UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DIPLOMOVÁ PRÁCE

2010 Bc. Jakub Čapský

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Systém pro ověřování plagiátorství prostřednictvím vyhledávací služby Google

Bc. Jakub Čapský

Diplomová práce
2010

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2009/2010**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jakub ČAPSKÝ**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Systém pro ověřování plagiátorství prostřednictvím vyhledávací služby Google**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

V úvodní části práce bude představena problematika analýzy obsahu ve vazbě na potřeby konstrukce dotazu pro vyhledávací služby. Pro potřeby vyhledávání budou vytvořeny algoritmy, které budu analyzovat vstupní texty a na základě kterých budou vytvořeny dotazy pro vyhledávací službu Google.Výsledkem diplomové práce bude vytvoření plně hostované aplikace, která je schopna jednoduchým a přehledným způsobem předkládat výsledky vyhledávání podezřelých míst. Aplikace by měla být schopna pracovat s více typy vstupních souborů (txt, doc, pdf). Práce my měla ověřit možnosti kontroly plagiátorství prostřednictvím vyhledávací služby Google. Diplomová práce by měla odpovědět na otázku, zda je možné vyhledávat plagiáty v celosvětové síti na rozdíl od ověřování plagiátorství vůči centrálnímu repositáři.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. WEST, MARK D. Theory, Method, and Practice in Computer Content Analysis. Westport, Ablex publishing, 2001 2. On-line: http://code.google.com/apis/ajaxsearch/documentation/

Vedoucí diplomové práce: **Ing. Lukáš Čegan**
Katedra informačních technologií

Datum zadání diplomové práce: **30. října 2009**

Termín odevzdání diplomové práce: **21. května 2010**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

doc. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 16. listopadu 2009

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně Univerzity Pardubice.

V Pardubicích dne 17.8.2010 Bc. Jakub Čapský

Poděkování:

Rád bych poděkoval vedoucímu práce Ing. Lukáši Čeganovi, PhD. za jeho čas, ochotu a profesionální přístup, který mi poskytnul při řešení standardních i nestandardních problémů při zpracování diplomové práce.

SOUHRN

Práce se zabývá problematikou plagiátorství zejména, avšak nejen, pro potřeby vysokých škol. Vysvětluje základní postupy a principy při odhalování plagiátů a seznamuje s již existujícími softwarovými produkty, které danou problematiku řeší. V praktické části práce je vytvořena komplexní webová aplikace, využívající celosvětovou síť Internet a vyhledávací službu Google, která má sloužit pro odhalování plagiátorství. Sekundárním cílem práce je zároveň odpovědět na otázku, zdali je možné a efektivní vyhledávat plagiáty v celosvětové síti, vůči systémům které fungují na principu centrálního repozitáře.

KLÍČOVÁ SLOVA

Plagiátorství, Detekce, Odhalování, Nelegální kopie, Autorská práva, Citace, Analýza obsahu, Webová aplikace, Kompilátor, Framework, Google API, HTML, CSS, JavaScript, PHP, AJAX, ExtJS, RIA, XML, XSD, XSLT, UML

TITLE

System for verification plagiarism by search service Google

ABSTRACT

The work is concerned with problematic of plagiarism especially, but not only, for needs of university. It explains basic sequences and principles within detection of plagiarism and introduce to yet existing software's products, which this problematic solves. In the practical section of this work it's developed complex web application, which utilizes worldwide network Internet and search service Google, and it should be useful for detection of plagiarism. Secondary purpose of this work is answer to question, if it's possible and effective to search plagiarism in worldwide network, compared with systems, which are based on central repository.

KEYWORDS

Plagiarism, Detection of plagiarism, Illegal copy, Copyrights, Citation, Analysis of content, Web application, Compiler, Framework, Google API, HTML, CSS, JavaScript, PHP, AJAX, ExtJS, RIA, XML, XSD, XSLT, UML

OBSAH

Úvod

Hlavním cílem této diplomové práce je objasnit problematiku plagiátorství, zejména pak jeho negativními dopady a také příčinami a řešeními tohoto problému. Dále je snaha seznámit s nejznámějšími existujícími softwarovými nástroji a metodami, které jsou určeny pro odhalování plagiátorství.

Po praktické části, má být výsledkem této práce softwarový produkt, jakožto vyhledávací systém, který má sloužit k odhalování plagiátů pomocí sítě Internet a vyhledávací služby Google. Tento systém má podporovat několik vstupních formátů (testovaného dokumentu) a obsahovat metody určené k analýze obsahu dokumentu. Nový produkt, resp. vyhledávací systém by zároveň měl mít uživatelsky příjemné a intuitivní rozhraní a samotný proces ověřování existence plagiátorství provádět pokud možno co nejrychleji a nejefektivněji. Tato práce má tedy odpovědět na otázku, zdali je možné a účinné odhalovat plagiáty prostřednictvím celosvětové sítě Internet, na rozdíl od konvenčního přístupu, který pro účely detekce plagiátorství využívá lokální archív dokumentů.

V současné době se předpokládá, že systém by prozatím měl sloužit pro účely pedagogických pracovníků, kteří tak budou schopni jednoduchou formou ověřit, zdali jim odevzdaná semestrální, bakalářská či jiná akademická práce není plagiátem. V neposlední řadě je cílem této práce poskytnout kvalitní uživatelskou a také programátorskou příručku k novému systému, tak aby jeho instalace, používání a případné budoucí rozšiřování bylo pokud možno co nejjednodušší a bez komplikaci.

1 Problém plagiátorství

Plagiátorství je nezanedbatelným problémem dnešní společnosti, který zasahuje nejen do školské sféry. Jedná se o **všeobecný problém**, který primárně vzniká ve světě duševního vlastnictví. Můžeme konkretizovat na modelových příkladech.

Nejmenovaný automobilový výrobce si objedná a zaplatí věhlasného designéra, aby pro něj navrhl nový model automobilu. Automobil se, po uvedení na trh, stane rychle oblíbeným a automobilka z něj dosahuje vysokých zisků. Problém však nastává tehdy, pokud jiný výrobce automobilů, na druhém kontinentě, se rozhodne tohoto cizího úspěchu využít ve svůj prospěch tak, že model převezme, provede kosmetické úpravy a považuje jej za vlastní. Co se stane? Jelikož jsou oba modely aut od různých výrobců téměř shodné a parazitující výrobce s největší pravděpodobností bude svůj model prodávat s nižší cenou, bude výrobce, který přišel s modelem na trh jako první, přicházet o nemalou část svých zisků.

Druhý příklad plagiátorství může pocházet ze světa hudebního průmyslu. V tomto odvětví jsou totiž praktiky daleko důmyslnější a hůře napadnutelné, než by se mohlo na první pohled zdát. Také se Vám někdy stalo, že jste slyšeli píseň, jejíž melodii jste už někdy dříve slyšeli? Pokud ano, pravděpodobně jste neposlouchali autora, ale plagiátora.

Obr. 1 – Plagiát designu staré kalkulačky jako SW produkt (zdroj: Oyayubizoku)

Zdaleka největším problémem je však definovat, kdy se jedná o skutečný plagiát a kdy je shoda dvou produktů (ať už materiálních či duševních) způsobena jiným ovlivňujícím faktorem. U duševního vlastnictví může být tímto faktorem např. skutečnost, že daná informace je považována za všeobecné známý fakt.

Tento problém např. řeší autorský zákon. Ten říká, že: „*Autorem je fyzická osoba, která dílo vytvořila, nebo v případě souborných děl (resp. databází) uspořádala či vybrala, což je ale nutno činit tvůrčím způsobem, protože jinak by nešlo o autora, ale plagiátora*“ [20]. Důležité je zde zejména dvousloví „tvůrčím způsobem“, které sděluje, že autor bezesporu musel na daném díle zanechat svůj čas a úsilí.

Plagiátorství je definováno i v ČSN pod označením **ISO 5127-2003**. Její doslovné znění je následující: „*Představení duševního díla jiného autora půjčeného nebo napodobeného v celku nebo z části, jako svého vlastního*“ [6]. V této definici, byť se to na první pohled nemusí zdát, je nejpodstatnější poslední trojsloví. Je zde totiž chápáno ve smyslu zatajení, resp. neuvedení autora původního díla. Jinými slovy řečeno; použít cizí vědomosti, myšlenky nebo poznatky lze pouze za předpokladu, že neopomeneme uvést původního autora.

1.1 Příčina problému

S odkazem na první řádky úvodní kapitoly, je potřeba uvést, že největší koncentrace plagiátorství se nachází v oblasti školství a studia všeobecně. Proč je tomu tak, lze objektivně dokázat dvěmi skutečnostmi.

První je založena na faktu, že lidstvo se neustále vyvíjí a to, co se dnes objeví nebo vymyslí, může být za krátkou dobu několikrát zdokumentováno a poměrně snadno přístupno. Zjednodušeně řečeno, na spoustu otázek už se dávno našla odpověď a tu již není potřeba, z pohledu studenta, hledat znovu. Je zřejmé, že tato skutečnost neplatí pro všechny studijní obory stejnou mírou. Existují obory, např. z odvětví filozofie, které mají své kořeny poměrně v dávných dobách, a tudíž nalezení odpovědi na většinu témat, je pouze jistým „přežvýkáním“ odpovědí již dříve nalezených. Student tak již z principu ztrácí chuť vymýšlet „vymyšlené“ a přichází o tak důležité složky studia jako je **motivace** a smysluplný cíl.

Obr. 2 – „Objevovat Ameriku“ není motivující ani efektivní (zdroj: Ctemploymentlawblog.com)

Na druhou stranu existují obory, např. z odvětví informačních technologií, které se rozvíjejí dynamičtěji, mají poněkud širší záběr a zadávat tak nová témata k vyřešení by pro kantory neměl být problém.

Druhou skutečností, je všeobecně známý fakt, že tím jak **rostou informační technologie**, rostou i možnosti jejich použití. V minulosti, když ještě neexistovaly nebo nebyly tolik dostupné osobní počítače, samozřejmě také docházelo k plagiátorství, ovšem k jeho uskutečnění bylo potřeba daleko větší úsilí. V dnešní době, kdy na internetu existuje nespočet elektronických archívů plných dokumentů (semestrálních prací, bakalářských a diplomových prací, atd.) není problém si dokument s požadovaným tématem vyhledat a metodou „copy-paste“ si tak usnadnit svoji práci.

Obr. 3 – S novými technologiemi, vznikají nové možnosti (zdroj: Codinghorror.com)

Za konkrétními, ne globálními, příčinami problému plagiátorství může stát např. **systém zadávání prací**. Pokud uvážíme situaci, kdy škola studenty v daném školním období „bombarduje“ různými seminárními pracemi, bývá mnohdy problém, všechny práce samostatně vypracovat a do daného termínu odevzdat. Při velkém počtu zadaných prací díky tomu neprojdou dva ze tří typů studentů. Těmi prvními jsou extrémně líní studenti, kteří ani nejsou schopni požadované téma najít a zkopírovat a druhým typem jsou bohužel poctiví studenti, kteří v záplavě prací a velkém stresu nestihnou všechny práce odevzdat.

Jediní, kdo dané požadavky „splní“ jsou ti studenti, kteří v tom umějí chodit a ani je moc netrápí, že se vědomě dopouští plagiátorství. Školu tedy opouštějí podprůměrní absolventi a výsledkem je pak **celková degenerace studia**.

1.2 Řešení problému

Je pochopitelné, že řešení tohoto problému se nebude hledat jednoduše. Důležité je si však uvědomit, že pokud nechceme, aby se plagiátorství stále víc a víc rozrůstalo, je potřeba studenty určitým způsobem varovat a informovat o daných nebezpečí. Hlavním nástrojem je tedy **prevence**.

Jedním takovým řešením je, aby studenti již od dětství byli vedeni k vlastní tvorbě a zároveň by byli upozorňování na problematiku plagiátorství. Je potřeba si ovšem uvědomit, že situace, kdy student při psaní zadané práce by měl používat pouze vlastní myšlenky, je téměř nereálná. Především je tedy nutné studenty naučit, jak správně používat cizí myšlenky, aby se nedopouštěli plagiátorství.

Definice říká, že plagiátem se dílo stává i tehdy, pokud s původní citace dojde k jedné z následujících tří operací a zároveň nikde není uveden původní autor:

- *Vypuštění vět nebo jednotlivých slov*
- *Záměně pořadí slov ve větě*
- *Nahrazení slova jeho synonymem*

Možnosti jak použít informace, které nepocházejí z naší hlavy, spočívá v použití:

1. **Přímé citace** – postup kdy daný text beze změny zkopírujeme, vložíme do uvozovek a za uzavírající uvozovku umístíme odkaz (nejlépe číselný) na původní zdroj informace. Také je vhodné daný text vizuálně odlišit od ostatního, např. použitím *kurzívy*. V případě, že je text rozsáhlejší, je vhodné jej umístit do samostatného odstavce.
2. **Nepřímé citace** (též „parafráze“) – znamená vyjádření stejného obsahu jiným způsobem. Získanou informaci si upravíme do námi požadované podoby, popř. do ní přidáme vlastní myšlenky. Původní zdroj informace je potřeba uvést v seznamu zdrojů na konci celé práce.
3. **Obecně známá fakta** – informace, které jsou známy široké veřejnosti. Může jim být např. informace: „Společnost Microsoft vypustila do světa nový operační systém Windows 7“. Na druhou stranu, sdělení typu: „Tržby ze systému Windows 7 dosahují v tomto čtvrtletí 100 mld.“ je již konkrétní údaj a je potřeba jej doložit zdrojem.

Dalším způsobem, jak se dopředu vyhnout plagiátorství, je upravit systém zadávání prací pro studenty. V případě, že by kantor při zadávání práce specifikoval i zdroj (nebo okruh zdrojů), ze kterého má být práce vytvořena, student by se musel, již od začátku tvorby, na daný zdroj soustředit.

V neposlední řadě existuje i řešení, které má především odstrašující účinek. A tj. zavedení informačních systémů, které by případné plagiátorství odhalovaly. Při jejich existenci je zřejmé, že by studenty, kteří chtějí profitovat na cizím úsilí, velmi znepokojovalo případné odhalení plagiátu a následné sankce.

1.3 SW systémy pro podporu odhalování plagiátorství

Boj proti plagiátorství, který má být efektivní, by se v dnešní době již těžko obešel bez nástrojů, které dokáží případné plagiátorství odhalovat. Tyto nástroje fungují na principu porovnávání (testování na shodu) textového obsahu souborů.

Aby mohl být podezřelý soubor testován, zdali náhodou není plagiátem, je potřeba mít k dispozici pokud možno co nejrozsáhlejší registr již hotových prací.

V dnešní době existuje řada systémů, které pracují s tzv. **centrálním repozitářem**. Jedná se o elektronický archív, tj. databázi odevzdaných prácí, do které postupem času neustále přibývají nové a nové práce. Z hlediska dynamiky je tedy tento přístup *expanzivní*, tj. plynutím času se dá předpokládat, že se datový objem tohoto registru bude zvětšovat. Výhodou tohoto archívu je možnost nastavit přístup buď jako *veřejný* nebo *soukromý*. V případě, že je nastaven jako soukromý, budou práce v něm obsažené přístupné pouze pro potřeby oprávněných osob, jako jsou např. kantoři nebo vedení příslušného orgánu. Pokud bude nastaven jako veřejný, můžou být (v závislosti na dalších specifikací) všechny práce přístupné nejenom pro studenty, ale i pro širokou veřejnost. Tímto však rapidně rostou bezpečnostní rizika a vzniká tak rodná půda pro možné plagiátorství.

Pokud se tedy zamyslíme nad charakterem centrálního repozitáře, jakožto databází elektronických dokumentů je možné, aby touto databází byl celý **Internet**? Jelikož máme k dispozici sofistikované vyhledávací nástroje, jako např. Google, Yahoo nebo Bing, potom by to neměl být po technické stránce problém. Otázkou však zůstává, zdali to bude mít pro nás nějaký přínos. Je zřejmé, že Internet jako takový je největší databází, jaká může být. Obsahuje nepřeberné množství informací a dokumentů nejenom ze školských institucí, které své práce zveřejňují, ale i od ostatních autorů, kteří samozřejmě také nechtějí, aby byla jejich duševní vlastnictví zneužívána. Toto je bezesporu velký potenciál a přínos tohoto řešení. Z hlediska dynamiky datového objemu se dá Internet hodnotit jako nestabilní, či *nepředvídatelný*. Nikdo totiž nemůže zaručit, že dokument který se na dané webové stránce nachází dneska, tam bude i zítra. Přístupnost k daným dokumentům a informací je však stejně variabilní jako u předchozího řešení, tj. můžou být jak *veřejné* tak *soukromé*.

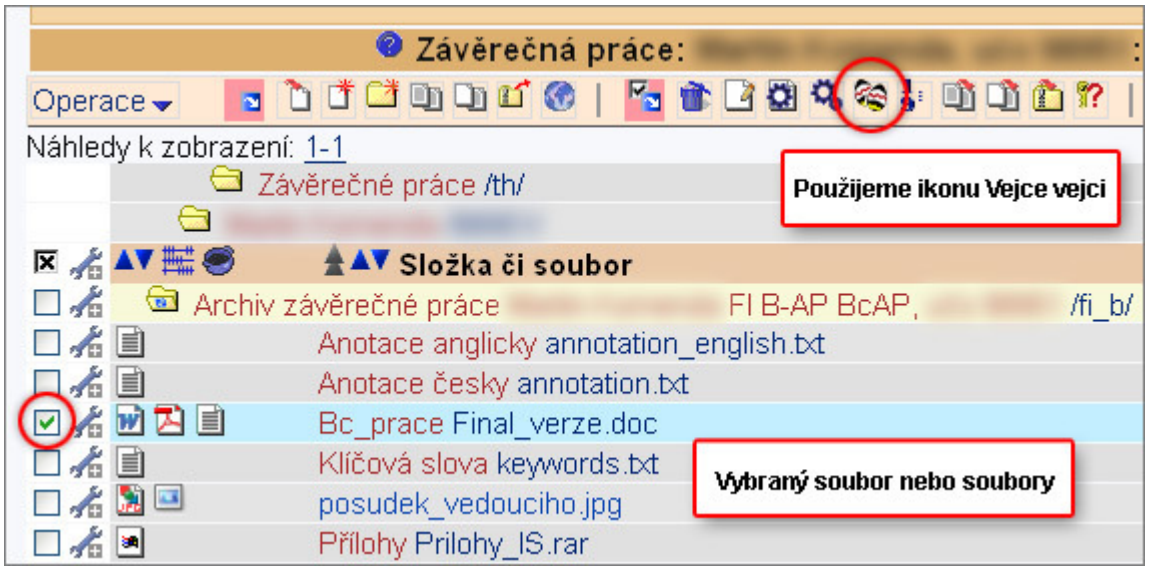
Obr. 4 – Nejpoužívanější vyhledávací služby současnosti

1.3.1 Theses.cz

Mezi nejznámější softwarové nástroje odhalující plagiátorství v naší zemi patří projekt Theses.cz. Je vyvíjen a provozován Masarykovou univerzitou v Brně a zároveň plní funkci **národního registru závěrečných prací**.

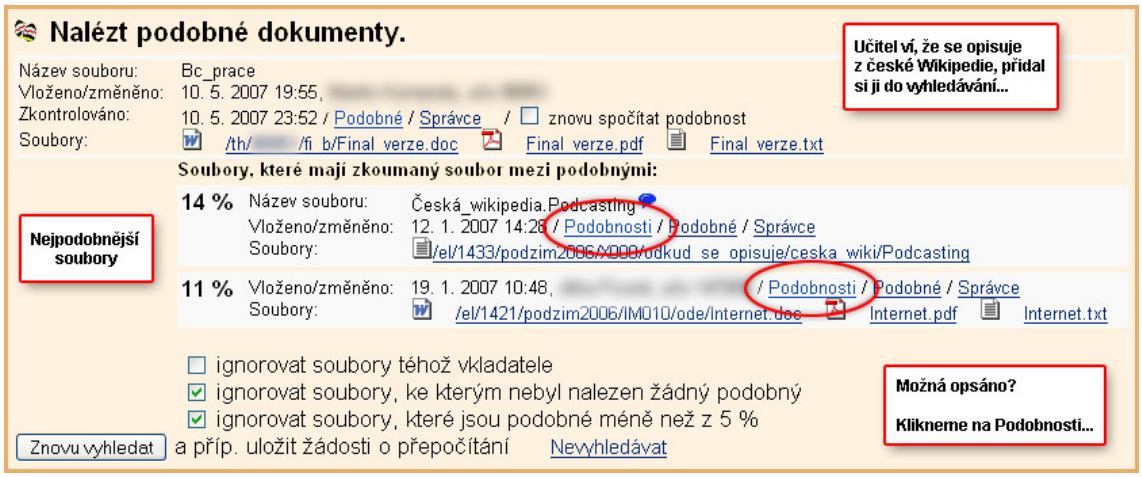
Historie tohoto projektu sahá do roku 2004, kdy v informačním systému (IS) této univerzity vznikl archív závěrečných prací, který prohloubil podporu odevzdávání prací elektronickou formou. Přes 40 000 uživatelů IS tedy mělo přístup ke všem pracím. Poté, co byl na škole vyvinut e-learningový systém pro elektronickou podporu výuky, učitelé si přáli, aby jejich pracně vytvořené publikace k přednáškám a studijním účelům nemohli být lehce zneužity. Přání bylo vyslyšeno, a 16. srpna 2006 dostali kantoři a ostatní uživatelé IS nástroj na odhalování plagiátů. Jelikož měl tento nástroj na konferenci *Inforum 2007* velký ohlas a dokonce byl oceněn, akademická obec rozhodla o vytvoření celonárodního projektu. Začala spolupráce s VŠE v Praze a ke konci roku 2007 získal projekt podporu ministerstva školství a vznikl nový centralizovaný rozvojový projekt MŠMT C1/2008 s názvem „*Národní registr VŠKP a systém na odhalování plagiátů*“, který je institutem ministerstva školství financován.

Tento systém je založen na **principu centrálního repozitáře**, ve kterém se nacházejí všechny odevzdané bakalářské, diplomové, disertační a rigorózní práce. Učitel, který má podezření ohledně původu odevzdané práce, pak jednoduše podezřelou práci přes webové rozhraní nahraje, přičemž jako výsledek systém vrací seznam prací, které jsou více, či méně obsahově podobně s prací testovanou. Vyhledávání plagiátů je implementováno specifickým algoritmem a probíhá *vícefázově*. „*Po vložení souboru do systému je tento analyzován a zpracován tak, že je připraven pro vyhledání podobností. Vyhledání nastává ve finální fázi, kdy uživatelé použijí jednu z funkcí pro vyhledání potenciálních plagiátů. Díky více fázím je vyhledání rychlé a proběhne během několika sekund*“ [9]. Uživatel má možnost použít funkci „jako vejce vejci“, která vyhledává plagiáty typu „copy-paste“, nebo může využít globálního vyhledávání plagiátů, který vypisuje všechny nalezené podobnosti.



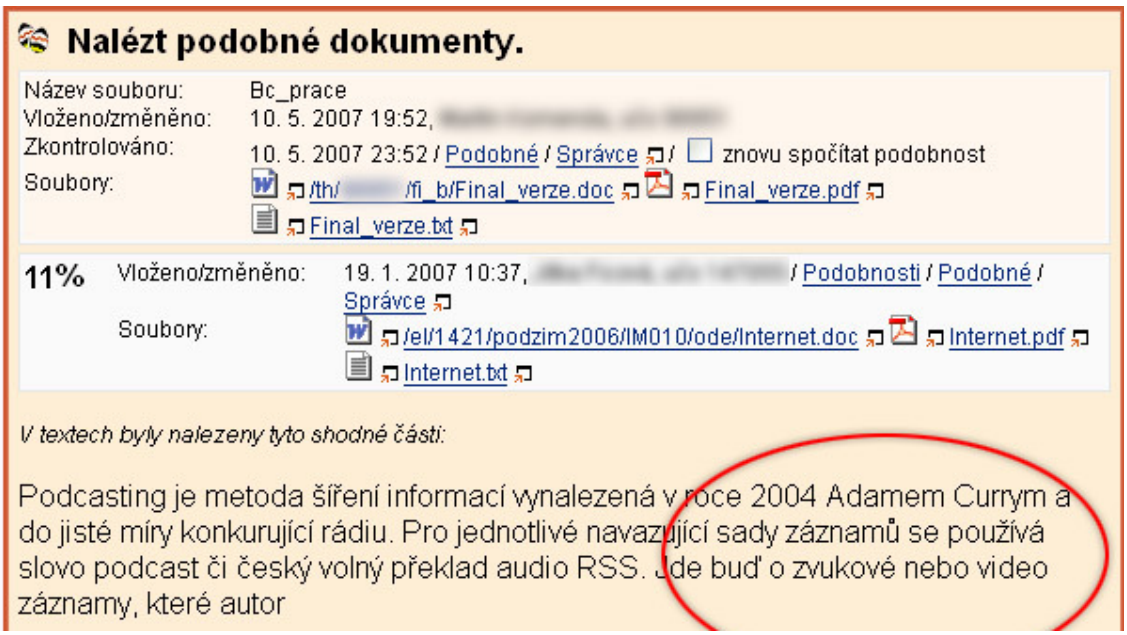
Obr. 5 – Příklad odhalování plagiátorství v systému Theses.cz [12]

Po zaškrtnutí souborů k otestování a vybrání porovnávací metody se uživateli zobrazí detailní výpis všech souborů, které mají danou míru podobnosti se souborem testovaným.



Obr. 6 –Příklad odhalování plagiátorství v systému Theses.cz [12]

Učitel má dále možnost, u všech nalezených souborů, prohlednout si nalezené podobnosti, přičemž v případě, že se jedná o delší odstavce textu, které nejsou řádně ocitovány tak jak to uvádí pravidla (viz. první kapitola), je s největší pravděpodobností testovaný dokument plagiátem.



Obr. 7 – Příklad odhalování plagiátorství v systému Theses.cz [12]

Kromě výše uvedeného základního principu odhalování plagiátorství, systém dokáže odhalit podobnosti mezi jednotlivými texty i za předpokladu, že autor text částečně pozměnil (změnil slovosled, vynechal některá slova, apod.). Funkčnost systému je také založena na tom, že eviduje sekundární údaje závěrečných prací, tzv. **metadata**, které později také slouží pro účely vyhledávání. Těmito metadaty může být např. datum vytvoření souboru, vlastník souboru, klíčová slova atd.

Za poměrně dlouhou dobu svého vývoje systém nabyl několika funkcí a rozšířeních, které jsou uvedeny v seznamu níže:

- *Fulltextové nebo tématické (kategorické) vyhledávání*
- *Vyhledávání dle kritérií*
  - *procentuální podobnost*
  - *cíl prohledávání (v rámci systému, univerzity, jednotlivých fakult)*
  - *stáří prohledávaných prací*
- *Vyhledávání v metadatech*
- *Správa uživatelů a dat*
- *Možnost importu prací hromadně správci nebo individuálně studenty*
- *Podpora pro evidenci posudků k závěrečným pracím*
- *Antivirová ochrana souborů a systém přístupových práv*
- *Rozšíření systému o další zdroje prací (internetové servery)*
- *Doplňkové aplikace (vývěska, diskuse, úschovna)*

V současné době se v systému nachází několik **desítek tisíc** závěrečných prací, přičemž jejich počet neustále narůstá. Toto číslo však není celkový počet dokumentů, se kterými dochází k porovnávání. Počet všech dokumentů se pohybuje v řádech miliónů.

Na projektu se v současné době podílí 26 českých veřejných vysokých škol a dále pak další soukromé, státní a zahraniční vysoké školy (např. ze Slovenska). Všechny tyto instituce mají možnost si systém nakonfigurovat k obrazu svému, tj. nastavit si jej dle svým vlastních specifických požadavků (např. přístupová práva k pracím pro jednotlivé subjekty). Myšlenkou a zároveň hlavním cílem projektu je **zapojit všechny VŠ** do boje proti plagiátorství. Bohužel se zatím ne všechny školy zapojili do tohoto projektu. Např. Karlova univerzita v Praze se zpočátku zdráhala do projektu zapojit, údajně kvůli možným problémům s autorskými právy a citlivosti dat.

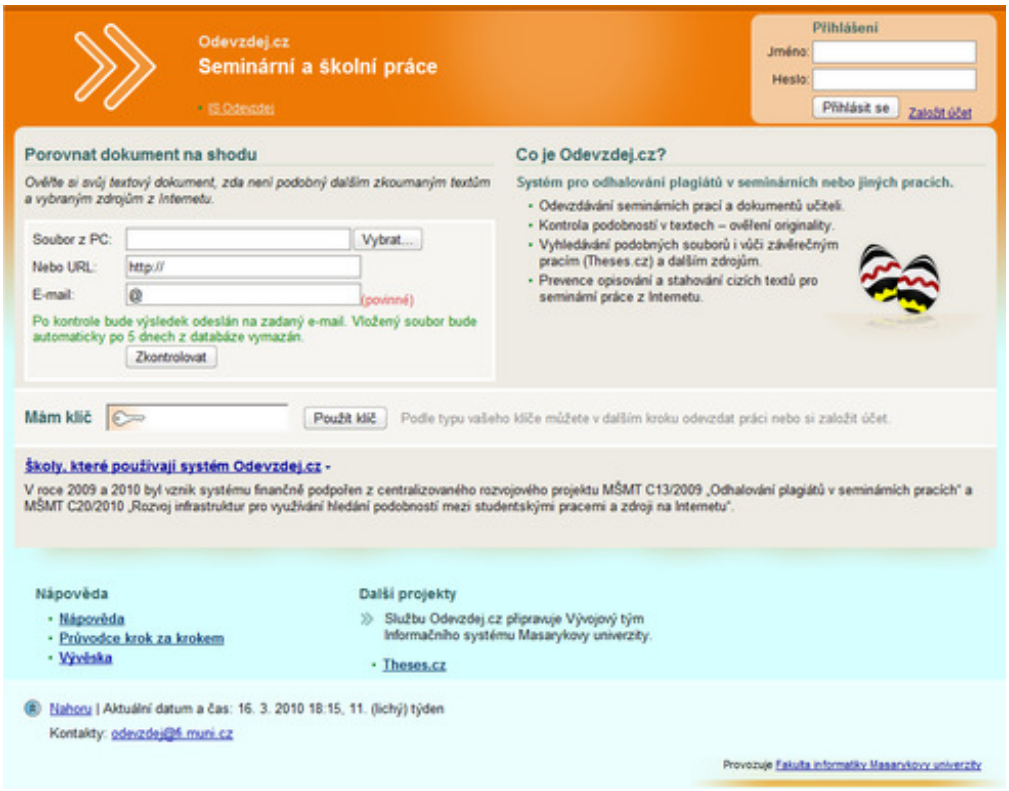
Pokud se tedy zamyslíme nad samotnou ideou projektu, je jasné, že systém stále nenaplňuje původní cíle. Tím, že nejsou připojeny všechny vysoké školy na našem území, systém nemůže kontrolovat všechny odevzdané práce, tudíž mohou být vytvářeny plagiáty z prací odevzdaných právě na školách, které se doposud nezapojily.

1.3.2 Odevzdej.cz

Tento projekt se dá považovat za „**dceřiný**“ k projektu Theses.cz. Je vyvíjen a provozován rovněž Masarykovou univerzitou v Brně. V zásadě projekt vznikl ze dvou důvodů. Tím prvním důvodem byly podněty ze stran učitelů, kteří upozorňovali na to, že studenti neopisují pouze při závěru studia. Tím druhým byly opakované žádosti ze stran veřejnosti, která stále častěji prosila o **jednorázové otestování práce** systémem Theses.cz. A tak se vývojové oddělení Masarykovi univerzity rozhodlo vytvořit systém, který bude přístupný všem.

V dobách, kdy byl vyvíjen systém Theses.cz, bylo původní myšlenkou kontrolovat systémem nejenom vysokoškolské závěrečné práce, ale i různé seminárky apod. Jelikož se později ukázalo, že by to bylo poměrně komplikované řešení, bylo jasné že pro tyto účely musí vzniknou samostatný projekt. A tak se celkem 10 vysokých škol (včetně Karlovi univerzity) zapojilo do centralizovaného rozvojového projektu MŠMT s názvem „*Systém na odhalování plagiátů v seminárních pracích*“.

Projekt umožňuje vyhledávat plagiáty nejen mezi závěrečnými prácemi, ale i mezi semestrálními prácemi, referáty, laboratorními a slohovými cvičeními a ostatními, kde je potřeba jistá tvůrčí činnost. Slouží nejen pro vysokoškolské pedagogy, ale i pro středoškolské profesory a ostatním soukromým osobám (bez nutnosti autentizace), které si chtějí ověřit, zdali daný dokument není plagiátem. Stačí, když do systému vloží soubor s příslušným formátem (Word, OpenOffice, *TeX*), který je následně převeden do jednotného formátu *PDF*, přičemž systém provede analýzu textového obsahu a po určité době uživateli zašle výsledky do jeho e-mailové schránky.



Obr. 8 – Uživatelské rozhraní domovské stránky systému Odevzdej.cz

Velkou výhodou systému je testování podezřelého souboru na **více místech**. Prvním místem jsou samozřejmě lokální úložiště škol, které jsou zapojené do projektu. Toto je zároveň efektivnější přístup, než ověřování na internetu pomocí nějaké z vyhledávacích služeb (např. Google), jelikož některé školy těmto vyhledávačům přístup k jejich databázím nepovolují. Dalším místem je rodičovský projekt Theses.cz, který obsahuje archív všech závěrečných prací a se kterým je projekt Odevzdej.cz přirozeně spjat. V neposlední řadě je systém schopen porovnávat soubor i vůči jiným všeobecně známým a ověřeným informačním zdrojům jako je např. otevřená encyklopedie Wikipedia.

Dalším přínosem pro zapojené školy jsou určité **e-learningové prvky**. Učitelé těchto škol totiž mohou vytvářen tzv. „odevzdávárny", které fungují jako úložiště, kam studenti mohou své práce vkládat.

To celý proces odevzdávání prací automatizuje a zejména učitelům přináší následující výhody:

- *Minimalizace administrativní zátěže (z hlediska odevzdávání a archivace)*
- *Ušetřený čas vynaložený na osobním sbíráním prací*
- *Není potřeba práce posílat jako přílohy e-mailem*
- *Učitel není zatěžován vytvářením adresářových složek na svém PC*
- *Pro jednotlivé práce lze nastavit tzv. „deadline“, nejzazší termín odevzdání (kontrola probíhá automaticky)*

Poté, co student práci do daného úložiště nahraje, je systémem zkontrolována a učitel po její otevření ihned vidí, zdali se nejedná o plagiát. Na učiteli pak zbývá práci ohodnotit, resp. označit případné nedostatky. Tím pádem odpadají i většinou bezpředmětné diskuze mezi žákem a učitelem o kvalitě odevzdané práce.

Školy se mohou rozhodnout pro jednu ze tří variant, jak do systému práce nahrávat. První možností je přenechat veškerou iniciativu na učiteli, který bude sám provádět hromadné vkládání prací (např. zabalených v zip archívu). Je zřejmé, že tímto způsobem se škola ochuzuje o některá z hlavních výhod tohoto systému, uvedených v seznamu výše. Dále škola může studenty informovat, kam mají práce vkládat a celý proces se tak zjednoduší. A konečně poslední, zato však nejvíce sofistikovaná varianta, je jakýsi „**auto-import**“, který mezi lokálním systémem školy a systémem Odevzdej.cz vytvoří jakýsi neviditelný komunikační kanál, kterým se automaticky budou posílat odevzdané práce do systému Odevzdej.cz a ten bude příslušný systém školy automaticky informovat o možných podezření na plagiáty.

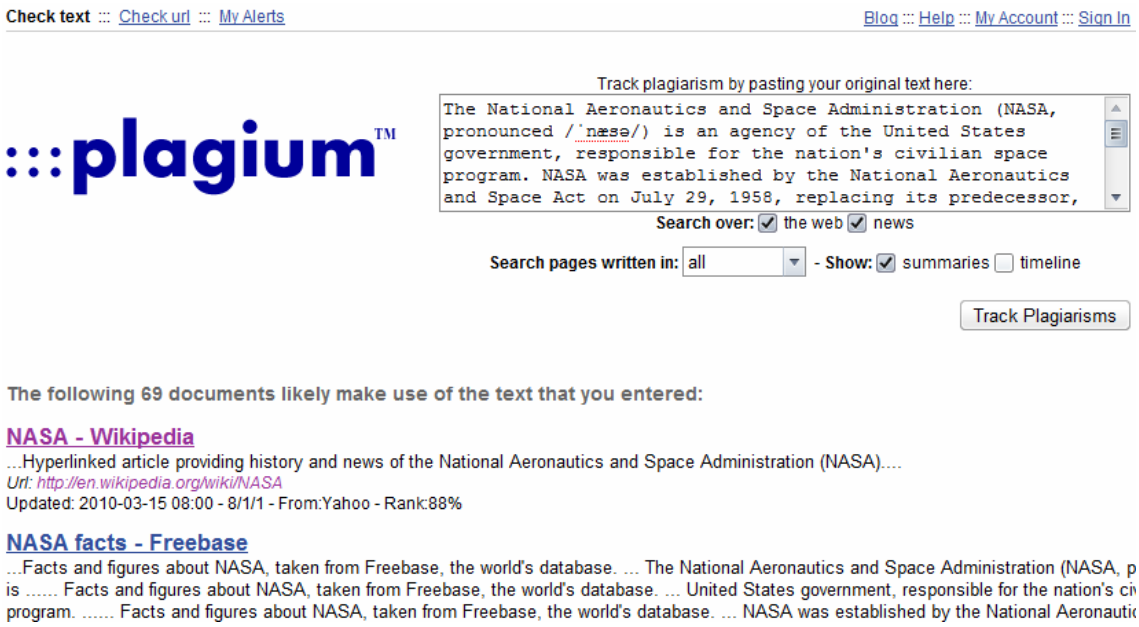
Budoucnost má systém Odevzdej.cz slibnou. Důležitou roli bude hrát i to, kolik škol se v následujících letech do projektu zapojí. Učinit tak mohou nejenom vysokoškolské, ale i střední a základní školy. V současné době se pracuje na algoritmu pro vyhledávání podobných souborů na Internetu a jeho následné integraci do systému.

1.3.3 Systémy v zahraničí

Je pochopitelné, že s problémem plagiátorství se nepotýkáme jenom v naší zemi, ale jedná se o **problém globální**. V zahraničí mají pro nástroje, které odhalují plagiátorství, označení „detection tools“. Těchto systému je na výběr poměrně velké množství, avšak většina z nich je zpoplatněna. Vesměs existují dvě základní kategorie těchto nástrojů.

Nástroje z první kategorie jsou **krabicové produkty**, přičemž v závislosti na verzi tohoto produktu, resp. pro jaký subjekt je verze určena, je odvozena jeho cena. Verze pro jednotlivce, např. studenty, kteří si chtějí pouze jednorázově zkontrolovat svoji práci (zdali se náhodou nedopustili plagiátorství), jsou většinou po registraci nabízeny zdarma, zatímco verze pro rozsáhlejší subjekty, jakými mohou být např. vydavatelství, zpravodajské agentury, právní firmy a ostatní neziskové organizace jsou zpoplatněny podle určitých specifikací.

Druhou kategorií nástrojů jsou **on-line aplikace** a naprostá většina z nich je poskytována zdarma. Téměř všechny aplikace mají jednotnou koncepci. Na úvodní stránce uživatelského rozhraní jsou dominantní dvě vstupní pole, přičemž první slouží pro vložení celého dokumentu (nebo URL adresy kde se dokument nachází) a druhé pro vložení konkrétních textových pasáží z dokumentu. Poté, co v aplikaci na pozadí proběhne vyhledávací algoritmus, je zobrazen jednoduchý seznam všech zdrojů, kde se nachází jisté podobnosti s testovaným dokumentem, resp. textem. U všech těchto aplikací je vyhledávací algoritmus založen na použití některé z vyhledávací služby (např. Google). U žádného z těchto nástrojů však nejsou k dispozici pokročilé možnosti, které jsou standardem u předchozí kategorie, jako např. porovnání zdrojového a podezřelého dokumentu ve dvou horizontálně sousedících panelech, barevné zvýraznění podezřelých míst nebo výběr metody pro porovnání souborů.



Obr. 9 – Ukázka jednoho z online nástrojů (http://www.plagium.com)

Jelikož nástrojů z obou kategorií existuje opravdu mnoho, a představení každého z nich by bylo nad rámec této práce, je v tabulce uvedené níže zobrazen přehled těch nejznámějších produktů včetně jejich základních specifikacích.

Tab. 1 – Přehled zahraničních produktů na odhalování plagiátorství

Název	Popis a technologie	Cena
Turnitin (WriteCheck)	Používá databáze tzv. „rukopisů", knih, časopisů a webových stránek. Tyto databáze obsahují desítky milióny záznamů. Podezřelé texty jsou označeny a předány učitelům ke kontrole ve formě tzv. „reportů", ve kterých jsou uvedeny všechny nalezené shody s testovaným textem. Tyto shody jsou barevně zvýrazněny a obsahují odkaz k jejich zdroji. Jako u některých ostatních produktů, je u testovaného souboru odstraněno formátování, a tak je potřeba mít vytvořenou záložní kopii.	$1,000 – $10,000 za rok v závislosti na velikosti univerzity
	http://www.turnitin.com	
Ithenticate	Podobný princip jako u předchozího produktu. Jedná se však o webovou aplikaci a tudíž je jeho velkou výhodou, že nevyžaduje instalaci nebo další podpůrné programy a je plně kompatibilní na všech platformách. Používá specializovaných vyhledávacích algoritmů a také prohledává web. Výsledem prohledávání jsou opět reporty. Ty podporují porovnávání typu „side-by-side" a zvýrazňování nalezených shod.	Odvozena od četnosti použití a ročního poplatku
	http://www.ithenticate.com/	

Copycatch	Systém vyvíjen ve Velké Británii. Opět pracuje s interní databází a vyhledáváním na webu. Na rozdíl od jiných produktů lze testovat všechny běžné textové formáty. Není potřeba vytvářet lokální kopie testovaných souborů.	$700
	http://www.cflsoftware.com/	
EVE2	„Essay Verification Engine". Nejprve nalezne internetové stránky, od kterých by testovaný soubor mohl pocházet, a poté provádí porovnávání testovaného souboru s textem, který se nachází na nalezených stránkách. Výsledné reporty obsahují procentuální vyjádření podobnosti eseje se stránkami a červenou barvou zvýrazňují nalezené shody.	$29
	http://www.canexus.com/eve/index.shtml	
JPlag	Slouží k odhalování plagiátorství ve zdrojových kódech programů. Může sloužit k hledání podobností i v běžných textech.	Zdarma (nutná registrace)
	http://wwwipd.ira.uka.de:2222/	
Moss	„Measure of Software Similarity". Podobně jako JPlag odhaluje plagiátorství v oblasti programování. Podporuje jazyky C, C++, Java, Pascal, Ada, ML, Lisp. Spravován Stanfordskou univerzitou.	Zdarma (na žádost; pouze pro pedagogy a potřeby výuky)
	http://theory.stanford.edu/~aiken/moss/	
My Drop Box	Tento produkt se velmi podobá tuzemskému produktu Odevzdej.cz. Jedná se o kolekci programů, které plně podporuje tzv. „digitální výukové prostředí". Učitelé mohou vytvářet účty, kam studenti odevzdávají své práce. Učitelé následně mohou ověřit, zdali se nejedná o plagiáty a poté práce ohodnotit.	Samostatná licence - $90 Institucionální licence - $0.50 za každého studenta
	http://www.mydropbox.com/	
Plagiarism Detector	Nástroj pro odhalování „copy-paste" plagiátorství. K prohledávání používá vyhledávací služby Google, Bing a Altavista. Jeho zvláštností je, že lze integrovat do příslušného textového editoru. Podporuje celou řadu formátů (DOC, PDF, RTF, HTML, PPT). Ve výsledku zobrazuje přehledné grafy z barevným rozlišením nalezených shod. Jedná se o zdařilý produkt s kvalitní podporou na jeho webových stránkách.	Demo verze zdarma $30 – 90$ v závislosti na typu licence
	http://www.plagiarism-detector.com	
Plagiarism Finder	Jedná se o aplikaci spustitelnou pod operačním systémem Windows a vyvíjenou v Německu. K běhu je potřeba přístup na Internet. Podporuje přímí import formátů DOC, PDF, RTF, HTML, TXT.	$200
	http://www.plagiarismfinder.com	
WCopyfind	Slouží pouze k porovnání sady vložených souborů. Ve vygenerovaném HTML reportu jsou zvýrazněny (podtržením) shodující se fráze. Vyvíjen na Univerzitě Virginii.	zdarma
	http://www.plagiarism.phys.virginia.edu/Wsoftware.html	

Scan My Essay (Viper)	Podezřelý dokument testuje proti 10 bilionům zdrojům. Možnost testování proti souborům na lokálním PC a také proti esejem publikovaných na Internetu. Velikost testovaného dokumentu je neomezená. Podporuje „side-by-side" porovnávání, zvýrazňování shod a zobrazování odkazů na plagiáty. Produkt na svých webových stránkách dokonce prohlašuje, že je lepší než jeho zpoplatněný konkurent „WriteCheck".	zdarma
	http://www.scanmyessay.com/	
On-line webové aplikace	http://www.plagium.com/ http://www.copyscape.com/ http://www.articlechecker.com/ http://www.dustball.com/cs/plagiarism.checker/ http://searchenginereports.net/articlecheck.aspx	zdarma

Velkou motivací je skutečnost, že v současné době neexistuje aplikace, která by poskytovala stejné možnosti jako ta, která má být vytvořena v praktické části této práce. Zároveň však výše uvedené softwarové produkty obsahují prvky, které mohou být inspirací právě pro nově vznikanou aplikaci.

1.4 Uplatňované postupy při analýze textu

V dnešní době si již většina autorů uvědomuje riziko případného odhalení, v případě, že jejich dílo nepochází z jejich tvůrčí činnosti, a tak při pokusu o obelstění systému nezůstávají pouze u metod typu „copy-paste", resp. zkopírování celého bloku cizí práce a jeho následné vložení do své práce bez jakýchkoliv úprav. A proto musejí přijít na řadu systémy se sofistikovanějšími metodami, které dokáží zamaskované plagiátorství odhalit.

Stěžejním všech systémů pro podporu odhalování plagiátorství je postup, ve kterém je potřeba z poměrně rozsáhlé množiny vstupních dat podezřelého souboru, vybrat **výstupní množinu**, která bude sloužit pro účely testování daného souboru s jinými. Jinými slovy, právě a pouze tato množina bude sloužit jako jakési „měřítko podobnosti" mezi testovaným souborem a množinou ostatních souborů, vůči kterým porovnávání proběhne.

Nejprve je potřeba stanovit, z jakých elementárních prvků bude daná množina vytvořena. V dnešní době dokáží výpočetní stroje velice rychle vzájemně porovnat dva řetězce znaků, avšak dokáží víceméně zjistit pouze to, zdali jsou či naopak nejsou tyto řetězce shodné. Dá se tedy říci, že toto je základní princip fungování všech nástrojů pro odhalování plagiátorství – určitým způsobem testují shodu dvou řetězců, přičemž tato shoda může být prováděna na úrovni odstavců, vět, slov, sousloví a nebo jednotlivých písmen, resp. znaků.

Mezi hlavní kritéria detekčních nástrojů patří **přesnost** a **rychlost výpočtu**, a proto se jejich porovnávací algoritmy dělí do dvou skupin. Algoritmy z první skupiny porovnávají řetězce na úrovni jednotlivých znaků, zatímco v druhé skupině se porovnávání provádí na úrovni jednotlivých slov.

Př.: Změna slovosledu učiní vyhledávací algoritmus pracující na úrovni vět nefunkční

Zdroj: V 80tých letech vznikla skupina standardů X.500 (DAP, DSP, DISP, DOP), které pokrývají adresářové služby.

Plagiát: Standardy X.500 (DAP, DSP, DISP, DOP), které pokrývají adresářové služby, vznikly v 80tých letech.

Předpokladem, k použití algoritmu z první či druhé skupiny, je stejná délka porovnávaných řetězců z obou testovaných souborů, tj. soubor podezřelý a potenciální zdrojový soubor. Tato délka je vyjádřena ve znacích, resp. v počtu slov, budeme ji značit **m**.

Pokud uvážíme situaci, kdy délka porovnávacího řetězce je m = 20 znaků a testovaný dokument je dlouhý čtyři stránky (cca 3 000 znaků), bude celkem 2 980 porovnávacích řetězců (3 000 – m). Zvětšením čísla m, resp. délky porovnávacího řetězce bychom vzniklou situaci o moc nezlepšily (3 000 – 40 = 2 960). Z toho vyplívá, že kdybychom takto vzájemně porovnávali dva soubory o této velikosti, museli bychom provést test na shodu u $2\ 980^2$ dvojic (8 880 400 kombinací). V případě, že bychom chtěli provést porovnání jednoho souboru vůči rozsáhlému archívu souborů, výsledný počet dvojic k porovnání by byl ještě mnohonásobně vyšší a praktické provedení by bylo nemožné.

Př.: Možné kombinace znakových řetězců k porovnání na shodu (při m = 20)

„V 80tých letech vzni" ⇔ „Standardy X.500 (DAP"
„V 80tých letech vzni" ⇔ „tandardy X.500 (DAP,"
„V 80tých letech vzni" ⇔ „andardy X.500 (DAP, "

Základní problémy jsou tedy následující. Je potřeba snížit celkový počet kombinací a také stanovit délku porovnávacího řetězce m. V závislosti na této délce bude mít finální porovnávací algoritmus příslušnou **přesnost**.

Pokud bychom totiž nastavili m na velkou hodnotu, systém se sice stane přesnější, resp. bude odhalovat pouze skutečné případy plagiátorství (kdy jsou kopírovány celé věty nebo odstavce textu), avšak jeho schopnost odhalit nějakým způsobem zamaskované plagiátorství se sníží. Navíc pokud bychom hodnotu m stále zvětšovali, od určité hranice by systém ztratil schopnost vůbec nějaký plagiát odhalit (při m = 300 by zkopírovaný odstavec dlouhý 299 znaků nebyl odhalen). Na druhou stranu, při malé hodnotě m bude systém vracet velký počet nalezených shod a tato příliš velká citlivost se může stát nežádoucí.

1.4.1 Fingerprint

Jednou z možností, jak vyřešit problém stanovení délky porovnávacího řetězce m, je neporovnávat mezi sebou samotné řetězce (vyextrahované z textu), ale z každého vytvořit tzv. „fingerprint" a ty následně porovnat. Jedná se o jakési **jedinečné digitální otisky** z textu, které jsou vytvářeny příslušnými metodami. Algoritmů, které se zabývají vytvářením těchto otisků, je mnoho a jejich kompletní výčet by byl nad rámec této práce. Pro ilustraci si však uvedeme algoritmus, který je založen na vynechání samohlásek, diakritiky a mezer.

Př.: Algoritmu vytvoření „fingerprintu" založený na vynechání samohlásek

„které pokrývají adresářové služby" ⇨ „ktrpkrvjdrsvslb"
„Které pokryvaji Adresarove Sluzby" ⇨ „ktrpkrvjdrsvslb"

Použitím této metody lze tedy říci, že na místo porovnání samotných textů se porovnávají jejich **reprezentace**. To nám přináší následující výhody:

- *zmenšení datového objemu až o 30%*
- *nezahrnutí nežádoucích znaků do porovnávání (např. diakritická znaménka, náhodné překlepy, formátovací značky, apod.)*
- *po převodu textu na číslo další datová úspora (viz. Hoffmanovo kódování jednotlivých znaků nebo LZ kódování skupin znaků)*

Jako i ostatní, i tento přístup není zcela dokonalý. Asi nejvýznamnější nevýhodou je skutečnost, že ze dvou různých řetězců znaků mohou vzniknou stejné otisky. Další nevýhodou je zvýšená náročnost na výpočet, a to jak z hlediska CPU, který otisky musí generovat, tak i z hlediska paměti, kdy je potřeba uložit původní text, ze kterého je fingerprint generován.

1.4.2 Redukce kombinací porovnávacích řetězců

Samotná metoda fingerprintu nám však stále neřeší druhý problém a tím je ohromné množství kombinací porovnávacích řetězců, který při porovnání dvou a více souborů mezi sebou vznikají. Je sice možné stanovit určité redukční pravidlo, podle kterého se sníží celkový počet porovnávacích řetězců – např. bude vybrán pouze každý desátý možný porovnávací řetězec, ze kterého bude vytvořen otisk, avšak toto řešení vytváří určitou „nepředvídatelnost“. Nelze totiž z určitou pravděpodobností definovat, že plagiátoři opisují právě námi zvolenou oblast.

Jedno z možných řešení, které pochází z Cornellovy univerzity, je založeno na **porovnávání skupin celých slov**, místo skupin řetězců. Z principu je toto řešení správné, jelikož lidé nepřemýšlí v řetězcích, ale ve slovech. Tento přístup tedy minimalizuje počet otisků k porovnání a pokud bychom se posunuli o další úroveň výš, resp. testovaný text by byl rozdělen na věty, ze kterých by byly vybírány slova, přineslo by to další zrychlení zpracování.

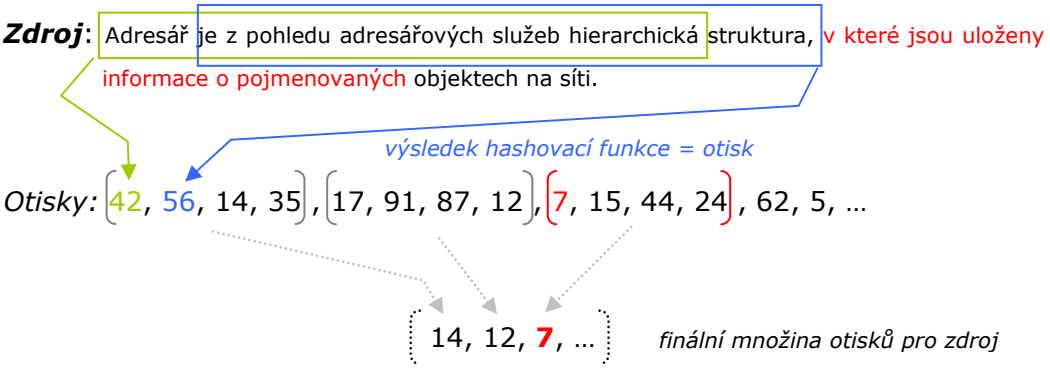
Po několika experimentech byl optimální počet slov v porovnávacím řetězci stanoven na $\underline{m} = 7$, přičemž je zahrnuto okolí těchto slov o délce $\underline{w} = 4$.

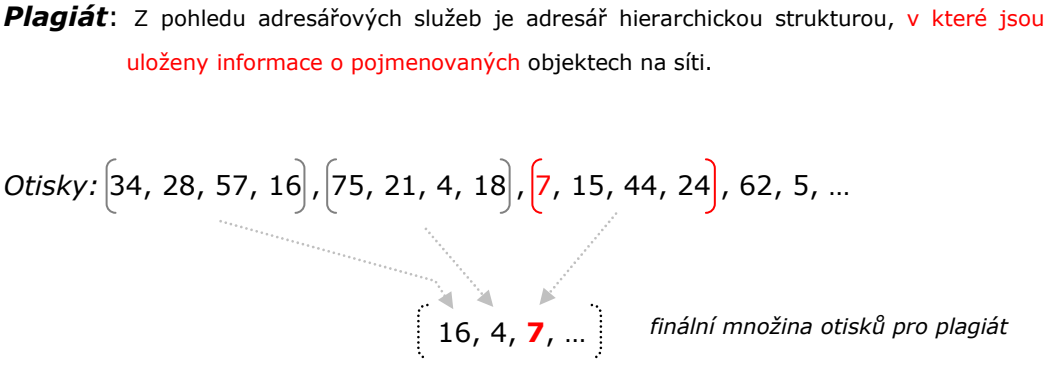
Jinými slovy, je vybrán text o délce 11 slov, přičemž z tohoto textu je dále vybráno 7 slov a těchto pak vytvořen digitální otisk (fingerprint). Zjednodušeně pak lze celý algoritmus definovat následovně [2]:

1. *Systém text nejdříve rozdělí na věty.*
2. *Pro každou větu bude postupovat (lineárně) od začátku.*
3. *Přitom vybírá vzorky (v našem případě jednotlivá slova) o délce m=7 (sedm slov), vždy se posune o jedno slovo doprava - tak vytvoří množinu otisků (z věty dlouhé 23 slov vytvoří 17 otisků - tj. 23 - m + 1).*
4. *"Lokální winnowing algoritmus" z této množiny otisků vybere několik reprezentativních vzorků. Opět zjednodušeně řečeno: skupinu otisků rozdělí na určité úseky (okna), které se nepřekrývají - z každého úseku pak vybere nejmenší otisk (záleží tedy na kontextu ostatních).*
5. *Vybrané otisky pak systém uloží do indexu spolu s číslem dokumentu (ID); s jejich pomocí bude prováděna detekce.*

Předchozí algoritmus lze jednoduše osvětlit na následujícím příkladu. Předpokládejme přitom, že daný výčet čísel slouží pouze pro ilustraci. Jejich skutečná hodnota je výsledkem hashovaní funkce.

Př.: Princip lokálního winnowing algoritmu





Na grafické ilustraci uvedené výše je vidět, jak algoritmus nejprve vytváří digitální otisky ze sousloví o sedmi slovech a následně tyto otisky rozdělí do skupin, kde každá skupina obsahuje 4 po sobě jdoucí otisky. Z každé této skupiny je poté vybrán otisk s nejmenší hodnotou a vložen do finální množiny otisků k porovnání. Nakonec jsou obě finální množiny, jak pro zdroj (14, 12, 7, …) tak i pro plagiát (16, 4, 7, …), vzájemně porovnány a pokud je nalezen otisk, jehož hodnota se vyskytuje v obou množinách, byl nalezen důkaz o plagiátorství. V našem případě otisk s hodnotou 7, která je výsledkem digitálního otisku sousloví „*v které jsou uloženy informace o pojmenovaných*“. V tomto případě budou spolehlivě odhaleny plagiáty, které se skládají z 11 slov (7 + 4, resp. „základ + okolí“), přičemž texty v rozmezí 7 – 11 slov nemusí, ale mohou být odhaleny.

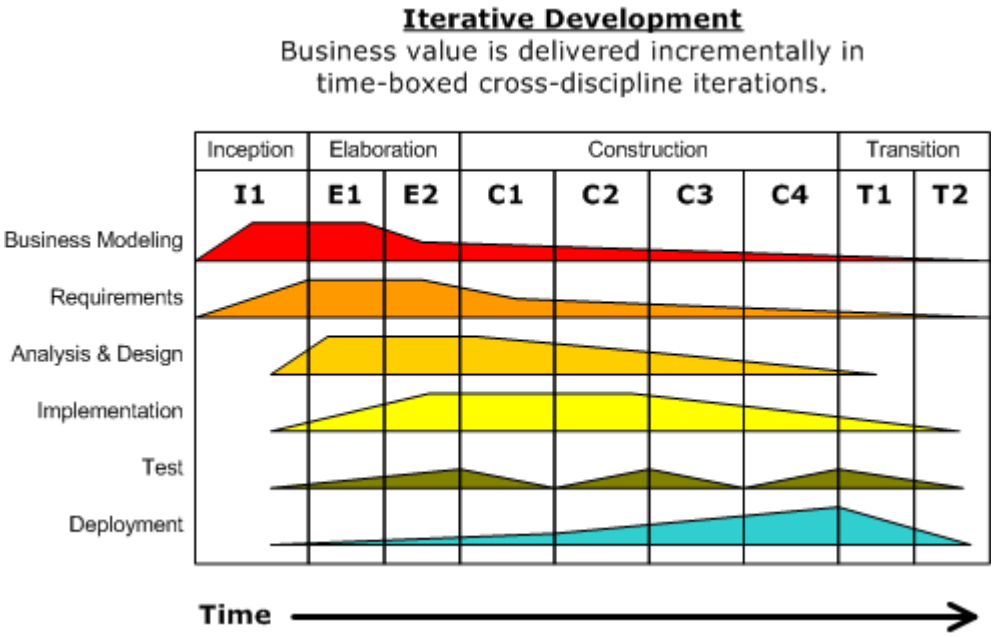
Mezi hlavní výhody této metody patří výrazná **redukce** počtu kombinací k porovnání a také jistý **determinismus**, kdy je zaručeno, že zkopírovaný text o m slovech a nachází se v kterékoliv části dokumentu, bude odhalen. Mezi nevýhody se dá považovat trochu složitější zpracování, kdy ke každému otisku je potřeba uchovávat informaci k jakému sousloví v daném dokumentu patří.

Metoda lokálního winnowing algoritmu zajisté není jediná, která se touto problematikou zabývá. Velmi známá a používaná je také metoda Séman, která využívá tzv. sémantické analýzy, jejichž princip detekce je založen na překladu slov v textu do tzv. univerzálního kódu, přičemž dvěma zdánlivě různým slovům je přiřazen stejný kód (např. slovu „skočil“ a „vyskočil“ bude přiřazen kód „dc2fx5“). Po integraci sémantické analýzy do nějaké metody pro odhalování duplicit v textových souborech (např. winnowing) může vzniknout opravdu silný nástroj pro odhalování plagiátorství.

2 Analýza nového systému

Vývoj každé nové aplikace, nezávisle od její složitosti a komplexnosti, se v dnešní době řídí jistými pravidly a postupy. Dodržování těchto metodik si vývojář, či tým, nejen ujasní a zpřehlední následující vývoj, ale zároveň zjednoduší budoucí možná rozšíření. Jedna z těchto metodik se nazývá UP (*Unified Process*).

Princip této metodiky spočívá v rozdělení vývoje aplikace do několika **hlavních aktivit**: sběr požadavků, analýza, návrh, implementace a testování. Seskupení těchto pěti hlavních aktivit je označováno jako **iterace**. Každá s těchto aktivit je více, či méně aplikována v jednotlivých **fází** vývoje: zahájení, rozpracování, konstrukce a zavedení do provozu. Dále je důležité poznamenat, že každá z těchto fází může obsahovat libovolné množství iterací (v závislosti na složitosti projektu).



Obr. 10 – Vztah hlavních aktivit tvořící iterace a fází v UP (zdroj: Chps.com)

Následující kapitoly v této části se budou zabývat prvními třemi aktivitami, tj. sběr požadavků, analýza a návrh, přičemž z obrázku výše je zřejmé že tyto aktivity se odehrávají ve fázi zahájení a rozpracování.

2.1 Základní princip a funkcionalita

2.1.1 Motivace

Každý vývoj nového produktu, ať už se jedná o věc ze světa informačních technologií či z jiné oblasti, je podmíněn základním faktorem – a tím je **motivace**. Motivačních otázek v daných případech existuje celá řada; „Co nám nový produkt přinese? Jaký bude jeho smysl a využití? Vrátí se investice vynaložené do produktu?“, apod., přičemž najít reálné odpovědi je nejenom důležité ale mnohokrát, velmi obtížné. Někdy může být přínosem zautomatizování a zjednodušení jistého procesu, jindy zas ukládání a sběr dat a jejich následné statistické vyhodnocení.

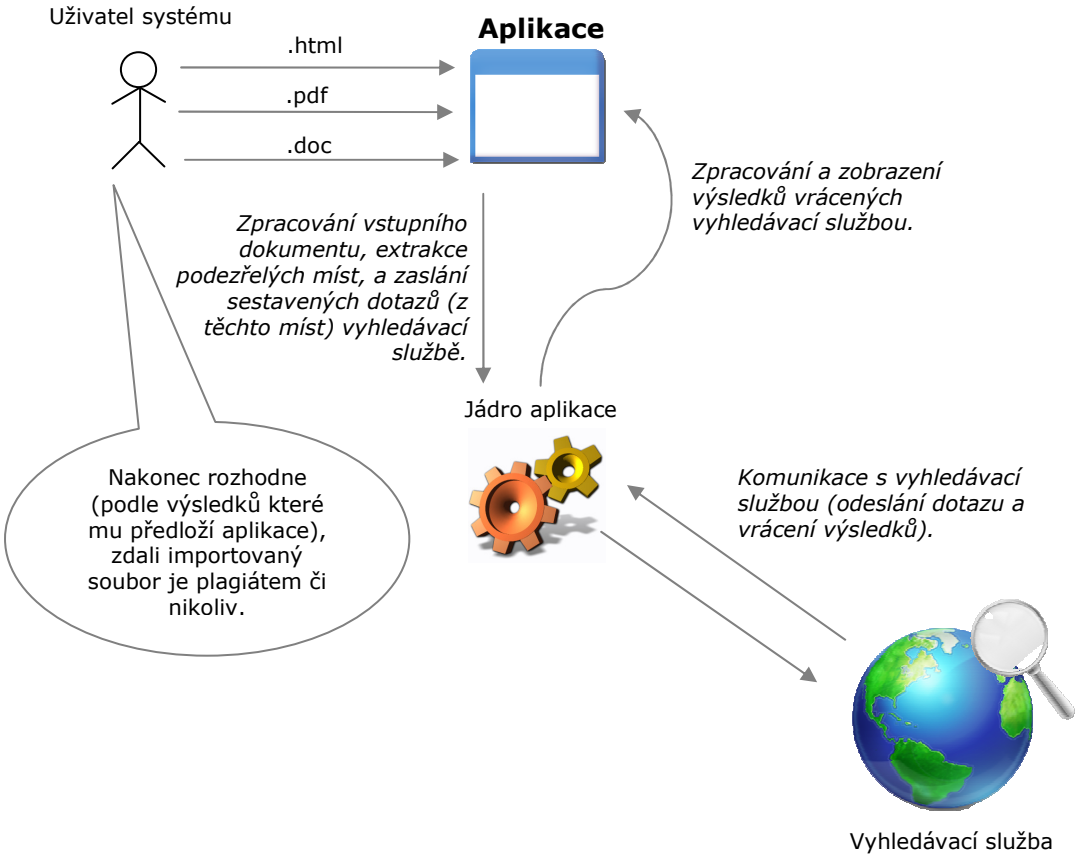
Motivačním důkazem pro vytvoření našeho systému pro odhalování plagiátorství budiž úryvek z jednoho internetového článku: „*Průšvih s opsanou celou jednou částí diplomové práce z internetu řešila před několika lety Ekonomicko – správní fakulta Univerzity Pardubice. Tehdejší proděkance bylo divné, že práce byla napsána až příliš dokonale. Když zadala do vyhledávače čtyři klíčová slova, podezření se jí potvrdilo*“ [15]. Co kdyby existovala aplikace, kterou by měli k dispozici všichni kantoři a mohli obdobným způsobem testovat odevzdané práce? Ba co víc – co kdyby celý proces odhalování plagiátorství zautomatizovala a daný kantor by se např. nemusel vůbec starat do kterého vyhledávače má testovaný text zadat a už vůbec ne jaká „*čtyři klíčová slova*“ má do něj vložit, tak jak je uvedeno v předchozí citaci.

Výsledek by byl zřejmý – nejenom že by nová aplikace měla jistý **preventivní účinek**, ale zároveň by reálně dokázala postihnout a potrestat případné hříšníky. Jistě by tak došlo ke zlepšení kvality nejen vysokoškolských studií.

2.1.2 Požadavky

Tak jak je uvedeno v zadání této práce, je hlavním požadavkem vytvořit SW produkt, aplikaci která bude schopna odhalit plagiátorství prostřednictvím celosvětové sítě Internet. Jestli se bude jednat webovou aplikaci, či tzv. „okenní“ aplikaci, odhalí až následující analýza a v tuto chvíli tento fakt není podstatný.

Co ovšem podstatné je, že aplikace by měla umět pracovat s různými formáty vstupního souboru, tj. souboru, který uživatel do aplikace nahrál a chce jej otestovat zdali náhodou není plagiátem. Po nahrání tohoto souboru, by měla aplikace uživateli přehledně zobrazit tzv. **podezřelá místa**, přičemž uživatel by měl mít možnost jedno z těchto míst vybrat a poslat tak předem sestavený dotaz (z podezřelého místa) vyhledávací službě, která následně vrátí výsledky tohoto dotazu (odkazy na soubory). Uživatel by pak měl mít možnost jeden z těchto souborů vybrat a vzájemně jej porovnat s dříve importovaným testovaným souborem. Konečný verdikt, zdali testovaný soubor je, či není plagiátem, má záležet na samotném uživateli. Je to právě on, který nakonec musí zkontrolovat, ověřit, a porovnat nalezené podobnosti. Je tedy na něm, zda-li jako plagiát označí soubor, ve kterém mu aplikace „znázornila“ nalezenou shodu v celém odstavci o 100 slovech nebo jenom v jedné větě. V následující kapitole budou však popsány kroky, kterými by mohla aplikace dále do budoucna směřovat a celý proces dále zautomatizovat a zefektivnit.



Obr. 11 – Základní funkcionalita nové aplikace

2.1.3 Vývojové stupně systému

S trochou nadsázky se dá říci, že tento nově vznikaných systém má několik **evolučních fází** (tyto stupně samozřejmě můžou mít i jiné sofistikované systémy). Tyto fáze lze chápat jako cestu od nejzákladnějších operací a funkcí (kdy téměř vše řídí a ovládá uživatel), až po plné zautomatizování požadovaného procesu (v našem případě proces odhalování plagiátorství). Následující seznam níže, lze chápat jako základní vodítko pro budoucí inovátory.

I. **Google** – tento pravděpodobně nejlepší vyhledávač lze považovat (ikdyž k tomu vůbec není určen) za nejelementárnější nástroj pro odhalování plagiátorství na Internetu. Nevýhody toto přístupu odstraňují až další stupně – samozřejmě nejzásadnější je fakt, že běžný uživatel neví jak sestavit dotaz, tak aby byla detekce co neefektivnější.

II. **Jednoduché on-line detektory** – nástroje stavěné už pro účel odhalování. Mají jednoduché uživatelské rozhraní, nejčastěji s dvěmi vstupními poli (URI k testovanému souboru a pole pro přímé vložení testovaného textu) a hlavně již obsahují jistý algoritmus, metodiku pro sestavení dotazu pro vyhledávač. Po spuštění vyhledávání většinou pouze vrací odkazy obdržené od vyhledávače.

III. **Pokročilé detekční nástroje** – tyto produkty jsou již maximálně přizpůsobeny pro efektivitu práce a komfort uživatele. Podporují celou řadu vstupních formátů, zjednodušenou formou nabízejí několik metod analýzy textu (resp. způsob sestavení vyhledávacího dotazu) a často komunikují s více než jednou vyhledávací službou. Již obsahují jeden velmi důležitý a praktický rys, který je většinou doménou off-line detektorů, tj. porovnávání souborů „side-by-side“ (samozřejmě včetně zvýrazňování shodných pasáží).

   Je důležité poznamenat, že tento vývojový stupeň (a část z dalšího) bude cílem nově vznikané aplikace této práce.

IV. **Plně automatizované systémy** – jedná se o nejvyšší možnou úroveň detekce plagiátorství. Celý proces odhalování je zautomatizován, tj. vyžaduje minimální (v podstatě nulovou) součinnost uživatele. Jak by podobný proces mohl fungovat popisují následující řádky.

   - *Kantor studentům zpřístupní FTP server, kam následně studenti nahrávají odevzdané práce*
   - *Systém má svůj proces, spuštěn v operačním systému, který v předem nastavených časových smyčkách kontroluje, zdali v úložišti nepřibyli nějaké soubory – pokud ano, oznámí to systému*
   - *Systém následně soubory převezme a na těchto pak provede „kontrolu“*
   - *Pokud systém podle výsledků kontroly rozhodne, že daný testovaný soubor je s největší pravděpodobnosti plagiátem (např. byla nalezena shoda v sousloví o 30 slovech) – informuje o tom kantora (např. prostřednictvím e-mailu)*
   - *Kantor má pak možnost tento výsledek zpětně ověřit v uživatelském rozhraní systému a pokud i on schválí, že se jedná o plagiát, může konat*

   Samozřejmostí těchto systému je maximálně propracované uživatelské rozhraní. Může např. obsahovat funkcionalitu záložek, obdobné řešení jako v internetových prohlížečích, což uživateli umožňuje otevřít a pracovat s více testovanými soubory najednou.

2.2 Návrh technologií

Ocitáme se ve fázi, kdy je potřeba podrobně rozebrat funkčnost nově vznikaného systému. Je potřeba rozhodnout a navrhnout vnitřní strukturu aplikace, zejména pak role uživatelů a jejich správu, na jakých technologií bude nový systém postaven, jaký bude jeho datový model a v neposlední řadě vzhled a rozvržení uživatelského rozhraní.

2.2.1 Základní koncept

2.2.1.1 Webová aplikace

Nejprve bylo potřeba rozhodnou na jaké základní architektuře bude aplikace postavena. V dnešním světě softwarového inženýrství existují dva základní proudy vytváření nových aplikací – webové aplikace vs. „okenní“ aplikace. Obě samozřejmě mají svoje výhody a nevýhody a rozhodnutí, který z těchto přístupů zvolit, vždy závisí na konkrétních požadavcích a použití.

Jelikož je vysoce pravděpodobné, že systém bude v budoucnu poměrně často rozšiřován, modifikován a aktualizován, bylo by vhodné, kdyby byl **zpracován centrálně** – tj. z jednoho místa. Kvůli těmto aspektům, se nejvíce nabízí volba webové aplikace. Systém tedy poběží na nějakém webovém serveru a klienti si jej budou spouštět prostřednictvím svého internetového prohlížeče. Tento způsob je zároveň v dnešní době trendem, kdy přináší pro uživatele jistý komfort a pohodlí, jelikož sám uživatel se nemusí starat o nějaké instalace a následné budoucí aktualizace, a pouze zadá *URL* adresu ve svém prohlížeči. Prohlížeč tak zde figuruje jako „tenký klient“, využívající síťovou architekturu „klient-server“. Díky tomuto přístupu se tak nová aplikace stane absolutně **multiplatformní**, jelikož webový prohlížeč je naprosto běžnou součástí všech moderních operačních systémů.

2.2.1.2 Apache & PHP

Nově vznikající systém bude webovou aplikací – tzn. že je potřeba rozhodnou na jakém webovém serveru poběží a v jakém skriptovacím jazyce bude napsána. Pro tyto účely byla vybrána kombinace *Apache* + *PHP* a to z několika důvodů. Tím prvním je skutečnost, že obě technologie jsou široce rozšířeny a používány, jsou **multiplatformní** a konečně jejich hlavním faktorem je cena. Získání a provoz je zcela **zdarma**.

To ovšem neznamená že by tím nějak utrpěli na kvalitě – naopak. Webový server Apache je vůbec nejrozšířenějším svého druhu, podporuje obrovské množství modulů (včetně PHP) a v neposlední řadě se jedná o *open-source*. Skriptovací jazyk PHP je plně objektově orientovaný jazyk, obsahuje nepřeberné množství funkcí a rozšiřujících modulů, podporuje mnoho databázových systémů (MySQL, Oracle,

apod.), lze jej nasadit v naprosté většině hostingových službách, není náročný na pochopení a má kvalitně vypracovanou dokumentaci. Další nepřehlédnutou výhodou PHP je, že pro něj bylo vytvořeno spoustu kvalitních *frameworků*, jako např. Zend, CakePHP, Kohana, Nette (český tvůrce), a spoustu dalších, které podporují moderní programovací metodiky, jako např. používání návrhového vzoru *MVC*.

2.2.1.3 AJAX

S odkazem na analýzu a návrh uživatelského rozhraní, je zřejmé, že kdyby při každém požadavku ze strany uživatele (kliknutí na odkaz) docházelo k znovunačtení celé stránky (včetně všech knihoven, stylů a skriptů), nejen že by se zpomalil běh celé aplikace, ale zároveň by bylo používání takovéto aplikace poněkud „těžkopádné“.

Co když ale existuje technologie, díky které můžeme odeslat požadavek (resp. žádost o zprostředkování určitých dat) webovému serveru, následně obdržet odpověď, zpracovat a zobrazit ji uživateli **bez obnovení** celé stránky. Ano, taková to technologie skutečně existuje a nese název AJAX (*Asynchronous Javascript And XML*). Podle akronymu lze poměrně snadno vyčíst, jakých pod-technologií využívá, přičemž jako rozhraní pro zasílání žádostí a odezev je použito tzv. XHR (*XMLHttpRequest*). Díky této technologii se používání webových aplikací v jistých ohledech stává plynulejší (proč např. načítat celý rozsáhlý web v případě, že si chce uživatel zobrazit fotografii z galerie, po kliknutí na její miniaturu) a celkově se tak používání webové aplikace přibližuje ke komfortu používání okenní aplikace.

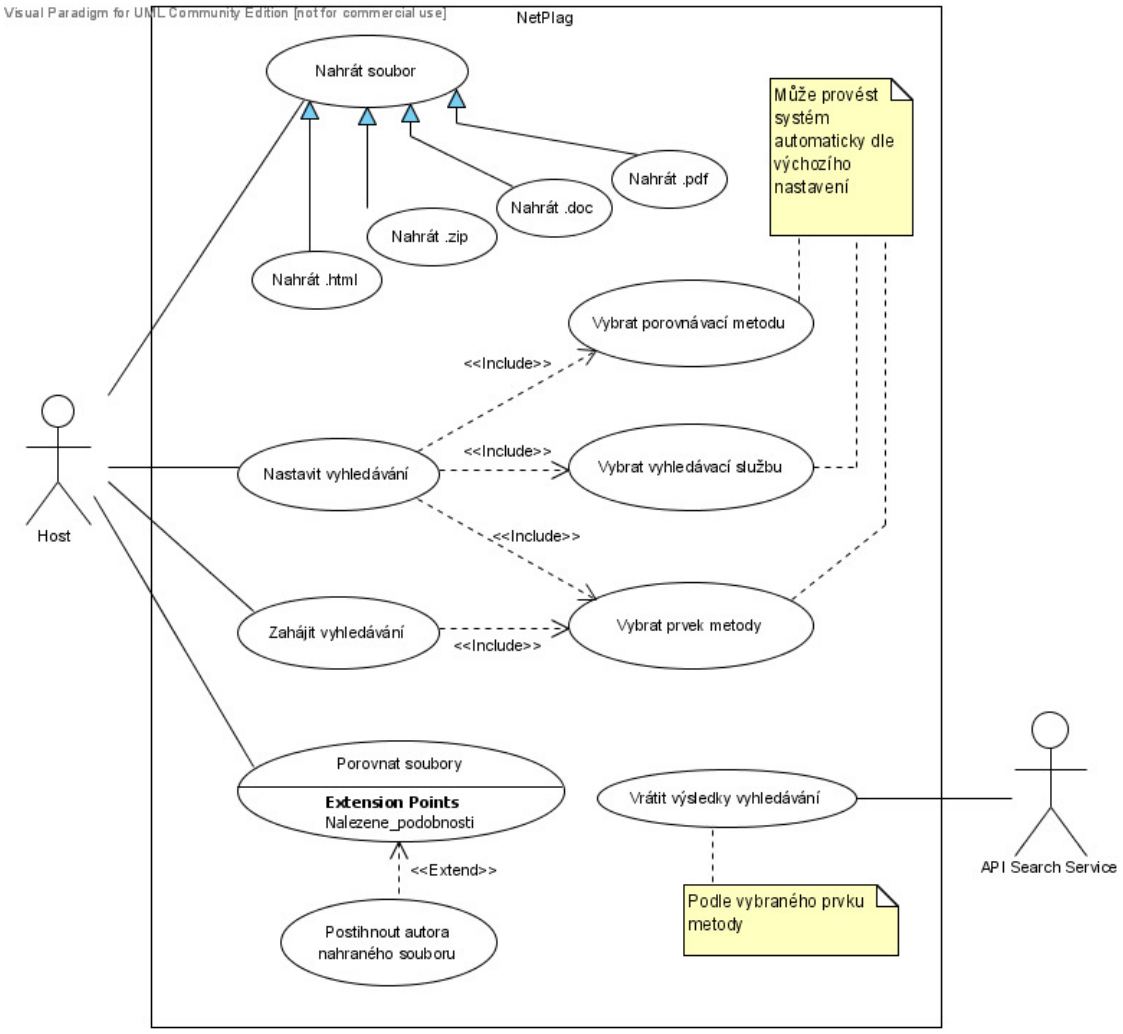
V tuto chvíli lze tedy říct, že tato technologie bude v nové aplikaci opravdu hojně využívaná a to např. v případě odeslání dotazu vyhledávací službě a následné obdržení odpovědi, její zpracování a zobrazení uživateli.

2.2.2 Správa uživatelů

Většina SW produktů slouží pro nějakou předem definovanou skupinu, či skupiny uživatelů. Právě definice těchto skupin je z hlediska budoucí implementace velmi důležitá a je nepostradatelnou součástí fáze analýzy a návrhu.

Samozřejmě tak jako při vývoji jiné části aplikace, i zde se vše odvíjí o **požadavků**. Je požadavkem v naší aplikaci uživatele ukládat? Je požadavkem rozlišovat a spravovat jejich role, či mít komplexní administrační rozhraní? Na rozdíl od vývoje např. intranetového systému, kde každý budoucí uživatel má svoje předem známe funkce a pravomoce a odpověď by nejspíš byla kladná, v našem případě tak není zapotřebí – alespoň v tuto dobu, z hlediska základních požadavků. V našem případě má systém fungovat v podstatě jako **jednorázový nástroj**, u kterého by případné přihlašování uživatelů postrádalo svůj základní význam – tj. zabránit neoprávněné osobě dostat se k cizím a citlivým informacím – náš systém žádné takovéto informace obsahovat nebude.

Naproti tomu by bylo vhodné nezapomenout při vlastní implementaci na to, že koncept a funkcionalita se v budoucnu může rozšířit a systém by mohl obsahovat rysy, které jakousi základní správu uživatelů vyžadují. V tuto chvíli tomu není za potřebí, a pro následující účely nám postačí, pokud zavedeme jedinou roli uživatele – **host**. Tato role nevyžaduje *autentizaci* ani *autorizaci*, a pod touto rolí si můžeme představit např. profesora na vysoké škole nebo autora odborných článků publikující pro technologický server. Na diagramu níže, který je součástí jazyka *UML* a spadá do kategorie diagramů chování, je zobrazen diagram případů užití (use-case) a znázorňuje elementární interakce uživatele se systémem.



Obr. 12 – Diagram případů užití (use-case)

Na diagramu výše je poměrně jasně znázorněná **vazba** mezi uživatelem, samotným systémem (jehož hranice tvoří velký obdélník) a také vyhledávací službou, která jednak přijímá požadavek na zahájení vyhledávání (podle vybraného prvku metody), ale zároveň do systému zasílá výsledky daného vyhledávání. Za zmínku také stojí případ užití „*Vybrat prvek metody*“, který nejenom slouží k nastavení vyhledávacího dotazu, ale také slouží k zahájení samotného vyhledávání.

2.2.3 Formát dat

Dalším důležitým parametrem při vývoji jakéhokoliv sofistikovanějšího systému, je formát a správa dat, resp. **datový model**. Pro většinu aplikací je v dnešním světe velice vhodné a efektivní použít pro správu a řízení dat tzv. relační databáze. Jejich hlavními výhodami je, že pomocí příslušných nástrojů dokáží zajistit integritu databáze, konzistenci dat a v neposlední řadě zabraňují redundanci dat. Také zahrnují standardizovaný dotazovací jazyk *SQL*, který nabízí široké možnosti pro výběr, editaci, přidání a odstranění dat z těchto databází. Zjednodušeně se dá tedy říci, že se nasazují všude tam, kde je potřeba uchovávat a pracovat s poměrně velkým objemem komplexních a heterogenních dat, přičemž je také rozhodující rychlý výběr a zpracování těchto rozsáhlých dat.

Pokud bychom si poslední větu z předchozího odstavce vyložili jako otázku, tak jelikož náš systém s těmito daty pracovat nebude (v současné době), nasazení relační databáze by bylo zbytečné. S odkazem na diagram případů užití se jako nejvhodnější úložiště, zejména a hlavně pro soubory, jeví použit **souborový systém**. Zároveň tak umožníme snadné budoucí rozšíření, kdy se tento typ úložiště dá poměrně snadno propojit např. s protokolem FTP.

2.2.3.1 NPL – interní formát

Pokud bychom se podívali na diagram případů užití, lze z něj vyčíst, že pro libovolný vstupní formát (HTML, PDF, apod.) je potřeba navrhnout a vytvořit jeho vlastní rozhraní, tak aby bylo možné přistupovat k jeho prvkům (např. vybrat 2. větu s 1. odstavce). Pokud bychom tak museli pro všechny tyto formáty implementovat jejich rozhraní, byl by tento přístup, s postupem času a přidávání dalších formátů, velice neefektivní a systém by přibíral na složitosti a nečitelnosti.

Daleko vhodnější se jeví varianta vytvoření **vlastního interního jazyka**, do kterého budou všechny vstupní formáty „transformovány“, a se kterým bude systém umět pracovat, resp. bude mít pro tento formát vytvořené rozhraní. V budoucnu tak po příchodu nového formátu bude stačit napsat pouze jeho transformaci do našeho interního formátu.

Jelikož v našem novém formátu bude zapotřebí určitým způsobem rozlišovat jednotlivé prvky (odstavce, věty, citace, apod.), jeví se jako nejvhodnější varianta, aby náš nový jazyk dodržoval standardy a předpisy již existujícího a velmi podporovaného jazyka **XML** (*Extensible Markup Language*). V současné době se jedná o velmi rozšířený a používaný *značkovací jazyk*, ze kterého je odvozen velmi známý jazyk HTML. Jeho asi největší síla spočívá v tom, že se v podstatě stal standardem pro výměnu informací a umí s ním pracovat nepřeberné množství systémů a aplikací. Další výhodou je skutečnost, že odděluje strukturu dat od jejich vzhledu. Náš nový systémový formát bude tedy odvozen od jednoho z nejpoužívanějších jazyků současnosti a pojmenujme jej **NPL** (*NetPlag Language*).

2.2.3.2 XSD – struktura dat

Jelikož náš interní formát pro popis dokumentu je jazykem XML, je potřeba u něj nějakým způsobem definovat jeho přípustnou strukturu, aby bylo zřejmé, které elementy a atributy může obsahovat. Naštěstí pro jazyk XML existuje spoustu možností pro popis takovéto struktury. Jedním z nich je **XSD** (*XML Schema Definition*). Jeho výhodou oproti ostatním alternativám je široká podpora (oproti např. *Relax NG*) a také to že samotné XSD je samo o sobě jazykem XML. Následující kód popisuje strukturu jazyka NPL a pod ním je k vidění zjednodušený ukázkový soubor NPL, který tuto strukturu dodržuje a je tudíž validní.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- koren -->
  <xs:element name="dokument">
    <xs:complexType>
       <xs:sequence>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element ref="nadpis" />
          <xs:element ref="odstavec" />
        </xs:choice>
        <xs:element ref="zdroje" maxOccurs="1" minOccurs="0" />
       </xs:sequence>
    </xs:complexType>
  </xs:element>

<!-- 1. uroven -->
  <xs:element name="nadpis">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="veta" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="odstavec">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="veta" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="citace" type="xs:boolean" use="optional" />
    </xs:complexType>
  </xs:element>

  <xs:element name="zdroje">
   <xs:complexType>
    <xs:choice minOccurs="1" maxOccurs="unbounded">
     <xs:element ref="kniha" />
     <xs:element ref="web" />
    </xs:choice>
   </xs:complexType>
  </xs:element>

<!-- 2. uroven -->
 <xs:element name="veta">
   <xs:complexType mixed="true">
     <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="citace" type="xs:string" minOccurs="0" />
        <xs:element name="odkaz" type="xs:anyURI" minOccurs="0" />
        <xs:element ref="zvyrazneni" minOccurs="0" />
     </xs:choice>
     <xs:attribute name="citace" type="xs:boolean" use="optional" />
   </xs:complexType>
 </xs:element>

<!-- 3. uroven -->
  <xs:element name="kniha">
    <xs:complexType>
      <xs:all>
       <xs:element name="nazev" type="xs:string" minOccurs="0" />
       <xs:element name="autor" type="xs:string" minOccurs="0" />
       <xs:element name="nakladatelstvi" type="xs:string" minOccurs="0" />
       <xs:element name="isbn" type="isbnType" minOccurs="1" />
      </xs:all>
    </xs:complexType>
  </xs:element>
```

```
  <xs:element name="web">
    <xs:complexType>
      <xs:all>
       <xs:element name="titulek" type="xs:string" minOccurs="0" />
       <xs:element name="autor" type="xs:string" minOccurs="0" />
       <xs:element name="odkaz" type="xs:anyURI" minOccurs="1" />
      </xs:all>
    </xs:complexType>
  </xs:element>

  <xs:element name="zvyrazneni">
   <xs:complexType mixed="true">
    <xs:attribute name="typ" type="zvyrazneniType" />
   </xs:complexType>
  </xs:element>

<!-- elementarni typy -->
  <xs:simpleType name="isbnType">
   <xs:restriction base="xs:string">
    <xs:pattern value="[ \d-]+"/>
   </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="zvyrazneniType">
   <xs:restriction base="xs:string">
    <xs:enumeration value="tucne" />
    <xs:enumeration value="kurziva" />
    <xs:enumeration value="podtrzeni" />
   </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<dokument>
	<nadpis>
		<veta>Kapitola 1</veta>
	</nadpis>
	<nadpis>
		<veta>Podkapitola 1.1</veta>
	</nadpis>
	<odstavec>
		<veta>Veta1.</veta>
		<veta>Veta2.</veta>
	</odstavec>
	<nadpis>
		<veta>Podkapitola 1.2</veta>
	</nadpis>
	<odstavec>
		<veta>Veta4.</veta>
		<veta>Veta5.</veta>
		<veta>Veta6.</veta>
	</odstavec>
</dokument>
```

2.2.3.3 XSL – vzhled dat

V tuto chvíli, kdy již máme definován interní formát a jeho strukturu je potřeba rozhodnout, jak jej budeme zobrazovat uživateli. Je pochopitelné, že uživatel, který do systému nahraje např. textový dokument, který je např. výstupem z programu Microsoft Word a má jistý vzhled a formátování, by nebyl příliš potěšen, kdybychom mu na obrazovku místo tohoto souboru zobrazili nenaformátovaný XML soubor. Na druhou stranu by bylo po technické stránce velmi náročné (a není to ostatně ani v požadavcích), zobrazit věrný klon uživatelem nahraný dokument, a tak je potřeba najít kompromis.

Naštěstí pro účely stylování a nastavování vzhledu libovolného XML souboru existuje standard **XSL**. Tento standard zahrnuje tři hlavní nástroje a to jazyk *XSLT*, *XSL-FO* a *XPath*. První z jmenovaných je jazyk, který slouží k převodu dat v XML souboru do libovolného formátu, nejčastěji do HTML. Druhý nástroj slouží k přesnému popisu vzhledu jednotlivých prvků výstupního formátu. A konečně třetím nástrojem je jazyk, díky kterému jsme schopni adresovat a přistoupit k libovolnému elementu či jeho atributu a hodnotě ve vstupním XML souboru.

Jsme tedy schopni zajistit, aby např. všechny elementy s názvem `<odstavec>` v našem souboru NPL byli převedeny do tvaru `<div class="odstavec" id="odstavec-{$i_odstavec}">`, kde zástupná proměnný parametr `{$i_odstavec}` supluje pořadí generovaného odstavce mezi všemi odstavci. Takto je učiněno z důvodu, abychom v budoucnu mohli k těmto elementům přistupovat prostřednictvím *DOM* operací, např. pro účely zvýrazňování textu. Podrobný a celý proces transformace je vidět na následujícím kódu níže.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="UTF-8" />

 <!-- hlavni sablona -->
 <xsl:template match="/dokument">

  <div class="testSoubor" id="testSoubor">
   <xsl:for-each select="./nadpis | ./odstavec">
    <xsl:variable name="elemName" select="name(self::node())"/>
    <xsl:choose>
     <xsl:when test="$elemName = 'nadpis'">
      <xsl:apply-templates select="self::node()" />
     </xsl:when>
```

```
      <xsl:when test="$elemName = 'odstavec'">
       <xsl:apply-templates select="self::node()" />
      </xsl:when>
     </xsl:choose>
    </xsl:for-each>
   </div>

 </xsl:template>

 <!-- sablony hlavnich elementu (nadpis, odstavec) -->
 <xsl:template match="nadpis">
   <!-- cislovani -->
   <xsl:variable name="i_nadpis">
    <xsl:number count="nadpis" level="single" format="1" />
   </xsl:variable>

   <h3 id="nadpis-{$i_nadpis}">
    <xsl:apply-templates select="./veta">
     <xsl:with-param name="parentType" select="'nadpis'" />
     <xsl:with-param name="parentPos" select="$i_nadpis" />
    </xsl:apply-templates>
   </h3>

 </xsl:template>

 <xsl:template match="odstavec">
   <xsl:variable name="i_odstavec">
    <xsl:number count="odstavec" level="single" format="1" />
   </xsl:variable>

   <div class="odstavec" id="odstavec-{$i_odstavec}">

     <xsl:apply-templates select="./veta">
      <xsl:with-param name="parentType" select="'odstavec'" />
      <xsl:with-param name="parentPos" select="$i_odstavec" />
     </xsl:apply-templates>

   </div>

 </xsl:template>

 <xsl:template match="veta">
  <xsl:param name="parentType"/>
  <xsl:param name="parentPos"/>
  <xsl:variable name="i_veta">
   <xsl:number count="veta" level="single" format="1" />
  </xsl:variable>

    <div class="veta" id="{$parentType}-{$parentPos}_veta-{$i_veta}"
         onmouseover="">
      <xsl:value-of select="text()" />
    </div>

</xsl:template>
</xsl:stylesheet>
```

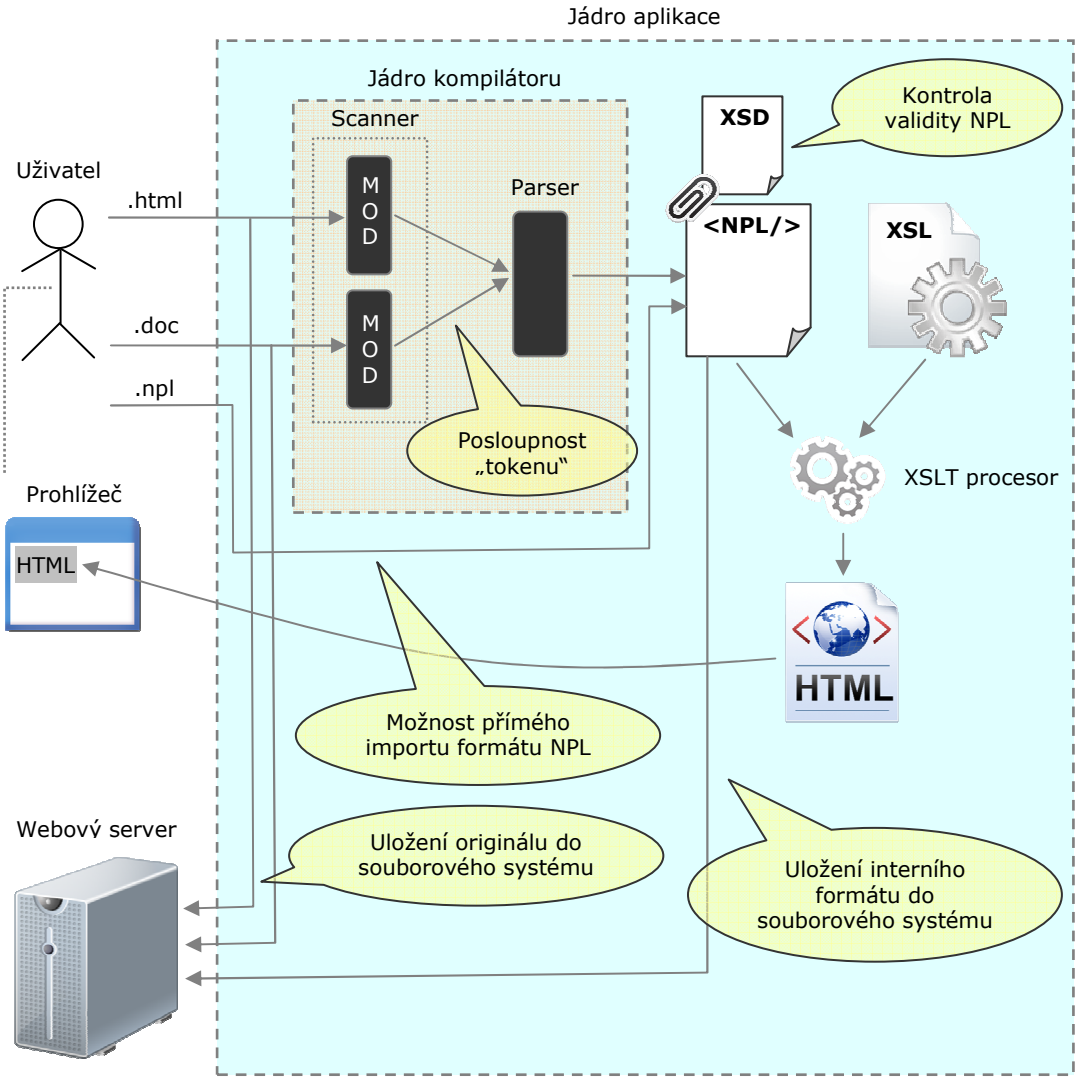
2.2.3.4 Kompilátor – transformace do NPL

Datový model již máme téměř kompletní, avšak zbývá nám vyřešit poslední otázku a tj. jakým způsobem budeme transformovat vstupní testovaný soubor do našeho interního formátu NPL. Podobně jako při analýze zpracování vstupního souboru, kde by bylo velice problematické a neefektivní vytvářet pro každý vstupní formát samostatné, nezávislé rozhraní, jsme došli k závěru zavedení interního formátu i zde vyvstává obdobný problém. Implementovat transformaci pro každý vstupní formát by bylo poněkud pracné. Řešení však existuje – použít jednoduchý **kompilátor**.

Principielně se totiž každý kompilátor skládá ze dvou nástrojů: **scanner** a **parser**. Přičemž základní a hodně zjednodušená funkcionalita je taková, že scanner zpracovává a analyzuje vstupní text a vybírá z něj tzv. **tokeny** a ty pak následně posílá parseru. Token si lze představit jako nedělitelnou posloupnost znaků, která ve vstupním (kompilovaném) textu plní určitou funkci. Pokud bychom jako vstupní text měli HTML soubor, pak jako token si lze představit posloupnost znaků „`<div>`“, přičemž tento token by mohl být pojmenován jako „`odstavec`“. Parser následně tento tok tokenů zpracovává, podle své implementace a konfigurace a může již vytvářet zkompilovaný výstup. Toto je opravdu nejelementárnější popis fungování kompilátoru a jeho další popis by daleko přesahoval rámec této práce. Další informace, zejména o implementaci našeho kompilátoru, lze nalézt v programátorské příručce této práce.

Výhoda použití tohoto nástroje spočívá v tom, že pokud bychom takto implementovali transformaci např. pro formát HTML, musíme vytvořit příslušný HTMLscanner a parser. Pak pro další formát, např. dokument textového procesoru, stačí vytvořit pouze daný scanner (v tomto případě DOCscanner) a zajistit, aby vracel stejný, resp. jednotný formát tokenů pro parser. To lze po implementační stránce zařídit poměrně snadno, např. použitím *interface*, ze kterého musí každý scanner dědit. Pokud uvážíme, že daný kompilátor zároveň obsahuje obslužné knihovny pro práci se vstupem a výstupem (např. načtení souboru, atd.) a v jistých případech také není potřeba tyto knihovny modifikovat, stává se požadovaná transformace do NPL, použitím kompilátoru, poměrně elegantní a efektivní.

Obrázek níže částečně rozšiřuje původní obrázek (Obr. 11) a popisuje vnitřní funkcionalitu aplikace z pohledu datového modelu.



Obr. 13 – Datový model nové aplikace

2.2.4 Metody analýzy textu

S odkazem na návrh uživatelského rozhraní lze prohlásit, že jedno z hlavních stěžení nového systému by mělo být poskytnutí více či méně sofistikovaných metod, které budou analyzovat NPL dokument (vytvořený kompilací vstupního testovaného souboru), extrahovat příslušné elementy (nadpisy, odstavce, apod.), sestavovat pro tyto prvky vyhledávací dotazy a ty pak následně předkládat uživateli. Pro další účely nazvěme sloučení elementu a dotazu jako **vyhledávací prvek**.

Uživatel poté bude mít výběr, který z těchto vyhledávacích prvků, použije pro vyhledávací službu. Díky tomuto přístupu může uživatel velice rychle (pouhým kliknutím) otestovat velké množství vyhledávacích prvků z různorodých metod a hloubkově prověřit, zdali testovaný dokument neobsahuje části, které jsou plagiáty.

V kapitole 1.4 jsme se seznámili se základními postupy a metodami při obsahové analýze textu. Metody typu *fingerprint* či *winnowing* však počítají nejen s přístupem k testovanému souboru, ale i k určitému archívu dokumentů, vůči kterým je testovaný soubor porovnáván. Bohužel pro nás je tímto archívem **nepředvídatelná množina dokumentu**, které jsou výstupem vyhledávací služby, a aplikování těchto sofistikovaných metod by bylo po technické stránce poměrně komplikované a nad rámec této práce (může být však součástí dalšího rozšiřování).

Našim hlavním cílem tedy bude vytvoření několika základních metod a hlavně implementace zjednodušeného *API* pro **vytváření a přidávání nových metod**. Účelem tohoto API bude zajistit, aby budoucí vývojář mohl implementovat a integrovat novou metodu do systému, aniž by musel znát jeho vnitřní strukturu. Je důležité poznamenat, že vestavěné metody nebudou nikterak vynikat svoji komplexností. Lze si představit, že by daná metoda vyextrahovala všechny cizí termíny dle slovníku, ale její implementace by nebyla triviální a přesahovala by zadání této práce.

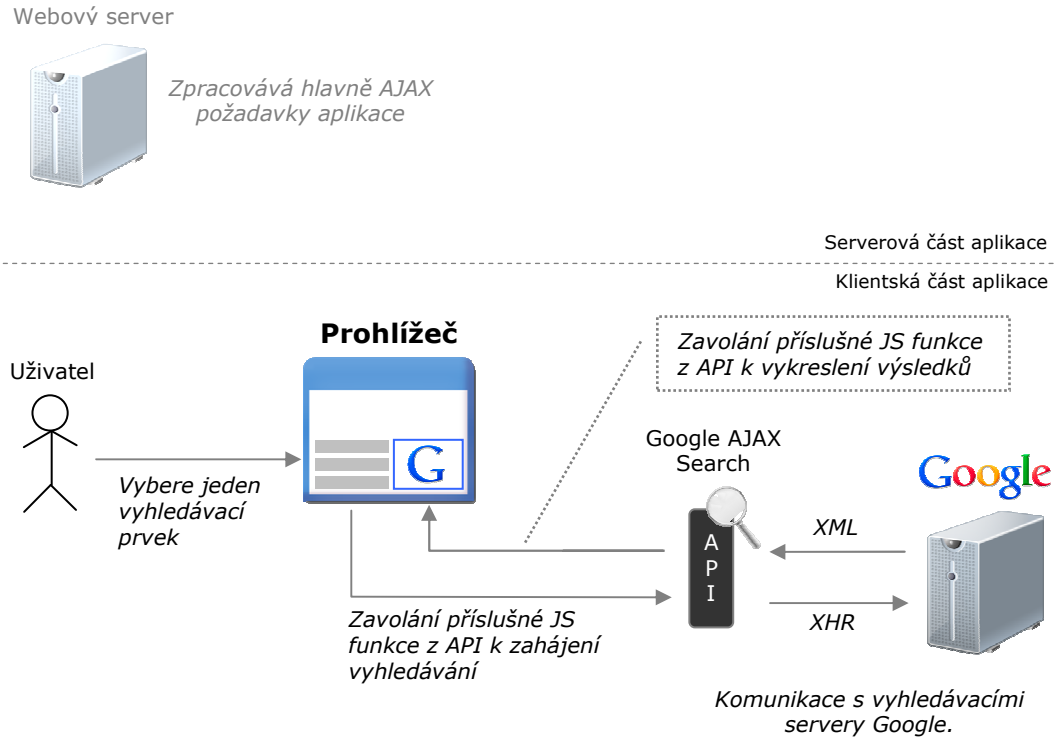
2.2.5 Komunikace s vyhledávací službou

Zajištění komunikace s vyhledávací službou je nepostradatelným motorem nové aplikace. Hlavním požadavkem této části je zajistit komunikaci s vyhledávací službou Google. Pro tyto účely se nám skvěle hodí již existující rozhraní **Google AJAX Search API**, vytvořené pro vývojáře, kteří chtějí využít vyhledávače Google pro svoje potřeby.

Již z názvu tohoto rozhraní vyplívá, že je postaveno na technologii AJAX. Jedná se tedy o knihovnu napsanou v jazyce *JavaScript*, která poskytuje poměrně široké spektrum objektů a metod, tak aby vývojář mohl rozhraní pohodlně implementovat do své aplikace. Hlavní princip (popsaný ve vývojářské příručce) spočívá v zahrnutí takzvaného *Google AJAX API Loader* do hlavičky webové stránky, který umožňuje jednoduchým způsobem použít více existujících API (mapy, vyhledávání, RSS feed) pomocí jednoho jmenného prostoru a jednoho **API klíče**. Tento klíč není povinný při

používání Google API a jeho užitečnost spočívá pouze z hlediska podpory ze strany společnosti Google, která je díky tomuto klíči schopná identifikovat stránku, v případě vzniku jakéhokoliv problému s Google API.

Zajímavé na této knihovně je, že pro svoji činnost **nepotřebuje žádné skripty na straně serveru běžící aplikace**, jinými slovy – běží pouze v klientské části aplikace, resp. v prohlížeči uživatele. Tato knihovna lze také použít pro jiné než javascriptové prostředí (např. *Flash*), přičemž základní princip spočívá v odeslání *HTTP* požadavku ve formátu *GET* a zpracování odpovědi ve formátu *JSON*. Bližší informace o této knihovně a její konkrétní použití lze najít v programátorské příručce této práce.



Obr. 14 – Komunikace s vyhledávací službou

2.3 Návrh uživatelského rozhraní

Správně navrhnout a realizovat uživatelské rozhraní nové aplikace rozhoduje o budoucí spokojenosti uživatelů této aplikace. Při navrhování je kladen důraz nejenom na logickém rozmístění jednotlivých kontejnerů s daty a ovládacích prvků, ale i na případné budoucí doplnění dalších elementů a s tím spojenou předchozí rezervaci volného místa. Samozřejmě se nesmí zapomínat i na barevnou konzistenci aplikace, přičemž barevné schéma může být odvozeno buď od konkrétního způsobu použití aplikace, nebo např. podle CI (*Corporate Identity*).

2.3.1 Požadavky & základní layout

Nejprve je potřeba rozhodnout o základní koncepci ovládání aplikace. Naše webová aplikace by se koncepčně měla skládat ze **čtyř hlavních částí**. Tou první by měl být kontejner s původním testovaným dokumentem ve zkompilované podobě, tj. ve formátu NPL. Druhá část by měla být jakási tabulka s vyhledávacími prvky, které vznikly na základě analýzy textového obsahu NPL dokumentu a ze kterých si uživatel může vybrat pro účely vyhledávání. Ve třetí části by měli být k dispozici výsledky vyhledávací služby. A v poslední části nalezený dokument, jakožto jeden z výsledků vyhledávací služby, vůči kterému je testovaný dokument porovnáván.

Možných variant, jak tuto základní koncepci realizovat, je několik. Pokud si ovšem uvědomíme, že chceme aby aplikace byla přehledná a ušetřila uživatele od zbytečného klikání (způsobené např. přechody mezi jednotlivými hlavními částmi), bude nejvýhodnější tyto čtyři hlavní části umístit do jednoho okna, resp. zajistit jejich **dlaždicové uspořádání**.

Dále by bylo vhodné implementovat v uživatelském rozhraní roletové menu s jednotlivými akcemi a nastavením, tak jak jej známe z běžných *GUI* aplikací. Výhoda této komponenty je nesporná – v budoucnu lze téměř neomezeně rozšiřovat o další akce a možnosti. S ohledem na požadavek, aby do systému mohl být nahrán celý archív (např. v *ZIP* formátu) s libovolným počtem testovaných souborů, které po nahrání budou automaticky otevřeny, je potřeba rezervovat místo pro pruh s jednotlivými záložkami těchto otevřených souborů.

Obr. 15 – Základní layout nového systému

2.3.2 Použitelnost a flexibilita

Z pohledu předchozí kapitoly vyplívá, že velmi důležitým kritériem bude **použitelnost** & **flexibilita** uživatelského rozhraní. Aby bylo porovnávání skutečně efektivní a použitelné, je potřeba zajistit zvýrazňování vyhledávacích prvků jak v testovaném tak nalezeném dokumentu, aby nakonec uživatel mohl rychle rozhodnout, zdali je shoda natolik vysoká, že testovaný dokument je více či méně plagiátem. Dále by bylo velmi vhodné a elegantní, kdyby uživatel mohl libovoně jednotlivé hlavní části skrývat, resp. zobrazovat. To by se uplatnilo např. v situaci, kdy by uživatel vzájemně porovnával rozsáhlé soubory (testovaný a nalezený dokument) – mohl by tak dočasně skrýt tabulku s vyhledávacími prvky a panel s výsledky vyhledávací služby a tím by získal více prostoru pro účely vzájemného porovnávání souborů.

Pokud bychom s implementací uživatelského rozhraní začínali na „zelené louce“ a měli bychom splnit všechny předchozí požadavky, cíle a funkcionality, bylo by to velmi pracné. Naštěstí existuje javascriptový framework s názvem **ExtJS**, který je nezávislý na prohlížeči (*cross-browser*) a umožňuje vývoj tzv. RIA (*Rich Internet Application*). Tento framework existuje ve dvou licencích (komerční a open-source), obsahuje velké množství přizpůsobitelných UI (*User Interface*) komponent a mezi jeho hlavní výhody patří snadné použití API a kvalitně zpracovaná dokumentace. Další informace a ukázky použití tohoto frameworku lze najít v programátorské příručce.

3 NetPlag

Následující odstavce pojednávají o samotném produktu, který byl vytvořen na základě předchozí analýzy. Jejich úkolem je zajistit pokud možno bezproblémovou instalaci systému, seznámit uživatele s ovládáním a používáním systému a samozřejmě umožní budoucím vývojářům nahlédnout „pod kapotu“ systému a nastínit základní principy fungování systému, tak aby jeho budoucí rozšiřování bylo co nejpohodlnější. Název systému byl stanoven na **NetPlag**, což je akronym od sousloví „Internet Plagiarism“, resp. „Internetové Plagiátorství“.

3.1 Instalační příručka

3.1.1 HW požadavky

Po hardwarové stránce není systém nikterak náročný. Ze serverového pohledu samotný systém vyžaduje přibližně **60 MB** volného místa na disku, přičemž je potřeba zajistit volné místo pro importované dokumenty (které jsou uchovávány právě v souborovém systému). Samotný webový server Apache (s podporou PHP) není náročný na zdroje, avšak platí že čím výkonnější stroj, tím rychlejší zpracování. Z uživatelského pohledu je systém náročný především na paměť a pro svůj optimální běh si vyžádá alespoň **100 MB** paměti RAM. To je způsobeno především nahráním rozsáhlého frameworku ExtJS při každém spuštěním systému.

Pozn.: Systém byl vyvíjen a testován na konfiguraci s procesorem AMD Athlon 64 3000+ 1,81 GHz, pamětí DDR 3GB 400MHz a grafickou kartou NVidia GeForce 6600 128MB, a jeho běh se dal považovat za slušný. Mírná „neplynulost“ byla objevena během přizpůsobování uživatelského rozhraní (skrývání jednotlivých panelů), což se pravděpodobně dá přičíst na vrub uvedené grafické kartě.

3.1.2 SW požadavky

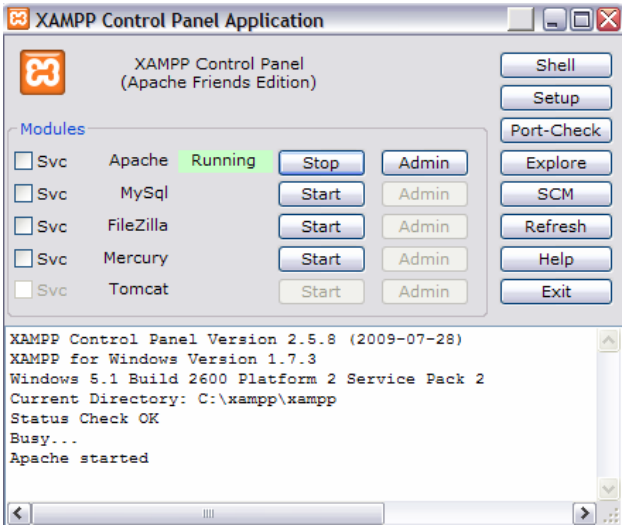
Jak již z analýzy vyplívá, hlavním softwarovým požadavkem pro běh systému bude zprovoznění webového serveru Apache a jeho modulu PHP, přičemž v budoucnu se předpokládá i možná spolupráce s databázovým systémem MySQL (dále jen „služby“). Tyto služby jsou multiplatformní, jinými slovy lze je provozovat na libovolném *OS*. Pro naše další účely ale budeme předpokládat, že služby poběží na operačním systému Windows XP SP2.

Po uživatelské stránce, je jediným SW požadavkem mít k dispozici pokud možno moderní internetový prohlížeč, jako je např. Internet Explorer 6, Mozilla Firefox 1.5, Opera 9, Chrome 3 nebo Safari 3. Samozřejmě s novějšími verzemi stejnojmenných prohlížečů by neměl být problém, naopak ve starších prohlížečích není podpora zaručena (zejména ze strany knihovny ExtJS).

3.1.3 Instalace

Existuje několik způsobů jak služby zprovoznit a zkušený správce sítě by si s tímto úkolem hravě poradil. Pro naše účely si vybereme ten nejjednodušší a nejrychlejší způsob, tj. prostřednictvím nástroje **XAMPP**.

Tento nástroj má ve svém výchozím nastavení všechny služby a moduly povoleny (např. modul PHP pro Apache), což je výhodou zejména ve fázi vývoje a testování, naopak v produkčním provozu se toto nastavení stává nevýhodou. K dispozici je na domovské stránce http://www.apachefriends.org/en/xampp.html, odkud je možno si jej zdarma stáhnout pro verzi OS Windows. Po stažení a spuštění instalace je potřeba do pole „Destination folder“ zadat cestu „*C:\xampp*“ a kliknout na tlačítko „Install“. Po dokončení instalace a spuštění nástroje, se zobrazí jednoduchý ovládací panel, ze kterého je potřeba nastartovat námi požadované služby (Apache). Poté si lze správnost tohoto kroku, tj. instalace a spuštění webového serveru Apache, ověřit zadáním adresy *„http://localhost“* do prohlížeče, přičemž by se nám měla zobrazit domovská stránka nástroje XAMPP.



Obr. 16 – Ovládací panel nástroje XAMPP

Následující kroky popisují instalaci, konfiguraci a první spuštění již samotného systému NetPlag. Tento postup předpokládá, že je k dispozici instalační CD systému a je již nainstalován a spuštěn webový server Apache (např. podle návodu v předcházejícím odstavci).

1. Rozbalení archívu „*netplag_install.zip*“ (obsahující systémové soubory), nacházejícího se na instalačním CD, do umístění „*C:\xampp\xampp\htdocs*“

2. Přidat následující úryvek kódu, resp. záznam, do souboru „*httpd-vhosts.conf*“ nacházející se v adresáři „*c:\xampp\xampp\apache\conf\extra*“

```
<VirtualHost *:80>
    ServerAdmin Stork71@seznam.cz
    DocumentRoot "C:/xampp/xampp/htdocs/netplag_dp"
    ServerName netplag.cz
    ServerAlias www.netplag.cz
    ErrorLog "logs/netplag-error.log"
    CustomLog "logs/netplag-access.log" combined
</VirtualHost>
```

3. Přidat následující úryvek kódu, resp. záznam, do souboru „*hosts*“ nacházející se v adresáři „*C:\Windows\System32\drivers\etc*“

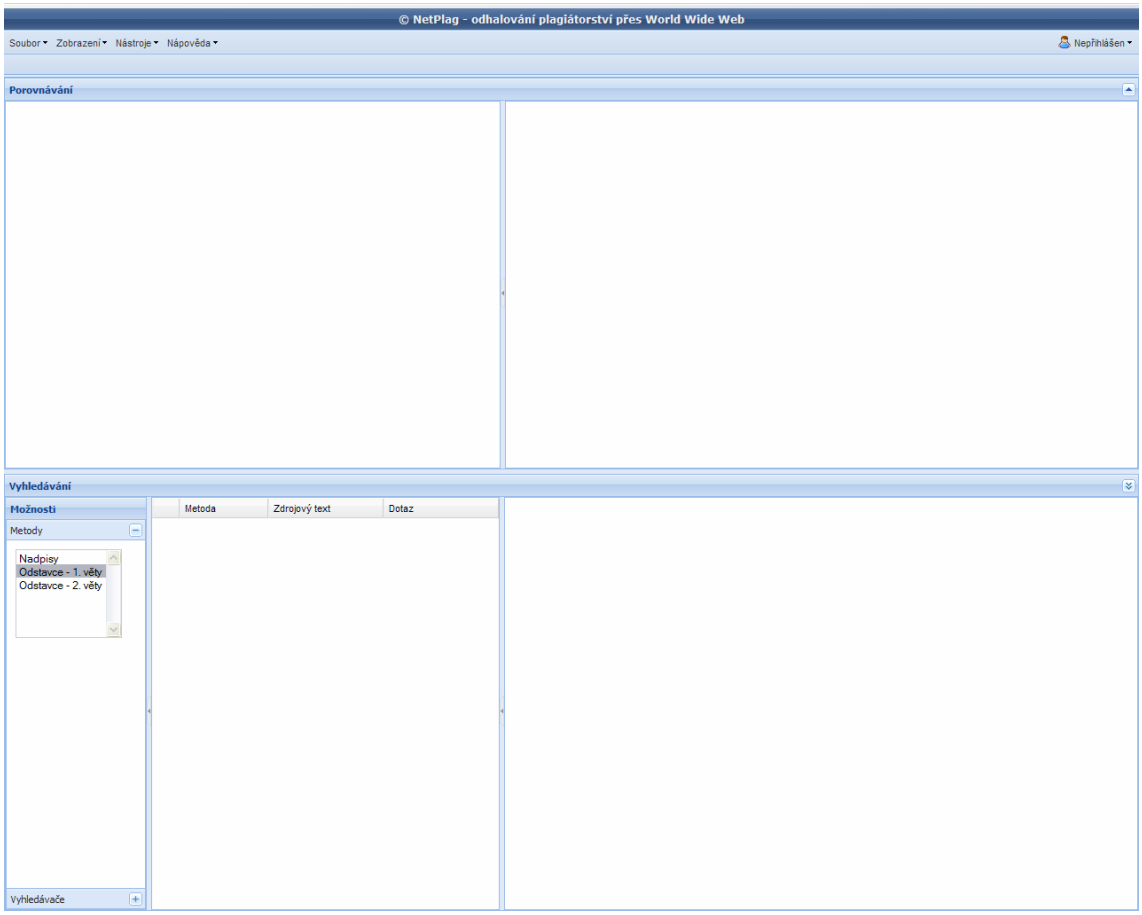
```
127.0.0.1       www.netplag.cz
```

4. Zprovoznit knihovnu *cURL* v modulu PHP
    a. Odkomentovat řádek, resp. odstranit znak středníku na začátku řádku, který obsahuje text „`extension=php_curl.dll`“ v konfiguračním souboru „*php.ini*“ v adresáři „*c:\xampp\xampp\php*“
    b. Zkopírovat soubory „*ssleay32.dll*“, „*libeay32.dll*“ z adresáře „*c:\xampp\xampp\php*“ do adresáře „*C:\Windows\System32*“
    c. Zkopírovat soubor „*php_curl.dll*“ z adresáře „*c:\xampp\xampp\php\ext*“ do adresáře „*C:\Windows\System32*“
    d. Restartovat webový server Apache (např. pomocí konzole XAMPP)

5. Spustit systém zadáním adresy „http://www.netplag.cz“ do internetového prohlížeče

Pozn.: Tento postup zajišťuje spuštění systému NetPlag na počítači, který zároveň plní funkci webového serveru (má spuštěnou službu Apache). Systém NetPlag lze jednoduše spustit i z jiného počítače, který se nachází ve stejné vnitřní síti (s neveřejnými IP adresami) jako webový server, a to pouhým provedením kroku č. 3 na daném počítači. Pouze místo IP adresy 127.0.0.1 (označující loopback) je potřeba zadat IP adresu webového serveru, ve kterém se nachází systém NetPlag (např. 192.168.1.100).

Po provedení předchozího postupu by se po posledním kroku měla zobrazit domovská stránka systému NetPlag, která je zároveň hlavním uživatelským rozhraním. Pokud se tak nestalo, je potřeba kontaktovat autora této práce, nebo jinou kvalifikovanou osobu (programátora), která bude schopna problém odstranit.



Obr. 17 – Domovská stránka systému NetPlag

3.2 Uživatelská příručka

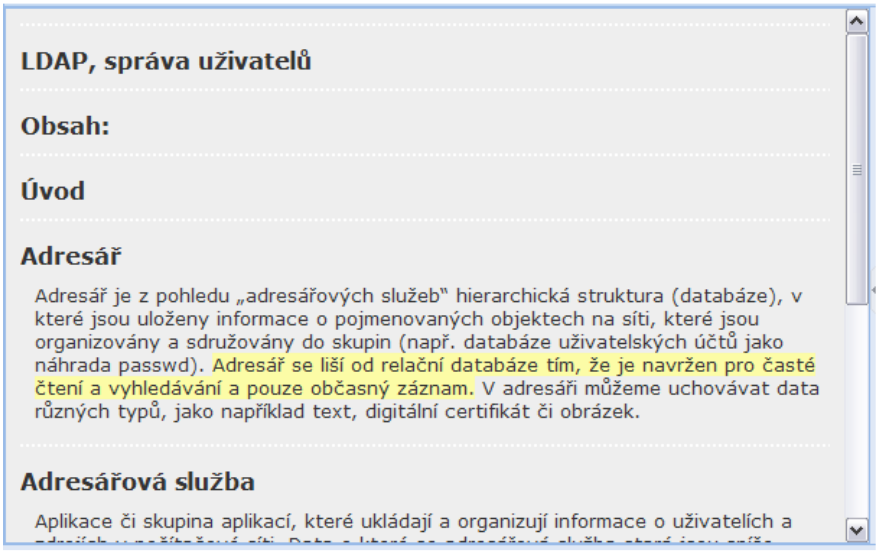
3.2.1 Ovládání

V této kapitole jsou podrobně vysvětleny všechny části a sekce uživatelského rozhraní tak, aby se práce s ním stala pro uživatele pokud možno pohodlnou a bezproblémovou.

Základní vlastností uživatelského rozhraní je jeho **flexibilita**. Hlavní okno aplikace nemá fixně zadané rozměry, tzn. že ať již je aplikace zobrazena na monitoru s jakýmkoliv rozlišením, vždy bude maximálně využita zobrazovací plocha daného monitoru, resp. prohlížeče. Zároveň pružnost celé aplikace podporují jednotlivé čtyři hlavní panely, u kterých lze dle potřeby měnit jejich šířku a výšku nebo je popřípadě dočasně celé skrýt.

3.2.1.1 Panel „vstupu“

Tento panel může být také označen jako **panel testovaného dokumentu** a z hlediska pozice mezi hlavními panely se nachází na imaginárním severozápadně (dále jen „SZ“). Jeho význam je zřejmý – je v něm zobrazen nahraný testovaný dokument, který je zároveň přetransformován do systémového formátu (tzv. NPL formát). Tento převod se vyznačuje hlavně tím, že z původního dokumentu jsou vyextrahovány pouze textové elementy (nadpisy, odstavce, apod.) a naopak prvky jako obrázky, grafy nebo tabulky jsou zahozeny. Zároveň je odstraněno i formátování textových elementů. Textové prvky jsou dále rozděleny podle jednotlivých vět, což si lze ověřit při pohybu myší nad těmito pod-elementy, které jsou rázem zvýrazněny žlutou barvou.



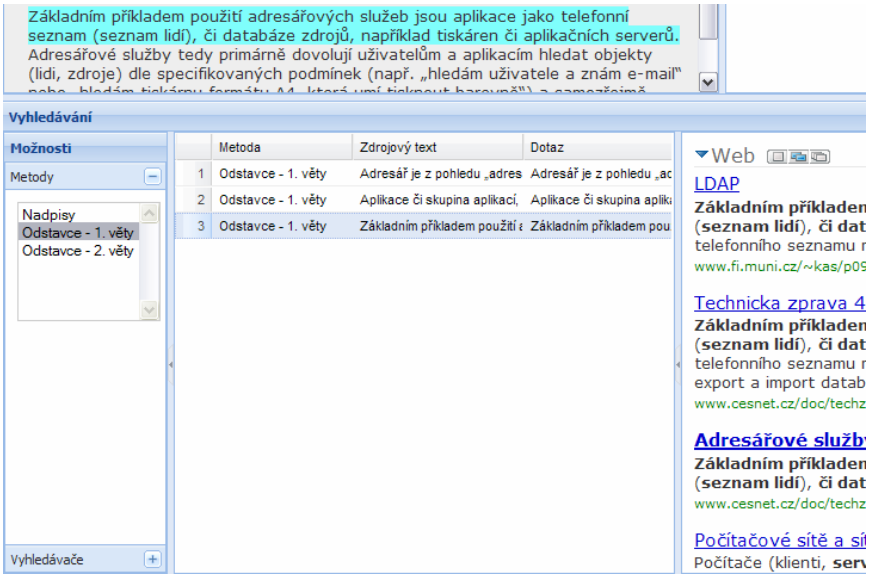
Obr. 18 – Panel vstupního dokumentu („severozápadní“ hlavní panel)

3.2.1.2 Panel analýzy obsahu

Tento se z hlediska pozice mezi hlavními panely nachází na imaginárním jihozápadně (dále jen „JZ“) a skládá se ze dvou částí. Tou první částí je roletové menu, které nabízí výběr metody analýzy obsahu a výběr vyhledávačů, které se „zapojí“ do prohledávání. Ve druhé části se nachází „grid“, resp. **tabulka s vyhledávacími prvky**.

Tato tabulka se skládá se **tří sloupců**. U každého vyhledávacího prvku lze tedy zjistit, která metoda ho vygenerovala (1. sloupec), z jaké části zdrojového textu byl sestaven vyhledávací dotaz (2. sloupec) a konečně samotný vyhledávací dotaz, který bude zaslán vybranému vyhledávači (3. sloupec).

U některých elementárních metod se může stát, že obsah 2. sloupce bude shodný s obsahem 3. sloupce, pro daný řádek. Nejedná se tedy o chybu aplikace a vše záleží na tom, jak je která metoda v systému implementována. Po výběru jakéhokoliv řádku (vyhledávacího prvku) z této tabulky se provedou **dvě zásadní operace** – tou první je zobrazení výsledků, resp. nalezených stránek, pro daný dotaz a vyhledávač do panelu vyhledávacích výsledků (JV) a ve druhé operaci se provede zvýraznění elementů, resp. vět, v panelu vstupu podle hodnoty ve druhém sloupci. Na následujícím obrázku je toto chování částečně nastíněno.



Obr. 19 – Panel analýzy obsahu („jihozápadní" hlavní panel)

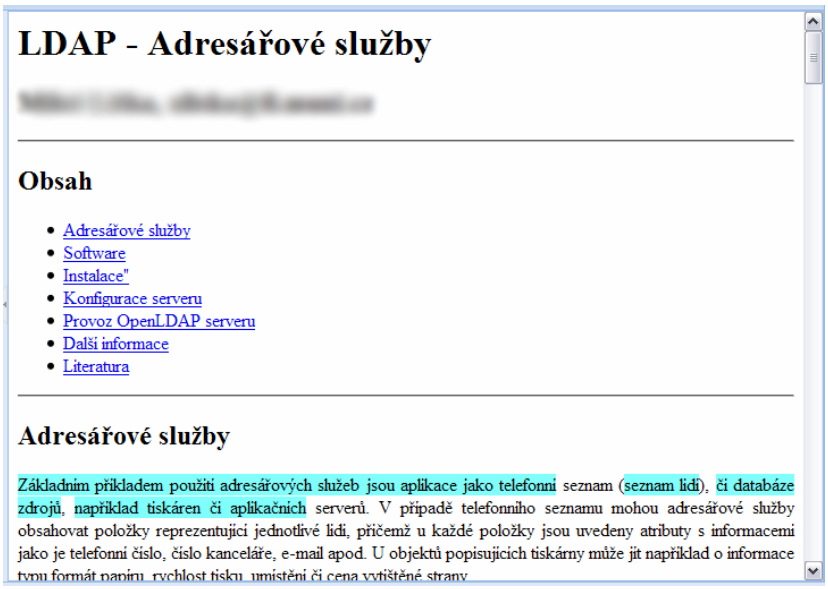
3.2.1.3 Panel výsledků vyhledávání

Tento kontejner plní jednoduchý účel. Zobrazuje výsledky vrácené vyhledávací službou po vybrání vyhledávacího prvku z tabulky, která se nachází v panelu analýzy obsahu. V současné době je zajištěna podpora pouze pro vyhledávací službu Google a proto je vzhled výsledků nastaven právě podle tohoto vyhledávače.

Obr. 20 – Panel výsledků vyhledávání („jihovýchodní“ hlavní panel)

3.2.1.4 Panel „výstupu“

Tento panel může být také označen jako **panel nalezeného dokumentu**. Jedná se o vnořený rám, přičemž uvnitř tohoto rámu se zobrazuje internetová stránka (obdobně jako se zobrazuje po zadání adresy do prohlížeče), podle toho, na který odkaz bylo kliknuto v panelu výsledků vyhledávání. Po tomto kliknutí a načtení stránky se automaticky provede **zvýraznění** a **odrolování** na takové textové sousloví ve stránce, které je zároveň tučně zvýrazněno v odkazu, na který bylo kliknuto v panelu výsledků vyhledávání. Uživatel má mimo jiné možnost si nastavit, aby stránky byly zobrazovány bez stylování (CSS), čímž mimo jiné urychlí načítání této stránky.



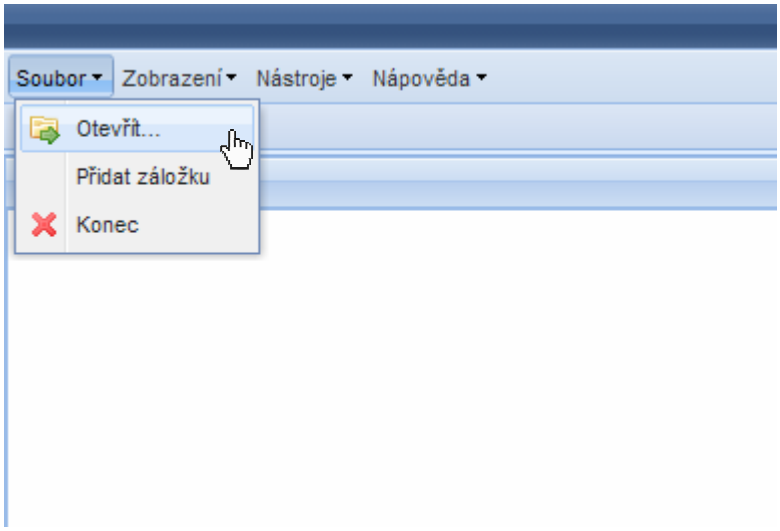
Obr. 21 – Panel výstupu („severovýchodní“ hlavní panel)

3.2.2 Ukázkový příklad odhalení plagiátu

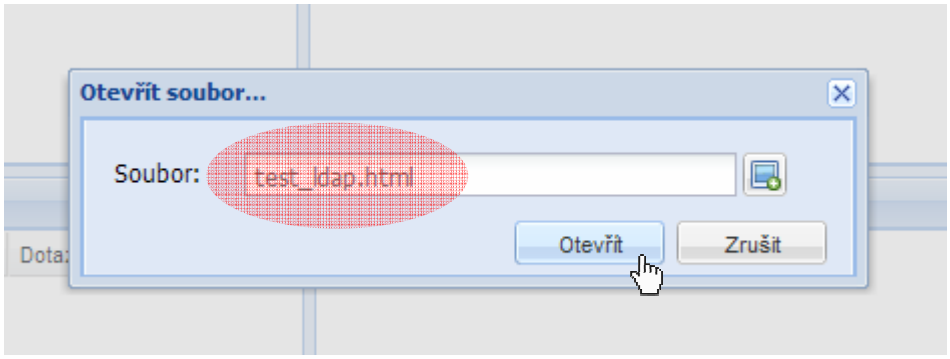
Následující postup je pro méně zdatné uživatele a ukazuje, jakým způsobem lze ověřovat, zda daný dokument je či není plagiátem. Výchozím předpokladem k vyzkoušení tohoto postupu je mí nainstalovaný a spuštěný systém NetPlag (o tom pojednává úvodní kapitola této uživatelské příručky).

1. Nahrajte do systému dokument pro účely testování s názvem „*test_ldap.html*“, který se nachází v systémové složce (*c:\xampp\xampp\htdocs\netplag_dp*) v adresáři „*TEMP*“. Pokud k systémové složce nemáte přístup, lze adresář „*TEMP*“ také najít v instalačním CD systému NetPlag.

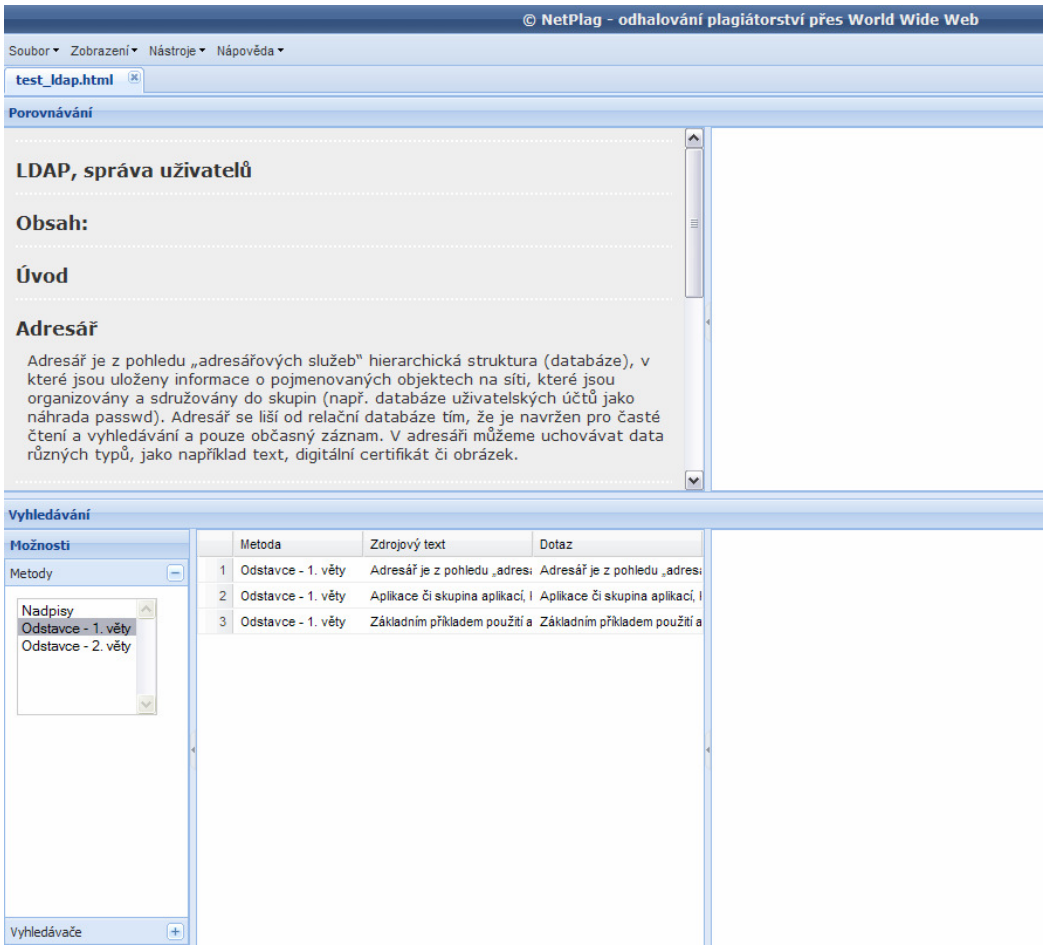
- *Klikněte na položku v menu „Soubor“ a dále „Otevřít“*



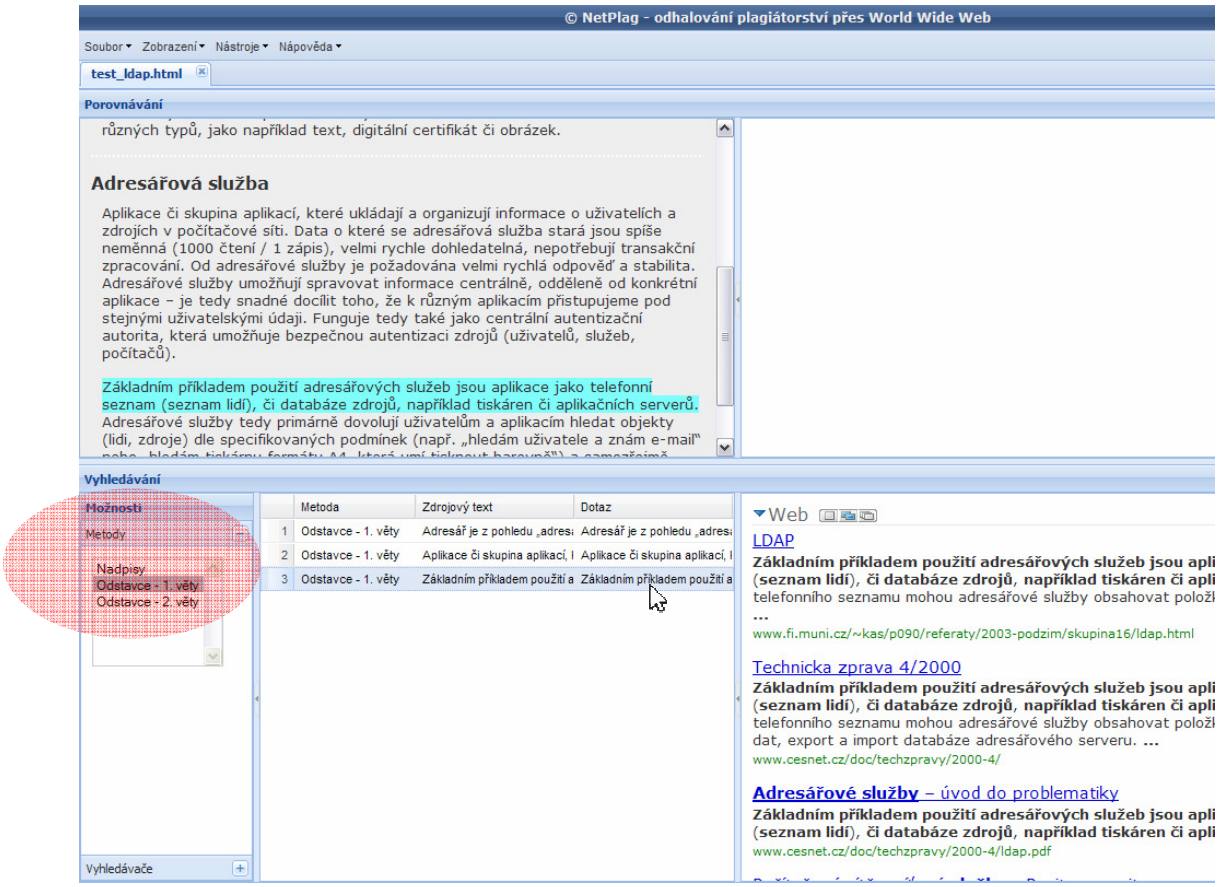
- *Vyberte příslušný soubor v daném adresáři a klikněte na „Otevřít“*



Pozn.: Po otevření se testovaný dokument převede do formátu NPL, zobrazí v panelu vstupu (SZ), vytvoří se pro tento dokument záložka s jménem a automaticky se naplní tabulka vyhledávacími prvky (JZ), dle výchozí metody analýzy obsahu.

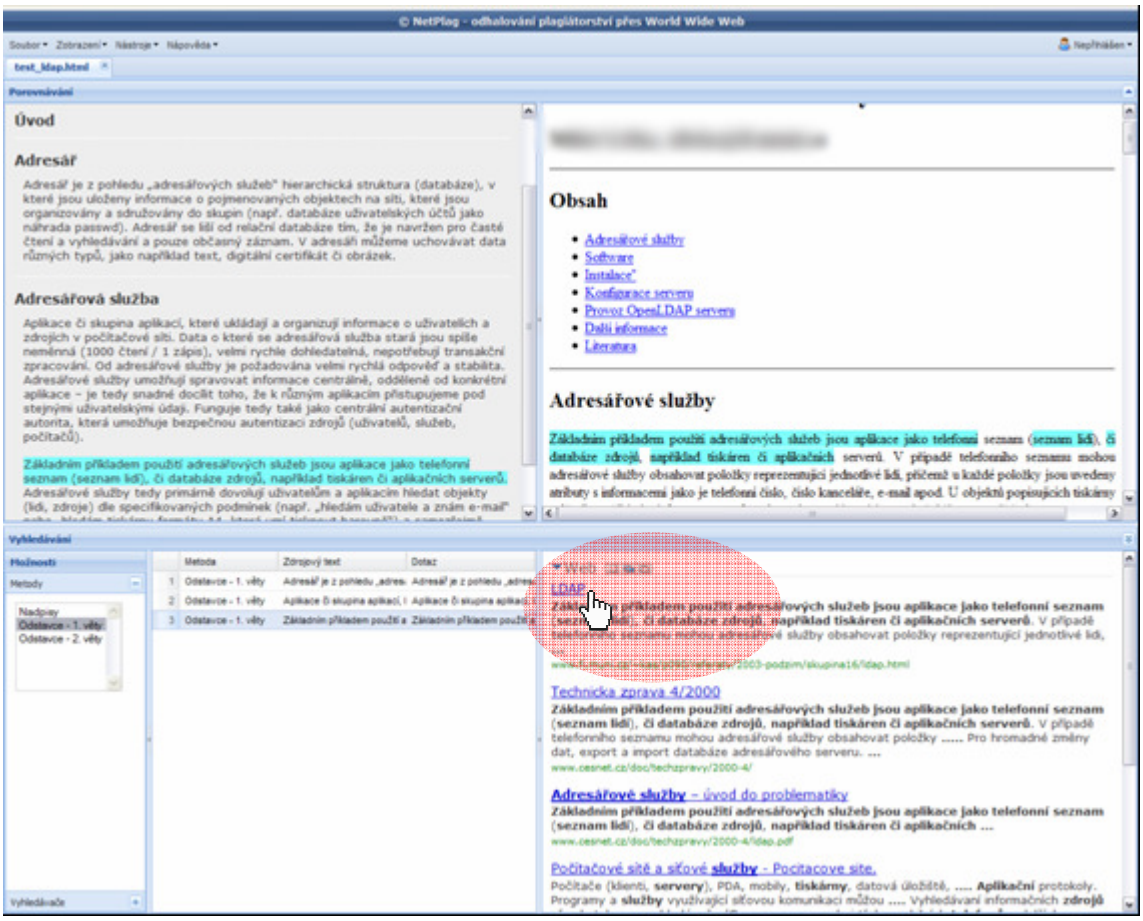


2. Pokud tomu tak není, vyberte metodu „*Odstavce – 1. věty*“ (přístupná otevřením položky „*Metody*“ v levém roletovém menu v JZ) a klikněte na 3. řádek v tabulce vyhledávacích prvků v JZ.



Pozn.: Kliknutím na vyhledávací prvek se zahájí vyhledávání (jehož výsledky se zobrazí na JV) a automaticky se zvýrazní zdrojový text vyhledávacího prvku na SZ (včetně automatického odrolování na tento text).

3. Pokud tomu tak není, rozbalte sekci „*Web*“ na JV a klikněte na první odkaz v této sekci. Všechny odkazy jsou vráceny vyhledávačem, který je zvolen v roletovém menu „Možnosti“ na JZ.



Pozn.: Kliknutím na odkaz se cílová stránka zobrazí na SV a v této stránce se zvýrazní takové sousloví (včetně automatického odrolování na toto sousloví), které je shodné s tučným textem, který se nachází pod odkazem, na který bylo kliknuto.

4. Následně, podle zvýrazněné shody (v testovaném a nalezeném dokumentu) a absenci citačního označení u této shody ve vstupním dokumentu (SZ), můžeme prohlásit, že zvýrazněný textový element v testovaném dokumentu je plagiátem.

3.3 Programátorská příručka

V poslední kapitole této práce jsou objasněny všechny základní principy a technologie na kterých je systém NetPlag postaven. Tato kapitola je určena zejména pro budoucí vývojáře, kteří se chystají systém NetPlag rozšiřovat či modifikovat. Zároveň je důležité poznamenat, že dalším vodítkem může být samotný zdrojový kód, jelikož při vývoji byl také kladen důraz na kvalitní okomentování všech implementovaných objektů a jejich metod.

3.3.1 Použité technologie

Následující tabulka uvádí verze všech technologií a nástrojů, pomocí kterých byl systém NetPlag vyvíjen a testován.

Tab. 2 – Verze všech podpůrných technologií při vývoji

Operační systém	Windows XP Professional SP2
Vývojové prostředí (IDE)	Eclipse PDT (PHP Development Tools) 2.1.1 http://www.eclipse.org/pdt/downloads/
Webový server	Apache 2.2.14 (součástí XAMPP 1.7.3) http://www.apachefriends.org/en/xampp.html
Skriptovací jazyk	PHP 5.3.1 (součástí XAMPP 1.7.3) http://www.apachefriends.org/en/xampp.html
PHP framework	Kohana 2.3.4 http://kohanaframework.org/download
JS/RIA framework	ExtJS 3.1.1 http://www.sencha.com/products/js/download.php
Vyhledávací služba	Google AJAX Search API 1.0
Prohlížeč	Mozilla Firefox 3.6.6 http://firefox.mozilla.cz/

3.3.2 Framework Kohana

Jedná se o jednoduchý, rychlý a bezpečný PHP framework, který plně podporuje OOP/PHP5 a tvorbu webových aplikací pomocí návrhového vzoru MVC. Tento framework plně podporuje princip *DRY*.

Hlavní výhodou tohoto frameworku je paradoxně jeho **nekomplikovanost**, jelikož nabízí pouze skutečně obecné a široce použitelné principy (URL směrování na controller), moduly (autorizace & archivace) a knihovny (cache, database, *ORM*) a tzv. „helpery“ (pomocníky), přičemž spoléhá že konkrétní a specifické knihovny, pro daný účel použití, budou implementovány programátorem. Tento framework vychází ze svého staršího bratříčka *CodeIgniter*, který je ovšem vyvíjen společností a je bohužel napsán ve starší verzi PHP 4. V neposlední řadě je tento framework velmi slušně dokumentován a není náročný na pochopení.

Obr. 22 – Princip URL směrování

3.3.3 Framework ExtJS

Na rozdíl od frameworku Kohana, který je určen pro tvorbu „server-side“ aplikací, tento framework usnadňuje život vývojářům při tvorbě na **klientské straně**. V podstatě se jedná o rozsáhlou JavaScriptovou knihovnu. Jeho velkou výhodou oproti ostatním javascriptový frameworku (např. *JQuery*) je specializace na tvorbu UI widgetů a již v základu (bez přídavných pluginů) obsahuje rozsáhlé množství těchto komponent.

Implementace tohoto RIA frameworku plně dodržuje principy OOP, čehož si lze všimnou u samotných komponent, kde u podobných komponent se většina metod opakuje, což je samozřejmě způsobeno dědičností od jednoho předka. Další výhodou tohoto frameworku je jeho kvalitně zpracovaná dokumentace.

Systém NetPlag v sobě obsahuje jakési zjednodušené **výukové prostředí** (již zahrnující několik komponent) pro snazší a rychlejší seznámení a zároveň testování této knihovny. Pro vstup do tohoto prostředí stačí zadat http://www.netplag.cz/meet_extjs, přičemž jako pohled je použit soubor „*./application/views/learn_extjs_view.php*“ a jako hlavní konfigurační soubor všech použitých komponent na stránce je „*./js/extjs/extstart.js*“. Na následujícím úryvku kódu je vidět zkrácena verze tohoto prostředí (těchto dvou souborů), včetně příslušných komentářů.

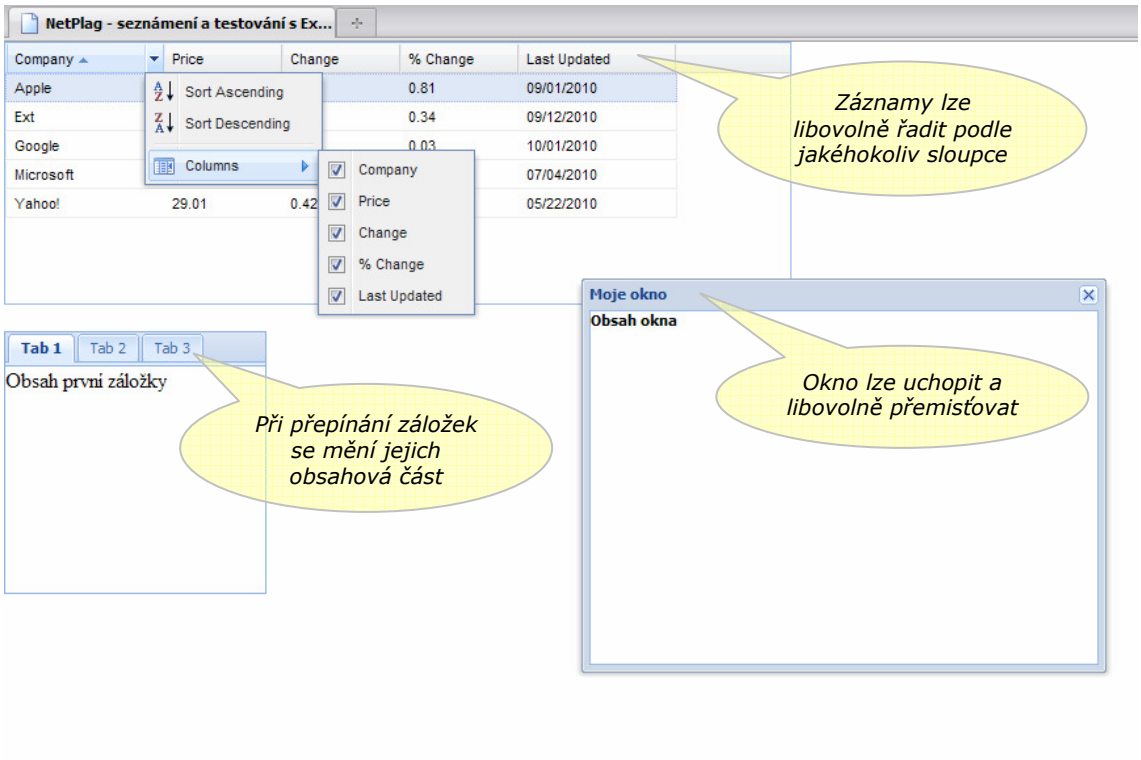
```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title>NetPlag - seznámení a testování s ExtJS</title>

        <!-- *** CSS *** -->
        <!-- hlavni knihovna -->
        <link rel="stylesheet" type="text/css"
                               href="/js/extjs/resources/css/ext-all.css" />

        <!-- *** Javascript *** -->
        <!-- ExtJS knihovna: zaklad/adapter -->
              <script type="text/javascript"
                      src="/js/extjs/adapter/ext/ext-base.js"></script>
        <!-- ExtJS knihovna: vsechny komponenty -->
              <script type="text/javascript"
                      src="/js/extjs/ext-all-debug.js"></script>

        <!-- prepisy hlavni knihovny -->
        <!-- ostatni rozsireni -->

        <!-- definice struktury stranky -->
             <script type="text/javascript"
                     src="/js/extjs/extstart_simple.js"></script>
    </head>
```

```
    <body>
            <div id="tabulka1">
                  <!-- zde bude umisten HTML zdroj komponenty "Grid" -->
            </div>
            <br />
            <div id="zalozky1">
                  <!-- zde bude umisten HTML zdroj komponenty "Tabpanel" -->
            </div>

            <!-- nasledujici kontejnery definuji "obsahovou cast" zalozek v
                 komponente "Tabpanel" -->
            <div id="tab1" class="x-hide-display">Obsah první záložky</div>
            <div id="tab2" class="x-hide-display">Obsah druhé záložky</div>
            <div id="tab3" class="x-hide-display">Obsah třetí záložky</div>
    </body>
</html>
```

```
// relativni cesta k prazdnemu obrazku (pruhledny GIF o rozmerech 1x1)
// slouzi k vytvareni radkovych ikon s pouzitim CSS
Ext.BLANK_IMAGE_URL = '/js/extjs/resources/images/default/s.gif';

Ext.onReady( function(){
          // *** Grid - tabulka s daty ***
          // deklarace a definice jednoducheho pole
          // (vetsinou jsou data nacitana napr. pomoci XHR (AJAX))
          var myData = [
              ['Apple',29.89,0.24,0.81,'9/1 12:00am'],
              ['Ext',83.81,0.28,0.34,'9/12 12:00am'],
              ['Google',71.72,0.02,0.03,'10/1 12:00am'],
              ['Microsoft',52.55,0.01,0.02,'7/4 12:00am'],
              ['Yahoo!',29.01,0.42,1.47,'5/22 12:00am']
          ];

          // vytvoreni tzv. "DataStore" (datove ulozite) ktery urcuje
          // jak maji byt cteny a formatovany data
          var ds = new Ext.data.SimpleStore({
              fields: [
                  {name: 'company'},
                  {name: 'price', type: 'float'},
                  {name: 'change', type: 'float'},
                  {name: 'pctChange', type: 'float'},
                  {name: 'lastChange', type: 'date', dateFormat: 'n/j h:ia'}
              ]
          });
          // nahrani pole do datoveho uloziste
          ds.loadData(myData);

          // vytvoreni tzv. "ColumnModel" (sloupcoveho modelu) ktery nam
          // umozni konfiguraci kazdeho sloupce v budoucim gridu
          var colModel = new Ext.grid.ColumnModel([
              {header: "Company", width: 120, sortable: true,
                dataIndex: 'company'},
              {header: "Price", width: 90, sortable: true, dataIndex: 'price'},
              {header: "Change", width: 90, sortable: true,
                dataIndex: 'change'},
              {header: "% Change", width: 90, sortable: true,
                dataIndex: 'pctChange'},
              {header: "Last Updated", width: 120, sortable: true,
               renderer: Ext.util.Format.dateRenderer('m/d/Y'),
               dataIndex: 'lastChange'}
          ]);
```

```
        // vytvoreni samotne komponenty "Grid", vcetne definice datoveho
        // uloziste a sloupcoveho modelu
        var grid = new Ext.grid.GridPanel({
               el: 'tabulka1',
               height:200,
               width:600,
               ds: ds,
               cm: colModel}
        );
        // vykresleni komponenty do kontejneru s ID="tabulka1"
        grid.render();
        // vybrani prvniho radku v gridu
        grid.getSelectionModel().selectFirstRow();


        // *** Tabpanel - system zalozek ***
        var panel = new Ext.TabPanel({
                width: 200,
                height: 200,
                activeItem: 0, // index or id
                items:[{
                    title: 'Tab 1',
                    // ID kontejneru ktery bude obsahovou casti zalozky
                    contentEl: 'tab1'
                    // prima HTML definice obsahove casti zalozky
                 // html: 'This is tab 1 content.'
                },{
                    title: 'Tab 2',
                    contentEl: 'tab2'
                },{
                    title: 'Tab 3',
                    contentEl: 'tab3'
                }]
        });
        panel.render(Ext.get('zalozky1'));


        // *** Window - klasicke okno (podobne jako v GUI aplikaci) ***

        var mySimpleWindow = new Ext.Window({
                id:'mySimpleWindow',
             width:400,
             height:300,
             autoScroll:true,
             title: 'Moje okno',
             html: '<div style="background-color:white; height:100%;">'
                        + '<h1>Obsah okna</h1>'
                    + '</div>'
        });
        mySimpleWindow.show();
    }

); // konec funkce "onReady"
```

Pozn.: Uvedený zdrojový kód, resp. tyto dva soubory, jsou nezávislé na frameworku Kohana, tzn. měly by být funkční i po osamostatnění. Samozřejmě je potřeba, aby všechny URL adresy vedly ke správnému cíly (zejména v hlavičce HTML dokumentu). Tato zjednodušená verze je také přístupná prostřednictvím adresy http://www.netplag.cz/meet_extjs/index/simple, přičemž používá stejnojmenné soubory, avšak se sufixem „_simple“. Po zadání adresy do prohlížeče by měl výsledek vypadat stejně jako na obrázku níže.



Obr. 23 – Zjednodušené výukové a testovací prostředí frameworku ExtJS

3.3.4 Google API

Vývojáře, kteří s tímto rozhraním pracují, zaujme na první pohled jeho relativní jednoduchost použití. V ideálních případech není potřeba vytvářet žádné „server-side“ skripty, k použití API nejsou za potřebí žádné registrace a ke všem objektům a metodám, které jsou k dispozici, je slušná dokumentace.

Jednoduchost použití však nikterak nesnižuje kvalitu a široké možnosti právě tohoto API. Rozhraní lze využít na prohledávání www, map, obrázků, videí, zpráv, elektronických knih, blogů, patentů nebo vlastní webové stránky (lokální vyhledávání). Pro všechny tyto kategorie jsou vytvořeny příslušné vyhledávače, resp. **vyhledávací objekty** (`google.search.WebSearch`, `google.search.ImageSearch`, atd.), které po vyhledání vrací stejnorodé objekty (`GwebResult`, `GimageResult`). Právě tyto objekty obsahují atributy které nesou např. titulek odkazu, URL odkazu nebo úryvek textového obsahu (ve kterém jsou tučně zvýrazněny shody s vyhledávacím dotazem) dané stránky. A konečně hlavní „gró“ spočívá v definici **návratové funkce** (*callback*), ve které lze se všemi těmito objekty pracovat. Tato definice se realizuje pomocí jedné z metod hlavního objektu celého API, který je instancí ze třídy `SearchControl`. Tento objekt také obsahuje velmi užitečnou metodu `draw(element, opt_drawOptions)`, diky které lze zařídit, aby hlavní objekt vykresloval výsledky vyhledávání do definovaného elementu. Samozřejmě lze, prostřednictvím CSS, těmto výsledkům upravovat vzhled.

Dále jsou k dispozici poměrně **široké možnosti nastavení vyhledávání**, jako např. omezení na konkrétní doménu, nastavení rozsahu vráceného výsledku, jazyk ve kterém mají být nalezené stránky a mnoho dalších. Velkou výhodou tohoto rozhraní je použitelnosti i pro jiná než javascriptová prostředí, přičemž princip spočívá v sestavení URL adresy (včetně příslušných parametrů) a po zavolání HTTP požadavku s touto URL je vrácena odpověď ve formátu JSON, která je dle potřeb zpracována (např. v PHP pomocí knihovny cURL). Pro používání tohoto API v systému NetPlag, bylo nejprve potřeba zahrnout tuto knihovnu do hlavičky stránky.

```
<skript src=http://www.google.com/jsapi type="text/javascript"></script>
```

Dále bylo potřeba nahrát modul (s požadovaným API) pomocí metody `load(moduleName, moduleVersion, optionalSettings)` objektu `google`. Poté bylo potřeba vytvořit funkci, která inicializuje hlavní objekt ze třídy `SearchControl`. Tento objekt byl zároveň zvolen jako globální proměnná, aby při každém požadavku o zahájení vyhledávání (kliknutí na vyhledávací položku v JZ) nemusel být tento objekt znovu inicializován (což vede k rychlejšímu zpracování).

Tato funkce je pojmenovaná `initSearchControl()` a v této funkci se nastavují veškeré parametry vyhledávání, vykreslování a také callback funkce zavolaná po dokončení vyhledávání.

```
// nahrani modulu s příslušným API
google.load('search', '1');

// globalni promenna
var searchControl;
function initSearchControl(){

  // pripoji vizitku "powered by Google"
    google.search.Search.getBranding(document.getElementById("branding"));

  // nastaveni "rozbalovatek" u jednotlivych vyhledavacich objektu
    var options = new google.search.SearcherOptions();
    options.setExpandMode(google.search.SearchControl.EXPAND_MODE_CLOSED);

    var optionOpened = new google.search.SearcherOptions();
    optionOpened.setExpandMode(google.search.SearchControl.EXPAND_MODE_OPEN);


  // vytvoreni hlavního objektu ze tridy „SearchControl"
    searchControl = new google.search.SearchControl(options);

  // nastaveni atributu „target" u vsech vracenych odkazu (ID iframu v SV)
    searchControl.setLinkTarget('frameSV');


  // pridani vsech „vyhledavacu", kteri se zapoji do vyhledavani
    var webSearch = new google.search.WebSearch();
    var localSearch = new google.search.LocalSearch();

    var extendedArgs = google.search.Search.RESTRICT_EXTENDED_ARGS;
    webSearch.setRestriction(extendedArgs, {gl: 'cs', lr:'lang_cs'});

    searchControl.addSearcher(webSearch, optionOpened);
    searchControl.addSearcher(localSearch, options);
    searchControl.addSearcher(new google.search.VideoSearch(), options);
    searchControl.addSearcher(new google.search.BlogSearch(), options);
    searchControl.addSearcher(new google.search.NewsSearch(), options);
    searchControl.addSearcher(new google.search.ImageSearch(), options);
    searchControl.addSearcher(new google.search.BookSearch(), options);
    searchControl.addSearcher(new google.search.PatentSearch(), options);


  // nastaveni vykreslovani (DRAW_MODE_LINEAR nebo DRAW_MODE_TABBED)
    var drawOptions = new google.search.DrawOptions();
    drawOptions.setDrawMode(google.search.SearchControl.DRAW_MODE_LINEAR);

  // nastaveni kontejneru do kterého budou vykreslovany vysledky hledani
    searchControl.draw(document.getElementById("hlavniJihVychod"),
                       drawOptions);

  // nastaveni callback funkce, která bude zavolana po dokonceni vyhledavani
    searchControl.setSearchCompleteCallback(this, searchComplete);
}

// zaregistrovani funkce ktera se spusti pouze jednou po nacteni stranky
google.setOnLoadCallback(initSearchControl);
```

Návratová funkce s názvem `searchComplete(searchControl, searcher)`, která je spuštěna po dokončení vyhledávání, plní jednoduchý účel – ke každému vrácenému odkazu přidává handler na funkci s názvem `resultClicked`, která je spuštěna po kliknutí na tento odkaz. Tato funkce již tak triviální není. Plní **dva základní úkoly**. Prostřednictvím jiné funkce nastavuje pole s tučnými elementy (nacházející se v textovém úryvku pod odkazem na který bylo kliknuto), které je později využito pro účely zvýrazňování ve stránce na kterou vede odkaz (na který bylo kliknuto). Druhým úkolem je nastavení atributu `src`, elementu `iframe`, na řadič a jeho metodu, která vrací HTML obsah stránky, na kterou vede odkaz (na který bylo kliknuto) v parametru této metody.

Tento pomocný řadič s názvem `simpleproxy` musel být zaveden, jelikož bezpečnostní standardy moderních prohlížečů **neumožňují přístup** k objektu `dokument` ve stránce, která pochází z jiné domény než ze které pochází „naše stránka“. Tato vlastnost se projeví např. právě v použití vnořených rámů. Pokud nemůžeme přistoupit k objektu `dokument`, ztrácíme tak možnost jakékoliv editace, včetně námi požadovaného zvýrazňování.

```
function searchComplete(searchControl, searcher){
   var resLinks = Ext.select("a.gs-title", true);

   for (var i=0; i < searcher.results.length; i++) {
       if (searcher.results[i].GsearchResultClass ==
            google.search.WebSearch.RESULT_CLASS) {
          resLinks.on('click', resultClicked);
          break;
       }
   }
}

var arrBoldTexts = null;
function resultClicked (e, t)
{
   // zruseni kliknuti (po zakomentovani bude odkaz posilat cil do iframu)
     e.preventDefault();

   // ziskani <b> elementu z google result
     var linkDOM = e.getTarget();
     var iframeDOM = Ext.get("frameSV").dom;
     arrBoldTexts = getBoldElems(linkDOM);

   // nastaveni atributu src u iframu na kohana controler
     var url = linkDOM.getAttribute("href");
     var url_b64 = Base64.encode(url);
     iframeDOM.src = "/simpleproxy/getweb/" + url_b64;
}
```

3.3.5 Popis kompilátoru

Jedná se o nejkomplexnější a nejrozsáhlejší nástroj v celém systému. Jeho účel je však překvapivě jednoduchý – převádět vstupní testované dokumenty do interního formátu NPL. Díky hlavním principům kompilátorů (zejména sekvenční zpracování), však tento úkol dokáže plnit mimořádně rychle a bez větších paměťových nároků. Jeho bezesporu další největší výhodou je **modularita** – systém NetPlag tak lze efektivně rozšiřovat o další vstupní formáty (DOC, PDF, atd.).

Kompilátor se skládá z jednoho řídícího kontroléru a pěti obslužných tříd nacházející se v adresáři. Základní význam všech těchto komponent objasňuje tabulka uvedená níže.

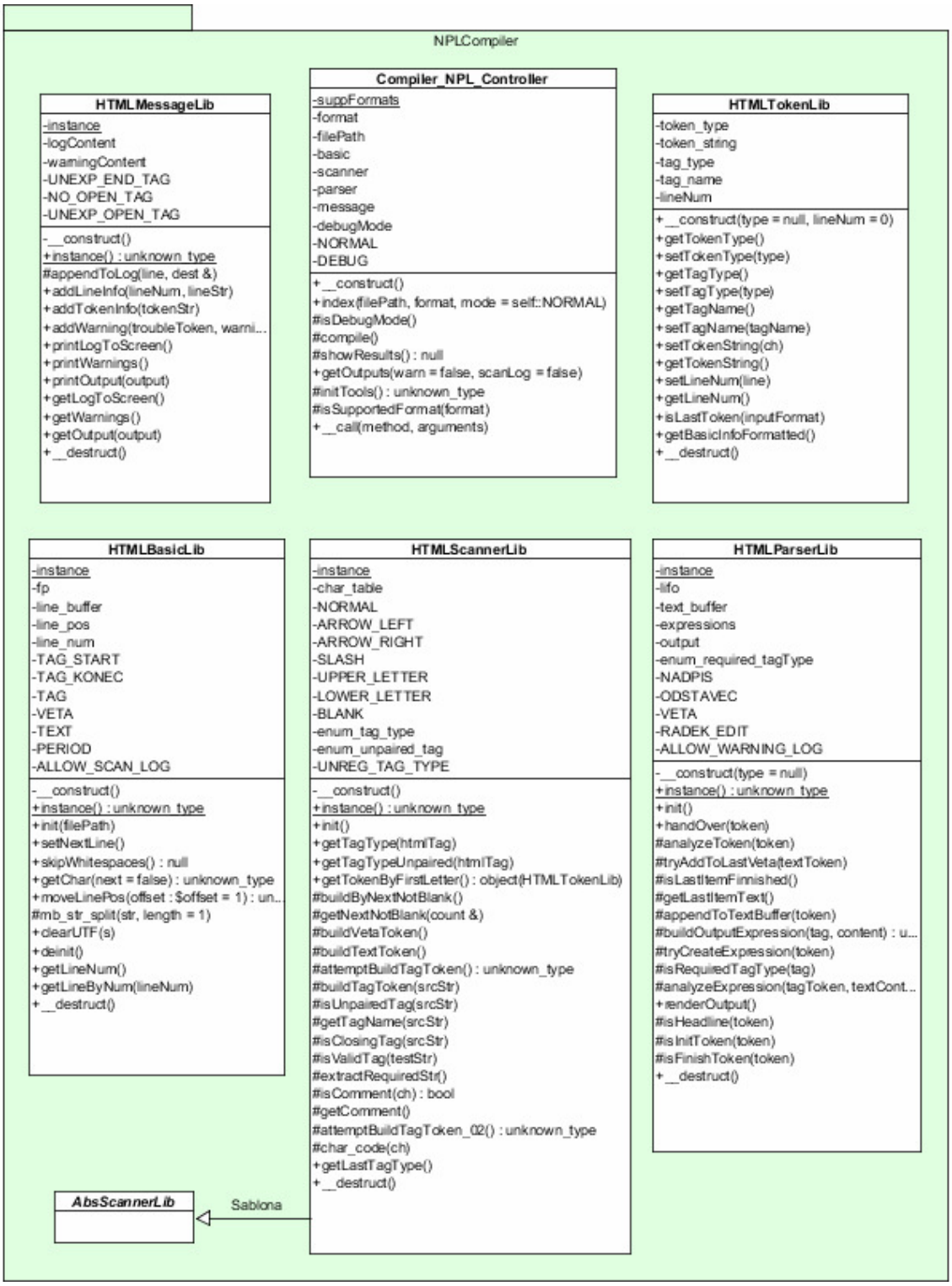
Tab. 3 – Základní komponenty kompilátoru

Compiler_NPL_Controller	• v *„/application/controllers/compiler_NPL.php“* • wrapper zajišťující součinnost pěti obslužných tříd • hlavní metoda `compile()` • umožňuje zobrazování warningů a logy scanneru • je uplatněn třídou „`Extjs_request`“ v metodě `compileFile()`
HTMLBasicLib	• v *„/application/libraries/HTMLBasicLib.php“* • obsahuje základní operace se vstupním souborem • sekvenčně čte (po řádcích) vstupní soubor • definuje výčet povolených typů tokenů • jedná se o návrhový vzor „singleton“
HTMLScannerLib	• v *„/application/libraries/HTMLScannerLib.php“* • čte vstupní soubor a sestavuje příslušné tokeny • hlavní metoda `getTokenByFirstLetter()` • dědí z abstraktní třídy `AbsScannerLib` • jako jediná je odvozena podle formátu vstupního dokumentu • jedná se o návrhový vzor „singleton“
HTMLParserLib	• v *„/application/libraries/HTMLParserLib.php“* • přebírá a analyzuje tokeny a následně sestavuje jednotlivé fragmenty/uzly NPL • hlavní metoda `handOver($token)` • definuje výčet možných elementů v NPL • je schopen ukládat warningy při výskytu syntaktických chyb ve vstupním testovaném souboru • jedná se o návrhový vzor „singleton“

HTMLMessageLib	• v *„/application/libraries/HTMLMessageLib.php“* • pomocná třída pro obsluhu výstupu • definuje přípustné syntaktické chyby, které mohou vzniknout během parsingu (v HTMLParserLib) • obsahuje vzniklé warningy a logy scanneru během kompilace (ve formátu HTML) • jedná se o návrhový vzor „singleton“
HTMLTokenLib	• v *„/application/libraries/HTMLTokenLib.php“* • vytvořený objekt zahrnuje veškeré informace o vzniklém tokenu • metodu `isLastToken()` využívá samotný `Compiler_NPL_Controller`

Pokud tedy v budoucnu vznikne snaha o přidání nového vstupního formátu, např. DOC, mělo by stačit pouze vytvořit novou třídu `DOCScannerLib`, která bude odvozena z abstraktní třídy `AbsScannerLib`, resp. bude definovat její atributy a implementovat příslušné metody. Autor však zároveň doporučuje podrobněji si prostudovat princip a fungování již existující třídy `HTMLScannerLib` a její vazby na ostatní komponenty v kompilátoru.

Činnost kompilátoru je poměrně komplikovanou záležitostí a výskyt chyby v systému NetPlag může způsobovat právě tento kompilátor, který je do systému NetPlag vestavěn. Proto bylo vytvořeno **ladící rozhraní** pro vývojáře, dostupné na adrese „*http://www.netplag.cz/Compiler_npl/index/[název souboru]/[přípona]/debug*“, které umožňuje testovat tento vestavěný nástroj s různými vstupními soubory, osamostatněný od dalších činností systému NetPlag. Toto rozhraní má zapnutý výpis všech vzniklých chyb, varování a činnosti scanneru během procesu kompilace. Je důležité poznamenat, že vstupní kompilované soubory se musí nacházet v adresáři „*./data/testovaci_data/*“. Kompletní strukturu kompilátoru, resp. seznam všech atributů a metod jeho jednotlivých komponent, přehledně popisuje jeho diagram tříd, který je zobrazen níže.



Obr. 24 – Diagram NPL kompilátoru

3.3.6 Konfigurátor metod

Tento nástroj je v podstatě zjednodušeným API pro **vytváření nových metod**, které analyzují vstupní dokument převedený do formátu NPL a vracejí vyhledávací položky, které se vkládají do gridu na JZ. Toto API se v současné době skládá ze dvou knihoven, dostupných v „*/application/libraries/*“, a to „*NPLanalyzerLib.php*“ a „*QueryBuilderLib.php*“.

Hlavní princip spočívá v definici privátní statické proměnné s názvem `$methodsConfig` v třídě `NPLanalyzerLib`. Jedna se o vícerozměrné pole, které na první úrovní obsahuje identifikátory metod (ty musí být deklarovány jako konstanty v téže třídě), které jsou hodnotami elementu `option` ve formulářovém prvku `select` (pomocí tohoto prvku uživatel vybírá požadované metody) a ve druhé úrovni konfiguraci těchto metod. Význam jednotlivých konfiguračních položek je dokumentován v tabulce níže.

Tab. 4 – Význam konfiguračních položek

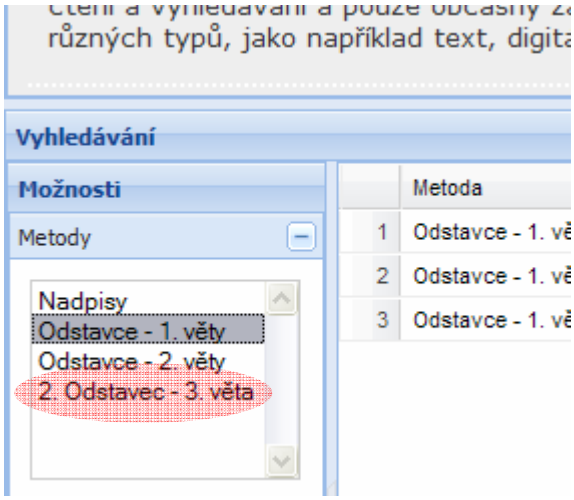
`methodName`	• popisek metody (může obsahovat diakritiku, mezery, apod.) • používá se jako popisek u elementu `option`
`itemName`	• určuje kterých elementů s tímto „tagName“, ze vstupního NPL dokumentu, se metoda týká • tyto elementy musejí být přímým potomkem kořene NPL dokumentu • tyto elementy budou z NPL dokumentu extrahovány pro další účely metody • zatím lze zadat pouze jednu hodnotu (nelze zadat pole) • možné hodnoty: "nadpis", "odstavec"
`includeItem`	• určuje kolikáté prvky (v pořadí) s danou hodnotou `itemName` v NPL dokumentu budou zahrnuty pro další účely zpracování • možné hodnoty: pole s libovolným počtem prvků, které jsou celá čísla od 1 – ∞ • prázdné pole znamená zahrnutí všem prvků s daným `itemName`
`includeSentence`	• určuje kolikáté elementy (v pořadí) s „tagName“ `veta` uvnitř elementu s danou hodnotou `itemName` v NPL dokumentu budou zahrnuty pro další účely zpracování • možné hodnoty: pole s libovolným počtem prvků, které jsou celá čísla od 1 – ∞ • prázdné pole znamená zahrnutí všem elementů s „tagName“ `veta` uvnitř elementu s danou hodnotou `itemName`

queryBuilder	className	• jméno třídy která přebírá další zpracování • v současné době lze využít existující knihovnu "QueryBuilderLib"
	funcName	• jméno metody patřící třídě className která přebírá další zpracování • v současné době lze využít existující metodu "simpleSentence"
	params	• specifikující parametry pro metodu funcName • Pozn.: pole vět se vkládá automaticky jako první parametr

Pokud bychom tedy chtěli přidat novou metodu, která ze vstupního NPL dokumentu vybere pouze druhý element s „tagName“ odstavec, z něho následně vybere pouze druhou, třetí a čtvrtou větu a jako finální dotaz pro vyhledávač použije právě zmiňovanou třetí větu, vypadala by konfigurace následovně.

```
// …
const ODSTAVEC_2ST_VETA_3ST = "odstavec_2st_veta_3st";
// …

private static $methodsConfig = array(

	// …předchozí metody

	self::ODSTAVEC_2ST_VETA_3ST => array(
		"methodName" => "2. Odstavec - 3. věta",
		"itemName"          => "odstavec",
		"includeItem"             => array(2),
		"includeSentence"   => array(2, 3, 4),
		"queryBuilder" => array(  "className"  => "QueryBuilderLib",
		                          "funcName"   => "simpleSentence",
		                          // 2. prvek z pole includeSentence
		                          "params"     => array(2))
	)
);
```

Pozn.: Selektivita v konfigurátoru je striktního charakteru. Pro tuto novou metodu to tedy znamená, že pokud např. v daném NPL dokumentu neexistuje 2. odstavec, nebo neexistuje jedna z vět definovaná v položce „includeSentence“, metoda nevrátí žádné vyhledávací položky. Tak by se tedy mohlo např. stát, pokud by existoval 2. odstavec s pouze třemi větami (neexistovala by 4. věta). Takto definovaná metoda se zároveň automaticky přidá do selectboxu k dalším registrovaným metodám a uživatel má možnost ji ihned použít.



Obr. 25 – Automatické zaregistrování nové metody

3.3.7 XMLBuilder

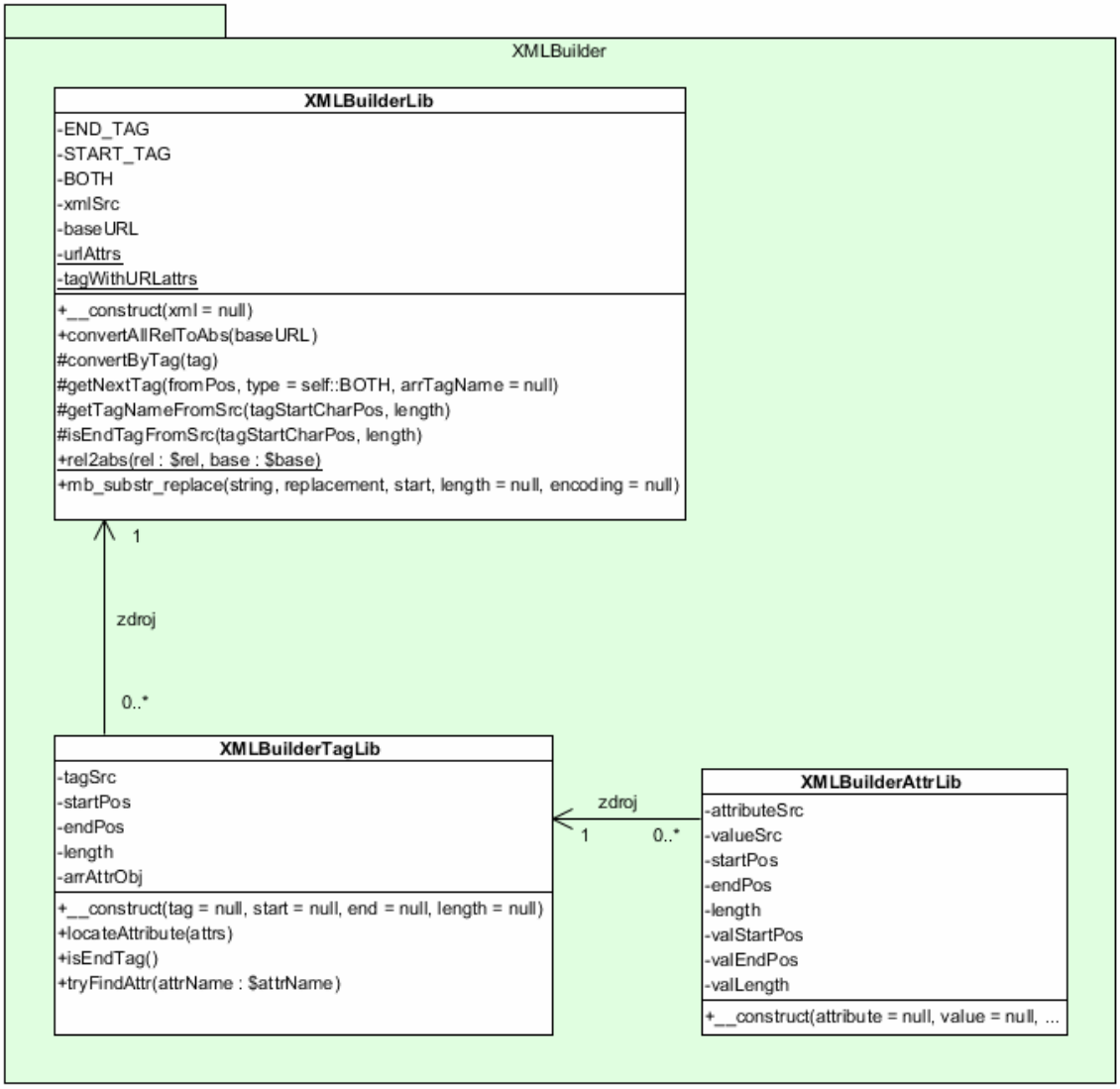
Jedná se o nástroj, který v podstatě supluje činnost již existující PHP knihovny s názvem *XMLReader*. Principielně se tedy jedná o tzv. „pull parser“, který po částech čte vstupní XML dokument (popř. HTML), předává jednotlivé uzly a ty následně mohou, avšak nemusí být zpracovány. To je zásadní koncepční rozdíl oproti „push parseru“ (*XML Parser*), který nám posílá proud událostí (uzlů), které my jsme povinni obsloužit. Tento přístup je však poměrně náročný na implementaci a pro většinu případů si programátor bohatě vystačí s knihovnou XMLReader.

Na druhou stranu existuje ještě rozhraní DOM (*Document Object Model*), které však není založeno na principu sekvenčního čtení souboru, ale vytváří v operační paměti jakousi **stromovou reprezentaci dokumentu**, přičemž uzly tohoto stromu jsou objekty. Tyto objekty pak reprezentují elementy, jejich atributy nebo textové uzly. Pomocí metod a vlastností těchto objektů můžeme pak u daného uzlu v dokumentu zjišťovat jeho typ, název, přímé i nepřímé potomky, rodiče, hodnoty atributů a mnoho dalšího. Tento přístup je tedy poměrně pohodlný, avšak paměťově náročný, jelikož pro každý uzel v dokumentu musí v paměti existovat jeho zástupný objekt. Konkrétní implementaci tohoto rozhraní najdeme např. v JavaScriptu, ve kterém můžeme např. pomocí jediné metody `getElementsByTagName` získat všechny odkazy na stránce.

Náš nástroj byl vytvořen za účelem **převodu všech relativních URL adres na absolutní**. Pokud totiž chceme v nalezené stránce, načtené v kontejneru `iframe` s atributem `src` vedoucí na náš kontrolér `simpleproxy` (vracející obsah této stránky), zobrazovat její CSS, zprovoznit její odkazy a ostatní prvky používající URL, je potřeba právě tyto URL adresy nastavit na absolutní.

Připomeňme, že celý proces s načítáním obsahu nalezené stránky a převodu všech relativních URL adres, se realizuje kvůli bezpečnostním standardům internetových prohlížečů, které neumožňují přístup k objektu `document` ve stránce, která pochází z jiné domény než ze které pochází „naše stránka“ (např. při použití rámů). Existující knihovny XMLReader nebo DOM jsme nepoužili z toho důvodu, jelikož tyto knihovny striktně vyžadují validitu zpracovávaného XML dokumentu, kterou mi u nalezeného dokumentu nejsme schopni zajistit. Při pokusu o zpracování HTML dokumentu s chybami vnitřní parser těchto knihoven oznámí „fatal error“ a ukončí svůj běh.

Knihovnu XMLBuilder v současnosti tvoří tří hlavní třídy, tj. `XMLBuilderLib`, `XMLBuilderTagLib` a `XMLBuilderAttrLib`, které se nachází ve stejnojmenných souborech v adresáři „*./application/libraries/*“ a jejich bližší popis je k dispozici v následujícím diagramu tříd.



Obr. 26 – Diagram tříd knihovny XMLBuilder

3.4 Budoucí rozšíření

Před samotným závěrem této práce je zde ještě seznam nových vlastností a funkcionalit systému NetPlag, které mají potencionál zvětšit kompatibilitu systému, zrychlit zpracování, zdokonalit ovládání a co nejvíce zautomatizovat celý proces odhalování plagiátorství. Tento seznam je určen především pro vývojáře webových aplikací a je doporučeno, aby budoucí programátor si kompletně prostudoval celou tuto dokumentaci a v případě větších problémů kontaktoval autora systémů NetPlag.

- ✓ *přidání nových metod analýzy obsahu testovaného dokumentu*
- ✓ *přidání podpory pro další známé formáty testovaného dokumentu*
- ✓ *načítání testovaných dokumentů z adresáře, popř. propojení s FTP (položka v menu „Načíst z adresáře“)*
- ✓ *ukládání výstupu z XSLT procesoru na disk (převod NPL na HTML), aby příště nemusela být znovu spuštěna transformace*
- ✓ *možnost přidání poznámky k zvýrazněnému elementu na SZ a následný export NPL dokumentu s poznámkou*
- ✓ *přidání možnosti se zvýrazněním dalších shod (resp. souslovi) mezi testovaným a nalezeným dokumentem*
- ✓ *uživatel bude vizuálně upozorňován na významné shody u vyhledávacích položek (JZ) a nalezených záznamů (JV) – např. zvýrazněním daných řádků v gridu nebo záložek*
- ✓ *přidání „status-baru“ zobrazující informace k danému porovnávání*
- ✓ *propojení vyhledávání s technologií YQL*
- ✓ *zrychlení zpracování knihovny XMLBuilder*

Závěr

Hlavním cílem této práce bylo objasnit problematiku plagiátorství, tj. proč vůbec tento jev vzniká a jak se mu vyvarovat. Dále bylo potřeba čtenáře seznámit s již existujícími SW produkty pro podporu odhalování plagiátorství a jejich principy a také s několika známými metodami, používané těmito produkty, které analyzují obsah dokumentu a konstruují porovnávací položky.

V praktické části této práce bylo cílem vytvořit podpůrnou aplikaci, která bude schopna odhalovat plagiáty prostřednictvím celosvětové sítě Internet a vyhledávací služby Google. Zároveň bylo otázkou, zdali takovéto odhalování je vůbec možné (oproti vyhledávání vůči lokální databázi). Aplikace měla podporovat několik vstupních formátů testovaného dokumentu a také být schopna přehledně zobrazovat podezřelá místa, resp. nalezené shody.

S trochou nadsázky se dá říci, že všechny výše uvedené cíle byli maximálně splněny. Je vytvořen nový systém s názvem **NetPlag**, spadající do kategorie RIA, který je schopen efektivně a účinně odhalovat plagiáty prostřednictvím sítě Internet. V současné době je tento systém kompatibilní se dvěma vstupními formáty (HTML a NPL) a obsahuje pár základních metod porovnávání, avšak díky modulárnímu konceptu, není velkou překážkou, přidávat další používané formáty souborů a nové porovnávací metody. Systém má jednoduché intuitivní ovládání a také velmi flexibilní uživatelské rozhraní. Samozřejmě ve světě informačních technologií není nic dokonalé a tak i tento systém má potenciál stát se daleko lepší a celý proces odhalování plagiátorství maximálně zautomatizovat.

Obr. 27 – Logo nového systému pro odhalování plagiátorství

Seznam pojmů a zkratek

- ***Open-source***
  Počítačový software s otevřeným zdrojovým kódem. Otevřenost zde znamená jak technickou dostupnost kódu, tak legální dostupnost – licence software, která umožňuje, při dodržení jistých podmínek, uživatelům zdrojový kód využívat, například prohlížet a upravovat [21].
- ***SQL***
  Structured Query Language. Standardizovaný dotazovací jazyk používaný pro práci s daty v relačních databázích [21].
- ***Apache***
  Webový server s otevřeným kódem pro Linux, BSD, Microsoft Windows a další platformy. V současné době dodává prohlížečům na celém světě většinu internetových stránek [21].
- ***MySQL***
  Databázový systém, vytvořený švédskou firmou MySQL AB. Pro svou snadnou implementovatelnost (lze jej instalovat na Linux, MS Windows, ale i další operační systémy), výkon a především díky tomu, že se jedná o volně šiřitelný software, má vysoký podíl na v současné době používaných databázích [21].
- ***PHP***
  PHP: Hypertext Preprocesor. Skriptovací programovací jazyk určený především pro programování dynamických internetových stránek. PHP skripty jsou prováděny na straně serveru, k uživateli je přenášen až výsledek jejich činnosti. V kombinaci s databázovým serverem (především s MySQL nebo PostgreSQL) a webovým serverem Apache je často využíván k tvorbě webových aplikací [21].
- ***JavaScript***
  Multiplatformní, objektově orientovaný skriptovací jazyk. Program v JavaScriptu se obvykle spouští až po stažení WWW stránky z Internetu (tzv. na straně klienta) [21].
- ***AJAX***
  Asynchronous JavaScript and XML. Obecné označení pro technologie vývoje interaktivních webových aplikací, které mění obsah svých stránek bez nutnosti jejich znovunačítání [21].
- ***XHTML***
  eXtensible HyperText Markup Language. Značkovací jazyk pro hypertext. Je jedním z jazyků pro vytváření stránek v systému World Wide Web, který umožňuje publikaci dokumentů na Internetu [21].
- ***CSS***
  Cascading Style Sheets. Jazyk pro popis způsobu zobrazení stránek napsaných v jazycích HTML, XHTML nebo XML. Jazyk byl navržen standardizační organizací W3C [21].
- ***XML***
  Extensible Markup Language. Obecný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. Umožňuje snadné vytváření konkrétních značkovacích jazyků (tzv. aplikací) pro různé účely a různé typy dat [21].
- ***JSON***
  JavaScript Object Notation. Způsob zápisu dat (datový formát) nezávislý na počítačové platformě, určený pro přenos dat, která mohou být organizována v polích nebo agregována objektech [21].
- ***PDF***
  Portable Document Format. Souborový formát vyvinutý firmou Adobe pro ukládání dokumentů nezávisle na softwaru i hardwaru, na kterém byly pořízeny. Může obsahovat text i obrázky [21].
- ***TeX***
  Sázecí systém, který naprogramoval profesor Donald Knuth [21].
- ***URL***
  Uniform Resource Locator. Řetězec znaků s definovanou strukturou, který slouží k přesné specifikaci umístění zdrojů informací (ve smyslu dokument nebo služba) na Internetu [21].

- ***HTTP***
  Hypertext Transfer Protocol. Internetový protokol určený původně pro výměnu hypertextových dokumentů ve formátu HTML [21].
- ***FTP***
  File Transfer Protocol. Protokol aplikační vrstvy z rodiny TCP/IP. Je určen pro přenos souborů mezi počítači, na kterých mohou běžet rozdílné operační systémy (je platformně nezávislý) [21].
- ***Framework***
  Softwarová struktura, která slouží jako podpora při programování a vývoji a organizaci jiných softwarových projektů.
- ***MVC***
  Model-view-controller. Softwarová architektura, která rozděluje datový model aplikace, uživatelské rozhraní a řídicí logiku do tří nezávislých komponent tak, že modifikace některé z nich má minimální vliv na ostatní [21].
- ***API***
  Application Programming Interface. Rozhraní pro programování aplikací. Tento termín používá softwarové inženýrství. Jde o sbírku procedur, funkcí či tříd nějaké knihovny (ale třeba i jiného programu nebo jádra operačního systému), které může programátor využívat [21].
- ***GUI***
  Graphical User Interface. Uživatelské rozhraní, které umožňuje ovládat počítač pomocí interaktivních grafických ovládacích prvků. Na monitoru počítače jsou zobrazena okna, ve kterých programy zobrazují svůj výstup [21].
- ***UML***
  Unified Modeling Language. Grafický jazyk pro vizualizaci, specifikaci, navrhování a dokumentaci programových systémů [21].
- ***DRY***
  Don't Repeat Yourself. Princip vývoje softwarů zaměřující se na redukci opakování různorodých informací zejména v oblasti vícevrstvých architektur [21].
- ***ORM***
  Object-relational mapping. Programovací technika která umožňuje konverzi dat mezi různými nekompatibilními systémy, které zajišťují typovou kontrolu. Vytváří jakousi virtuální databázi, která může být použita v daném programovacím jazyce [21].
- ***Layout***
  Rozvržení, uspořádání, struktura www stránky.
- ***Upload***
  Přenesení dat do jiných systémů, nahrání dat na server
- ***Autentizace***
  Proces ověření identity [21].
- ***Autorizace***
  Je postup, který omezuje přístup k informacím, funkcím a dalším objektům. Přístup mají pouze oprávněné subjekty. Proces autorizace subjektu vyžaduje autentizaci subjektu a vyhledání jeho role a práv [21].

Seznam obrázků

Seznam tabulek

Použitá literatura

[1] Programmer to programmer. PHP5, MySQL, Apache : Vytváříme webové aplikace. Martin Domes; Bogdan Kiszka. Brno : CP Books, a.s, 2006. 816 s. ISBN 80-251-1073-7.

[2] Chýla, Roman. Detekce plagiátorství. Ikaros [online]. 2009, roč. 13, č. 3 [cit. 03.08.2010]. Dostupný na World Wide Web: http://www.ikaros.cz/node/5308. URN-NBN:cz-ik5308. ISSN 1212-5075.

[3] Chýla, Roman. Detekce plagiátorství. Ikaros [online]. 2009, roč. 13, č. 2 [cit. 03.08.2010]. Dostupný na World Wide Web: http://www.ikaros.cz/node/5253. URN-NBN:cz-ik5253. ISSN 1212-5075.

[4] *Strategie v boji proti plagiátorství* [online]. 2010 [cit. 2010-08-03]. Informační studie na téma Strategie v boji proti plagiátorství na akademické půdě. Dostupné z WWW: http://st.vse.cz/~XKOCP13/index.htm.

[5] Zadražilová Iva. Problém plagiátorství na vysokých školách. Inflow: information journal [online]. 2008, roč. 1, č. 5 [cit. 2010-08-03]. Dostupný z WWW: http://www.inflow.cz/problem-plagiatorstvi-na-vysokych-skolach. ISSN 1802-9736.

[6] Infogram. *Plagiátorství* [online]. 2009 [cit. 2010-08-03]. Úvod do problematiky plagiátorství. Dostupné z WWW: http://www.infogram.cz/findInSection.do?sectionId=1115&categoryId=1161.

[7] IS MUNI. *Do boje proti plagiátorství se zapojí polovina veřejných vysokých škol v ČR : Tisková zpráva* [online]. 21.8.2007 [cit. 2010-08-03]. Do boje proti plagiátorství se zapojí polovina veřejných vysokých škol v ČR. Dostupné z WWW: http://is.muni.cz/clanky/2007_plagiaty.pl?lang=en/.

[8] Brandejs, Michal; Brandejsová, Jitka; Krhutová, Růžena; Mikulášová, Zuzana; Pekárková, Lucie; Stančík, Martin. Kontrola plagiátů v seminárních pracích prostřednictvím Odevzdej.cz. Ikaros [online]. 2009, roč. 13, č. 8 [cit. 03.08.2010]. Dostupný na World Wide Web: http://www.ikaros.cz/node/5641. URN-NBN:cz-ik5641. ISSN 1212-5075.

[9] BRANDEJSOVÁ, Jitka, et al. Národní registr VŠKP a systém na odhalování plagiátů. *Čtenář : měsíčník pro knihovny* [online]. 01/2009, roč. 61, č. 1, [cit. 2010-08-03]. Dostupný z WWW: http://ctenar.svkkl.cz/clanky/2009-roc-61/01-2009/tema-narodni-registr-vskp-a-system-na-odhalovani-plagiatu-51-311.htm.

[10] BRANDEJSOVÁ, Jitka. *Odevzdej.cz: Odhalování plagiátů v seminárních pracích* [online]. [s.l.], 2009. 4 s. Seminární práce. Masarykova univerzita. Dostupné z WWW: http://www.evskp.cz/Seminar4/seminar4-Brandejsova-text.pdf.

[11] Zadražilová Iva. Problém plagiátorství na vysokých školách. Inflow: information journal [online]. 2008, roč. 1, č. 5 [cit. 2010-08-05]. Dostupný z WWW: http://www.inflow.cz/problem-plagiatorstvi-na-vysokych-skolach. ISSN 1802-9736.

[12] *Projekt Odhalování plagiátů v závěrečných pracích* [online]. 2009 [cit. 2010-08-05]. Jak učitel vyhledává plagiáty?. Dostupné z WWW: http://theses.cz/th_dok/plagiaty_demo.pl.

[13] Plagiát. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 14. 10. 2004, last modified on 16. 5. 2010 [cit. 2010-08-05]. Dostupné z WWW: http://cs.wikipedia.org/wiki/Plagiát.

[14] MÜNICH, Daniel Plagiátorství je někdy cestou jak přežít. In *Blog.aktualne.cz* [online]. [s.l.] : [s.n.], 22.08.2007 [cit. 2010-08-05]. Dostupné z WWW: http://blog.aktualne.centrum.cz/blogy/daniel-munich.php?itemid=1455.

[15] DUBSKÝ, Kamil; PLOCKOVÁ, Zuzana. S plagiátorstvím bojují i východočeské univerzity. *Pardubický deník : Moje Pardubicko* [online]. 21.10.2009, 1, [cit. 2010-08-10]. Dostupný z WWW: http://pardubicky.denik.cz/zpravy_region/s-plagiatorstvim-bojuji-i-vychodoceske-f363.html.

[16] *Assessing Student Learning* [online]. 2008 [cit. 2010-08-05]. Overview of plagiarism detection software. Dostupné z WWW: http://www.cshe.unimelb.edu.au/assessinglearning/03/plagsoftsumm1.html.

[17] *Shambles* [online]. 2010 [cit. 2010-08-05]. Plagiarism Tools. Dostupné z WWW: http://www.shambles.net/pages/staff/ptools/.

[18] The PHP Group. PHP [online]. c2001-2008, [cit. 2010-08-05]. Text v angličtině. Dostupný z WWW: http://php.net/.

[19] JANOVSKÝ, Dušan. Jak psát web : O tvorbě internetových stránek [online]. [2003-2008], [cit. 2010-08-05]. Text je v češtině. Dostupný z WWW: http://www.jakpsatweb.cz/. ISSN 1801-0458

[20] Autorský zákon. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 5. 7. 2006, last modified on 5. 8. 2010 [cit. 2010-08-10]. Dostupné z WWW: http://cs.wikipedia.org/wiki/Autorský_zákon.

[21] Wikimedia Foundation Inc. Wikipedie : Otevřená encyklopedie [online]. 2001-2008 [cit. 2010-08-05]. Dostupný z WWW: http://cs.wikipedia.org