webový formulář, který je možné uložit mezi záložky.

Bohužel v základní neplacené verzi aplikace není možné vyhledat článek jako takový, ale pouze referenci na něj. Reference neobsahuje abstrakt, ale je možné ho manuálně vložit do poznámky. V neplacené verzi je možné vyhledávat články pouze v 5 základních knihovnách a počet uložených referencí je limitován na 50 000. V placené verzi je možné hledat až v 5000 zdrojích a referencí je možné uložit neomezené množství. Limitovaný přístup do digitálních knihoven v neplacené verzi značně omezuje její využití.

3.1.4 Zotero a Juris-M

Zotero je projekt na správu referencí v podobě samostatné aplikace a doplňku do prohlížeče. Aplikace slouží pro správu referencí, které jsou vytvářeny buď manuálně nebo importovány z doplňku prohlížeče. Doplněk podporuje import pouze z některých stránek nebo ze souborů bibliografických dat. PDF nepatří mezi podporované formáty. Podporované prohlížeče jsou Chrome, Firefox a Safari. Bohužel doplněk je vyvíjen hlavně pro prohlížeč Firefox a doplňky do ostatních prohlížečů trpí problémy, kdy data získaná z prohlížeče potřebují manuální opravu.

Juris-M je open-source projekt, který rozšiřuje aplikaci Zotero o podporu dalších jazyků a umožňuje export referencí v různých formátech dle šablon. Projekt Juris-M také obsahuje nástroje na vytváření šablon pro reference. Tento projekt není bez projektu Zotero funkční.

3.2 Literatura a nástroje na parsování PDF

Při svém hledání souvisejících článků a prací jsem se soustředil na dva aspekty, podle kterých je možné články dělit do dvou skupin. První jsou články zaměřující se na tématiku strukturování PDF a jeho parsování. Zde nemohu nezmínit specifikaci PDF jako formátu takového [1]. Toto specifikace je základním kamenem pro práci s formátem PDF. PDF verze 1.7 je definováno ISO standardem ISO 32000-1. Dokument popisuje podrobně fungovaní a strukturu formátu PDF.

Následující články jsou zajímavé z hlediska extrakce informaci a textu z dokumentu PDF. Články se snaží různými přístupy řešit obecný problém formátu PDF, kterým je jeho strojové čtení a extrakce informací. PDF má velice volnou strukturu a existuje obvykle více způsobů jak dosáhnout stejného výsledku. Bohužel tato volnost znesnadňuje strojové zpracování tohoto formátu. V článku [4] autoři popisují přístup, kdy PDF pomocí pravidel přeskládává celý dokument do nové struktury, kde objekty mezi sebou mají silnější návaznosti. Toto přeskládání, v originále *juggling*, s jednotlivými objekty v rámci stránky dokumentu přispívá pozitivně k míře strojové čitelnosti dokumentu. Oproti tomu článek [5] se zabývá analýzou rozpadu stran na jednotlivé objekty. Hlavními parametry jsou typ objektu a jeho velikost a umístění v dokumentu. Autoři se snaží vytvořit původní strukturu objektů v dokumentu podobně jako ji má HTML. Informace o struktuře dokumentu totiž PDF neobsahuje. Článek [6] se rovněž zabývá získáním struktury a informací o rozmístění jednotlivých objektů v dokumentu. Autoři článku rozkládají PDF na tři základní části: text, obrázky a vektorovou grafiku. Následně je každá z částí samostatně uložena a metadata o objektu jsou zapsána do XML. Článek [7] se zabývá problematikou, jak číst vícesloupcové dokumenty. Autor řeší tento problém rozložením dokumentu na logické bloky podle určitých pravidel. Logickými bloky jsou následující typy: text, hlavičky, záhlaví, zápatí a neznámé bloky. Na základě tohoto rozdělení určuje pořadí, ve kterém by bloky četl čtenář, tedy i jak by měly být čteny strojově. Za zmínku rovněž stojí patent [8]. Tento patent popisuje teoretický princip vytváření záchytných bodů „Hotspotů“ pro specializované programy v dokumentech PDF. Na tyto záchytné body by pak reagoval program, který by PDF renderoval, a zobrazoval tak libovolný dodatečný obsah v místě hotspotu.

3.2.1 QPDF

QPFD [9] je C++ nástroj pro práci s PDF soubory. QPDF nemá žádné grafické rozhraní a je ovládán čistě přes příkazovou řádku. QPDF podporuje práci s vnitřními objekty PDF souborů, jako je vytváření a kopírování vnitřních objektů. QPDF není vhodný pro vytváření obsahu PDF souboru nebo jeho modifikaci. Jako nejzajímavější funkci programu považuji možnost dekomprimovat streamové objekty uvnitř souboru či možnost dešifrování celého souboru. Většina PDF souborů je implicitně komprimována, takže QPDF poskytuje rychlou a jednoduchou možnost, jak nahlédnout na zdrojový kód PDF souborů.

3.2.2 PDFMiner

PDFMiner[10] je jednoduchý nástroj pro příkazovou řádku, psaný v jazyce Python. Slouží k extrakci textu z PDF dokumentu do jiné podoby. Možnosti extrakce textu jsou čtyři následující:

- Text – Extrahuje čistý text
- XHTM – Extrahuje text otagovaný XHTML tagy
- XML – Extrahuje text taguje každý znak do XML notace
- Tag – Extrahuje text a taguje tagy dle PDF specifikace PDFMiner extrahuje text s ohledem na strukturu dokumentu a řeší některá úskalí při převodu odsahu PDF dokumentu zpět na plain text.

3.2.3 iText 7

iText 7 je komerčně vyvíjené SDK v jazyce Java nebo .NET pro práci s dokumenty formátu PDF. iText 7 je kromně komerční licence také možné získat pod licenci AGPL,

která umožňuje jeho využití v open-source projektech. SDK umožňuje komplexní práci s PDF od jejich vytvoření po editaci. iText také obsahuje nástroje pro vytvoření a zpracování PDF formulářů. Struktura objektů SDK je založená na struktuře *Carousel Object System*, kterou přebírá od PDF.

3.2.4 Apache PDFBox

PDFBox[11] je SDK pro práci se soubory PDF vyvíjené v jazyce Java komunitou Apache software foundation. Apache PDFBox je SDK umožňující komplexní práci s PDF dokumenty, podobně jako IText7.

3.3 Články a nástroje zabývající se strukturou vědeckých dokumentů

Druhá skupina článků je zaměřená na extrakci metadat a referencí z textů a dokumentů různých formátů. Články Bibliographic Attribute Extraction from Erroneous References Based on a Statistical Model [12] a A unified framework for automatic metadata extraction from electronic document [13] se zabývají extrakcí textu a bibliografických údajů za pomocí OCR – tedy rozeznávaní textu na základě jeho vizualizace. V prvním článku se jedná o extrakci metadat z různých zdrojů, převážně dokumentů a knih jež byly nascanovány, kde text samotný nemusí být dobře čitelný, nebo je nějakým způsobem neobvykle formátovaný. Druhý článek se zabývá problematikou parsování referencí z pohledu OCR. V článku autor popisuje vlastnosti modelu, jeho učení a vyhodnocuje přesnost této metody. V článku Reference Metadata Extraction from Scientific Papers [14] je popisován kombinovaný přístup strojového učení v podobě Conditional Random Fields (CRF) a Hidden Markov Model (HMM), knowledge-based, rule-based přístupu. Autoři ukazují, že kombinací uvedených metod jsou schopni dosáhnout vysoké úspěšnosti. Vynikajících výsledků, okolo 98% úspěšnosti při parsování referenci dosáhli autoři článku A New Approach towards Bibliographic Reference Identification, Parsing and Inline Citation Matching [15] kombinací learning based algoritmu, knowledge-based a regulárních výrazů. Článek A Simple Extraction Procedure for Bibliographical Author Field [16] se věnuje extrakci referencí čistě pomocí regulárních výrazů. V článku jsou popsány obecné regulární výrazy a několik pravidel, která mají za úkol omezit chybovost metody.

3.3.1 ParsCit

ParsCit [17] je nástroj na parsování referencí distribuovaný pod licencí GPL. ParsCit je psaný převážně psaný v jazyce Perl a využívá knihovnu CFR++ psanou v jazyce C++. CRF neboli Conditional random field je metoda klasifikace využívaná pro strojové učení a rozeznávání vzorů.ParsCit nejdříve pomocí heuristických pravidel nalezne sekci referencí a poté parsuje samotné reference na jednotlivé pole. ParsCist obsahuje datovou sadu, na níž je klasifikátor vycvičen na rozpoznávání polí reference. Změnou v trénovacích šablonách a datech může být docíleno zaměření ParsCitu na určitý styl článků, kde bude pak dosahovat větší přesnosti při rozpoznávání polí v referenci. Program je možné volat z příkazové řádky, nebo je také možné spustit serverový script, který bude odpovídat na SOAP volání na přednastavené adrese a portu. V obou případech parscit přijímá na vstupu pouze plain text s kódováním UTF-8 a vrací xml s naparsovanými referencemi.

3.3.2 FreeCite

FreeCite [18] je program určený na parsování referencí obdobně jako ParseCite, kterým byl také inspirovaný. Oproti ParseCitu je Freecite šířen pod licencí MIT. FreeCite je založen, obdobně jako ParseCite, na CRF a knihovně CRF++ a psaný v jazyce Ruby on Rails. Model FreeCitu je založen na trénovacích datech z datasetu CORA [19], který byl upraven pro účely učení modelu. Dataset FreeCitu je několikrát větší než pro ParseCite. FreeCite je psán hlavně jako webová služba na parsování reference samotné. FreeCite parsuje pouze reference a není schopen je nalézt v textu jako tomu je u ParseCitu.

3.3.3 ParaCite

ParaCite [20] se zaměřuje hledání článků online. ParaCite nejprve naparsuje referenci a poté vyhledává článek ve vyhledávačích Google, Google Scholar a CiteBase. CiteBase je experimentální služba pro indexaci vědeckých článků. ParaCite nejprve naparsuje referenci a poté vyhledává ve výše zmíněných vyhledávačích za pomoci naparsovaných polí. ParaCite je napsán v jazyce Perl a vyžaduje MySQL databázi, ve které uchovává data o článcích a referencích. ParaCite vystavuje SOAP rozhraní pro externí komunikaci.

Součástí projektu je také nástroj ParaTools. ParaTools je ve skutečnosti srdce ParaCitu, zveřejněné a zdokumentované pro použití v jiných projektech. Vstupem tohoto nástroje má být, stejně jako tomu je u ParaCitu, text reference a výstupem link ve formátu OpenURL. OpenURL je norma jak zapisovat jednotlivé parametry do URL.

Bohužel v době psaní této práce byl ParaCite stále v experimentálním režimu a program nepracoval dle očekávání. Některé zdroje programu byly nedostupné a ParaCite nebyl schopný najít článek, ani po manuálním vyplnění jednotlivých polí. Poslední update projektu proběhl v roce 2004, což znamená že projekt je už nějakou dobu mrtvý.

3.4 Shrnutí

V době vyhotovení této práce jsou dostupné aplikace určené převážně k organizaci odborných článků a jejich psaní. Tyto aplikace jsou schopny v současné době vyhledat a zobrazit metadata(autor, jméno článku, rok vydání) o daném článku. Tyto informace pak aplikace zvládají uchovávat a organizovat. Následně je pak možné reference exportovat nebo je přímo citovat v textových editorech.

Avšak ne vždy jsou tato metadata dostupná či zcela správná. Na základě nalezených metadat jsou tyto aplikace schopné najít omezené množství dalších relevantních článků na obdobné téma. Žádná zkoumaná aplikace neřešila zdroje čteného článku a jejich případné zobrazení.

Nástroje dostupné pro parsování citací dokáží jednotlivé citace rozpadnout na jednotlivá pole, jako jsou název, autoři, rok, atd. Tyto nástroje však nedokáží najít reference v článku a neposkytují graficky přívětivé zobrazení těchto informací. K dosažení těchto výsledků tyto nástroje využívají CRF, což je statistická modelovací metoda používaná při rozpoznávání vzorů a strojovém učení.

Strukturální analýza dokumentů formátu PDF je netriviální záležitost. Vyplývá to z článků zabývajících se formátem PDF. Další metodou jak dostat informace z PDF je nechat přečíst dokument za pomoci strojového učení a OCR. Nástroje věnující se extrakci dat z PDF obsahují sady pravidel, kterými se extrakce řídí. Nástroj, který by umožňoval extrakci referencí přímo z PDF jsem nenašel. Existují však nástroje na převod PDF dokumentu na text a nástroje na extrakci referencí z textu.

Kapitola 4
Návrh řešení

Cílem této práce je vytvořit PDF reader, který by zobrazoval informace o referencích v místě odkazu na referenci. Tyto informace by měly být zobrazeny ve formě kontextové nápovědy. Návrh počítá i s případným obohacením informací, zobrazovaných v kontextové nápovědě, o informace z externích zdrojů.

Tato funkcionalita by měla uživateli ulehčit od dohledávání těchto informací manuálně na konci vědecké publikace a případně v jiných zdrojích.

Pří návrhu aplikace jsem použil pro některé komponenty externí programy, které jsou ovšem publikovány pod otevřenou licencí. Používání knihoven je dnes nedílnou součástí vytváření softwaru. A stejně jako je zbytečné psát znovu funkce obsažené v knihovnách, tak považuji za stejně zbytečné psát znovu čtečku PDF, nebo parser referencí. Níže jsou popsány open-source projekty, které jsem použil jako základní kameny pro svou práci.

4.1 Architektura

Při návrhu architektury aplikace byly brány do úvahy tři možné architektonické návrhy. Každý návrh bude popsán spolu s výhodami a nevýhodami a na konci sekce bude rozhodnutí, který návrh bude použit.

4.1.1 Webová aplikace

Webová aplikace běžící na serveru by obstarávala veškeré zpracování a dotazy na externí zdroje a zobrazovala PDF i s dodatečnými informacemi přímo v prohlížeči. Výhodou tohoto návrhu je, že server má úplnou kontrolu nad daty a zobrazovaným obsahem. Tato kontrola pak umožňuje analyzovat obsah nejčastěji zobrazovaný pomocí aplikace a následně na základě těchto dat upravit model pro zpracování PDF a referencí. Dále je také toto řešení nezávislé na uživatelské platformě, což případně rozšiřuje základu uživatelů, pro které bude aplikace zajímavá. Nevýhody tohoto řešení jsou nutnost trvalého připojení k internetu, nároky na přenos dat mezi serverem a uživatelem a nároky na správu uživatelských účtů a skladování dokumentů na serveru.

4.1.2 Samostatná aplikace

Nejsilnější stránkou samostatné aplikace je její nezávislost na funkčnosti dalších služeb. Samostatná aplikace si výsledky vyhledávání třetích stran může kešovat do lokálního úložiště. Analýza a zobrazování informací o referencích je prováděna lokálně, takže základní funkcionalita nepotřebuje ani připojení k internetu. Uživatel má absolutní kontrolu na obsahem, který čte. Slabinou tohoto řešení je, že neumožňuje žádnou implicitní zpětnou vazbu ani data, která by mohla přispět k zlepšení aplikace. Další nevýhodou je, že aplikace je dostupná jen na jedné platformě. Vytvářet aplikaci pro každou platformu zvlášť je extrémně náročné a portace mezi platformami je nejisté řešení s mnoha možnými komplikacemi.

4.1.3 Klient–Server aplikace

Návrh Klient–server architektury, skládající se ze dvou částí, *PDF readeru* a *PDF parseru*, kdy *PDF Reader* bude v podobě zásuvného modulu do prohlížeče, nebo samostatné aplikace na klientské straně a bude odpovědný za správu a zobrazení dokumentů samotných. *PDF parser* pak bude serverová část aplikace, která bude zodpovědná za převod PDF na text a následná zpracování a poskytnutí informací o referencích *PDF readeru*. Toto řešení je kompromis mezi čistě serverovou aplikací a čistě standalone aplikací. Při klient–server architektuře je klient dostatečně samostatný, aby umožnil práci s již analyzovanými dokumenty při nedostupnosti serveru. Díky pojetí klienta jako doplňku do prohlížeče se zachová platformní kompatibilita, protože prohlížeče jako Mozilla Firefox a Google Chrome jsou multiplatformní.

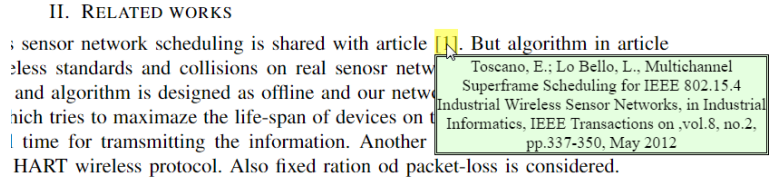
4.1.4 Shrnutí

Rozhodl jsem se pro použití Klient–server architektury, protože kombinuje výhody zbylých dvou možností. Dává možnost použít Reader v offline režimu pro články s uloženými výsledky a také umožňuje číst články neanalyzované, ty ovšem bez zobrazení informací o referencích. Dále také toto řešení umožňuje vylepšovat schopnosti Parseru na straně serveru bez nutnosti aktualizace klientských doplňků a sběr dat pro další vývoj. Toto řešení je také dostatečně platformově nezávislé.

4.2 PDF reader

PDF reader bude aplikace na straně klienta. Hlavním úkolem readeru, tedy klientské části aplikace, je zobrazení a správa dokumentů na lokálním úložišti uživatele. Reader bude posílat PDF dokumenty ke zpracování na server, který dokumenty zpracuje a vrátí pozici odkazů na reference s informacemi o jednotlivých referencích, a názvem článku obsaženém v dokumentu. Odkazy budou v textu označeny žlutým podbarvením čísla odkazu reference. Informace o referenci budou zobrazeny po kliknutí myší na odkaz reference v „bublině“ kontextové nápovědy. Dále také bude možné kliknout v kontextové nápovědě na tlačítko pro získání informací z externího zdroje. Do tohoto zdroje se bude dotazovat přímo klient.

Reader bude umožňovat otevřít PDF z disku počítače a výsledky již zpracovaných dokumentů si bude kešovat pro další otevření. Dále klient bude zobrazovat dokumenty pod názvem článku, nikoliv souboru.Název článku klient získá spolu s informacemi o referencích ze serveru.



Obrázek 4.1. Raný návrh bubliny s informacemi o referenci

Funkcionalita zobrazování PDF dokumentů není hlavním tématem této práce a existují mnohé readery, které tuto práci odvedou daleko lépe, než mi dovoluje rozsah diplomové práce. Proto jsem se rozhodl, že použiji již existující PDF reader. Vzhledem k zvolené architektuře bylo třeba se nejprve rozhodnout, zda se bude jednat o nezávislou aplikaci, jako jsou například Acrobat Reader nebo Foxit Reader, nebo zda k zobrazení

dokumentů použiji některý z enginů obsažených ve webových prohlížečích. Vzhledem k licenčním podmínkám a faktu, že neexistuje multiplatformní, otevřený reader, který bych mohl bez problémů použít, jsem možnost použití samostatného klienta zavrhl. V oblasti PDF readerů pro webové prohlížeče se nabízejí dva readery. PDFium a PDF.js, oba tyto projekty jsou součástí rozšířených a běžně dostupných prohlížečů. A těmi jsou Google Chrome, respektive Mozilla Firefox.

4.2.1 PDFium

PDFium je open–source reader, který je součástí prohlížeče Google Chrome. Tento Reader je psán v jazyce C++. PDFium vytvořila pro Chrome společnost Foxit, která se zabývá komerčním softwarem na vytváření a práci s PDF. Foxit má vlastní reader nazvaný Foxit reader, který je součástí celé sady nástrojů pro práci s PDF. Ztéto skutečnosti je zřejmé, že společnost Foxit je v oblasti práce s PDF zavedenou značkou. PDFium je nedílnou součástí prohlížeče Chrome. Tedy případné úpravy zpracování PDF dokumentů by bylo třeba zakomponovat přímo do jádra Chromu.

4.2.2 PDF.js

PDF.js je open–source reader vyvíjený pod záštitou Mozilla Foundation a jak název napovídá, je psaný v jazyce JavaScript. Jedná se o otevřené řešení vyvíjené komunitou pod licencí Apache v2.0. Zatímco PDFium vzniklo účelně jako engine pro zobrazení dokumentů, PDF.js vznikl jako samostatný projekt, který měl za cíl vytvořit reader nezávislý na jádru prohlížeče, a také jako demonstrace možností HTML5. Díky svému charakteru je možné tento projekt používat nejen jako reader na straně klienta, ale také jako součást stránky na straně serveru.

4.2.3 Shrnutí

Vzhledem k faktu, že PDFuim je vcelku pevně svázáno s prohlížečem Chrome a jeho úpravy by znamenaly přechod na otevřenou verzi Chromu, tedy Chromium a seznámení se s frameworkem tohoto prohlížeče, jako výhodnější mi vyšlo použití základu readeru od Mozilla Foundation PDF.js, který podporuje možnost vytvoření doplňku pro prohlížeče Chrome i Firefox, a jeho případný port na ostatní prohlížeče není extrémně náročný.

4.2.4 Offline dostupnost

Offline dostupnost je velice důležitá, je to jedna z klíčových vlastností, kvůli níž jsem se rozhodl pro architekturu klient–server. Požadavek je vcelku jednoduchý. Reader po otevření dokumentu a jeho úspěšném zpracování na straně serveru data uloží do zvláštního datového souboru spolu s cestou a kontrolním součtem souboru. Při otevření souboru z disku reader zkontroluje, zda tento konkrétní soubor již neotvíral, a pokud ano, tak zda se nezměnil kontrolní součet. Pokud kontrolní součet souboru je nezměněn, znamená to, že od posledního zpracování nedošlo ke změně souboru. Reader v tomto případě dohledá data, uložená v datovém souboru, a zobrazí je bez nutnosti volat server. Pokud byl však soubor změněn uložená data jsou zahozena a je třeba volat server. Tímto způsobem bude dosaženo dostupnosti i v případě, že počítač není připojený k internetu nebo je server nedostupný.

4.3 PDF parser

PDF parser je aplikace na straně serveru, psaná v jazyce Java EE. Jejím účelem je přijmout od klienta PDF dokument, přečíst jej a převést do formy strojově zpracovatelné.

Dále v dokumentu najít jeho nadpis, jednotlivé odkazy na reference v textu, včetně přesné pozice, a sekci referencí. Jednotlivé reference naparsovat na jednotlivá pole jako jsou název článku, autoři, rok vydání a tyto informace přiřadit k odkazům na reference v textu práce. Následně tyto informace vrátit klientovi.

Aplikace je rozdělena na jednotlivé části, které budou obstarávat různé aktivity při zpracování dokumentu. Seznam jednotlivých aktivit:

- Komunikace s klienty
- Převedení dokumentu na text
- Nalezení názvu práce
- Nalezení odkazů na reference
- Nalezení referencí
- Parsování referencí

Níže bych rád popsal aktivity, které z pohledu návrhu nebyly realizovatelné v rozumné funkční míře v rozsahu práce, a proto jsem se k jejich realizaci rozhodl použít existující nástroje či řešení.

4.3.1 Převedení dokumentu na text

Jelikož dokumenty ve formátu PDF nejsou jednoduše čitelné, je třeba je nejprve převést do formy, se kterou lze pracovat. K tomuto účelu je možné využít jeden z nástrojů zmíněných v kapitole *Související práce*. Tedy iText 7, PDFMiner nebo Apache PDFBox. Všechny tyto nástroje umí přečíst nativní strukturu dokumentů PDF a poskytnou informace o objektech a textu uvnitř dokumentu.

iText 7 je především framework pro vytváření dokumentů. Převod na text s dodatečnými informacemi o formátování není scénář s kterým tento framework počítá, a je tedy nutné tuto funkcionalitu vytvořit na základě informací z vnitřních objektů dokumentu, které framework interpretuje.

Apache PDFBox je framework psaný v Javě, velice podobný iTextu, ale iText je komplexnější, se silnější objektovou orientací. PDFBox je ve své implementaci daleko přímočařejší a jednotlivé metody řeší komplexní úkoly. Proto úprava funkcionality, aby poskytovala potřebné informace, je náročnější než je tomu u iTextu.

PDFMiner je program psaný v jazyce Python, přímo určený na extrakci textu z dokumentů. PDFMiner vrací text dokumentu s informacemi o fontech, umístění a velikosti písma. Tyto informace jsou dostačující k následné analýze dokumentu. Vnitřní struktura PDFMineru je podobná jako u PDFBoxu.

K parsování dokumentů jsem vybral PDFMiner. Bohužel neodpovídá platformě vybrané pro implementaci serveru, ale zato je navržen přesně pro účely extrakce textu a informací o něm. V tomto ohledu PDFMiner hravě překoná zbylé frameworky, ve kterých by bylo třeba tuto funkcionalitu vytvořit.

4.3.2 Parsování referencí

Nalezení sekce referencí a referencí jako takových je úkol, který je zvládnutelný pomocí sady pravidel. Je to možné díky tomu, že sekce referencí má vždy stejnou strukturu a pozici v rámci dokumentu. Stejné tvrzení ale neplatí o referencích samotných.

Reference jako takové mají pravidla na to, v jakém pořadí se mají informace v referencích uvádět, a případně jaký mají styl. Vezměme si například standard citací od *IEEE* [21]. Citace začíná číslem v hranatých závorkách a následuje seznam autorů. Co bude následovat záleží na typu práce. *IEEE* rozeznává 8 různých typů prací a pro každou platí trochu jiná pravidla citací. Tedy pro každý typ by bylo třeba vytvořit pravidlo,

a to zatím uvažujeme pouze jeden návod, jak citovat. A takovýchto předpisů existuje více. Např. Harvard nebo Chicagská univerzita mají vlastní pravidla pro citace. Problém nastává v momentě, kdy se pokusíme zjistit, jaké pravidlo máme použít, protože pravidla se neliší v použitém stylu textu, ale v pořadí informací, a ne všechny informace jsou pro citaci povinné. Tedy nastává problém, jak rozlišit jméno univerzity od jména žurnálu nebo organizace. Došel jsem k názoru, že bez použití strojového učení je tato úloha jen velice těžce proveditelná, a rozhodl jsem se využít k parsování jednotlivých citací již existující aplikaci.

V úvahu připadaly dva programy popsané v kapitole *Související práce* v sekci *Články a nástroje zabývající se strukturou vědeckých dokumentů* a to *ParsCite* a *FreeCite*. Vzhledem k faktu, že *FreeCite* vychází z *ParsCite* a je tedy jeho nástupcem, mají oba programy velice podobnou implementaci a rozhraní. Zvolil jsem novější a dle informací, uváděných na stránkách projektu, i všestrannější *FreeCite*.

4.4 Externí zdroje

V popisu klienta je zmíněno dotahování „informace z externích zdrojů“, které bude vykonáváno klientem. Jedná se o dotažení dodatečných informací, které reference v dokumentu neobsahuje. Tyto informace budou vyhledávány podle jména práce v referenci v externí databázi a následně dotaženy klientem a zobrazeny v patřičné kontextové nápovědě.

Při hledání externího zdroje s vystaveným API, který by poskytoval dostatečné množství informací, jsem byl neúspěšný. Google Scholar nevystavuje volně dostupné API, a digitální knihovny jako IEEE Xplore nebo Springler Link neposkytují skrz API informace jako je počet citací, ale pouze referenci, které je však již obsažena v dokumentu. Rozhodl jsem se tedy, že klient bude volat přímo Google Scholar a parsovat odpověď na HTTP Get. V tomto volání jsou obsaženy informace jako zkrácený abstrakt, počet citací článku a odkaz na zdroj článku nebo dokument samotný.

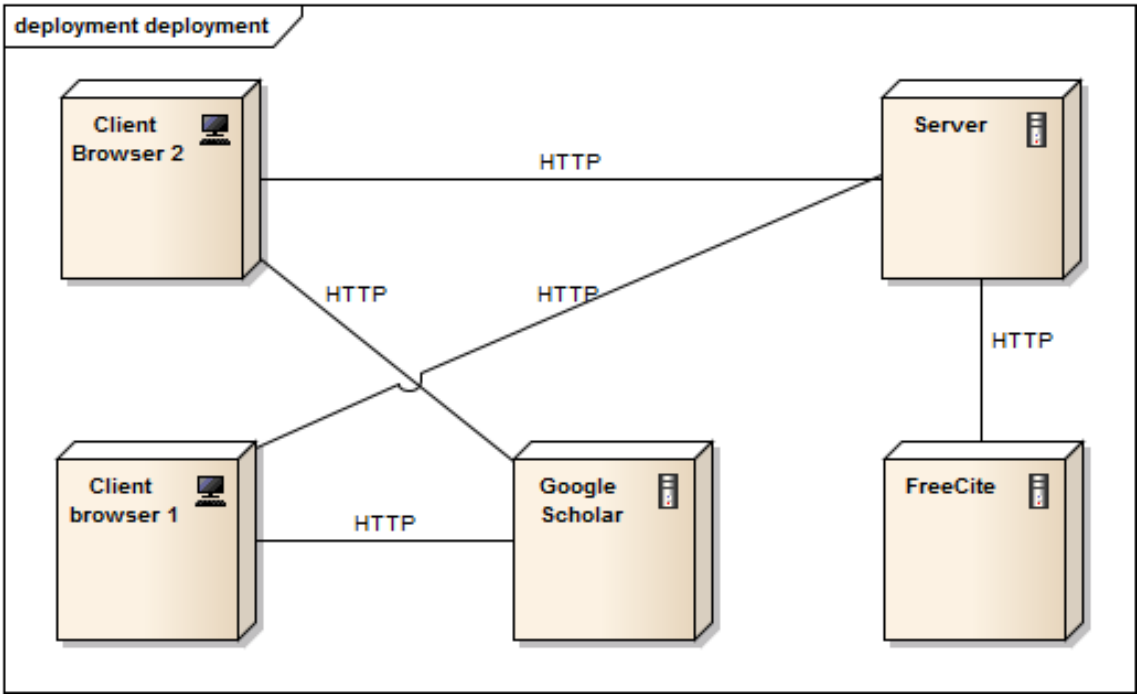
Kapitola 5
Architektura řešení

V této kapitole je popsána celá aplikace z pohledu architektury. Jak z pohledu celkového, kdy je celé řešení děleno na klientskou a serverovou část, tak i z pohledu jednotlivých celků. PDF reader je tvořen vrstvami, které budu popisovat. Serverová část je dělena na jednotlivé moduly z důvodu jednodušší výměny funkčních celků bez nutnosti upravovat ostatní moduly.

5.1 Klient-Server

Aplikace má architekturu klient–server. Důvody, spolu s výhodami a nevýhodami, které mě vedly ke zvolení právě této architektury, jsem uvedl v kapitole *Návrh řešení* v sekci *Architektura*.

Klient volá server přes HTTP. Obdobně volá klient i Google Scholar. Server volá při zpracování zprávy od klienta FreeCite pro naparsování referencí na jednotlivé pole. Tedy vztah klient–server se objevuje vícekrát a některé části vystupují jako klient pro jednu službu a jako server pro druhou.

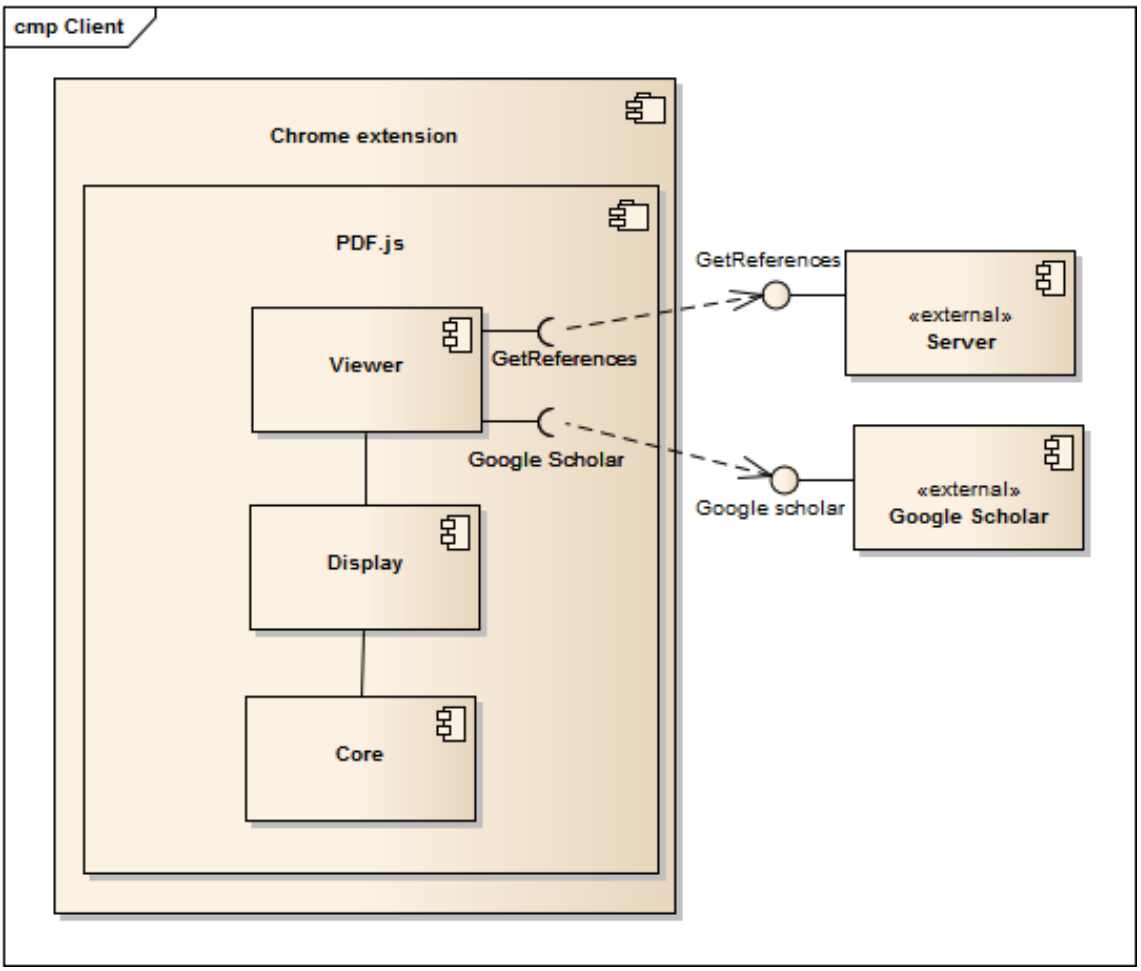


Obrázek 5.1. Diagram nasazení

5.2 Klient PDF reader

Klient je PDF reader v podobě zásuvného modulu do prohlížeče Chrome. Klient komunikuje se serverem i s Google Scholar pomocí AJAX rozhraní. AJAX je zkratka

z *Asynchronous JavaScript And XML* a jedná se o knihovnu pro HTTP komunikaci. Směrem k serveru posílá klient HTTP zprávu s příznakem POST. Součástí zprávy je dokument ve formátu PDF. V případě potřeby dodatečných informací ze strany uživatele se klient dotáže Google Scholaru, výsledek naparsuje a zobrazí v kontextové nápovědě.



Obrázek 5.2. Diagram komponent klienta

5.3 PDF.js

Celý reader je napsaný v JavaScriptu na frameworku Node.js. Reader je tedy určený pro zobrazení ve webovém prohlížeči. Tím, že je celý projekt psáný v JavaScriptu, podporuje použití buď na straně serveru, nebo jako zásuvný modul na straně prohlížeče.

PDF.js hojně využívá při své práci asynchronního volání, takzvaných slibů. V angličtině se tyto sliby nazývají „promises“. Jedná se konstrukci, kdy se vytvoří struktura kolem objektu nebo volání, která poskytuje okolí informaci o stavu volaného objektu, a dává možnost okolí na tento stav reagovat. Sliby jsou kombinací event listeneru a manuální kontroly stavu objektu. Díky slibům běží PDF.js v neblokujícím režimu a dovoluje, aby se spousta činností odehrávala na pozadí programu asynchronně.

PDF.js má následující tří základní vrstvy:

5.3.1 Core vsrtva

Core vrstva je základním stavebním kamenem celého readeru. Jedná se o implementaci jednotlivých objektů samotného formátu PDF. Tato vrstva převádí kód dokumentu do objektové struktury PDF, se kterou následně pracují další vrstvy.

5.3.2 Display

Display vrstva převádí objekty z core vrstvy do formátu, který je jednodušeji zobrazitelný samotným rendererem. Tato vrstva také zpracovává metadata zapsaná v PDF.

5.3.3 Viewer

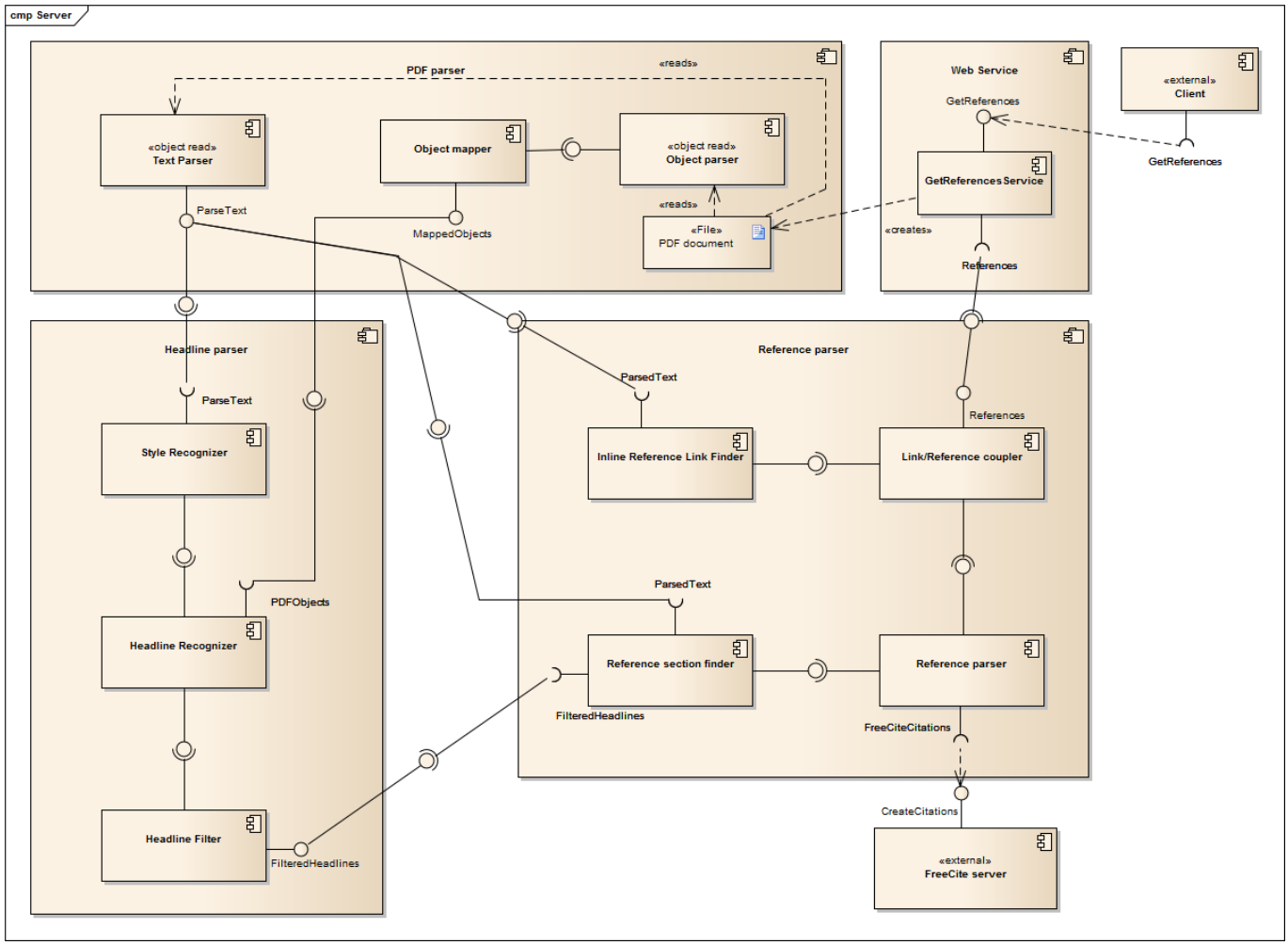
Viewer vrstva obstarává uživatelské rozhraní celého readeru. Tato vrstva obaluje celý reader. Konkrétní implementace se liší podle cílového prohlížeče. Jedná se o část projektu, nejčastěji upravovanou pro individuální potřeby vývojářů, kteří na PDF.js staví svá řešení.

5.4 Server

Modulární struktura serveru umožňuje jeho snadnou rozšiřitelnost a případné nahrazení jednotlivých celků jinými programy. Server se skládá z jednotlivých hlavních modulů, kde každý modul má svoji přesnou roli. Jsou to následující moduly:

- PDF parser
- Headline parser
- Reference parser
- WebService

Níže bude popsán každý z modulů spolu s jejich jednotlivými celky.



Obrázek 5.3. Diagram komponent serveru

5.5 PDF parser

Parser je modul, který převádí dokument ve formátu PDF do formy uchopitelné v dalším zpracování.

5.5.1 Documet parser

- Vstup: PDF dokument
- Výstup: Proud znaků s vlastnostmi

, Text parser čte jednotlivé textové objekty v dokumentu. Každý textový proud přečte znak po znaku, a pro každý znak zaznamená jeho pozici na straně, velikost, font a stranu. Pro extrakci textu z PDF je užíván nástroj PDFMiner. Text parser zaobaluje externí volání PDFMineru a následné čtení jeho výstupu.

5.5.2 Object parser

- Vstup: PDF dokument
- Výstup: List netextových objektů s vlastnostmi

Object parser identifikuje jednotlivé netextové objekty v dokumentu a zaznamenává jejich vlastnosti, jako jsou typ a umístění na stránce.

5.5.3 Object mapper

- Vstup: List netextových objektů s vlastnostmi, Proud znaků s vlastnostmi
- Výstup: Mapa objektů na straně

Object mapper zaznamenává grafickou reprezentaci objektů a textu na straně, a to na základě jejich umístění a velikosti. Poskytuje informace o ucelenosti jednotlivých bloků v dokumentu.

5.6 Headline parser

Headline parser analyzuje data z document parseru a hledá jednotlivé styly v textu, na základě kterých vyhledává název článku.

5.6.1 Rozpoznání stylů

- Vstup: Proud znaků s vlastnostmi
- Výstup: Seznam jednotlivých stylů

Tento modul má za účel na základě velikosti a fontu určit styly použité v dokumentu a podle pravidel určit jejich pravděpodobné užití.

5.6.2 Rozpoznání nadpisů

- Vstup: Proud znaků s vlastnostmi, Seznam jednotlivých stylů
- Výstup: Seznam nadpisů s vlastnostmi

Na základě stylů a dalších pravidel oddělí obyčejný text od nadpisů. Nadpisy vyhledá v textu a uloží je do seznamu společně s jejich vlastnostmi.

5.6.3 Filter nadpisů

- Vstup:Seznam nadpisů s vlastnostmi
- Výstup: Seznam nadpisů s vlastnostmi

Na základě textu nadpisů a jejich stylů filtr určí vhodnou úroveň členění na sekce. Nadpisy jednotlivých sekcí pak předá dál.

5.7 Reference parser

Úkolem tohoto modulu je najít sekci referencí a naparsovat ji na jednotlivé reference na jednotlivé pole.K parsování polí se volá FreeCite.

5.7.1 Hledání odkazů v textu

- Vstup: Proud znaků s vlastnostmi
- Výstup: Seznam odkazů s pozicí

Modul zpracovává proud znaků a hledá odkazy v textu ohraničené hranatou závorkou a s číslem uvnitř. Toto je způsob citace *IEEE*. Rozpoznávání odkazů jiných stylů citací bude předmětem dalšího rozvoje aplikace. Hranaté závorky obsahující jiné znaky než čísla a mezery nejsou vyhodnoceny jako odkaz na referenci.

5.7.2 Hledání sekce referencí

- Vstup: Proud znaků s vlastnostmi, Seznam nadpisů s vlastnostmi, Mapa objektů na straně
- Výstup: Sekce referencí

Na základě nadpisů a objektů nalezených na straně modul hledá začátek sekce referencí a snaží se vyfiltrovat cizí objekty (např. tabulky). Výsledkem je text obsahující jednotlivé reference.

5.7.3 Párování referencí a odkazů

- Vstup: Seznam naparsovaných referencí. Seznam odkazů s pozicí
- Výstup: Slovník odkazů s textací reference

Podle čísel odkazů a referencí modul vytvoří slovník, který bude obsahovat jednotlivé odkazy s pozicí a k nim příslušnou textaci.

5.7.4 Parsování referencí

- Vstup: Slovník odkazů s textací reference
- Výstup: Slovník odkazů s objekty obsahující jednotlivá pole citace

Tento modul vezme textace referencí ze slovníku a nahradí je citací rozdělenou na jednotlivá pole. Citace bude vygenerována pomocí API FreeCite. Bude učiněn jeden souhrnný dotaz na veřejný server FreeCite, na který bude aplikace posílat požadavky o naparsování jednotlivých referencí. Server je také možno zprovoznit lokálně spolu se serverovou částí. Doporučuji zprovoznit si vlastní serverovou část a lokální server FreeCite, pokud aplikaci bude využívat větší počet uživatelů na lokální síti. Následně pomocí JAXB bude výsledek převeden do objektu, z kterého se vygeneruje nová textace pro referenci.

5.8 Web service

Modul vystavuje webovou službu pro klienta.

5.8.1 GetReferences

- Vstup: HTTP zpráva
- Výstup: HTTP zpráva

Tato třída obstarává komunikaci pomocí webové služby. Přijímá dokument, který ukládá jako soubor a vrací naparsované reference v odpovědi na zprávu.

Kapitola 6
Implementace

Tato kapitola popisuje technologie a problémy, které jsem řešil v rámci implementace této práce.

6.1 Použité technologie

Hlavním účelem této sekce je dát čtenáři přehled o tom, jaké technologie a nástroje, a hlavně v jakých verzích, byly použity při implementaci této práce.

6.1.1 Java EE v7

Pro implementaci serverové části jsem zvolil Javu EE verze 7 pro její orientovanost na webové služby a robustnost. Dalším klíčovým faktorem byla jednoduchá škálovatelnost serveru při zátěži.

6.1.2 WildFly 10

Pro běh serverové části jsem zvolil aplikační server WildFly 10. WildFly je open–source aplikační server vyvíjený pod záštitou společnosti Red Hat Inc. Upřednostnil jsem plnohodnotný aplikační server před webovým kontejnerem, jakým je například Apache Tomcat, hlavně kvůli další rozšiřitelnosti serverové části, kde by již pouhý kontejner nemusel být dostačující.

6.1.3 Python 3

V programovacím jazyce Python 3 jsou psány některé podpůrné programy používané serverovou částí, jako například PDFMiner. Zvolil jsem Python, protože podporuje moduly pro strojové učení. Programy zabývající se strojovým učením jsou psány v Pythonu nejen pro jeho vlastnosti, ale také pro jeho širokou podporu nástrojů pro strojové učení a tyto nástroje jsou vyvíjeny pro Python hlavně kvůli jeho vlastnostem.

6.1.4 Chrome v57

Jako výchozí prohlížeč jsem zvolil Google Chrome. Aplikace byla vyvíjena na verzi 57 jako na nejaktuálnější verzi prohlížeče v době vývoje aplikace. Google Chrome jsem zvolil pro jednoduchou manipulaci s klientem jako s doplňkem. Dalším důvodem byla integrace projektu PDF.js přímo do prohlížeče Mozilla FireFox, obával jsem se možných konfliktů mezi různými verzemi.

6.1.5 Node.js v6.10

Node.js je běhové prostředí pro JavaScript určené převážně pro servery. Výhodou Node.js je jeho rychlost, škálovatelnost a hlavně neblokovatelnost. Node.js vychází z JavaScriptového enginu *V8* psaného pro Google Chrome. Tedy běh aplikací na tomto frameworku je podporován i prohlížeči, pokud je spouštěn jako zásuvný modul. Velkou výhodou Node.js je jeho jednotné API pro práci se zdroji jako jsou například databáze nebo JSON objekty. Framework je také uzpůsoben, aby podporoval použití návrhových vzorů jako jsou *Mode View Controller* , *Mode View Presenter* a *Mode View ViewModel*.

6.2 Serverová část

6.2.1 Rozpoznání sloupcovitosti textu

Vědecké články jsou povětšinou formátovány do jednoho nebo dvou sloupců. Čtení textu ve sloupcích je již dovednost nad, kterou se člověk ani nezamyslí, ale pro strojové zpracování to samozřejmost není. Proto bylo třeba pro správné přečtení textu z kódu dokumentu rozpoznat, zda článek je psaný v jednom či dvou sloupcích.

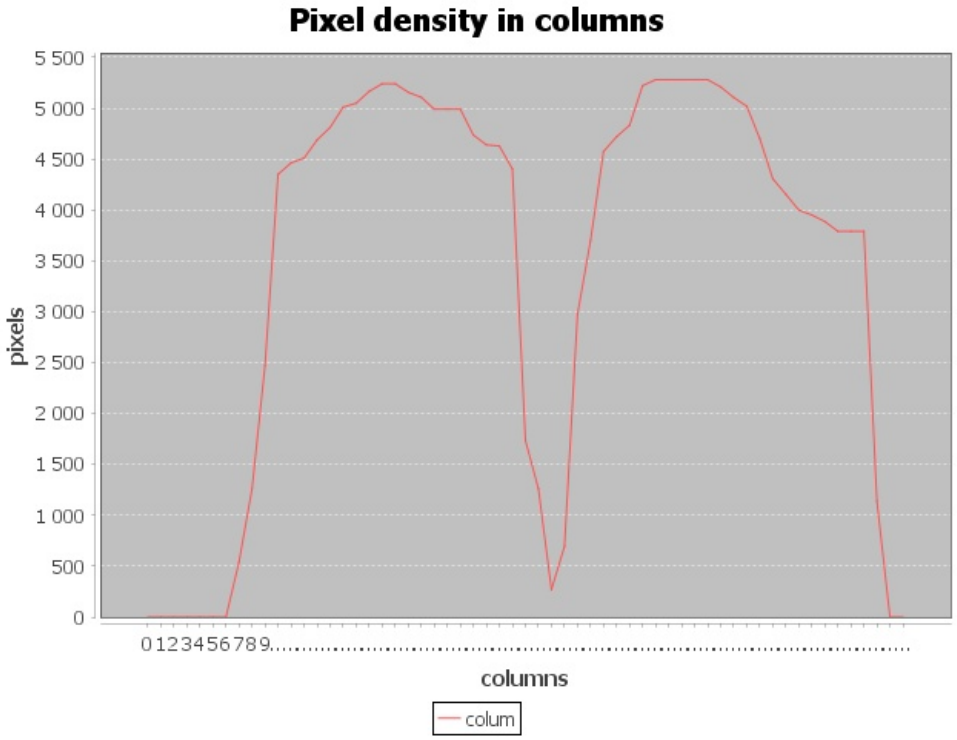
Každý znak má krom grafického vykreslení i svůj prostor, takzvaný *bounding box*. *Bounding box* je osobní prostor znaku, kam žádný jiný znak nebo objekt nezasahuje.

Pro potřeby rozpoznání sloupcovitosti jsem vytvořil otisk strany. Tento otisk je bitové pole, kde každý pixel strany odpovídá jedné buňce v poli. Tato buňka je buď 0, pokud se daný pixelu nenachází v žádném *bounding boxu*, nebo 1 pokud daný v *bounding boxu* nachází. Tedy pole je vyplněno souvisleji než, kdybychom brali pouze vykreslené pixely znaků.

Obrázek 6.1. Bitový otisk strany

Na základě bitového otisku je zhotoven histogram, který zachycuje počet pixelů v jednotlivých sloupcích. Jako optimální rozlišení se ukázala šířka sloupce 10 pixelů. Šířka 10 pixelů odpovídá přibližně šířce jednoho znaku psaného Fontem Arial ve velikostí 16,

nebo dvou znaků velikosti 11. Příliš široké sloupce byly náchylné na na zkreslení, pokud například text obsahoval obrázek, který se do textu nepočítá. Příliš úzké sloupce zase neobsahují dostatečný počet pixelů, aby se vytvořily dva jasně rozeznatelné sloupce bez náchylnosti na anomálie. V případě, že text obsahuje dva sloupce, je šířka sloupce 10 je vyhovující a poskytuje stabilní výsledek. Výkyv v počtu pixelů ve sloupci je pak patrnější a snáze detekovatelný. A právě na základě změny v hustotě zachycených pixelů je program schopný určit, zda je text na straně formátován do dvou či jednoho sloupce.

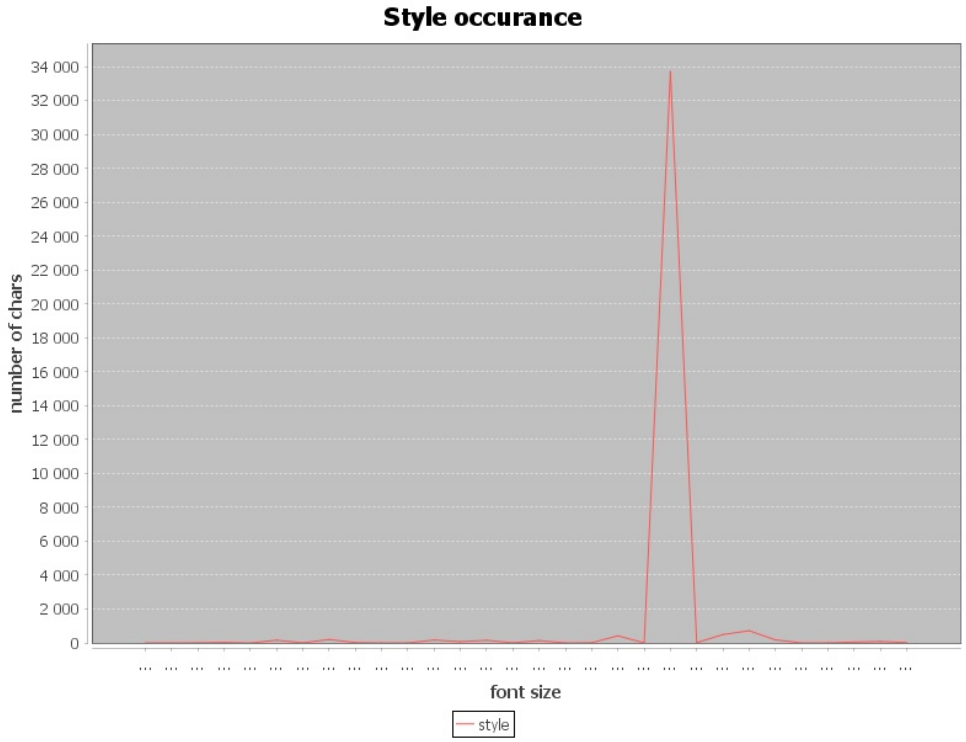


Obrázek 6.2. Graf pixelů ve sloupcích

6.2.2 Rozpoznávání nadpisů textu

K určení nadpisů v článku je třeba nejprve určit jednotlivé styly použité v celém článku. Styl je definován velikostí písma a fontem. Pro účely rozpoznání hlavního textu a nadpisů musíme nejprve vědět, v jakém stylu je dokument psán. Tento styl bude mít největší zastoupení, tedy bude jím napsáno nejvíce znaků. Všechny nadpisy budou mít větší velikost, než je použita pro běžný text. Chceme tedy odfiltrovat všechny znaky stejně velké a menší než je nejpočetněji zastoupený styl.

Celý dokument je projit znak po znaku, a pro každý znak se kontroluje, zda je jeho kombinace velikosti a fontu zařazena do seznamu stylů. Pokud seznam obsahuje tento styl, je k jeho počítadlu přičten výskyt znaku tohoto stylu. Po dokončení průchodu můžeme odříznout styl použitý pro tělo textu, tedy styl s největším počtem výskytů, a všechny styly s menší velikostí fontu. Zbylé styly jsou pravděpodobně nadpisy, nebo součásti titulní strany dokumentu.



Obrázek 6.3. Graf zastoupení stylů

Tato funkcionalita byla využita hlavně pro hledání titulu práce. Spolu s informacemi o stylech a pozicí textu je aplikace schopna určit název článku. Název článku bude jistě na první straně a bude začínat v její horní polovině, zarovnaný na střed nebo k levému kraji, s největší velikostí fontu. Informace o stylu také pomůže identifikovat víceřádkový nadpis, v tom případě vybere aplikace všechny navazující řádky stejného stylu.

6.2.3 Rozpoznávání sekce referencí a referencí

Jedním z hlavních problémů při realizaci této práce bylo nalezení sekce referencí. O této sekci víme, že obsahuje velké množství referencí, které mají specifickou strukturu znaků a řádkování. Sekci referencí není těžké pro čtenáře rozpoznat, ale už není tak jednoduché vytvořit pravidla pro její rozpoznání pro program.

Tento problém se mi nakonec nepodařilo zcela elegantně vyřešit. Nenašel jsem dostatek specifických znaků, které by odlišovaly sekci referencí od zbytku textu. Řešením bylo rozpoznávat sekci referencí podle názvu sekce. Všechny anglické články tuto sekci mají pojmenovanou „References“. Hledám tedy v nadpisech toto slovo a za sekci referencí pokládám text od tohoto nadpisu dále, případně po další nadpis.

Jednotlivé reference pak v sekci je možné rozeznat podle čísel v hranatých závorkách na začátku reference. Tyto hranaté závorky jsou dány směrnicemi pro citace IEEE, tedy u správně citovaného článku z oboru elektroniky a počítačů by tyto závorky na začátku referencí neměly chybět.

6.3 Klientská část

Veškeré změny a úpravy byly prováděny ve viewer vrstvě projektu PDF.js. Zbylé vrstvy se zaměřují na vykreslování dokumentů a do této funkcionality nebylo potřeba zasahovat. Zobrazování kontextové nápovědy a dotazování externích zdrojů architektonicky

patří do viewer vrstvy stejně jako ostatní funkce, které nejsou přímo asociované s vykreslováním PDF, jako je otevíraní dokumentů, jejich ukládání na disk atd.

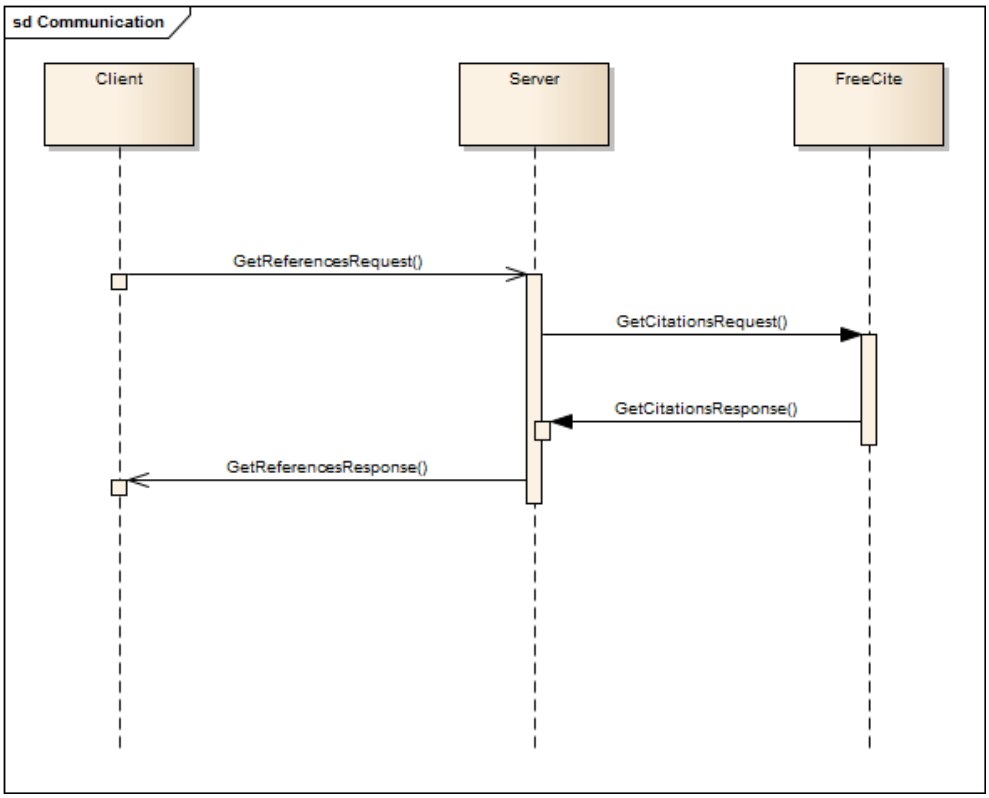
6.3.1 Vykreslování překryvu v PDF.js

Pro vykreslení elementů kontextové nápovědy bylo třeba vytvořit třídu překryvu a třídu pro komunikaci s externími zdroji dat. Vrstva překryvu je volána při každém překreslení strany kvůli úpravě měřítka a pozice odkazů na reference a kontextové nápovědy. Při prvním zavolání vykreslení překryvu se zahájí inicializace v podobě slibu, a v momentě, kdy jsou dotažená data o referencích a jejich pozicích, je strana překreslena a obohacena o tyto informace.

Pro dotažení dat z Google Scholar je princip stejný, pouze se volá jiný zdroj a výsledek je zapsán do kontextové nápovědy.

6.3.2 Ajax rozhraní

Ajax rozhraní obstarává HTTP komunikaci s parse serverem a stránkou Google Scholar. Ajax je zkratka pro „asynchronous JavaScript and XML“. Volání se zpracovávají jako asynchronní, klient tím pádem není v průběhu vyřizování požadavku blokován. Tento asynchronní přístup umožňuje načítat data v pozadí, zatímco funkčnost uživatelského rozhraní readeru není ovlivněna.



Obrázek 6.4. Sekvenční diagram komunikace serveru a klienta

6.3.3 Volání Google Scholar

Jako externí zdroj jsem zvolil Google Scholar. Bohužel Google scholar nevystavuje žádné API. Volání v klientu je prováděno pomocí tlačítka umístěného v kontextové nápovědě.

Klient sestaví URL z názvu článku a zavolá tuto adresu metodou GET, tzn. tváří se jako webový prohlížeč. Odpovědí je celá webová stránka, ze které je třeba dostat požadovaná data. Výsledek je uložen do DOM stromu. Výsledky dotazu jsou ve struktuře

stránky uloženy zvlášť od navigačních prvku a hlavičky, což usnadňuje jejich nalezení. Klient z DOM stromu vybere první výskyt této třídy výsledků. Z výsledku hledání jsou vybrány požadované informace, jako je odkaz na dokument, kolikrát byl článek citován a zkrácený abstrakt. Tyto informace jsou dodány do kontextové nápovědy a tlačítko pro volání Google Scholaru je skryto.

Kapitola 7
Testování

Každý software je třeba řádně otestovat. Jedna z věcí, kterou jsem se při studiu a práci naučil je, že testovat důkladně se vyplatí. Pokud píšete software a dokážete účinně hledat a odstraňovat chyby, ušetříte spoustu trápení sobě i ostatním, kteří budou Váš software používat.

V této kapitole je popsáno testování celé aplikace v několika krocích. Prvníkm rokem jsou unit testy obou částí. Poté na řadu přicházejí uživatelské testy a následují výkonnostní (zátěžové) testy.

7.1 Unit testy serveru

Jednotkové testy ověřují funkčnost jednotlivých tříd v programu. Jedná se o základní kontrolní mechanizmus při vývoji softwaru. Testy jsou psány v frameworku JUnit. Tento framework je již standardem v oblasti jednotkových testů. Jedná se o nejpoužívanější framework zaměřený na jednotkové testy.

Pro každý modul serverové části obsahuje projekt sadu jednotkových testů. Tyto testy ověřují funkčnost jednotlivých tříd a modulu jako celku. Je tedy možné jednoduše zaměnit třídy anebo rovnou celé moduly a jednoduše otestovat funkčnost těchto celků.

7.2 Testy klienta

PDF.js jako projekt vyvíjený komunitou obsahuje svou vlastní sadu testů, která prověřuje fungování jednotlivých celků. Tato testovací sada je nezbytně nutná k fungování projektu jako celku. V pokynech pro přispěvatele je výslovně uvedeno, že každá změna, kterou chce přispěvatel zahrnout do projektu, musí být otestována regresními testy projektu a úspěšně projít. Testovány jsou následující operace:

- load test — je kontrolováno správné načítaní dokumentu
- eq test — testuje renderování stránky vůči vzoru
- text test — testuje textovou vrstvu vůči vzorovým datům
- annotations test — testuje anotační vrstvu vůči vzorovým datům
- fbf test — forward-back-forward test testuje jednoznačnost výstupu
- unit testy — Unit testy ve frameworku Jasmine

Testy se spouští pomocí testovací stránky, která volá příslušné testovací JavaScripty. Tyto scripty testují jednotlivé funkce a výsledek přímo zobrazí na stránce. Unit testy mají zvláštní stránku řízenou frameworkem Jasmine. Zde je možné pouštět testy jednotlivě a případně kontrolovat jejich výpis.

V rámci práce byly přidány Unit testy pro kontrolu volání webových služeb a parsování výsledků.

Dále je prováděna také statická analýza kódu pomocí programu Lint pro JavaScript. Statická analýza kontroluje používání bílých mezer, nových řádků, názvů proměnných a tříd a správné umístění závorek v kódu. Tedy na kód jsou kladeny stylistické nároky udávané směrnicemi projektu.

7.3 Uživatelské testování

Automatizované testování vizualizace klientské aplikace je nemožné bez referenčních dat, vůči kterým by byl porovnáván testovaný stav. Tato data není možné bez funkční aplikace vytvořit. Proto jsem se rozhodl tuto funkcionalitu otestovat pomocí uživatelského testování.

Pro účely uživatelského testování byla vytvořena testovací sada vědeckých článků. Sada článků byla vytvořena náhodným výběrem článků z digitální knihovny IEEE Xplore. Sada obsahuje 40 různých článků. Testování probíhalo následovně:

- Uživatel otevřel článek
- Otevřel konzoly prohlížeče a zkontroloval, zda byl článek načten bez chyb.
- Počkal na načtení dat ze serverů.
- Zkontroloval v konzoli prohlížeče, zda při načítání dat ze serveru nedošlo k chybě.
- Prošel článek a náhodně zkontroloval odkazy, zda zobrazená data souhlasí s referencemi na konci článku.
- U náhodně vybraných článků si vyžádal dodatečné informace o článku pomocí tlačítka.
- Počkal na načtení dodatečných informací a zkontroloval v konzoli prohlížeče, zda při načítání dodatečných dat nedošlo k chybě.
- Zkontroloval správnost dodatečných informací.

U 90% článků byly reference nalezeny, naparsovány a zobrazeny. Ve zbylých 10% nastal alespoň jeden z následujících případů. Reference nebo jejich odkazy nebyly uzavřeny v hranatých závorkách. Sekce referencí nebyla pojmenována „References“. V 20% případů byly odhaleny drobné nepřesnosti v parsování referencí. Nejčastěji se jednalo o chybné parsování jmen autorů, kdy při použití počátečních písmen prvního a prostředního jména bylo toto jméno detekováno jako součást jiného pole. Rovněž diakritika ve jménech způsobuje problémy, protože při parsování dokumentu jsou některá písmena a háček nad nimi vnímány jako dva znaky.

7.4 Zátěžové testy

Zátěžové testy jsem prováděl za pomoci nástroje Apache JMeter. JMeter je open–source nástroj na zátěžové a behaviorální testy aplikací. Tento nástroj je napsaný čistě v Java SE, ale s jeho pomocí se dají provádět testy nejen na aplikacích psaných v Jave. Nástroj umožňuje posílat zprávy celou škálou protokolů a díky JDBC ovladačům je možné vykonávat zátěžové testy i nad databázemi. JMeter umožňuje uživateli sestavit si scénáře, kdy je možné nastavit i odpovědi na zprávy a různé podmínky. JMeter považuji za jeden z nejlepších open–source nástrojů na zátěžové testy.

Zátěžové testy jsem prováděl na notebooku s parametry popsanými v tabulce 7.1.

parameter	hodnota
Procesor	Intel Core i5-2520M
Taktovací frekvence	2,5 GHz
Počet jader	2
Počet vláken	4
Operační paměť	16 GB
Operační systém	Windows 7 64-bit

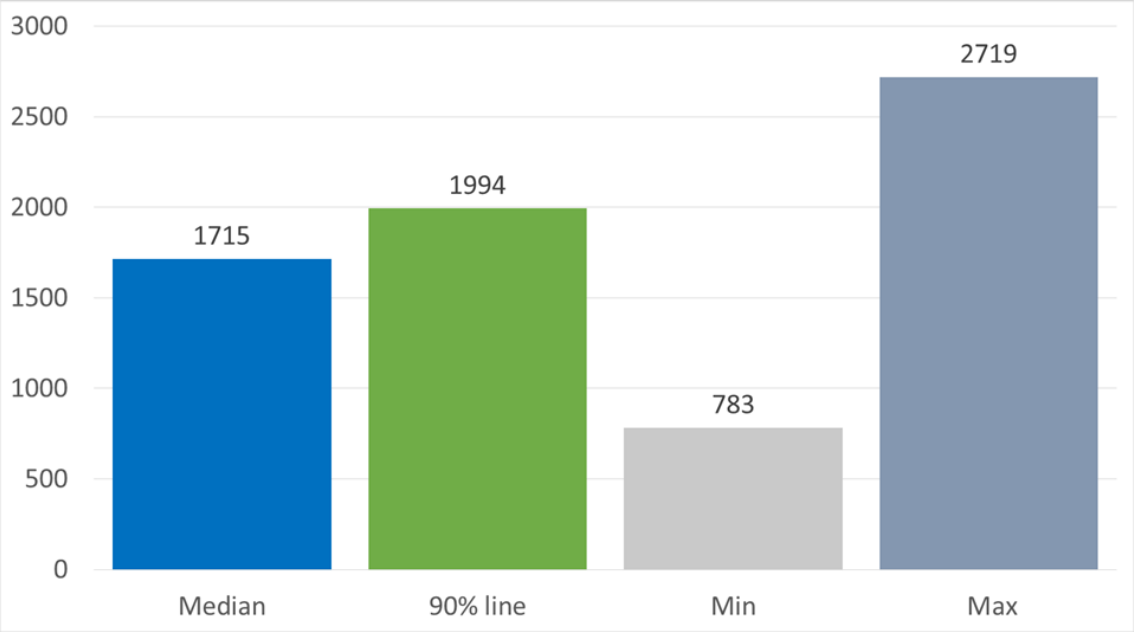
Tabulka 7.1. Specifikace testovací sestavy

Testování spočívalo v posílání různého množství množství zpráv a měření doby odezvy. Každý test má určitý počet vláken posílajících zprávy na server a měřících odezvu. Tato vlákna odešlou každou sekundu jednu zprávu ke zpracování. Každé vlákno pošle 20 zpráv. Vzhledem k tomu, že obě rozhraní jsou webové služby, testování se neliší pro Parser server a server FreeCite. Jediný rozdíl je v posílané zprávě.

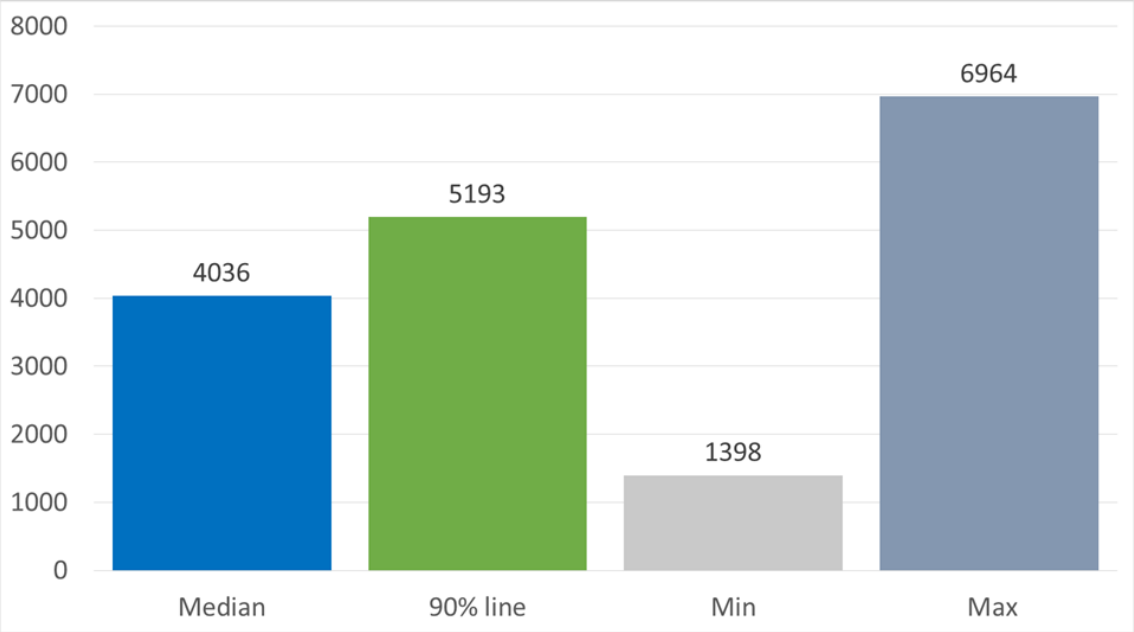
7.4.1 Zátěžové testy Freecite

Nejdříve jsem testoval propustnost aplikace FreeCite. Od propustnosti této aplikace se pak bude odvíjet propustnost celého serveru, protože k odpovědi na každý dotaz se volá i FreeCite. Jede tedy nezbytné vědět jaká je propustnost této aplikace ještě předtím, než začneme testovat celý server jako celek.

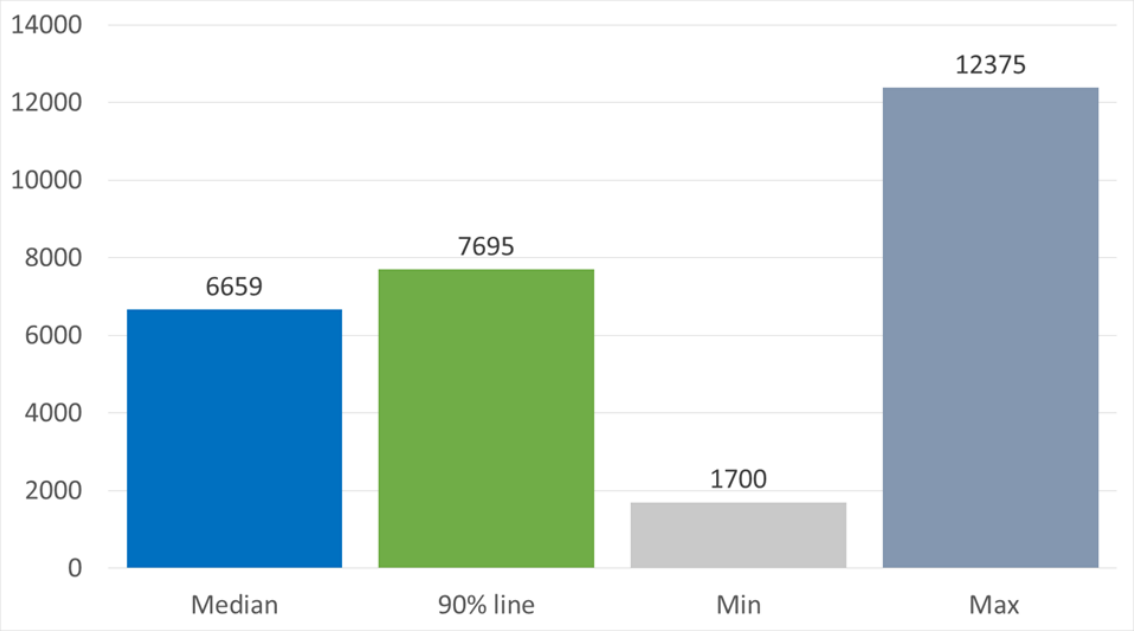
Provedl jsem testy pro 5, 10 a 15 vláken. Zobrazované výsledky jsou v milisekundách.



Obrázek 7.1. Doba odpovědí FreeCite pro 5 vláken



Obrázek 7.2. Doba odpovědí FreeCite pro 10 vláken



Obrázek 7.3. Doba odpovědí FreeCite pro 15 vláken

Threads	Samples	Average	Median	90% line	Min	Max	Througput
5	100	1703	1715	1994	783	2719	1,846/sec
10	200	3976	4036	5193	1398	6964	1,984/sec
15	300	6622	6659	7695	1700	12375	1,933/sec

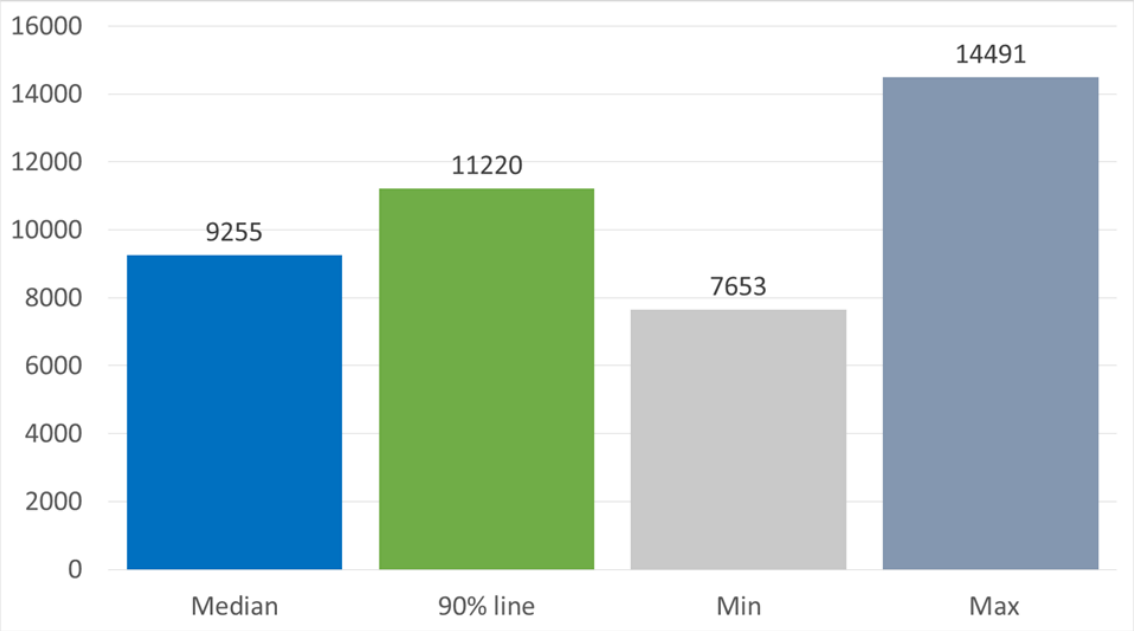
Tabulka 7.2. Zátěžové testy FreeCite

Z grafů můžeme vidět lineární nárůst latence serveru oproti počtu přicházejících požadavků. Z tabulky 7.2 je jasně vidět, že propustnost serveru FreeCite je přibližně stejná ve všech třech testovaných případech, ale požadavky se hromadí a proto narůstá doba, po kterou čeká požadavek na obsloužení.

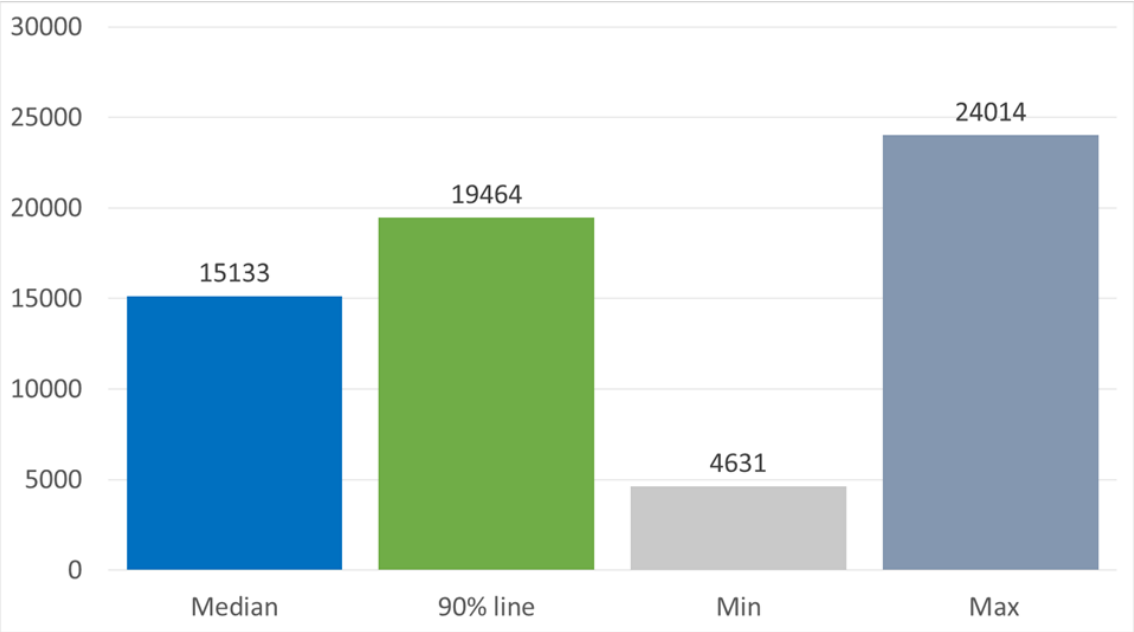
7.4.2 Zátěžové testy Serveru

Server jsem testoval jako celek včetně volání serveru Freecite. Toto volání jsem zahrnul do zátěžového testu, protože doba odpovědi serveru FreeCite je závislá rovněž na počtu příchozích zpráv. Mockovaním odpovědi z FreeCite bych mohl dostat zkreslené výsledky.

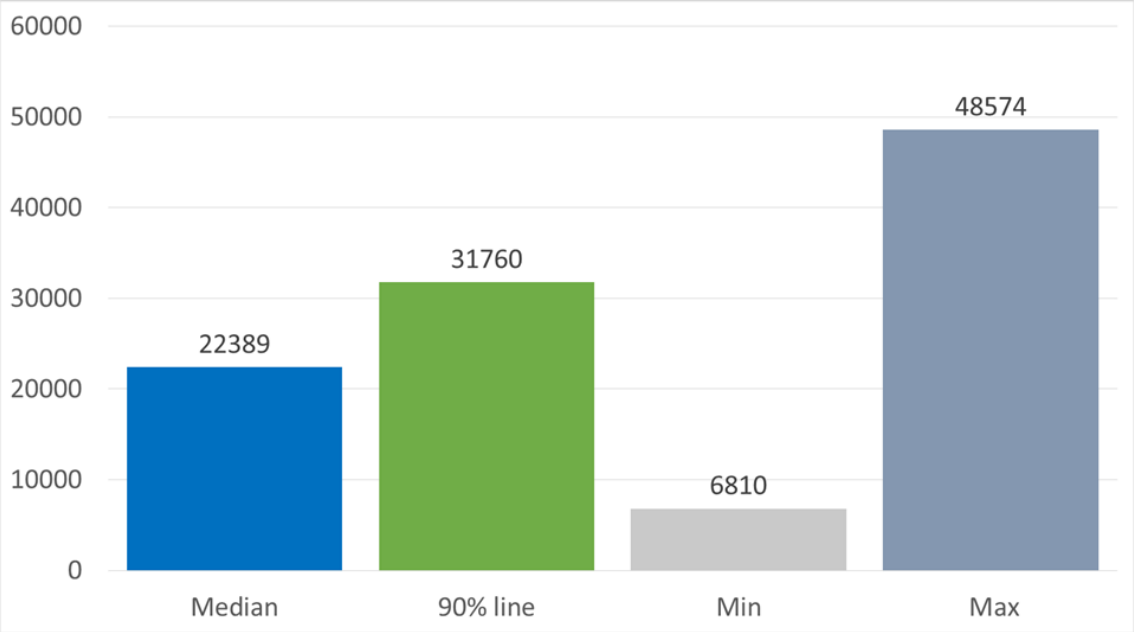
Provedl jsem testy pro 5, 10 a 15 vláken. Zobrazované výsledky jsou v milisekundách.



Obrázek 7.4. Doba odpovědí Serveru pro 5 vláken



Obrázek 7.5. Doba odpovědí Serveru pro 10 vláken



Obrázek 7.6. Doba odpovědí Serveru pro 15 vláken

Threads	Samples	Average	Median	90% line	Min	Max	Througput
5	100	9698	9255	11220	7653	14491	0,461/sec
10	200	15526	15133	19464	4631	24014	0,587/sec
15	300	23278	22389	31760	6810	48574	0,580/sec

Tabulka 7.3. Zátěžové testy serveru

Z grafů můžeme vidět nárůst latence serveru oproti počtu přicházejících požadavků. Z tabulky 7.3 je jasně vidět, že propustnost Parser serveru je přibližně stejná ve všech třech testovaných případech, ale požadavky se hromadí a tím narůstá doba, po kterou čeká požadavek na obsloužení.

Kapitola 8
Budoucí práce

V této kapitole je popsána vize možného dalšího vývoje této aplikace. Jedná se hlavně o návrhy, které by aplikaci posunuly o další kus blíže reálnému použití nebo o funkcionalitu, kterou jsem neimplementoval, ať už z časových či jiných důvodů. Návrhy jsou v kapitole řazeny sestupně podle míry přínosu. Míra přínosu je do jisté míry subjektivní.

8.1 Nastavení klienta

Momentálně klient nepodporuje změnu adresy serveru bez zásahu do kódu. Součástí klienta by dle mého názoru měla být možnost tuto adresu změnit. Adresa serveru je momentálně nastavená na localhost, tedy je počítáno, že server poběží na stejném stroji jako klient. Pro reálné použití je tuto adresu třeba nastavit na veřejnou doménu, na které server poběží, nebo na adresu serveru v lokální síti. Rovněž pokud by existovalo více veřejných serverů, měl by uživatel mít možnost si vybrat server se nejnižší latencí a nemusel by spoléhat na přednastavený server, či případně měnit zdrojový kód doplňku.

Tuto část většinou obstarává kód doplňku samotného, protože pracuje s lokálním úložištěm. Funkční část readeru by tedy touto funkcionalitou neměla být dotčena, aby byla zachována jednoduchá portovatelnost mezi prohlížeči.

8.2 Správce dokumentů na disku a Kešování výsledků

Jednou z hlavních myšlenek byla možnost implementovat v rámci doplňku do prohlížeče vlastní adresář článků. Doplňek by měl přehled o souborech na disku, které by uživatel buď do readeru importoval nebo otevřel pomocí readeru. Reader by udržoval data získaná o dokumentu, jejich cestu a kontroloval jejich kontrolní součet při každém otevření. Při změně kontrolního součtu by reader zavolal server a data updatoval. Data by se udržovala v odkládacím souboru v adresářové struktuře prohlížeče. Také by mělo být možné vymazat data uložená o dokumentech.

Při otevření doplňku by klient zobrazil stranu se seznamem dokumentů. Místo názvu souboru by doplňek zobrazoval jméno souboru získané ze serveru.

Tato funkcionalita by zaručila použitelnost celého řešení i při nedostupnosti serveru nebo výpadku internetu. Bohužel z časové náročnosti tohoto řešení a neznalosti API prohlížeče Chrome jsem tuto funkcionalitu neimplementoval.

8.3 Citace nejen nejen formátu IEEE

V současné době je vyhledává server pouze odkazy na citace a citace, které jsou dle směrnic pro citování *IEEE* [21]. Bohužel to omezuje využití této aplikace pouze na obory elektroniky a počítačů a další přidružené obory, které berou tento způsob citace jako normu. Pro ostatní styly citací je potřeba upravit funkcionalitu detekce odkazů a

detekce jednotlivých referencí v sekci referencí. Funkcionalitu parsování na jednotlivá pole není třeba upravovat, ta je na reference jiných stylů plně připravena.

8.4 Škálování

Testování ukázalo, že aplikace má dvě úzká hrdla. Prvním je převedení dokumentu formátu PDF na text. Tento úkon je výpočetně náročný a při velkém počtu dotazů poslaných současně dochází k vypršení některých dotazů. V tomto případě optimalizace algoritmu není možná. Je nutné tedy škálovat aplikaci. Aplikace je bezestavová, není tedy problém ji škálovat vertikálně či horizontálně. Při vertikálním škálování by bylo potřeba dodat více výpočetního výkonu pro zrychlení převodu PDF do textové formy. Horizontální škálování pomocí přidání více instancí serveru není problém, protože aplikační server WildFly tu to možnost podporuje a jedná se pouze o konfiguraci více aplikačních serverů a jejich spojení do clusteru. Přerozdělování jednotlivých dotazů pak řeší cluster sám.

Druhým úzkým hrdlem je dotazování na server služby FreeCite. Tento problém se dá vyřešit zřízením vlastní instance tohoto serveru. Zde není problém s vertikálním škálováním instance, podobně jakou u Parser serveru. Bohužel FreeCite není možné jednoduše škálovat horizontálně. Bylo by nutné tedy spustit několik instancí FreeCite na několika strojích. Loadbalancing by byl buď statický, tedy v závislosti na rychlosti zpracování dokumentů Parser serverem, nebo dynamický. Statický loadbalancing by probíhal způsobem, kdy pro každých n nodů parser serveru byla jedna instance FreeCite serveru. Tedy jednotlivé nody by měly nastavenou pevnou adresu svého FreeCite serveru. Což bohužel může vést k přetížení jednotlivých instancí Freecite serverů, nebo dokonce částečné nedostupnosti služby, pokud selže některý ze FreeCite serverů. Dynamické loadbalancing by nadruhou stranu vyžadoval vytvořit vlastní loadbalancer, který by jednotlivé nody spravoval a přeposílal by zprávy na méně vytížené servery.

8.5 Design

Během práce na klientské aplikaci jsem se snažil především vyvinout funkcionalitu do maximálně funkčního stavu. Bohužel na úkor celkového designu aplikace. Kromě kompletního designu uživatelských menu, pro potřeby vývoje zmíněného v této kapitole, je potřeba předělat, po grafické stránce, zobrazování kontextové nápovědy a dodatečných informací. V současné době je zobrazování výsledků z Google Scholaru nepřehledné. Tato práce nebyla zaměřena na uživatelské rozhraní a je potřeba provést celkový grafický návrh.

8.6 Statistika výsledků

Za užitečnou vlastnost systému považuji jeho vlastní analýzu provozu na serverové části. Není neobvyklé, že analýzou dat se zabývají nástroje třetích stran, které bývají placené. Informace, které je dle mého názoru užitečné sledovat, jsou čas příchodu zprávy, čas volání FreeCite, čas odpovědi freecite, velikost příchozího dokumentu, počet citací v dokumentu a počet odkazů v dokumentu. Na základě těchto dat se dá sestavit zátěž serveru v průběhu dne, nebo také analyzovat, zda uživatelé používají nástroj pro čtení krátkých vědeckých článků, či delších publikací. Pro tuto funkcinalitu záznamu dat jsem zvažoval následující dva scénáře.

První možností je, že systém bude pouze shromažďovat data o provozu. Data by byla ukládána do speciálního logu a interpretace by pak byla na provozovateli sytému. Pro zjednodušení následného zpracování by data byla ukládána ve formátu CSV.

Druhá možnost je zaznamenávat data do databáze a provádět vlastní analýzu přímo v aplikaci. To by vyžadovalo implementaci nejen jednotlivých funkcí pro výpočet statistik a analýz, ale také jejich vizualizaci. Vzhledem k tomu, že server je založený na platformě Java EE, navrhoval bych vytvoření webového rozhraní s jednoduchou autentizací pro zobrazení grafu a statistik.

Kapitola 9
Závěr

V rámci práce se mi podařilo vytvořit funkční nástroj pro zjednodušení čtení vědeckých článků. Tento nástroj poskytuje informace o referenci v místě odkazu na ní a je schopný na vyžádání získat další informace ze služby Google Scholar.

V rámci této práce jsem se seznámil s formátem PDF a problematikou strojového zpracování dokumentů formátu PDF. Dále jsem byl obeznámen se způsoby řešení těchto problémů. Většina řešení těchto problémů je postavena na formě strojového učení, což je bohužel obor mimo moji specializaci. V rámci řešení problémů jsem však tato řešení použil jako již hotové programy, které byly zakomponovány do celkové aplikace.

Bohužel kvůli časovým problémům a komplikacím, s kterými jsem se potýkal při psaní této práce, jsem nebyl schopný implementovat 100% funkcionality, kterou jsem původně plánoval. Nápady a návrhy jsem sepsal do kapitoly „Budoucí práce“. Rovněž aplikace není zcela bez chyb. Většina chyb je způsobena extrakcí textu z PDF dokumentu. Tyto chyby se nepodařilo odstranit úplně a dochází tak k překlepům nebo nepřesnostem v referencích.

Literatura

[1] Adobe Systems Incorporated. *Document Management — Portable Document Format — Part 1: PDF 1.7*. ISO 32000-1:2008. 2008 .

[2] Mendeley Ltd. *Mendeley Desktop*.
`https://www.mendeley.com/`.

[3] Inc. Labtiva. *ReadCube Desktop*.
`https://www.readcube.com`.

[4] Philip N. Smith a David F. Brailsford. Towards structured, block-based PDF. *Electronic Publishing – Origination, Dissemination and Design*. 1995, 8 (3), 153–165. Copyright transferred to Univ. of Nottingham in 1998..

[5] Hui Chao, Giordano Beretta a Henry Sang. *PDF document layout study with page elements and bounding boxes*. In: *Workshop on document layout interpretation and its applications (DLIA2001)*. 2001.

[6] ”Chao. In: *”Document Analysis Systems VI: 6th International Workshop*. ”Berlin: ”Springer Berlin Heidelberg”, ISBN ”978-3-540-28640-0”.
`"http://dx.doi.org/10.1007/978-3-540-28640-0_20"` .

[7] William S Lovegrove a David F Brailsford. Document analysis of PDF files: methods, results and implications. *Electronic Publishing–Origination, Dissemination and Design*. 1995, 8 (3), 207–220.

[8] Y. Horowitz a N. Arazi. *System and method for relating unstructured data in portable document format to external structured data*. 2008.
`https://www.google.com/patents/US20080084573`. US Patent App. 11/548,274.

[9] Jay Berkenbilt. *QPDF*.
`http://qpdf.sourceforge.net`.

[10] Yusuke Shinyama. *PDFMiner*.
`https://euske.github.io/pdfminer/`.

[11] The Apache Software Foundation. *Apache PDFBox*.
`https://pdfbox.apache.org/index.html`.

[12] A. Takasu. *Bibliographic attribute extraction from erroneous references based on a statistical model*. In: *2003 Joint Conference on Digital Libraries, 2003. Proceedings.*. 2003. 49-60.

[13] Asanee Kawtrakul a Chaiyakorn Yingsaeree. *A unified framework for automatic metadata extraction from electronic document*. In: *Proceedings of The International Advanced Digital Library Conference. Nagoya, Japan*. 2005.

[14] ”Min-Yuh Day, Richard Tzong-Han Tsai, Cheng-Lung Sung, Chiu-Chen Hsieh, Cheng-Wei Lee, Shih-Hung Wu, Kun-Pin Wu, Chorng-Shyong Ong a Wen-Lian Hsu”. ”Reference metadata extraction using a hierarchical knowledge representation framework ”. *”Decision Support Systems ”*. ”2007”, ”43” (”1”), ”152 - 167”. DOI ”http://dx.doi.org/10.1016/j.dss.2006.08.006”. ”Mobile Commerce: Strategies.

[15] ”Gupta. In: ”Berlin: ”Springer Berlin Heidelberg”, ISBN ”978-3-642-03547-0”. "http://dx.doi.org/10.1007/978-3-642-03547-0_10" .

[16] Pere Constans. A Simple Extraction Procedure for Bibliographical Author Field. *CoRR*. 2009, abs/0902.0755

[17] Isaac G Councill, C Lee Giles a Min-Yen Kan. *ParsCit: an Open-source CRF Reference String Parsing Package.*. In: *LREC*. 2008.

[18] Public Display Inc. *FreeCite*. http://freecite.library.brown.edu/welcome.

[19] LINQS. *Cora Dataset*. http://www.cs.umd.edu/~sen/lbc-proj/LBC.html.

[20] Mike Jewell. *ParaCite*. http://paracite.eprints.org.

[21] IEEE. *IEEE Citation Reference*. 2007. https://www.ieee.org/documents/ieeecitationref.pdf.

Příloha A
Instalační Manuál

V této části bude popsán postup jak nasadit obě části aplikace.

A.1 Klient

- Otevřete nastavení prohlížeče Chrome na adrese chrome://extensions/
- Zaškrtněte Developer mode
- Klikněte na Load unpacked extension
- Otevřete složku pdf.js/extensions/chromium
- Nyní je možné otevřít klientskou aplikaci v prohlížeči

A.2 Server

Pro funkcionalitu serveru je nutné stáhnout aplikační server a aplikaci na něj nasadit.

- Stáhněte WildFly 10 ze stránky http://wildfly.org/downloads/
- Rozbalte server
- Spusťe server prikazem Standalone.sh nebo Standalone.bat
- V prohlížeči otevřete stránku http://localhost:9990/console/App.html
- Přejděte na záložku Deployment
- Nasaďte ParserServer-1.0-SNAPSHOT.war

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Novotný Tomáš

Studijní program: Otevřená informatika
Obor: Softwarové inženýrství

Název tématu: Uživatelské rozhraní pro čtení vědeckých článků

Pokyny pro vypracování:

Proveďte podrobnou rešerši vědecké literatury zaměřené na extrakci informací z odborných textů. Zaměřte se převážně na parsování referencí, detekci nadpisů a dalších částí jako jsou tabulky a obrázky.
Proveďte rešerši a srovnání podobných řešení (aplikací).
Navrhněte architekturu systému pro extrakci metadat a jejich vizualizaci v dokumentu
Navrhněte a implementujte algoritmus pro extrakci metadat z vědeckého článku.
Implementujte vizualizační nástroj - tento nástroj, který bude zobrazovat a organizovat odbornou literaturu na mobilním zařízení nebo na PC ve webovém prohlížeči
Otestujte extrakční algoritmy a vizualizační nástroj na testovací sadě vědeckých článků v PDF.

Seznam odborné literatury:

[1] ISO 32000-1. Document Management Portable Document Format Part 1: PDF 1.7. Adobe, 2008.
[2] Alves, N. F., Lins, R. D., & Lencastre, M. 2012. A Strategy for Automatically Extracting References from PDF Documents. 2012 10th IAPR International Workshop on Document Analysis Systems.
[3] Gupta D., Morris B., Catapano T., Sautter G. (2009) ?A New Approach towards Bibliographic Reference Identification, Parsing and Inline Citation Matching?. In: Ranka S. et al. (eds) Contemporary Computing. IC3 2009. Communications in Computer and Information Science, vol 40. Springer, Berlin, Heidelberg
[4] Min-Yuh Day, Richard Tzong-Han Tsai, Cheng-Lung Sung, Chiu-Chen Hsieh, Cheng-Wei Lee, Shih-Hung Wu, Kun-Pin Wu, Chorng-Shyong Ong, Wen-Lian Hsu, Reference metadata extraction using a hierarchical knowledge representation framework, Decision Support Systems, Volume 43, Issue 1, February 2007, Pages 152-167, ISSN 0167-9236, http://dx.doi.org/10.1016/j.dss.2006.08.006.
[5] Chao,H. and Fan, J., 2004, September. Layout and content extraction for pdf documents. In International Workshop on Document Analysis Systems (pp. 213-224). Springer Berlin Heidelberg.

Vedoucí: Ing. Jan Šedivý, CSc.

Platnost zadání do konce letního semestru 2017/2018

L.S.

prof. Dr. Michal Pěchouček, MSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 20.2..2017