VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

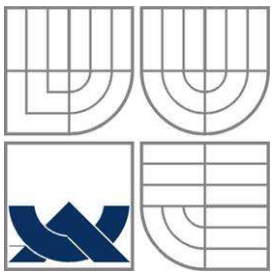
ZÍSKÁVÁNÍ DAT Z HTML STRÁNEK

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

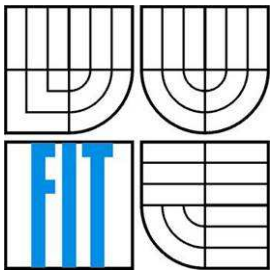
AUTOR PRÁCE
AUTHOR

TOMÁŠ KOMENDA

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ZÍSKÁVÁNÍ DAT Z HTML STRÁNEK

DATA ACQUISITION FROM HTML SITES

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ KOMENDA

VEDOUCÍ PRÁCE
SUPERVISOR

ING. MARTIN STRAKA

BRNO 2007

Zadání bakalářské práce/5623/2006/xkomen03

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Komenda Tomáš**
Obor: Informační technologie
Téma: **Získávání dat z HTML stránek**
Kategorie: Web

Pokyny:

1. Seznamte se s principy tvorby dynamických www stránek a jazykem HTML, PHP nebo Javascript případně databází MySQL.
2. Prostudujte způsoby extrakce informací z WWW stránek.
3. Promyslete implementaci skriptu či algoritmu pro získávání a filtraci dat ze zadaných webových stránek.
4. Takto získané informace tranformujte do vlastní databáze či souboru a generujte vlastní grafický výstup(webová stránka).
5. Vybrané části získaných dat se budou také pomocí skriptu automaticky posílat přes SMS bránu skupině příjemců.
6. Algoritmus implementujte ve vhodně zvoleném jazyku.
7. Demonstrujte činnost a zhodnoťte dosažené výsledky.

Literatura:

- Welling, L., Thomsonová, L.: PHP a MySQL - rozvoj webových aplikací, Softpress 2003. 910 s. ISBN 80-86497-60-7.
- Kosek, J.: HTML, tvorba dokonalých www stránek, Grada 1998. 296 s. ISBN 80-7169-608-0.

Při obhajobě semestrální části projektu je požadováno:

- splnění prvních 4 bodů zadání

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Straka Martin, Ing.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. **Pan**

Jméno a příjmení: **Tomáš Komenda**
Id studenta: 85445
Bytem: Lubnice 44, 671 07 Uherčice u Znojma
Narozen: 01. 02. 1985, Dačice
(dále jen "autor")

a

2. **Vysoké učení technické v Brně**

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Získávání dat z HTML stránek
Vedoucí/školitel VŠKP: Straka Martin, Ing.
Ústav: Ústav počítačových systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Práce popisuje možnosti tvorby aplikací pro získávání a pozdější sledování textových dat z HTML stránek. Zabývá se lexikální a syntaktickou analýzou jazyka HTML. Charakterizuje možnosti filtrace a uchovávání dat. Dále se práce zaměřuje na nástroje pro tvorbu dynamických WWW stránek, způsoby extrakce dat a periodické spouštění aplikací. V neposlední řadě popisuje vyspělé možnosti rozesílání elektronické pošty a SMS zpráv z prostředí webové aplikace. Závěrečná část je věnována popisu dvou aplikací demonstrujících uváděné principy.

Klíčová slova

HTML, PHP, JavaScript, MySQL, wget, fopen, cron, lexikální analýza, syntaktická analýza, HTML filtr, Mail PEAR, Mail_mime PEAR, SMS

Abstract

This work describes scope of creating application for extraction and following data from HTML sites. This work engages in lexical analyze and parsing HTML. This work describes filtration of data from HTML sites. This work describes saving of data in database and XML documents, creating of dynamic HTMP pages, timer cron, GNU utility wget, sending of SMS, sending email and extraction of date from internet. Final chapters describe two web applications. These applications follow data from HTML sites and inform users about changes.

Keywords

HTML, PHP, JavaScript, MySQL, wget, fopen, cron, lexical analyze, scanning, parsing, HTML filter, Mail PEAR, Mail_mime PEAR, SMS.

Citace

Tomáš Komenda: Získávání dat z HTML stránek, bakalářská práce, Brno, FIT VUT v Brně, 2007

Získávání dat z HTML stránek

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Martina Straky. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jméno Příjmení
Datum

Poděkování

V této části bych chtěl poděkovat všem lektorům VUT FIT, se kterými jsem během svého studia přišel do kontaktu, za jejich ochotu a obětavost učit studenty a pomáhat jim tak, aby dosáhli svých cílů. Obzvláště chci poděkovat svému vedoucímu Ing. Martinu Strakovi za konzultace a rady, které vždy ochotně poskytl.

Obsah

Úvod

Tato práce je zaměřena na získávání dat z HTML stránek. Toto téma je v dnešní době, kdy internet nabývání na významu, je stále více aktuální. Práce je zaměřena převážně na možnosti tvorby a samotný popis aplikací, které hlídají příslušné HTML stránky v síti internetu a sledují uživatelem zvolené části těchto dokumentů. Tyto aplikace jsou pak spouštěny v pravidelných intervalech a krom skenování vybraných stránek, porovnávají a ukládají sledovaná data. O případných změnách pak informují uživatele prostřednictvím elektronické pošty nebo zasílání krátkých textových zpráv na mobilní telefon. Práce je zaměřena na webové aplikace, tudíž je většina popisovaných jevů vsazena do této problematiky. Tedy většina kapitol ukazuje, jak je možno danou problematiku řešit jazykem PHP. Od čtenáře se tedy očekává základní znalost tohoto jazyka a také objektově orientovaného programování. V práci však najdeme i části, které problematiku charakterizují obecně a nastiňují její řešení v různých programovacích či skriptovacích jazycích.

První kapitola se zaměřuje na jazyky a nástroje pro tvorbu dynamických webových stránek. Protože se předpokládá čtenářova zběhlost v této problematice, není v kapitole kladen důraz na popis syntaxe ani sémantiky těchto jazyků. Kapitola je spíše zaměřena na charakteristiku, historický vývoj a budoucnost těchto nástrojů.

Následující kapitoly se věnují způsobům extrakce dat z HTML stránek a možnostem periodického spouštění aplikací. Na tuto část pak nepřímo navazují kapitoly věnované lexikální a syntaktické analýze jazyka HTML. Tyto kapitoly jsou pojaty spíše obecně, ale najdeme zde i popis nástrojů jazyka PHP, které jsem při tvorbě aplikací využil.

Poměrně rozsáhlou kapitolu tvoří pojednání o způsobech filtrace dat. Tyto kapitoly jsou také zaměřeny spíše teoreticky, ale odrážejí skutečné principy testované a používané v mých aplikacích. S těmito částmi práce bezprostředně souvisí uchovávání filtrovaných dat pro pozdější použití. Tímto tématem se zabývá další kapitola. V aplikacích jsem použil databázi MySql, tu však charakterizuji už v první kapitole, proto se zaměřuji hlavně na alternativní řešení ukládání dat pomocí jazyka XML a jeho nástrojů dostupných v jazyku PHP.

Protože vytvořené aplikace musí rozesílat výsledky sledování uživatelům, je jedna z kapitol zaměřena na problematiku pokročilých možností rozesílání elektronické pošty a krátkých textových zpráv SMS.

Poslední část práce popisuje dvě aplikace pro sledování dat na HTML stránkách, které jsem vytvořil v rámci této bakalářské práce. V těchto kapitolách nastiňuji implementaci, využití, testování a možnosti dalšího rozšíření. V závěru pak zhodnocuji dosažené výsledky a zabývám možnostmi dalšího vývoje.

1 Jazyky a nástroje pro tvorbu dynamických www stránek

V současné době je trendem uchovávat a zveřejňovat informace na internetu prostřednictvím internetových stránek. Existuje mnoho jazyků a nástrojů pro tvorbu těchto WWW stránek. Základním stavebním kamenem je jazyk HTML a dnes také stále modernější XHTML. Tyto jazyky by však pro tvorbu dynamických WWW stránek nestačily, proto se používají i jiné prostředky a jazyky rozvíjející dynamičnost stránek. Tyto nástroje a jazyky můžeme rozdělit do dvou základních skupin. První skupinou jsou klientské, které pracují na straně klienta (interpret prohlížeče). Většinou se jedná o událostmi řízený běh programu. Druhou skupinou jsou jazyky, jejichž kód je vykonáván na straně serveru, jako je PHP, ASP a některé druhy JavaScriptu. Na straně serveru se velmi často nutné uchovávat data a proto je nutné seznámit se i s různými druhy databází, jako je například stále oblíbenější MySql. Následující podkapitoly stručně charakterizují tyto nástroje.

1.1 HTML

Kapitola o HTML vychází z [1], [2] a [3].

HyperText Markup Language (HTML) je značkovací jazyk pro hypertext. Je jeden z mnoha jazyků pro tvorbu World Wide Web (WWW) stránek. Jedná se o programovací jazyk, který pomocí tágu popisuje formátování textu. Dokumenty v jazyku HTML mají přesně přepsanou strukturu. Od verze HTML 4.01 je povinná deklarace DTD. Dále musí obsahovat kořenový element, hlavičku elementu, jež je metadaty pro celý dokument a tělo dokumentu. Jazyk HTML je jazykem značkovacím. Z hlediska významu značek je můžeme rozdělit do tří skupin. První skupinou jsou strukturální značky, které dodávají dokumentu formu. Tyto značky rozvrhují strukturu dokumentu. Příkladem jsou odstavce (`<p>`) nebo nadpisy (`<h1>`, `<h2>`). Druhou skupinu jsou sémantické značky popisující povahu obsahu elementu. Příkladem je titulek (`<title>`) nebo adresa (`<address>`). Dnešní trend je orientován právě na sémantické značky, které usnadňují automatizované zpracovávání dokumentů a vyhledávání informací. Tato snaha dospěla až k jazyku XML. Třetí skupinou značek jsou stylistické značky určující vzhled elementu při zobrazení. Příkladem je tučné písmo (`<p>`) nebo font (`<font>`). Tyto značky však nejsou v současnosti doporučovány. Trendem je používání kaskádových stylů, které jsou odděleny od obsahu dokumentu. Dále se pokusím stručně nastínit historii jazyka.

V roce 1989 Tim Berners-Lee vytvořil HTML (které bylo přísně založeno na SGML) jako publikační jazyk pro síť. V CERNu, Evropské laboratoři pro fyziku částic ve švýcarské Ženevě, předložil Tim Berners-Lee svůj návrh původního HTML. Dosud byly poznatky z výzkumů dostupné pouze prostřednictvím souborů, které bylo možno stahovat do počítače. Návrh Tima Berners-Lee

umožňoval vědcům ze vzdálených míst zeměkoule organizovat a uspořádávat informace. Navrhnul, že můžeme zařídit, aby texty v souborech na sebe přímo odkazovaly. Už v roce 1980 rozvinul Tim Berners-Lee svůj první hypertextového systému "Enquire", který sloužil pro jeho osobní potřebu. Prototyp webového prohlížeče pro počítači NeXT byl zveřejněn v roce 1990. Nový jazyk potřeboval také protokol pro přenos dat po sítí. V roce 1990 byl vyvinut HyperText Transfer Protocol (HTTP).

V roce 1991 Dave Ragget z laboratoří Hewlett-Packard vytváří HTML+. Národní centrum pro superpočítačové aplikace (NCSA) ústavu na univerzitě Illionis at Champaign-Urbana vytvořilo svůj vlastní prohlížeč, který nazvali Mosaic. V březnu 1993 Lou Montulli uvolňuje prohlížeč Lynx verze 2.0a. V květenu 1994 se v Ženevě uskutečnila první WWW konference, na které bylo předvedeno HTML+. V červenci 1994 byla uvolněna specifikace HTML 2 a v listopadu téhož roku byl vytvořen Netscape. V roce 1995 se objevily nové druhy značek, které se do této chvíle zdály nepoužitelné pro HTML (BODY, FONT FACE, atribut BGCOLOR) jenž ovládají stylistické aspekty dokumentu. Byla táké přidána podpora pro kaskádové styly. V roce 1995 bylo také představeno HTML 3, do kterého byla přidána specifikace tabulky. V srpnu 1995 přichází Internet Explorer od Microsoftu.

V roce 1996 WWW konsorcium sestavilo vydavatelskou kontrolní asociaci pro HTML (ERB), která pomáhala se standardizačním procesem. Její snahou bylo spolupracovat a odsouhlasit všeobecný standard pro HTML , aby se vyhnula nejednotnostem ve vývoji prohlížečů. Prohlížeče do této chvíle implementovaly každý jinou podmnožinou jazyka. V dubnu 1996 W3C konsorcium vypracovalo návrh na publikování skriptů. V lednu roku 1997 W3C bez problémů a protestů největších vydavatelů prohlížečů Microsoft a Netscape schválilo standardní verzi HTML 3.2. HTML 3.2 použilo existující IETF standard HTML 2 a začlenilo HTML+ a HTML 3. HTML 3.2 zahrnovalo obtékání textu kolem obrázků, aplety, tabulky, skripty a superskripty. HTML ERB se stalo pracovní skupinou HTML a začali pracovat na Cougar, další verzi HTML. To bylo dokončeno až na přelomu roku 1997-1998. Bylo označeno jako verze HTML 4.0. Prohlížeče však v té době nebyly na novou verzi HTML připraveny. Prohlížeče Netscapu ani Microsoftu neimplementovaly kompletně šablony stylů předepsaným způsobem. Značka OBJECT pracovala ve všech prohlížečích rozdílně. Tyto nedostatky byly však brzy napraveny. Poslední verze HTML 4.01 byla vydána v prosinci 1999 a je pouhou revizí, která řeší některé chyby předchozí verze HTML 4.0. Také přidává několik málo nových elementů. Z verze HTML 4.01 byla vytvořena verze XHTML. Důvodem pro tento krok je překotný vývoj prohlížečů, které umožnily vznik obrovského množství dokumentů, jenž neodpovídají specifikaci HTML.

1.2 PHP

Kapitola o PHP vychází z [4],[5] a [6].

Hypertext Preprocesor (PHP, původně však Personal Home Page) je skriptovací programovací jazyk určený především pro programování dynamických internetových stránek, ale je možné jej použít i pro tvorbu konzolových a desktopových aplikací. PHP skripty jsou prováděny na straně serveru. K uživateli je přenášen až výsledek jejich činnosti. Jazyk PHP je dynamicky typový, to znamená že datový typ proměnné se určí až v okamžiku přiřazení hodnoty. Heterogenní pole PHP mohou obsahovat jakékoli údaje, stejně tak jako jejich indexy. To v kombinaci s dynamickým typováním výrazně usnadňuje práci programátora. Dále se pokusím stručně popsat vývoj jazyka.

Jazyk PHP se začal objevovat v roce 1994. Zakladatelem je Rasmus Lerdorf, který se rozhodl vytvořit vlastní a jednoduchý systém pro počítání přístupu ke svým stránkám. První pokus byl implementován v jazyku Perl. Perlovský kód však nadměrně zatěžoval server a proto byla implementace přepsána do jazyka C. Sada těchto vytvořených skriptů byla ještě později téhož roku vydána pod názvem Personal Home Page Tools (PHP). V roce 1995 Rasmus Lerdorf spojil své PHP s jiným programem, a to sice s nástrojem Form Interpreter (FI). Vzniklo tak PHP/FI 2.0. Rasmus Lerdorf se rozhodl uvolnit zdrojový kód PHP/FI pro všechny, takže kdokoli ho může používat, stejně jako opravovat chyby a obohacovat kód. Tento systém, který si postupně získal celosvětovou proslulost, se velmi rozšířil. Oficiálně bylo PHP/FI 2.0 uvolněno až na konci roku 1997, do této doby byly k dispozici pouze betaverze. Krátce nato byla vydána verze PHP 3.0.

PHP 3.0 bylo první verzí, jež se podobala dnešnímu PHP, tak jak jej známe. Autory byli Andi Gutmans a Zeev Suraski. Produkovali jej v roce 1997 jako kompletně přepsaný celek. PHP/FI 2.0 se jim totiž zdálo výrazně poddimenzované pro internetových aplikací. Verze PHP 3.0 byla mnohem rychlejší a vybavenější než PHP 2.0 a byla k dispozici rovněž pod operačními systémy Windows. Jednou z největších výhod PHP 3.0 byly jeho obrovské možnosti rozšíření. Poskytlo přístup a spolupráci k mnoha různým infrastrukturám a databázím, protokolům a API koncovým uživatelům. Mnoho vývojářů se připojilo a vytvořili nové rozšiřující moduly. Jiným klíčovým prvkem v PHP 3.0 byla podpora objektově orientované syntaxe a mnohem silnější a konzistentnější syntaxe jazyka. Po konci roku 1998, tedy v době svého vrcholu, bylo PHP 3.0 instalováno přibližně 10 % všech WWW serverů na Internetu.

PHP 3.0 nebylo navrženo pro efektivní práci náročnějších aplikací. Byl tedy navržen nový engine, nazvaný Zend Engine, jehož název byl sestaven z křestních jmen hlavních vývojářů Zeev a Andi. Nový engine úspěšně splnil cíle návrhu. Byl uveden až v polovině roku 1999 pod názvem PHP 4.0. Dále byl doplněn širokou škálou nových prvků, jako je podpora pro mnoho WWW serverů, buffering výstupu, bezpečnější způsoby zpracování vstupů a vstupů, HTTP sessions a mnoho nových jazykových konstruktů. Oficiální verze byla uvolněna v květnu roku 2000. A používalo jí asi 20% domén na internetu.

Verze PHP 5.0 byla uvolněna v roce 2004 a byla vybavena vyspělým objektovým přístupem podobným jazyku Java. Poslední verzí jazyka je PHP 5.2.1, která vyšla v roce 2007.

1.3 JavaScript

Kapitola o JavaScript vychází z [7] a [8].

JavaScript je objektově orientovaný, multiplatformní skriptovací jazyk. Syntaxe jazyka patří do rodiny jazyků C, C++, a Java, ale slovo Java je v názvu pouze z marketingových důvodů. Krom podobné syntaxe nemá JavaScript s jazykem Java nic společného. Jazyk je zpravidla používá jako interpretovaný programovací jazyk pro WWW stránky, a vkládá se přímo do HTML kódu stránky. Skript je pak zpracováván klientskou částí (verze pracuje s DOM, událostmi prohlížeče, dokumentu atd.), tedy na straně prohlížeče. Existují ale i JavaScript, který pracuje na straně serveru, jako například Live Wire.

Klientským JavaScriptem jsou obvykle ovládány různé interaktivní prvky GUI (tlačítka, textová políčka, zprávy, efekty) nebo tvořeny animace a efekty obrázků. Protože JavaScript pracuje na straňe serveru, plynou z toho jistá bezpečností omezení. JavaScript například nemůže pracovat se soubory, aby tím neohrozil soukromí uživatele (výjimkou jsou přístupy ke cookies). Syntax jazyka je jednoduchá, na jednom řádku může být více příkazů. Každý příkaz se ukončuje středníkem. Na konci řádku se sice středník může vynechat, ale nedoporučuje se to kvůli přehlednosti. JavaScript rozlišuje velikost písmen. Názvy funkcí a proměnných mohou začínat jen písmenem, podtržítkem nebo znakem "`$`" (dolar). Dále jen stručně popisuji historii a vývoj jazyka.

Autorem je Bernard Eich z tehdejší společnosti Netscape. JavaScript byl původně vyvíjen společností Netscape pod názvem Mocha, později pod názvem LiveScript. Ohlášen byl v prosinci 1995 ve spolupráci se společností Sun Microsystem, jako doplněk k jazykům Java a HTML. Microsoft jej distribuoval pod názvem JScript. V roce 1996 vydala firma Netscape první implementací JavaScriptu pracující na straně serveru pod jménem Live Wire (dnes existuje několik implementací včetně opensource implementace Rhinola založená na Rhino, Apache a gcj). V červenci roku 1997 byl JavaScript standardizován asociací Europen Computer Manufacturers Association (ECMA) a v srpnu 1998 jej standarizovala i asociace International Standards Organization (ISO). Standardizovaná verze JavaScriptu byla pojmenována ECMAScript. Později jsou odvozeny další verze jako například ActionScript.

Pro ujasnění pojmů uvádím, že pokud budu v dalším textu používat pojmu JavaScript, budu mít namysli formu fungující na klientské části, nebude-li uvedeno jinak.

1.4 MySql

Kapitola o MySql vychází z [9].

MySQL je multiplatformní databáze jenž je založena na komunikaci za pomocí jazyka SQL. Přesněji se jedná o dialekt tohoto jazyka s některými rozšířeními. Existuje ve dvou licencích, GPL nebo komerční licence. Právě díky bezplatné GPL licenci se tento druh databází ujal u mnoha uživatelů. Vyznačuje se snadnou implementovatelností a výkonem. MySQL bylo od začátku vyvíjeno především s ohledem na rychlost, i za cenu některých zjednodušení, jako například jednoduchého způsobu zálohování. MySql také až donedávna nepodporovalo pohledy, uložené procedury a databázové triggery. Tyto vlastnosti byly implementovány až u posledních verzí (od verze 5.0). MySql ukládá data do databázových tabule, které se liší svým použitím, možnostmi a způsobem ukládání dat do souborů. Nejpoužívanějším způsobem je MyISAM, který nepodporuje transakce, dále pak InnoDB, které transakce podporuje. Dále to jsou BerkeleyDB, MEMORY, NDB Cluster (od verze 5.0), ARCHIVE a CVS pro ukládání dat v prostých textových souborech. MySql bylo vytvořeno švédskou firmou MySql AB. Hlavními autory jsou David Axmark a Michael Widenius. Aktuální komerční verzí je MySql 5.0.30 Enterprise a GPL verzí je MySql 5.0.27, která vyšla v listopadu roku 2006. V únoru roku 2007 byla uvolněna zatím poslední betaverze MySql 5.1.14.

2 Extrakce informací z www stránek

Způsobů jak extrahovat informací z www stránek je mnoho. Snad každý programovací jazyk poskytuje nástroje pro extrakci dat, ať už se jedná o soubor umístěný na lokálním paměťovém médiu nebo na vzdáleném počítači v síti. Tato bakalářská práce je zaměřena na extrakci dat z HTML stránek umístěných na vzdálených serverech v síti internet, proto se budu v této kapitole věnovat tomuto tématu. HTML stránka není ve své podstatě nic jiného než soubor s textovými (v některých případech binárními) daty, a tímto způsobem s ní také pracujeme. V této kapitole jsem se zaměřil na několik možností jak tyto informace získávat v závislosti na zvoleném nástroji tvorby aplikace (programovacím či skriptovacím jazyku). V následujících podkapitolách je popsána například utilita *wget*, která je vhodná pro zapojení do skriptu, nebo funkce *fopen* dostupná v mnoha programovacích jazycích.

2.1 Utilita wget

Kapitola o wget vychází z [10] a [11].

Jednou z možností jak extrahovat data z www stránek je utilita *wget*. Jedná se o samostatný program, který je vhodný pro zapojení do skriptu. V zapojení ve skriptu společně z dalšími GNU programy pro filtraci textových informací, jako jsou *awk*, *grep* a *sed* může tento skript tvořit opravdu silnou zbraň pro zpracování dat z HTML stránek.

GNU *wget* je bezplatný a volně šiřitelný nástroj, který umožňuje jednoduché i pokročilé souborové stahování, jako je například rekurzivní download. Je to příkazový linkový program, který je určen hlavně pro unixové systémy. *Wget* byl ale pro svoji oblibu napsán a přeložen i pro Windows, tudíž je multiplatformní. Program je noninteraktivní, což znamená, že může běžet na pozadí. Existují i grafické nadstavby jako například *gwget*, ale ty jsou pro zapojení do skriptu nepodstatné. Utilita může také zapisovat časy posledního stahovaní, a poté porovnávat, zda-li se obsah souboru od poslední extrakce změnil, a tak automaticky získávat nejnovější verze. Program byl navržen pro stahování na pomalých a nestálých připojeních. Podporuje sekundární a mirror servery, které můžou odlehčit síťovému spojení, zrychlit stahování a poskytnout přístup za firewally. Program *wget* je schopen stahovat data přes HTTP i FTP protokol. Dále je program schopen http-autentifikace, což umožní stahování dat i z privátního adresového prostoru. V další části je popsáno základní použití programu a jeho parametrů.

2.1.1 Základní použití utility wget

wget http://www.komenda.cz/

- stáhne soubor *index.html* z výše uvedené adresy, nebo ten co se vrací na /

wget –p http://www.komenda.cz/

- stáhne soubor *index.html* z výše uvedené adresy a vše, co je nutné ke zobrazení stránky

wget –m http://www.komenda.cz/

- parametr -m slouží pro mirrorování celých webů, to znamená rekurzivní stahovaní do nekonečné hloubky se všemi prvky pro úplné zobrazení stránky

wget –r –l 2 –k http://www.komenda.cz/

- rekurzivně stáhne obsah stránek, to zajišťuje přepínač **–r**, stahování se provede do hloubky 2, to zajišťuje přepínač **–l 2**, poslední přepínač **–k** říká, že se mají všechny odkazy převést pro lokální prohlížení

wget –A jpg, bmp –r –l 1 http://www.komenda.cz/

- parametr **–A** říká, že se mají stáhnout pouze soubory s koncovkou *jpg* a *bmp*, vždy se ale stáhne soubor *index.html*

wget –R jpg, bmp –r –l 1 http://www.komenda.cz/

- parametr **–R** říká, že se nemají stáhnout soubory s koncovkou *jpg* a *bmp*, vždy se ale stáhne soubor *index.html*

wget --http-user = login --http-password=heslo http://www.komenda.org/

- příkazy **--http-user** a **--http-password** zasílají jméno a heslo pro přihlášení přes http-utentifikaci

wget ftp://ftp.komenda.cz/skola/bp.html

- stáhne soubor *bp.html* pomocí FTP protokolu

wget –c ftp://ftp.komenda.cz/skola/bp.html

- přepínač **–c** říká, že pokud dojde při stahování souboru *bp.html* k přerušení, má se soubor dostahovat, až bude zase přístupný

Program wget má sám jako takový stovky parametrů, uvedeno je několik málo z nich, které demonstrují základní funkce utility. Více informací můžete najít v manuálových stránkách programu wget nebo je získat pomocí přímo v programu wget pomocí přepínače `--help`. Další informace můžete najít na domácí internetové stránce GNU utility *http://www.gnu.org/software/wget/* nebo v [10] a [11].

2.2 Funkce fopen

Kapitola o wget vychází z [12] a [13].

Na rozdíl od utility *wget* (2.1) se nejedná o samostatně fungující program, ale o funkci v konkrétním programovacím jazyku. Tato funkce je implementována v mnoha programovacích i skriptovacích jazycích počínaje jazykem C až například po Live Wire, typ JavaScriptu firmy Netscape, který pracuje na straně serveru (1.3). Tato práce je převážně zaměřena na jazyky a nástroje pro tvorbu dynamických webových stránek, proto se budu věnovat konkrétně implementaci a použití funkce fopen v jazyku PHP (1.2). Funkce fopen je standardně implementovaná v PHP a není ji tedy nutno dodávat v externích knihovnách.

```
resource fopen( string $filename, string $mode [, bool $use_include_path
[, resource $context]] )
```

`string` **`$filename`** – Funkce *fopen* má dva povinné parametry, prvním z nich je jméno (popřípadě cesta k souboru, jež bude otevřen). Jestliže řetězec obsažený **`$filename`** začíná "http://" nebo "HTTP://", je navázáno spojení s příslušným server pomocí protokolu HTTP 1.0. Posílá se hlavička "Host: " pro přístup k virtuálním serverům založeným na jméně a je vrácen deskriptor ukazující na začátek těla dokumentu. Je třeba vložit koncové lomítko za název adresáře. Bohužel tato funkce nepracuje s přesměrováním. Stejným způsobem funguje přenos pomocí FTP protokolu (Jestliže řetězec obsažený **`$filename`** začíná "ftp://" nebo "FTP://"). Podmínkou je, že server musí podporovat pasivní FTP přenos, v opačném případě přenos selže. Od verze PHP 3.0.13 je možné nastavit **`$filename`** jako "php://stdin", "php://stdout", nebo "php://stderr". V těchto případech bude otevřen standardní vstup nebo výstup. V předcházejících verzích PHP se muselo nastavovat **`$filename`** jako "/dev/stdin" nebo "/dev/fd/0". Ve všech ostatních případech je otevřen soubor na lokálním paměťovém médiu. Je potřeba mít na paměti také pravidla pro zápis cesty ke hledanému souboru. Většinou, pokud není uvedena cesta k souboru, je hledaný v adresáři, ve kterém je umístěn skript pro otevření. Záleží však na konfiguraci serveru. Také je třeba mít na paměti, že OS Windows používají `"\"` (obrácených lomítek). Na závěr je dobré upozornit na jména domén v URL. Ty nejsou sace-sensitive, to znamená, že nerozlišují malá a velká písmena. Ale naopak jména adresářů a souborů jsou case-sensitive, tedy velikost znaků rozšiřují.

`string` **`$mode`** – Druhým povinným parametrem říkáme operačnímu systému, jaké požadavky máme na otevíraný soubor. Lépe řečeno, s jakými parametry má operační systém otevřít soubor.

r+ Čtení - otevření souboru pro čtení a zápis od začátku.

r Čtení - otevření souboru pouze pro čtení od začátku.

w+ Zápis - soubor se otevře pro zápis a čtení od začátku. Pokud soubor existuje, bude jeho obsah smazán, neexistuje-li bude vytvořen.

w Zápis- soubor se otevře pouze pro zápis od začátku. Pokud soubor existuje, bude jeho obsah smazán, neexistuje-li bude vytvořen.

a Soubor bude otevřen pouze pro doplnění nového zápisu za konec již existujícího zápisu. Pokud soubor ještě neexistuje bude vytvořen.

a+ Soubor bude otevřen pro doplnění nového zápisu a čtení. Zápis začne od konce předešlého zápisu. Když soubor neexistuje, pokusí se ho vytvořit.

b Tento parametr se používá pro rozlišení binárních souborů na operačních systémech Windows.

`bool` **`$use_include_path`** – Třetí parametr je nepovinný a pokud je nastaven na "1" (`TRUE`), bude se hledat soubor také v *include_path*.

`resource` **`$context`** – Posledním nepovinným parametrem je formát výstupního deskriptoru. Pokud dojde k chybě při otevírání, je vrácena hodnota `FALSE`.

2.2.1 Ukázka metody objektu HTML Parser pro načtení URL

```
function HtmlParser_ForURL () {
      //otevřu spojeni na URL, vrací deskriptor
      $fp = fopen ($this->url, "r");
      //pokud se nepovede otevřít soubor vrátím chybu
      if(!$fp)
            return $this->chyba;
      //vyprázdním obsah řetězce, kam budu načítat data z URL
      $this->content = "";
      for (;;) {
            //pokud načítám data v 8 Kb blocích a ukládám je
            $data = fread ($fp, 8192);
            if (strlen($data) == 0) {
                  //jsem na konci souboru, uzavřu spojení
                  fclose ($fp);
                  return $this->content;
            }//konec podmínky
            $this->kontent .= $data;
      }//konec cyklu for
}//konec metody
```

2.3 Další možnosti extrakce

Je mnoho dalších možností jak získávat data z webových stránek. Například v jazyku C# .NET 2.0 je možno použít třídu `HttpWebRequest`, `Stream`, `HttpWebResponse`. Je možné také použít vyspělou komponentu, kterou nabízí Microsoft Visual Studio 2005 pro WinForms. Jedná se o komponentu `WebBrowser`. Podobná komponenta je k dispozici například i v Borland Delphi 7 nebo Borland Builder C++.

3 Periodické spouštění aplikací

Další částí, kterou se zabývá tato práce, je spouštění skriptů, či aplikací pro extrakci dat. Periodické spouštění aplikací je důležité v situacích, kdy chceme v pravidelných intervalech extrahovat informace z HTML stránek. Jedním z nejčastějších řešení tohoto problému, bez ohledu na způsob implementace, je periodický časovač *cron*. Tímto programem a dalšími možnostmi se podrobně zabývá tato kapitola.

3.1 Periodický časovač Cron

Kapitola o cronu vychází z [14] a [15].
Téměř v každém vyspělém webovém systému je nutné používat periodicky naplánované akce. V mém případě to jsou situace, kdy skenuji určitou webovou stránku a zjišťuji odlišnosti od posledního skenování. K tomuto účelu je možné použít periodický časovač. Démon *cron* je systémový nástroj pro Unix a Linux. Tento démon spouští v předem naplánovaných časech a intervalech zaregistrované aplikace. Obdobou v OS Windows je plánování úloh.

Cron však není určen pouze pro webové systémy, je jej možné použít pro spouštění všech aplikací a příkazů. Například unixoví administrátoři používají *cron* pro úklid systému nebo pravidelnému zálohování dat. Pokud mluvíme o webhostingu, kde se *cron* také hojně využívá, nebudeme mít k programu přímý přístup. Pokud však pracujeme na vlastním systému (serveru, osobní PC atd.), máme přístup ke *cronu* přímo přes příkazovou řádku nebo přes jeho grafickou nadstavbu. Program však neumí spouštět aplikace či skripty v intervalu kratší jedné minuty. V těchto případech je výhodnější nechat aplikaci běžet permanentně (o této možnosti pojednává další podkapitola).

Pro nastavení spuštění aplikace pomocí *cronu* můžeme využít jednu ze dvou možností. První spočívá v tom, že nakopírujeme aplikaci do jednoho z těchto adresářů */etc/cron.daily*, */etc/cron.hourly, /etc/cron.weekly* a */etc/cron.monthly*. Poté je aplikace spouštěna v intervalech, jež vyplývají z názvů adresářů. Tento způsob je jednoduchý, ale nevýhodný v případech, kdy požadujeme pouze jedno spuštění nebo spouštění v jiném čase nebo jiných intervalech. Proto existuje další varianta. Pro tuto druhou možnost nastavování je nutné použít utilitu, která se nazývá *crontab*. Je to vlastně seznam úloh pro démona *cron*. Každý uživatel má přístup ke svému vlastní tabulce. Přistoupit k této tabulce můžeme pomocí textového editoru, nebo pomocí příkazu `crontab -e`. Záznam v tabulce se skládá ze šesti údajů oddělených mezerami nebo tabulátorem. Pořadí a hodnoty jednotlivých záznamů jsou popsány na konci tohoto odstavce. Pokud chceme říci, že se má aplikace spouštět každou hodinu, nahradíme v záznamu číslo hodiny symbolem `"*"` (hvězdička). Pokud chceme aby byla aplikace spuštěna dvakrát do hodiny, nastavíme minuty spuštění a oddělíme je

pomocí znaku "," (čárka). Jednotlivé záznamy jsou od sebe odděleny pomocí konců řádku. Takto to funguje se všemi údaji v tabulce.

1. **minuta (0 – 59 nebo *)**
2. **hodina (0 – 23 nebo *)**
3. **den v měsíci (1 – 31 nebo *)**
4. **měsíc (1 – 12 nebo *)**
5. **den v týdnu (0 – neděle ... 6 – sobota)**
6. **cesta k aplikaci, která se spustí**

3.1.1 Ukázka několika záznamů v Crontab

0 3 * * * /bin/php –f /www/komenda/sken.php

- vždy ve tři hodiny ráno spustí skript *sken.php*

0,30 3,5 * * * /www/komenda/sken.php

- vždy ve tři hodiny, v půl čtvrté, v pět a v půl šesté ráno spustí skript *sken.php*

1 2 3 * */program/save_me

- vždy třetí den v měsíci ve dvě hodiny a jednu minutu ráno je spuštěn program *seve_me*

*** 16 * * * wget http://www.komenda.cz/**

- každou minutu v od čtyř do pěti odpoledne každý den spustí program *wget* (2.1)

Problém však nastává při spouštění PHP skriptů. Aby bylo možno spouštět skript jako v prvním ukázkovém záznamu (pomocí PHP interpretu), je nutno, aby na serveru bylo kompilováno CGI (tedy existuje spustitelný program PHP). Pokud tomu tak není, je možno použít program *wget*, který skript spustí podobně jako internetový prohlížeč. Toto řešení však není ideální, protože skript příliš vytěžuje webový server a také se uplatňuje timeout, který může skript předčasně ukončit. Spouštění z příkazové řádky je rychlejší a efektivnější. Je li tedy kompilováno CGI je možno spouštět PHP skripty jako v prvním ukázkovém záznamu, bez nutnosti zásahu do skriptu. Druhou možností, kterou demonstruje druhý ukázkový záznam, je přidat specifikaci "`#!/bin/php`" do skriptu. Potom se skript chová jako spustitelný.

Pokud chceme používat Cron pro internetové aplikace a nemáme k dispozici přístup k příkazové řádce serveru a tudíž i k tabulce záznamů, můžeme například použít *cron* na jiném serveru nebo počítači, který má přístup do sítě internetu a spouštět tento skript pomocí programu *wget*. Nicméně většina poskytovatelů webhostingu *cron* zpřístupňuje, ale jeho nastavování je individuální.

3.2 Další možnosti periodického spouštění aplikací

Pokud nemáme *cron* k dispozici, a chceme využívat jeho služby pro spouštění PHP skriptů, můžeme využít služby *WebCron*. Tato služba je volně dostupná a je možné ji nastavovat přes přehledné webové rozhraní, to je dostupné na adrese *http://www.webcron.org/*.

Periodický časovač je výhodný, ale nelze ho vždy použít. Jedním z dalších možností je spuštění aplikací pomocí plánování úloh v OS Windows. Další možností je permanentně spuštěná aplikace. To se hodí pro velmi krátké intervaly (méně než jedna minuta). Čekání mezi akcemi je možno zajistit například pomocí uspání aplikace pomocí příkazu a metod jako jsou `sleep`, `pause` nebo `delay`. Například v jazyku C# .NET 2.0 je možno použít třídu `Timer`, které za pomocí delegáta přiřadím funkci, jež bude spuštěna v pravidelných intervalech. Podobná komponenta je k dispozici například i v C++, Borland Delphi nebo Borland Builder C++.

4 Lexikální analýza

Kapitola o lexikální analýze vychází z [16] a [17].
Dalším krokem po extrakci dat z webových stránek (2) je lexikální analýza. Mějme tedy zdrojový kód stránky načtený do proměnné přístupné z aplikace (je zřejmé, že tento postup by byl značně neúsporný, protože udržovat v paměti celý řetězec se zdrojovým textem by zbytečně zatěžovalo paměť, berme to tedy pouze jako jednu z možností pro uvedení do problematiky). Zdrojový kód se skládá z lexémů. Lexémy jsou logicky oddělené lexikální jednotky jako například identifikátory, klíčová slova, čísla, operátory atd. Lexémy jsou reprezentovány tokeny, ty mohou mít i atributy. Činnost lexikálního analyzátoru (scanneru) je tedy následující. Analyzátor prochází zdrojový kód uložený v proměnné a musí rozeznávat a klasifikovat lexémy a přiřazovat jim reprezentaci v podobě tokenů. Další činností lexikálního analyzátoru je odstranění komentářů a takzvaných prázdných míst. To jsou místa, která nemají na chod programu žádný význam a ve zdrojovém kódu se neuplatňují. Lexikální analyzátor úzce spolupracuje se syntaktickým analyzátorem o němž bude řeč v další kapitole. Protože je moje práce zaměřena na jazyk HTML, budu se dále zabývat lexikální analýzou jazyka HTML.

4.1 Lexikální analýza jazyka HTML

Co je to jazyk HTML, jaké jsou jeho vlastnosti, zařazení, vývoj a další informace bylo popsány v první kapitole této práce (1.1). Pro pochopení lexikální analýzy jazyka HTML je však nutné doplnit několik dalších poznatků.

HTML je součástí rodiny značkovacích jazyků a doposud vychází ze SGML (další verze bude s největší pravděpodobností vycházet z XML, protože se tyto dva jazyky stále více kombinují). Protože se jedná o značkovací jazyk, pro úplnost doplním, že značky jsou ohraničeny pomocí znaku "<" a ">". Části dokumentu jsou reprezentovány elementy, ty definují daný úsek dokumentu a přiřazují mu vlastnosti chování i vzhledu. Obecně se dá říci, že elementy mají začátek (počáteční tag), obsah a konec (koncový tag). Elementy se do sebe mohou rekurzivně vnořovat. Element může být i prázdný, ten nesmí v dokumentu obsahovat žádný obsah a může mít zkrácený zápis. XHTML již vyžaduje, aby všechny elementy byly reprezentovány počátečním a koncovým tagem nebo pomocí zkráceného zápisu. V jazyku HTML to není ještě explicitně vyžadováno. Jako příklad uvedu atribut konce řádku. V jazyku HTML je povalená tato značka `<br>`, ale v XHTML je vyžadována tato značka `<br />`. Elementy mohou být parametrizované pomocí atributů. Atributy jsou klíčová slova, nikoliv poziční a můžou tvořit celý seznam atributů. Pro úplnost uvedu, že atributy se zapisují v HTML a XHTML s následující syntaxí `jméno_atributu = "hodnota atributu"` nebo je možno uvozovky nahradit apostrofy a v některých verzích HTML je lze úplně vynechat. Dále je

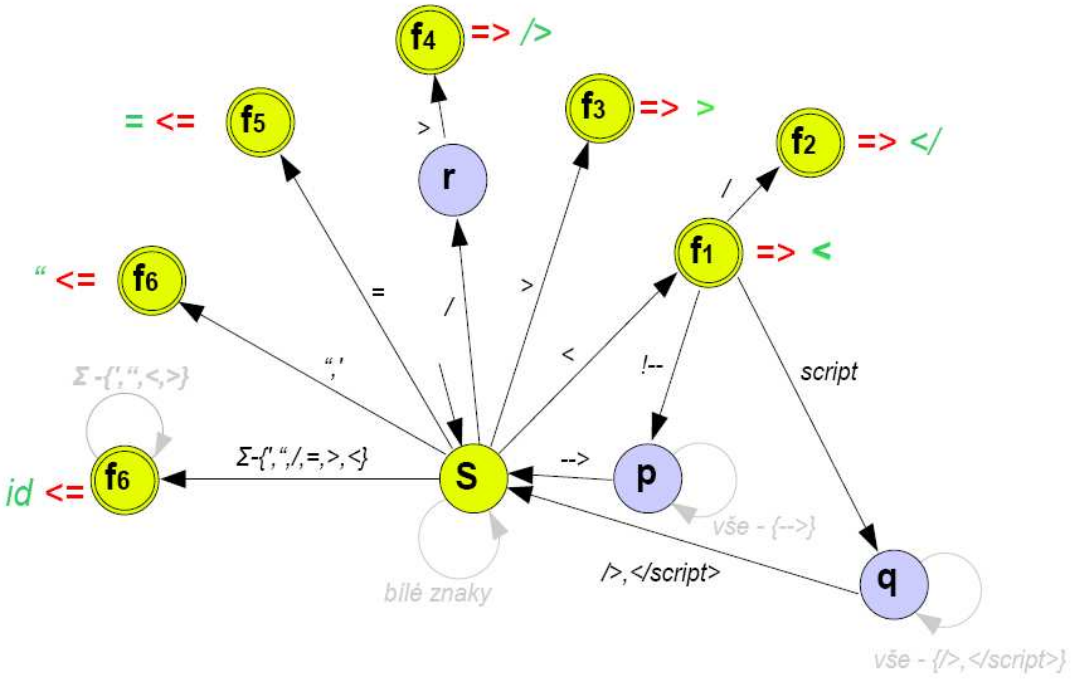
možno použít malá i velká písmena pro označení tagů a atributů (sace-sensitive). Obecně se doporučuje dodržovat jednotnou formu v celém dokumentu. Další poměrně specifickou součástí, kterou v jazyku HTML můžeme najít, jsou HTML komentáře. Komentáře se při lexikální analýze přeskakují a jen pro úplnost uvedu, že komentáře jsou uzavřeny do těchto značek "`<!--`" a "`-->`".

Zdrojový kód HTML je velmi často doplňován skripty v jazyku klientský JavaScript (1.3), JScript nebo VB. Tyto skripty mohou být přilinkovány z externího souboru. V tomto případě nenastává při lexikální analýze žádný problém. Pokud jsou však součástí zdrojového textu HTML stránky, doporučuje se dodržovat následující syntax.

```
<script language="JavaScript"> <!-- zdrojový kód JS //--> </ script>
```

Pokud je dodržena syntaxe jako v předcházejícím případě, lexikální analyzátor jazyka HTML přeskočí kód skriptu a ignoruje jej jako komentář. Problém však nastává, pokud není syntaxe dodržena (pokud chybí značky komentářů HTML). Je proto lépe připravit lexikální analyzátor i na tyto situace. Jako příklad jsem uvedl JavaScript, ale tato situace se týká například i kaskádových stylů a dalších.

Pro znázornění jedné z možností lexikální analýzy jsem se rozhodl použít schéma deterministického konečného automatu (Obrázek 4.1.1.). Pro přehlednost byla zjednodušena část, která zajišťuje přeskakování komentářů a skriptů. Podle automatu je možné implementovat lexikální analyzátor v různých jazycích. Nebo si jej nechat vygenerovat pomocí specializovaného nástroje, například v LEXu. Více informací o lexikální analýze a programu LEX se můžete dozvědět v [17].



Obrázek 4.1.1. Lexikální analýza pomocí DKA

5 Syntaktická analýza

Kapitola o syntaktické analýze vychází z [18], [19] a [20].
Poté co je připraven lexikální analyzátor (4), který vrací klasifikované tokeny, můžeme provádět sémantickou analýzu. Syntaktická analýza nám prozradí, je-li skenovaná stránka správně syntakticky napsána. Tato analýza pracuje s tokeny, které posílá lexikální analyzátor a sestavuje derivační strom. Vytváření tohoto stromu je založeno na gramatických pravidlech. Existují dva přístupy, shora dolů a zdola nahoru. Pokud je derivační strom sestaven správně, je i zdrojový kód syntakticky správný. Vzhledem k zaměření práce se budu věnovat syntaktické analýze jazyka HTML.

5.1 Syntaktická analýza jazyka HTML

Na úvod podkapitoly bych položil řečnickou otázku. Tématem této práce je získávání dat z HTML stánek, tak proč provádět syntaktickou analýzu? Asi hlavním důvodem je rozčlenění a identifikace objektů, které je možno na stránce rozeznávat. Jako příklad uvedu část zdrojového textu v jazyku HTML, na kterém dále rozvinu svoji myšlenku.

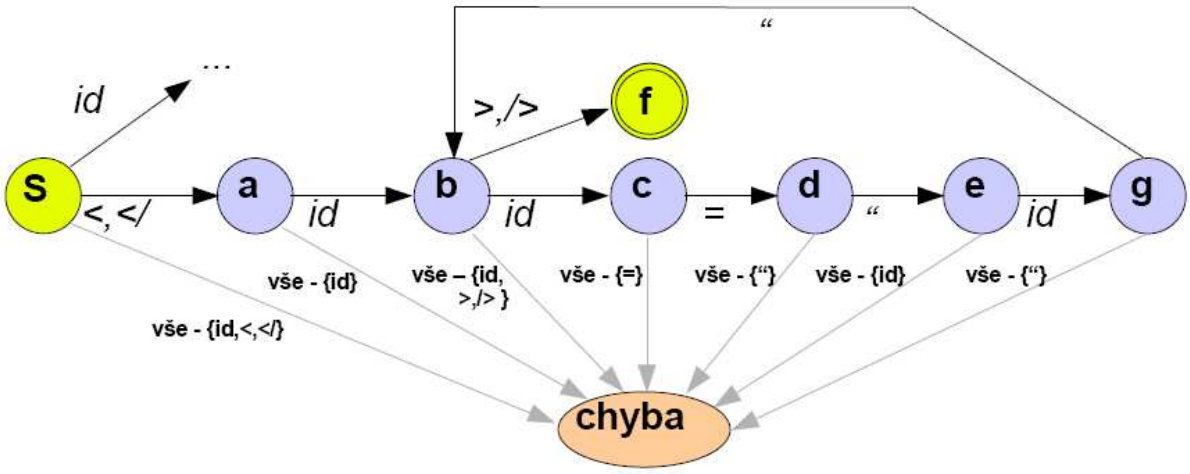
```
<p id="p1">
        <font id="f1" class="cerna">
                Bakalářská práce pojednává o získávání dat z HTML.
                <font id="f2" class="modra">
                        Moji práci můžete najít na webu na adrese:
                        <a id="a1" htref="http://komenda.bp.cz">
                                http://komenda.bp.cz
                        </a>
                </font>
                Další specifikace bude doplněna.
        </font>
</p>
```

Předcházející zdrojový kód může ukázat, jak rozlišovat objekty na webové stránce. Například můžeme vzít první element, kterým je odstavec (začíná tagem s identifikací **p1**). Pokud budeme chtít sledovat změny v tomto odstavci, zařadíme si jej jako objekt odstavec. Tento objekt však v sobě zanořuje další objekt, který reprezentuje element představující font (začíná tagem s identifikací **f1**) a ten v sobě zanořuje další elementy. Syntaktickou analýzu je tedy možno využít k pojmenování a ke klasifikaci objektů, které se na stránce nacházejí. Lépe se to dá objasnit na elementu fontu (začíná tagem s identifikací **f1**). Pokud si jej zařadíme jako objekt, vidíme, že v sobě zapouzdřuje i element

druhého fontu (začíná tagem s identifikací **f2**) a ten v sobě zapouzdřuje element odkazu (začíná tagem s identifikací **a1**). Kdybychom nevyužívali prvků syntaktické analýzy, objekt fontu by byl ukončen již na ukončovacím tagu vnořeného druhého fontu. Kdyby se tak stalo, objekt font by do svého obsahu nezahrnul text "`Další specifikace bude doplněna.`". Je také zřejmé, že všichni tvůrci stránek neopatřují všechny tagy atributem **id**, který by element jednoznačně identifikoval. Tyto atributy jsem uvedl pouze pro přehlednost a dobrou orientaci v ukázkovém zdrojovém textu. Rozdělení a pojmenování elementů stránky na objekty je tedy důležité pro následný výběr a filtraci dat a právě k tomu lze syntaktickou analýzu dobře využít.

Dalším důvodem, proč provádět syntaktickou analýzu HTML kódu je ověření jeho bezchybnosti. To sice není přímo předmětem této práce, ale úzce to s ní souvisí. Vždy se nemůžeme spoléhat na to, že je HTML kód napsán správně. Dnešní prohlížeče totiž tolerují některé chyby, jako jsou například chybějící značky, nebo nesprávně zapsané atributy. Na vzhledu stránky se to pak bohužel vůbec neprojeví. Ale při pozdějším zpracováním stránky, například pro filtraci dat, to může znamenat problémy.

Pro znázornění jedné z možností syntaktické analýzy jsem se rozhodl použít schéma dobře specifikovaného konečného automatu (Obrázek 5.1.1.). Tento automat provádí pouze syntaktickou analýzu počátečního či koncového tagu elementu. Reprezentuje tedy jen jednu část syntaktického analyzátoru (parseru). Také počítá s uzavřením hodnoty atributů do uvozovek. Tento automat předpokládá, že pracuje s lexikálním analyzátorem, který vrací podobně specifikované tokeny, jako analyzátor popsaný v kapitole věnované lexikální analýze jazyka HTML (4.1 obrázek 4.1.1.).



Obrázek 5.1.1. Syntaktická analýza pomocí DSKA

5.1.1 Syntaktická analýza jazyka HTML pomocí zásobníkového automatu

Díky schopnosti rekurzivního vnořování elementů je výhodné použít k syntaktické analýze zásobníkový automat. Tato metoda je poměrně jednoduchá na implementaci a není na rozdíl od jiných metod (rekurzivní sestup) příliš náročná na paměť.

Pro následující text předpokládejme, že lexikální analyzátor vrací tokeny, jež reprezentují počáteční a koncové tagy elementů, atributy, hodnoty atributů a obsahy elementů. Potom syntaktická analýza pomocí zásobníkového automatu funguje následovně. Na počátku je vyprázdněn zásobník. Syntaktický analyzátor volá lexikální analyzátor, ten vrátí klasifikovaný token. Pokud reprezentuje počáteční tag elementu, podívám se do tabulky gramatických pravidel (pravidla mohou vypadat například takto `table->th`, `table->tr`, `tr->td`), kde se pokusím najít pravidlo, jenž informuje, jaký token musí předcházet aktuálně vrácenému tokenu. Pokud takové pravidlo existuje, podívám se na vrchol zásobníku a pokud je zásobník prázdný, nebo token na vrcholu zásobníku nereprezentuje levou stranu alespoň jednoho nalezeného pravidla, jedná se o syntaktickou chybu. Pokud pravidlo souhlasí nebo pravidlo neexistuje, tak uložím aktuální token na vrchol zásobníku. Poté se volá dále lexikální analyzátor. Pokud je dalším tokenem identifikátor atributu, sáhne se do tabulky gramatických pravidel, kde se najdou příslušná pravidla pro konkrétní tag. Pokud je nalezeno pravidlo, které odpovídá dané situaci, pokračuje se dále, pokud se jedná se o chybu. Podobným způsobem je možno kontrolovat i hodnoty atributů. To však již souvisí s pokročilou syntaktickou a také se sémantickou analýzou. Tyto hodnoty však v případě mé práce nehrají významnou roli a na sledování textových informací na HTML stránkách nemají velký vliv. Proto na ně v mé práci neberu takový ohled a při předběžné syntaktické analýze je nekontroluji. Pokud lexikální analyzátor vrátí token reprezentující koncový tag elementu, je vyjmut z vrcholu zásobníku uložený token. Je-li párovým protějškem aktuálně vráceného tokenu, můžeme pokračovat v analýze. Pokud se však tokeny liší, na stránce se vyskytla syntaktická chyba. Můžeme vrátit vyjmutý tag na vrchol zásobníku a pokračovat v parsování nebo můžeme analýzu přerušit a nahlásit chybu. Pokud lexikální analyzátor nahlásí konec souboru, podívá se syntaktický analyzátor do zásobníku a pokud není prázdný, stránka není syntakticky bezchybná. Pokud je zásobník prázdný, zdrojový kód je bez chyb.

5.1.2 Syntaktická analýza jazyka HTML pomocí rekurzivního sestupu

Druhou možností provádění syntaktické analýzy, kterou jsem ve své implementaci také vyzkoušel, je metoda využívající rekurzivní sestup. Metoda je založena na schopnosti funkce volat sama s sebe.

Předpokládejme, že lexikální analyzátor vrací tokeny, jenž reprezentují počáteční a koncové tagy elementů a obsahy elementů. Pro zjednodušení vynechávám syntaktickou analýzu atributů a jejich hodnot. Dále mějme funkci, jež spolupracuje s lexikálním analyzátorem. Funkce má parametr, jenž nás informuje, jaký byl předcházející token. Návratová hodnota funkce informuje, zda nejsme již na konci parsovaného souboru. Tuto funkci, s nastaveným prázdným parametrem, pak spouštíme v cyklu tak dlouho, dokud lexikální analyzátor nenahlásí konec parsovaného souboru. Ve funkci je volán lexikální analyzátor. Ten vrátí funkci klasifikovaný token. Pokud reprezentuje počáteční tag elementu, funkce se podívá do tabulky gramatických pravidel (pravidla mohou vypadat například takto `table->th`, `table->tr`, `tr->td`) a pokusí se najít pravidlo takové, kde levé straně pravidla vyhovuje token vrácený parametrem funkce a pravé straně token vrácený analyzátorem. Pokud takové pravidlo existuje, a neexistuje-li pravidlo, kde se levá strana shoduje s parametrem nebo je vstupní parametr funkce prázdný, zavolá funkce sama sebe s parametrem aktuálně vráceného tokenu. Pokud parametr není prázdný a zároveň je nalezeno pravidlo, ve kterém se levá strana shoduje s parametrem a vrácený token nevyhovuje pravé straně ani jednoho takového pravidla, jedná se o syntaktickou chybu v parsovaném zdrojovém kódu. V kódu funkce za rekurzivním voláním sama sebe, se volá opět lexikální analyzátor, jež vrací token. Tento token musí být párovým protějškem tokenu, jež byl vrácen před rekurzivním voláním sama sebe. Pokud tokeny nesouhlasí, jedná se o syntaktickou chybu ve zdrojovém textu.

Výše popsaná syntaktická analýza je záměrně zjednodušená, protože její pochopení může být náročnější než u metody využívající zásobníkový automat. Její implementace je sice jednoduší, ale na druhou stranu má řadu nevýhod. Rekurzivní sestup značně zatěžuje paměť, protože se celá funkce v paměti může vytvořit několikrát. V extrémních případech může dojít k pádu aplikace z důvodu nedostatku paměti. Řešením v takovýchto situacích bývá počítadlo zanoření. Sledovali bychom počet zanoření a po překročení určité hranice bychom další zakázali. To si ale v tomto případě nemůžeme dovolit, protože bychom neprovedli kompletní analýzu. Z vlastní zkušenosti mohu doporučit metodu zásobníkového automatu. Vlastní testování těchto metod a dosažené výsledky popisuji v dalších kapitolách této práce.

5.2 HTML parser pro PHP 4

Kapitola o HTML parseru vychází z [21].

Ve své práci jsem implementoval vlastní lexikální analyzátor i vlastní syntaktický analyzátor. Vlastní syntaktickou analýzu jsem využil pro kontrolu správnosti stránek a pro filtraci na objekty v aplikaci pro sledování obsahu HTML stránek. Vlastní lexikální analýzu jsem využil v editoru pro úpravu vzhledu výstupního protokolu aplikace. Podrobněji se této problematice věnuji v dalších kapitolách o filtraci dat a v kapitole zaměřené na samotnou aplikaci. V samotné aplikaci jsem však využil volně dostupný produkt, který napsal Jose Solorzano v roce 2003 a zdrojový kód je možno získat na adrese *https://sourceforge.net/project/showfiles.php?group_id=91649*. Jedná se o implementaci třídy, která nese pojmenování *HTML parser,* a je určena, jak již název podkapitoly naznačuje, pro jazyk PHP (1.2) verze 4 (samozřejmě je funkční ve všech vyšších verzích).

Podle názvu by se mělo jednat o syntaktický analyzátor. Ve skutečnosti se jedná spíše o lexikální analyzátor, takže podle mého názoru by se měl jmenovat spíše *HTML scaner*, ale budu respektovat autorovo mínění. *HTML parser*, provádí jednoduchou syntaktickou analýzu, ale pouze do takové míry, jaká je znázorněna na obrázku 4.1.1. v kapitole o syntaktické analýze jazyka HTML (4.1). Navenek vrací spíše klasifikované tokeny a jejich atributy. To mě však pro moji aplikaci maximálně vyhovuje. Provedl jsem sice několik nepatrných zásahů do zdrojového textu, tak aby maximálně vyhovoval mým požadavkům (například jsem zařídil aby názvy tagu a atributů byly vraceny pouze malými písmeny atd.), to však nemělo na výsledné chování žádný významný vliv.

Jak jsem již uvedl jedná se o implementaci třídy, tedy o objektově orientované programování. Třída je navržena tak, že zpracovává kód ze vstupního řetězce. Nejlépe bude funkci *HTML paresu* demonstrovat na ukázkovém příkladě.

```
//vložíme zdrojové texty třídy HTMLparser
include "htmlparser.inc";
//do proměnné $zdroj přiřadíme parsovaný zdrojový kód
$zdroj = "<font color=\"red\" >Tento text je červený.</font>";
//vytoříme objekt HTMLparseru a předáme mu zrojový kód
$parser = new HtmlParser ($zdroj);
//voláme metodu parse, ta postupně parsuje zdrojový kód a pokud jsme
//na konci zdrojového kódu vrátí hodnotu FALSE
while ($parser->parse()){

        //v cyklu zpracovávám návratové hodnoty v podobě atributů

}
```

Zdrojový kód ukazuje, jak je možno *HTML parser* vytvořit a použít. Protože se jedná o skriptovací jazyk, není nutné objekt po vytvoření rušit. Všechny objekty jsou po skončení skriptu automaticky uvolněny.

Dalším krokem je získání informací o parkovných hodnotách. Tyto informace se zpracovávají uvnitř cyklu a jeden průchod smyčky cyklu patří jednomu vrácenému tokenu. V případě mého ukázkového příkladu se cyklus bude opakovat třikrát. V cyklu se asi nejčastěji budeme dotazovat na čtyři atributy objektu *HTML parseru*. Popis atributů hodnot jež můžou nabývat jsem vyjádřil následující tabulkou. Ještě upozorním, že pojmenování objektů v tabulce vychází z ukázkového příkladu.

Atribut	**Hodnoty atributu**	**Popis**
`$parser->iNodeType`	**0** = start typ **1** = start tag elementu **2** = end tag elementu **3** = text **4** = komentář **5** = neznámé	Atribut vrací číselnou hodnotu, která reprezentuje typ vráceného tokenu.
`$parser->iNodeName`	Například **td, table, font, a, text** atd.	Vrací název tokenu.
`$parser->iNodeValue`	Vrací hodnotu tokenu. Uplatňuje se pouze při tokenu, klasifikovaném jako text. V tomto případě nese textovou informaci. Jinak je prázdný.	
`$parser->iNodeAttributes`	Například **red, 400px, display:none;** atd.	Vrací asociativní pole hodnot atributů.

Objekt *HTML pareseru* má atributů více, ale výše uvedené jsou ty nejpodstatnější. Na závěr kapitoly ještě uvedu, jaké hodnoty vrátí *HTML pareseru* v případě ukázkového zdrojového kódu.

Průchod	**Hodnoty atributu**

1.	`$parser->iNodeType == 1`
	`$parser->iNodeName == "font"`
	`$parser->iNodeAttributes["color"] == "red"`

2.	`$parser->iNodeType == 3`
	`$parser->iNodeName == "text"`
	`$parser->iNodeValue == " Tento text je červený."`

3.	`$parser->iNodeType == 2`
	`$parser->iNodeName == "font"`
	`$parser->iNodeValue == ""`

6 Filtrace dat z HTML stránek

Další velmi důležitou součástí je filtrování dat. Mějme HTML stránku, na které je tabulka s výsledky fotbalových zápasu. Tabulka má tři sloupce. V prvním sloupci je název domácího mužstva, ve druhém název hostujícího mužstva a třetí sloupec obsahuje výsledky utkání. Před tabulkou a za tabulkou se nacházejí další prvky, jako například odstavec s úvodem do problematiky, několik hypertextových odkazů na jiné stránky, obrázky a další. Nás, jako uživatele aplikace pro sledování informací na HTML stránkách, však zajímají pouze výsledky zápasu v tabulce. Je tedy nutné odfiltrovat přebytečné informace ze stránky a vrátit pouze požadovaná data ve strukturované podobě (tedy v tomto případě trojice domácí, hosté, výsledek). Následující podkapitoly se pokusí naznačit několik způsobů, jak toho dosáhnout. Protože je moje práce zaměřena na HTML stránky a jazyky pro tvorbu dynamického webu, budou následující kapitoly věnovány implementaci filtrů pomocí jazyka PHP (1.2).

6.1 Filtr pro konkrétní HTML stránku

Pokud víme, jakou HTML stránku budeme sledovat, známe její strukturu a bude-li se jednat o sledování dlouhodobějšího charakteru, jednou z možností je vytvořit (naprogramovat) filtr na míru pro danou HTML stránku. Takový filtr většinou bývá efektivnější a jednodušší než filtr univerzální a v jistých případech je to nejideálnější řešení. Na druhou stranu jeho implementace je specifická a je použitelný jen pro jeden typ HTML stránek.

Jako součást mé práce jsem takový filtr implementoval a jeho výsledky byli vynikající (samotné aplikaci, využívající tento druh filtru a jeho výsledkům je věnována samostatná kapitola). Pokusím se tedy nastínit jednu z mnoha možností, jak tento filtr implementovat. Předpokládejme, že máme k dispozici *HTML parser pro PHP*, kterému byla věnována část v minulé kapitole (4.2). Dalším krokem je dobře se seznámit se strukturou a zdrojovým textem sledované stránky.

Vycházejme tedy z příkladu s tabulkou výsledků fotbalových zápasů, který je uveden v úvodním odstavci této kapitoly. Chceme tedy dosáhnout efektu, který nám vrátí strukturované informace z tabulky a ostatní data bude ignorovat. Pro názorné vysvětlení použiji algoritmus napsaný v jednoduchém pseudokódu. Pro přehlednost jsem volil variantu textu s barevným rozlišením. Tento algoritmus vrací dvourozměrné pole, kde každý prvek pole odpovídá textovému obsahu jedné buňky tabulky. Kód počítá s označením tabulek ve zdrojovém textu pomocí atributu `id`. Prochází tedy kód, dokud nenarazí na tabulku označenou hledaným identifikátorem. Pokud by tagy na stránce nebyly označeny identifikátory, musíme zvolit jiný způsob. Například vím-li, že se jedná o jedinou tabulku na stránce, procházím stránku tak dlouho, dokud nenarazím na element tabulky. Tento způsob je však

nebezpečný, protože přidá-li tvůrce stránek před hledanou tabulku ještě další tabulku, budeme skenovat právě tu přidanou. Algoritmus prochází tabulku až do jejího konce a textové informace přiřazuje do dvourozměrného pole. Nepočítá tedy se zanořenou tabulkou. Předpokládejme, že pseudoparser funguje podobně jako *HTML parsec pro PHP*.

```
While ( (parser->dalsi)->id !=  id_tabulky ){};

While ( (parser->dalsi)->typ != konec_tabulky ){

        If( parser->typ == konec_bunky )
                        bunka++;

        If( parser->typ == konec_radku ){
                        radek++;
                        bunka = 0;
        }

        vystup [bunka][radek] += parser->text;
}
```

Tato filtrace je velmi efektivní. Podle algoritmu je možné implementovat filtr v mnoha jazycích. Já jsem se rozhodl pro implementaci v jazyku PHP.

Jednou z mnoha dalších možností, jak vytvořit takový filtr je využití GNU utilit. Pomocí programu *wget* (2.1) je možno extrahovat informace ze sítě. Program také umí sám hlídat změny a pokud se extrahovaná stránka od poslední extrakce nezměnila, *wget* ji nebude stahovat a zajistí tak ukončení skriptu. Poté, když máme data pro filtraci připraveny na lokálním paměťovém médiu, můžeme využít například programů *awk*, *grep* a *sed* pro filtraci požadovaných dat.

6.2 Univerzální filtr

Druhou možností je filtr univerzální. Myšlenka spočívá v tom, že uživatel si vybere vlastní HTML stránku, kterou chce sledovat. Nemusí znát strukturu kódu stránky, ani nemusí ovládat žádný programovací jazyk. Pouze nastaví vlastnosti filtru, a ten zajistí filtrování požadovaných dat. Pokud k tomuto problému přistoupíme z vyšší abstrakce, jsou takovými filtry samotné programy *awk*, *sed* nebo *grep*. Jejich nastavování však není zas až tak triviální, a nastudovat manuály k takovým programům by zabralo spoustu času. Zaměřím se tedy na filtry, které je možno nastavit přes přehledné uživatelské rozhraní.

Implementace takového filtru není narozdíl od filtru specializovaného pro konkrétní stránku triviální. Tyto filtry nedosahují takové rychlosti a efektivnosti a mají daleko větší paměťovou náročnost. Filtry, které budu dále popisovat, jsem implementoval a odzkoušel v jazyku PHP a jejich funkčnost a dosažené výsledky popisuji v kapitule zaměřené na samotnou aplikaci. Z hlediska implementačního

a uživatelského se může univerzální filtrování rozdělit na dvě fáze. V první fázi, je celá HTML stránka pomocí filtrů rozdělena na objekty, aby umožnila uživateli výběr těch, které bude na stránce sledovat. V druhé fázi, která probíhá při sledování změn, již známe zvolené objekty, a proto si můžeme dovolit optimalizace, jako přeskakování nepodstatných objektů a vynechávání informací, které na sledovanou informaci nemají vliv. V následujících podkapitolách se tedy pokusím popsat jednu z mnoha možností, jak tento filtr realizovat.

6.2.1 Identifikace objektů

Nejprve je nutné ujasnit si, co filtr bude na stránce rozeznávat. Zaměřil jsem se hlavně na textové informace. To znamená, že filtr umí hlídat obsahy elementu jako jsou například odstavec, font, tabulka, seznam, odkaz, nadpis, div a další. Dále je filtr schopen sledovat i atributy těchto elementů. To však není pro tuto podkapitolu podstatné.

Jak ale pojmenovávat a rozdělit obsah na objekty? Ideální řešení by bylo, kdyby byl každá element jednoznačně identifikován pomocí čísla nebo jiného klíče. To je sice možné, ale většina tvůrců stránek to nedělá, proto se na tento způsob nelze spoléhat. Je nutné každý prvek na stránce, který bude reprezentovat objekt, jednoznačně identifikovat. Jednou z možností je pojmenovávání objektů podle zanoření. Princip a výhody této metody se pokusím vysvětlit na následujícím kódu HTML.

```
<div>
        <font class="cerna"> Černá </font>
        <font class="cervena"> Červená </font>
</div>
<div>
        <font class="zluta"> Žlutá </font>
        <font class="modra"> Modrá </font>
</div>
```

Jak tedy funguje tato metoda. Ve zdrojovém textu je celkem deset objektů, které je možno sledovat. Prvním objektem je element prvního divu a vše, co obsahuje. Řekněme, že jej pojmenujeme **div1**. Tento objekt v sobě zapouzdřuje objekt elementu font, ten pojmenujeme **div1-font1** a objekt druhého fontu **div1-font2**. Objekt **div1-font1** v sobě zapouzdřuje objekt samotné textové informace. Ten pojmenujme **div1-font1-text1**. A objekt **div1-font2-text1**. Stejným způsobem se pojmenuje i dalších pět objektů v druhém elementu divu **div2**, **div2-font1**, **div2-font2**, **div2-font1-text1** a **div2-font2-text1**.

Tímto způsobem každý objekt, který je možno samostatně sledovat, opatříme unikátním identifikátorem. Další z možností by bylo jednoduše očíslovat výskyty různých elementu jako

například **div1**, **font1**, **text1**, **font2**, **text2**, **div2**, **font3**, **text3**, **font4** a **text4**. Tento způsob je implementačně jednodušší, ale skrývá v sobě celou řadu nevýhod. Jedna z nich se pokusím uvést. Řekněme, že uživatel se rozhodne sledovat změny textu ve druhém divu v elementu prvního fontu. Tedy text "Modrá". Pokud by jsme použili pojmenování objektu pomocí vnořování, je tento objekt pojmenován jako **div2-font1-text1**. Pokud použijeme jednoduchou variantu, má tento objekt identifikaci jako **text3**. Pokud se na stránce nic nezmění, fungují oba způsoby bezproblémově a první způsob se zdá být příliš komplikovaný. Pokud se však tvůrce sledovaného kódu rozhodne přidat do elementu prvního divu další element fontu s textovým obsahem (například "Fialová") pod druhý element fontu, nastane problém. Při použití jednoduchého způsobu pojmenování objektů, by text v tomto novém elementu nesl identifikaci **text3**. Což je námi zvolený objekt ke sledování. Ve skutečnosti to však není námi zvolený objekt, ten je identifikován jako **text4**. A při sledování se tedy chová, jako by byl text "Modrá" změněn na "Fialová". Ve skutečnosti tomu tak není. Pokud ale použijeme způsob identifikace založený na vnořování, nese tento objekt stále stejnou identifikaci **div2-font1-text1**.

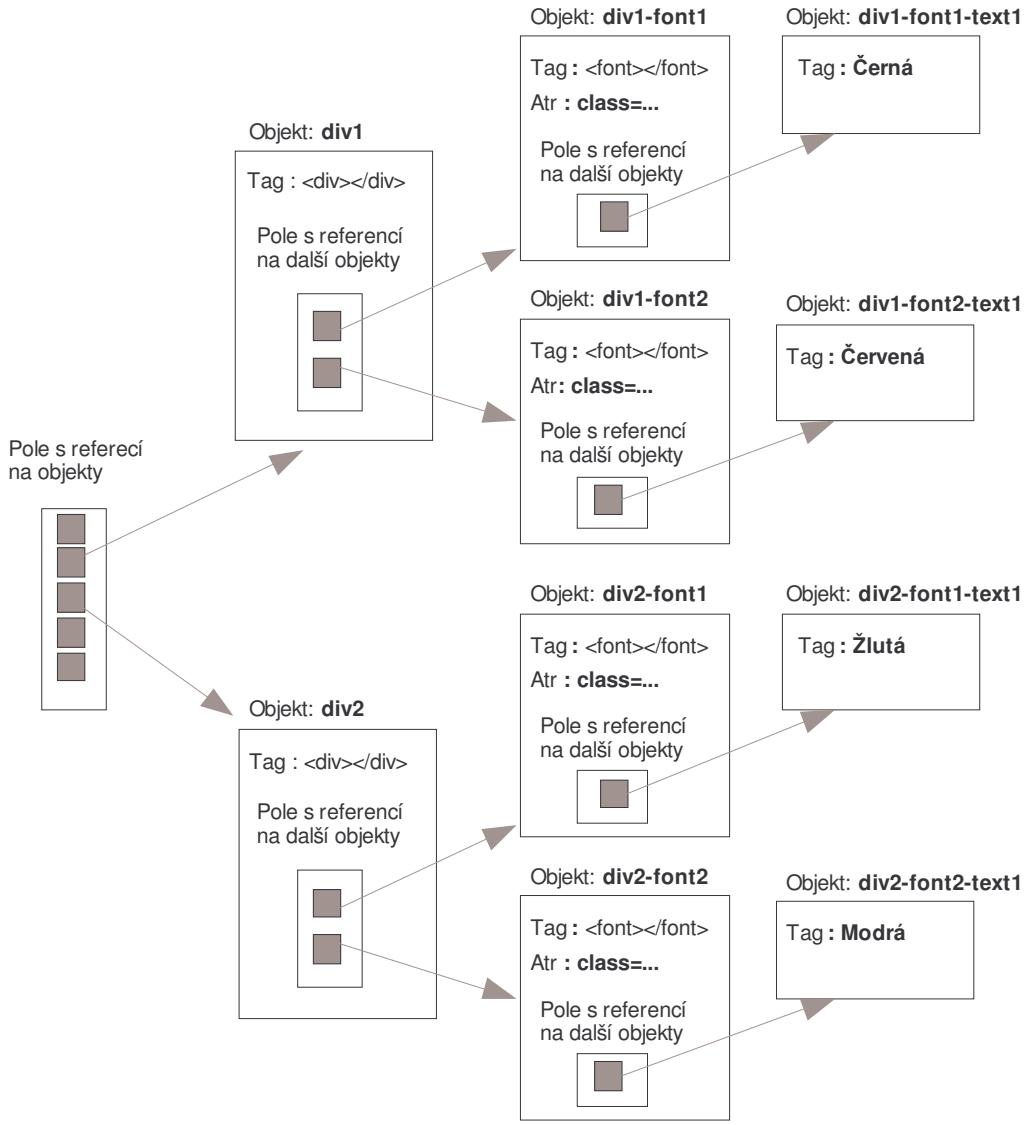
6.2.2 Uchovávání identifikovaných objektů

Jako způsob pro pojmenovávání objektů, jsem se rozhodl použít způsob rekurzivního zanoření. Dalším krokem je rozdělení obsahu zdrojového kódu stránky na objekty. Ty je nutné uchovávat v paměti a umožnit uživateli výběr objektů, které budou sledovány. Jako příklad opět využijeme zdrojový kód HTML z předchozí kapitoly (6.2.1). Řekněme, že uživatel chce sledovat obsah celého elementu prvního divu. Musíme tedy zajistit, aby při výběru objektu **div1** byly vybrány i objekty **div1-font1**, **div1-font2**, **div1-font1-text1** a **div1-font2-text1**.

6.2.2.1 Metoda na bázi vytváření objektů

V paměti musí být tedy připraven a rozdělen obsah stránky tak, aby byl výběr rychlý a jednoznačný. Jedním ze způsobů, jak data do paměti mapovat, je vytvářet skutečné objekty a pomocí referencí je pospojovat. Objekty by již obsahovaly všechny příslušné informace o elementech, včetně zdrojového kódu v HTML. Znázorněna je tato situace na následujícím obrázku (obrázek 6.2.2.1.1.). Obrázek ukazuje, jak by byl touto metodou rozdělen zdrojový HTML kód z předcházející kapitoly. Pokud by se tedy rozhodl uživatel sledovat element prvního divu, byl by vybrán objekt s identifikací **div1** a tím zároveň i všechny objekty, které obsahuje. Při porovnávání objektů pak můžeme využít rychlého binárního porovnávání celých úseků paměti nebo si vytvořit vlastní funkci, která bude procházet paralelně dva objekty a porovnávat textové informace. Tímto způsobem okamžitě víme, v jakém objektu došlo ke změně. Také se dá využít binárního ukládání dat

do databáze. Pro rozdělení na objekty a jejich naplnění se dá dobře využít syntaktická analýza na bázi rekurzivního sestupu. Při testování skriptu PHP, který vykonával právě toto rozdělení, jsem se velmi často setkal s nedostatkem paměti. Tato skutečnost byla zapříčiněna faktem, že rekurzivní sestup společně s rychle rostoucím stromem objektu brzy vyčerpal paměť, jež byla skriptu stanovena limity. Závěrem pro tento způsob filtrace je konstatování, že se příliš nehodí pro skripty PHP. Nicméně pokud zdrojový HTML kód není příliš obsáhlý nebo nejsme omezeni limity pro PHP skripty, funguje tato metoda bez problémů.



Obrázek 6.2.2.1.1. Objekty v paměti.

6.2.2.2 Metoda na bázi dvourozměrného pole

Metoda, která vytvářela stromovou strukturu objektů, příliš zatěžuje paměť a také je zbytečně složitá pro implementaci. Problémem může být také průchod stromové struktury objektů a extrakce dat pro porovnávání a ukládání do databáze. Druhá metoda, kterou jsem implementoval a vyzkoušel, dělila obsah podobně jako u stromové struktury. Tyto informace se však ukládají do dvourozměrného pole tak, jak jdou za sebou ve zdrojovém textu. Tato metoda vyžaduje, aby byly klasifikovány a identifikovány i koncové tagy elementů jako samostatné objekty (aby bylo možno určit kde daný element končí). Tuto metodu je vhodné využít v kombinaci se syntaktickou analýzou pomocí zásobníkového automatu (funguje ale i v kombinaci s metodou rekurzivního sestupu). Následující obrázek (obrázek 6.2.2.2.1.) pak ukazuje, jak vypadá v paměti dvourozměrné pole získané ze zdrojového textu uvedeném v předcházející podkapitole (6.2.1).

div1	** **	
div1-font1	****	**class=…**
div1-font1-text1	Černá	
e-div1-font1	****	
div1-font2	****	**class=…**
div1-font1-text2	Červená	
e-div1-font2	****	
e-div1	** **	
div2	** **	
div2-font1	****	**class=…**
div2-font1-text1	Žlutá	
e-div2-font1	****	
div2-font2	****	**class=…**
div2-font1-text2	Modrá	
e-div2-font2	****	
e-div2	** **	

Obrázek 6.2.2.2.1. Dvourozměrné pole v paměti

Pokud bude chtít uživatel sledovat element prvního divu, je z paměti použita jen ten úsek pole, který začíná identifikátorem **`div1`** a končí identifikátorem **`e-div1`**. Tato část je poté optimalizována pro uložení do databáze. Při dalším sledování je pak vždy ze stránky odfiltrován hledaný objekt, ostatní jsou ignorovány. Ze sledovaného objektu je vytvořeno opět pole a údaje v něm se porovnávají s údaji z minulého skenování. Atributy nejsou součástí tagů, protože někteří uživatelé chtějí porovnávat pouze textové informace a zařazení objektů.

7 Uchovávání dat

Sledovaná data z HTML stránky, stejně jako výsledky s porovnávání, je nutno uchovávat k dalšímu použití. Proto je tato kapitola věnována dlouhodobému uchovávání dat a manipulací s těmito daty. Aplikace vytvořené v rámci této práce jsou webového charakteru, proto se budu zaměřím na uchovávání dat na webu. Jednou z dnes nejpoužívanějších variant je databáze MySql. Tuto databázi využívám i v rámci svých aplikací, ale protože jsem této databázi již věnoval celou podkapitolu (1.4), nebudu ji zde znovu charakterizovat. Zaměřím se na další možnosti.

7.1 Uchovávaní dat pomocí XML

Kapitola o XML vychází z [22] a [23].

Další možností, jak dlouhodobě uchovávat data, je stále populárnější XML. Xtensible Markup Language je obecným značkovacím jazykem, umožňuje snadné vytváření konkrétnějších značkovacích jazyku. Bylo vyvinuto konsorciem W3C a tato specifikace je zdarma dispozici. Je vhodný především pro výměnu a uchovávání dat mezi dokumenty a jejich publikování ve snadno zpracovatelné a zároveň člověku přehledné formě. Jak název napovídá, jedná se o značkovací jazyk. Nemá však žádné předefinované značky. Tyto značky v dokumentu vyznačují význam jednotlivých částí textu. To je samozřejmě velkou výhodou při prohledávání dokumentu, kdy můžeme určit i jaký význam má mít hledaný text. Aktuální verzí XML je 1.1. Verze se od sebe liší v požadavcích na použité znaky, v názvech elementů, atributů atd. XML počítá s podporou mnoha jazyků, proto je implicitní znakovou sadou Unicode, lze však využít i další, například windows-1250, iso-8859-2 atd.

PHP poskytuje mnoho prostředků pro zpracování XML dokumentu. Mezi ně patří například modul XML_PARSER. Ten poskytuje objektově orientované rozhraní. V PHP 4 používá rozšíření externí knihovnou *expat*, PHP 5 zase knihovnu *libxml2*.

V zásadě rozlišujeme dva typy parserů. Již zmíněný XML_PARSER a rozšíření XML jsou ve stylu Simple API for XML (SAX). Tyto partery jsou řízené událostmi. XML dokument je procházen a pokud dojde k události, například nalezení uvozovacích značek, je vyvolána akce pro zpracování dané událost. Druhým typem jsou parsery založené na objektovém modelu dokumentu DOM. Mezi ně můžeme zařadit rozšíření domxml a smplexml PHP. Tyto parsery vybudují stromovou reprezentaci dokumentu s rodičovskými a dceřinými prvky. Poté je možno stromovou strukturu procházet a manipulovat s ní.

XML je moderní a zřejmě velmi výhodné řešení. Nehodí se však pro manipulaci s velkým množstvím dat. Vyhledávání v rozsáhlém XML dokumentu je náročné jak časově, tak paměťově. V tomto ohledu stále vítězí MySql databáze, nebo jiné komerční produkty. XML však může být vhodnou volbou pro jednoduché aplikace, kde není k dispozici výkonná databáze.

8 Rozesílání dat

V předcházejících kapitolách bylo naznačeno, jak je možné data z HTML stránek získávat, filtrovat porovnávat a uchovávat. Dalším krokem je zasílání informací o zjištěných změnách. Tato kapitola se tedy věnuje rozesílání dat uživatelům prostřednictvím elektronické pošty a pomocí krátkých textových zpráv SMS.

8.1 Rozesílání elektronické pošty

Kapitola o elektronické poště vychází z [23].
Jednou z možností jak uživatele informovat o průběhu sledování, je zasílání výsledků pomocí elektronické pošty. V jazyku PHP je několik prostředků jak rozesílat elektronickou poštu. Asi nejsnadnější řešení je použití vestavěné funkce *mail*. Pomocí této funkce však nemáme odesílání pošty plně pod kontrolou. Lepším řešením je použití modulů Mail a Mail_mine PEAR. Ve svých aplikacích jsem testoval moduly PEAR Mail 1.1.2 a Mail_mine 1.2.1 PEAR v PHP 5.

8.1.1 Modul Mail PEAR

Modul Mail PEAR se používá k odesílání dat čistě textového charakteru. Zároveň poskytuje kontrolu nad odesíláním dat. Pokud chceme odeslat zprávu, je nutné vytvořit nový objekt `Mail` metodou `Mail::factory()`. Poté zavoláme metodu `send()` a předáme ji adresu příjemce, tělo zprávy a další patřičná záhlaví. Následuje část zdrojového kódu, na které je ukázáno, jak je možno odeslat zprávu.

```
//vložíme zdrojové texty třídy Mail
require "Mail.php";
$mailer =& Mail::factory(´mail´) ;
$to ="komendatomas@seznam.cz";
$head = array(´Subject´ => "Pokus", ´Cc´ => "další příjemci");
$message ="tělo zprávy";
$res = $mailer->send($to, $head, $message);
//kontrola odeslání
if( PEAR::isError($res)){
 //ošetření chybné situace
}
```

První argument `Mail::factory()` určuje, jaký ovladač se použije pro odeslání zprávy. V ukázkovém případě je použit ovladač `mail`, který říká, že se má používat vestavěná funkce PHP

mail. PHP tedy spoléhá na konfigurační direktivy vztahující se k poště, tedy `sendmail_path` a na Windows `sendmail_from`. Tyto direktivy by měly mít nastaveny cesty k nějakému externímu poštovnímu programu. Systém Windows nedisponuje vestavěným poštovním programem, proto ve většině případech dává funkci *mail* pokyn k odeslání přes nějaký SMTP server. Tato varianta není tedy příliš výhodná, zdrojový kód je mnohem komplikovanější než při použití samotné funkce *mail* a výsledek je stejný. Dalšími ovladači jsou `smtp` a `sendmail`. Pokud tedy použijeme ovladač `smtp`, je zpráva odeslána pomocí SMTP serveru. Poslední ovladač je vhodné použít, pokud je na serveru přístupný *sendmail* nebo jiný obdobný program.

Nejvýhodnější je tedy použití ovladače `smtp`. Vyhneme se tak nastavováním direktiv. Při použití tohoto ovladače jsou však vyžadovány moduly `NET_SMTP` a `NET_Socket PEAR`. Tyto moduly nejsou standardně k dispozici a je nutné je doinstalovat. Při použití tohoto ovladače je nutné nastavovat i další argumenty pro `Mail::factory()`. Jejich přehled je shrnut v následující tabulce.

Volba	**Výchozí**	**Popis**
host	Localhost	Název hostitelského servru, k němuž se bude připojovat.
port	25	Port připojovaného SMTP serveru.
auth	FALSE	Použije li se použít autentizace SMTP.
username		Login pro autentizaci.
password		Heslo pro autentizaci.
localhost	Localhost	Jakým způsobem se má hostitel identifikovat SMTP serveru.

Posledním ovladačem je `sendmail` využívající poštovní program, který odešle zprávu. Při použití tohoto ovladače je nutné nastavit několik parametrů, které se pak předají příkazové řádce shellu. Dále ovladač požaduje, aby byla uvedena v parametrech hlavičky adresa odesilatele *From*. Nastavení ostatních parametrů demonstruje následující zdrojový kód.

```
$param = array(´sendmail_path´ => "user/lib/sendmail",
               ´sendmail_args´ => "-i");
```

První parametr předává umístění programu pro odesílání pošty. Druhý přepínač `-i` říká programu pro odesílání pošty, aby nepovažoval znak tečky ve zprávě za ukončení zprávy. Více podrobných informací včetně ukázkových zdrojových kódu poskytuje publikace Davida Sklara [23] v kapitole o Odesílání pošty.

8.1.2 Modul Mail_mime PEAR

Při posílání zpráv přes SMTP servery se očekává, že tělo zprávy bude tvořeno pouze znaky ASCII. Nejsou tedy povoleny žádná binární data. MIME však poskytuje způsob jak toto obejít. Standart mime určuje, jak reprezentovat jako čistý text ostatní dokumenty, které čistý text neobsahují (obrázky, PDF dokumenty atd.) a také jak zapouzdřit několik dokumentů do jedné zprávy. Hodí se tedy pro odesílání zpráv s přílohami. Tato práce se však nezabývá odesíláním příloh. Zajímavá je však jiná vlastnost modulu. Protokol o skenování dat na HTML stránkách má svoji vlastní grafickou reprezentaci. Ta je tvořena pomocí jazyka HTML. Modul Mail_mime umí generovat HTML zprávu. Zdrojový kód pro vygenerování takovéto zprávy by vypadal následovně.

```
//vložíme zdrojové texty třídy Mail
require "Mail/mime.php";
//přepokládejme, že v proměnné $message je klasická
//textová zpráva a v proměnné $message_HTML je zdrojový HTML kód
$mime = new Mail_mime();
//načtení textové zprávy do těla
$mime->setTXTBody($message);
//načtení HTML zprávy do těla
//obdobně lze využít načtení ze souboru
//$mime->setHTMLBody(´/user/document/zprava.html´, TRUE);
$mime->setHTMLBody($message_HTML);
//vygenerujeme tělo zprávy
$body = $mime->get();
```

Odeslání takovéto zprávy by pak vypadalo stejně, jako v předcházející podkapitole a využilo by se funkce *mail* nebo modulu Mail PEAR. Tento druh zprávy se řadí do skupiny `multipart/alternative`. Jedná se o zprávu se dvěma těly. Jak je patrné z ukázkového zdrojového kódu, jedno tělo zprávy je čistě textového charakteru a druhé se skládá z HTML elementů. O tom, jaké tělo zprávy bude ve výsledku zobrazeno, rozhodne až poštovní klient. Pokud se jedná pouze o textového klienta (Mutt, Pine), je zobrazena textová informace. Pokud je klient grafický (Outlook, Eudora), zobrazí tělo HTML. Více podrobných informací včetně ukázkových zdrojových kódu poskytuje publikace Davida Sklara [23] v kapitole o Odesílání pošty.

8.2 Odesílání SMS

Další možností jak informovat uživatele o sledovaných datech, je zasílání krátkých textových zpráv na mobilní telefon. Oproti elektronické poště toto řešení postrádá možnost zasílání dlouhých protokolů. Informace musí být pouze textového charakteru a velmi krátká. Další nevýhodou je, že tyto služby nejsou ve většině případů zdarma. Jednou z mála výhod tohoto systému je rychlost doručení. Pokud nosí uživatel mobilní telefon stále u sebe, má naprostý přehled o dění na sledovaných stránkách bez přístupu k počítači.

8.2.1 Odesílání SMS zdarma

Jednou z možností, jak odesílat SMS z webové stránky zdarma, je využití SMS brány. Toto řešení však neposkytují všichni operátoři a ti, kteří toto řešení poskytují, omezují své služby. Například společnost O2 omezuje zprávy pouze na šedesát znaků a společnost T-Mobile tuto službu nepovoluje vůbec. Toto řešení tedy značně omezuje některé uživatele a není pro internetové aplikace příliš vhodné.

Dalším řešením je využít takzvaného mobilního emailu. Tento způsob rozesílaní SMS pracuje na principu zasílání emailu příslušnému operátoru a ten je dále odešle konkrétnímu uživateli na mobilní telefon. Tento princip je dostupný u všech našich operátorů, ale s určitým omezením. Společnost O2 omezuje délku zprávy na šedesát znaků a nedovoluje zasílat více jak pět zpráv z jednoho emailu. Tvar emailové adresy, na kterou se zpráva zasílá je u každého operátora odlišný. Pro názornost uvedu adresu společnosti O2. Ta vypadá takto *00420xxxxxxxxx@sms.o2.cz*. Křížky jsou pak nahrazeny mobilním číslem konkrétního adresáta. Záměrně uvádím tvar adresy společnosti O2. Je to totiž jediná společnost, u které je jisté, že mobilní email existuje. Všichni zákazníci O2 totiž mají mobilní email automaticky vytvořený. To však neplatí u zákazníků ostatních operátorů, ti si jej musí aktivovat sami. Proto je tato služba nevhodná pro použití do internetových aplikací.

8.2.2 Odesílání SMS z soukromé SMS brány

Kapitola o odesílání placených SMS vychází z [24].

Zasílaní SMS zdarma je značně omezené a komplikuje tak dostupnost kvalitních služeb internetových aplikací. Řešením je využití placených SMS brán. Toto řešení jsem využil ve své aplikaci pro sledování HTML stránek. Používám variantu odesílání SMS přes rozhraní serveru SMS Midlet. Službu lze nastavovat přes webové rozhraní na adrese *http://smsmidlet.com*. Pro zpřístupnění služby je nutné nabití kreditu. SMS pak lze rozesílat v libovolném množství bez omezení na většinu evropských operátorů. Komunikaci skriptu a serveru SMS Midlet je možné uskutečnit přes HTTP rozhraní. Pro předávání dat je pak využito metody GET. Druhou variantou je otevřít socket přímo na

serveru *smsmidlet.com*. A zapisovat přímo do tohoto socketu. Tato varianta je vhodnější při použití skriptů které nespouští přímo prohlížeč (ale například démon Cron). Na následujícím zdrojovém textu v jazyku PHP se pokusím demonstrovat princip odesílání SMS.

```
//číslo příjemce v mezinárodním formátu
$n = "420606431679";
$u = "login";
$p = "heslo";
//obsah zprávy musíme rozdělit do sekcí po 160 znacích a posílat je
//postupně
$m = "Obsah zprávy";
//typ zasílané zprávy SMS,MMS
$st = 1;
$ps  = "Heslo na SMS Midlet";
//zašifruji heslo, mobilní číslo, uživatelské jméno
$kod = md5($ps.$n.$u);
//sestavím požadavky pro odeslání pomocí metody GET
$poz = "username=".$u."&password=".$p."&body=".$m."&number=".$n."
        &smstype=".$st."&hash=".$kod."&show_SMSID=1";
//vložení požadavků do kontextu HTTP pro zápis do socketu
$zap = "GET /post2/?".$poz." HTTP/1.0\r\nConnection: Close\r\nHost:
         smsmidlet.com\r\nUser-Agent: Mozilla/4.72 [en]
         (Win98;I)\r\n\r\n";
//otevřu socket na portu 80
$fp = fsockopen("smsmidlet.com", 80, $errno, $errstr, 30);
if(!$fp){ //socket se neotevřel
}
else {
      //zapiši informace do socketu
      fputs($fp, $zap);
      //zavřu socket
      fclose($fp);
}
```

Server SMS Midlet umožňuje zasílání několika SMS najednou. Zároveň je také možno číst ze socketu stav informací o odeslání SMS a o aktuálním stavu kreditu. Pro přehlednost jsem však tyto možnosti v ukázkovém zdrojovém kódu neuváděl. Více informací je možno najít na stránkách společnosti SMS Midlet [24].

9 Aplikace pro sledování HTML stránek se sportovními výsledky

V rámci této bakalářské práce jsem implementoval dvě aplikace pro sledování dat na HTML stránkách. První z aplikací je určena pro dvě konkrétní internetové stránky. Aplikace je napsána v jazyku PHP a je umístěna na webovém serveru. Je přístupná přes webové rozhraní na adrese *http://watch-dog.wz.cz*. Uživatel, který chce služeb aplikace využívat může přistoupit za pomocí všech moderních internetových prohlížečů na výše uvedenou adresu a přidat své uživatelské jméno a mobilní telefonní číslo do databáze uživatelů. Při změně obsahu sledovaných stránek je pak informován. Zasílání SMS zpráv je založeno na použití mobilního emailu (8.2.1), proto je funkční pouze pro zákazníky firmy O2. Uživatelé využívající služeb jiných operátorů se musí spokojit pouze s informativními emaily.

Aplikace sleduje výsledky zápasů malé kopané okresu Blansko. Sleduje několik tabulek, které informují o odehraných zápasech a také o statistických výsledcích všech mužstev. Skenování je prováděno vždy v jedenáct hodin večer. Skript provádějící skenování využívá třídy *HTML parseru pro PHP* (5.2). Filtr je napsán speciálně pro tyto stránky (6.1), skenování je tedy efektivní a rychlé. Výsledky jsou ukládány do databáze MySql (1.4). Skript je aktivován pomocí služby *WebCron* (3.2). Aplikace vždy generuje protokol o skenování. Tento protokol je pak uložen na serveru jako samostatná webová stránka. Pokud skript odhalí odlišnosti od posledního sledování, ukládá tyto změněná data do databáze. Zároveň rozešle informativní emaily a SMS zprávy s odkazem na WWW stránku protokolu.

Volba tohoto způsobu implementace je velmi výhodná. Uživatel, jenž chce využít služeb, nemusí shánět a instalovat tento program na vlastní počítač. Odpadá tak nutnost překladu na různých architekturách a operačních systémech. Služba je dostupná ze všech počítačů mající přístup do sítě internet, musí však být vybavené jedním z moderních prohlížečů. Volba implementačního jazyka byla založena na myšlence, že celá práce se zabývá webovými aplikacemi, tudíž i jazyk implementované aplikace by měl být z této oblasti. ASP.NET je z komerční sféry, proto jsem se rozhodl pro volně použitelné PHP.

Aplikace je již několik měsíců v provozu a funguje spolehlivě. Nevýhodou jsou však časté výpadky služby *WebCron.* Skript pak není spouštěn v přesně naplánovaný čas nebo není v daný den spuštěn vůbec. Další nevýhodou jsou anti-spamové filtry, které nedovolují rozesílat emaily velkému množství uživatelů. Tyto problémy je však možno po domluvě s poskytovatelem hostigu nebo umístění aplikace na vlastní server vybavený démonem *cron* zcela odstranit. Protože je skript rychlý a efektivní, nenastávají problémy s paměťovými limity, ani s timeoutem.

10 Aplikace pro sledování textových informací z HTML stránek

Druhou aplikací vytvořenou v rámci této práce je program sledující textové obsahy libovolně zvolených HTML stránek. Stejně jako v předcházejícím případě se jedná o webovou aplikaci. Aplikace je opatřena univerzálním filtrem pracující na principech popsaných v jedné z předcházejících kapitol (6.2). Filtr je implementován v jazyku PHP za pomocí *HTML parser pro PHP* (5.2). Data jsou uchovávány v databázi MySql. Pro rozesílání emailů se využívá pokročilých možností elektronické pošty Mail_mine PEAR (8.1) v PHP. Krátké textové zprávy SMS jsou zasílány prostřednictvím předplacené služby SMS Midlet (8.2.2). Důvody volby nástrojů pro tvorbu této aplikace jsou stejné jako v předcházejícím případě (9).

10.1 Popis funkce aplikace

Hlavním úkolem aplikace je sledování textových informací na libovolných HTML stránkách. Aplikace umí rozlišovat objekty tabulek, seznamů, divů, nadpisů, odkazů, spanů, odstavců, fontů a samotných textů. Je možno si libovolně zvolit část nebo části dokumentu, které budou sledovány. Sledována pak bude i samotná struktura zvolené části dokumentu. Libovolně je možné sledovat i atributy elementů.

Uživatel zadá adresu sledované stránky. Aplikace pak provede předběžnou syntaktickou analýzu za pomocí zásobníkového automatu (5.1.1) a sama přednastaví parametry skenování. Toto doporučení však může uživatel ignorovat a zvolit si parametry podle vlastního uvážení. Dále je možno nastavit i další volby (rozlišování velikosti písmen, ignorace bílých znaků, převod na HTML entity, ignorace diakritiky atd.). Filtr je implementován ve dvou verzích. První využívá zásobníkový automat (5.1.1) a je standardně přednastavena. Druhá varianta funguje za pomocí rekurzivního sestupu (5.1.2). Druhou variantu je možné také zvolit, ale nedoporučuji to pro velkou paměťovou náročnost metody, která velmi snadno překročí limity stanovené na serveru.

Filtr naparsuje a převede na objekty všechny zvolené elementy na dané stránce (6.2). Za pomoci přehledného grafického rozhraní umožní zvolit objekt nebo objekty, které budou dále předmětem skenování. Dynamické chování výběru je vytvořeno za pomocí DOM modelu JavaScriptu (1.3). Poté co jsou vybrány objekty, je jejich obsah optimalizován a uložen do databáze.

V uživatelském menu je dále možno nastavit další vlastnosti úlohy (aktivita, interval spouštění, seznamy SMS čísel, seznamy emailů, zasílat nebo nezasílat SMS, porovnávat tagy, ukládání zjištěných změn atd.). Dále je možno nastavit vlastní šablonu pro vzhled generovaného protokolu pomocí integrovaného HTML editor. Tomu se věnuji v další podkapitole (10.3). Tato šablona je pak

použita při generování protokolu. V menu je také možno najít historii všech skenování i seznam všech protokolů ke konkrétní úloze.

Samotné skenování je pak spouštěno pomocí démona *cron*. K filtrování je pak vždy použit filtr využívající zásobníkový automat. Tento filtr je však již optimalizován. Neskenuje všechny objekty. Na stránce sice projde a pojmenuje všechny elementy, ale do paměti načítá pouze ty, které má uložené v seznamu skenování. Porovná je s daty uloženými v databázi a generuje příslušné emailové a SMS protokoly. Ty pak rozesílá zaregistrovaným uživatelům (8.1, 8.2.2). Protokoly ukládá do databáze pro případné pozdější použití. Uživatel má však možnost zakázat tvorbu historie nebo uchovávat jen protokoly, ve kterých je záznam o změně. Podrobný manuál k aplikaci, a popis testu najdete v přílohách k této práci. V přílohách je také umístěn ER diagram databázových tabulek této aplikace.

Aplikace je nachystána i na multiuživatelský režim. Každý se může zaregistrovat přes webové rozhraní aplikace. Aplikace mu pak vytvoří účet a zašle mu heslo na email. Každý uživatel má přidělen kredit pro zasílání SMS. Superuživatel *root* pak má po přihlášení kontrolu nad všemi účty i nad všemi úlohami. Má právo omezovat uživatele a měnit hodnoty jejich kreditu. Každý uživatel může z menu měnit své údaje. Pro dobytí kreditu pak musí informovat superuživatele prostřednictvím připravené volby v menu. Multiuživatelský režim však není v ukázkové aplikaci zpřístupněn. Lépe řečeno povolen je, ale ostatní uživatelé nemají právo zadávat úlohy. Je to z důvodu vytíženosti webového serveru, při provádění mnoha skenování ve stanovenou dobu. Významnou roli hrají nastavené časové a paměťové limity serveru, které by mohli skenování předčasně ukončit. Možné řešení tohoto problému jsem popsal v následující podkapitole (10.2). Další možností je zrušení limitů pro skript, ve kterém se provádí skenování. Toho lze dosáhnout přesunutím aplikace na vlastní server, nebo individuální domluvou s poskytovatelem serveru.

Aplikaci jsem odzkoušel na zadaných webových stránkách a fungovala bez problémů. Skenování stránek je rychlé a pro jednu úlohu jsem se nesetkal s případem, že by byly překročeny stanovené limity (výjimkou byla situace, kdy byla použita filtrace objektů pomocí metody rekurzivního sestupu, zdrojový kód měl přes 10000 řádků a bylo na něm rozlišováno přes 3500 objektů). Data prochází před uložením do databáze optimalizačním filtrem, který optimalizuje text, tak aby se dal uložit do jednoho záznamu a nezabíral mnoho místa v paměti databáze. Manuál a konkrétní popis testu je uložen jako příloha k této práci.

Vývojem této aplikace do budoucna by mohlo být zpřístupnění multiuživatelského režimu a dokonalejší optimalizace filtrování.

10.2 Paralelní zpracování v PHP

Kapitola o paralelním zpracování vychází z [24].

Velmi často je potřeba provést pomocí PHP skriptu nějaký náročný výpočet, či akci, která trvá dlouhou dobu. Problémem však je, že na většině PHP serverů jsou nastavené limity (časový timeout, který po určité době ukončí činnost skriptu, nebo limit maximální velikosti využitelné paměti). Tyto omezení mohou komplikovat dokončení skriptu. Je možné se individuálně dohodnout se správci serveru na odstranění limitů. Ve většině případů však neúspěšně. Osobně jsem se na tuto možnost dotazoval u třech poskytovatelů webhostingu a pouze jeden z nich souhlasil, a to za podmínek příplatků.

Další možností je ošálit server tak, že rozdělíme výpočet do několika skriptů a voláme je zapojené v do série. Každý skript má pak nastaven vlastní limit a timeout. Mějme však situaci, kdy jeden skript představuje jeden z pěti kompletních výpočtů nezávislých na sobě. Pokud by se tyto výpočty prováděly zapojené do série a v jednom z nich nastane chyba, jenž ukončí skript, další výpočty se již neprovedou. Ideálnějším a rychlejším řešením by bylo tyto výpočty provádět paralelně. Toho je možné dosáhnout několika způsoby. Já se zaměřím na způsob vytváření nových procesů pomocí funkce `pcntl_fork()` přístupné od PHP 5 díky rozšíření PCNTL (ve Windows CygPHP). Tato funkce vytvoří nový proces. Toto rozšíření je bohužel k dispozici pouze v prostředích CGI a CLI, takže se uplatňuje hlavně při spouštění skriptů z příkazové řádky. Na následujícím kódu se pokusím demonstrovat použití této funkce.

```
for($i = 0; $i < 5; $i++){
      //vytvořím nový proces
      $pid = pcntl_fork();
      if($pid == -1){
            //chyba, nelze forknout
      }
      if(!$pid){
            //kód potomka
      }
}//konec cyklu for
while(pcntl_wait($stat)){
            //čekáme na dokončení synovských procesů, aby nevznikali
            //zombie procesy
}
```

10.3 HTML editor

Proto, aby bylo možné k protokolům o skenování přidávat vlastní poznámky, obrázky a celkově upravovat výsledný vzhled dokumentu, jsem zařadil do aplikace vlastní HTML editor. Ten je založen na jazyku JavaScript (1.3). Je tedy přístupný přes webové rozhraní. Umožňuje tvorbu a úpravu vlastních šablon. Vstupem a výstupem je HTML kód dané šablony. Je tedy zřejmé, že editor musí obsahovat vlastní lexikální a syntaktický analyzátor. Editor tedy také umí získávat a filtrovat potřebná data z HTML kódu pro další použití, proto bezprostředně souvisí s tématem bakalářské práce. Lexikální a syntaktický analyzátor je implementován v jazyku JavaScript a funguje přesně tak, jak je popsáno v kapitolách o lexikální (4.1 obrázek 4.1.1) a syntaktické (5.1 obrázek 5.1.1) analýze. Ovládání editoru je založeno na objektech, jež představují jednotlivé elementy a části zdrojového kódu. Editor umí zobrazovat části dokumentu rozdělené a přístupné jako samostatné celky. Umožňuje měnit jejich obsah, přesouvání, vkládání nových objektů a mazání. Dále zobrazuje barevně rozlišený HTML kód výsledné šablony. Ovládání editoru je rychlé a přehledné.

Pomocí DOM modelu JavaScriptu mohu výsledky reprezentovat okamžitě prostřednictvím jazyka HTML. Hlavní nevýhodou JavaScriptu je absence práce se soubory. To komplikuje předávání dat do databáze. Dále bylo nutné předat například fotografie, které budou využity v šabloně, serveru. K těmto účelu jsem se rozhodl využít jazyk PHP (1.2). Existuje několik možností, všechny však mají své omezení. Cookies je pro tento účel nevhodné, protože nastává problém s předání příliš dlouhých zpráv a s předáním některých znaků. Další metodou je GET, princip spočívá v předáním pomocí URL, to je však omezeno svou délkou a opět nastane problém pří předání některých znaků (tento problém je globální nevýhodou značkovacích jazyků). Další možnost, kterou jsem v editoru využil, je metoda POST. Pomocí DOM modelu v JavaScriptu se mohu dostat k hodnotě formulářových proměnných a ty předat pomocí PHP.

Editor pracuje velmi dobře a je možné ho využít jako samostatný celek v jiných webových aplikacích (například pro tvorbu článků nebo celých stránek). Díky své implementaci a vlastnostem se hodí jako součást aplikace pro sledování dat HTML stránek.

Závěr

V dnešní době, kdy je internet největší skladiště informací, je nutné orientovat se v nástrojích pro tvorbu dynamických HTML stránek. Při tvorbě této práce jsem se s těmito nástroji velmi podrobně seznámil a využil jsem je k implementaci vlastních aplikací. Prostudoval jsem možnosti lexikální a syntaktické analýzy značkovacích jazyků. Tento přínos mohu v budoucnu velmi dobře využít při tvorbě podobných aplikací a hlavně při tvorbě samotných HTML stránek. Implementace filtru dat z HTML kódu mě přiměla ke studiu objektově orientovaného programování v jazyku PHP a JavaScript. A nejen to, donutila mě k mnoha úvahám o možnostech členění HTML kódu a o tom, jak jej strukturovat. Při tvorbě aplikací jsem využíval JavaScript a jeho DOM model. Tato skutečnost byla velmi přínosná a umožnila tvorbu velmi flexibilní a dynamické aplikace. Tyto poznatky jsem také využil při tvorbě HTML editoru a jistě je budu využívat i do budoucna. Seznámil jsem se také s mnoha užitečnými GNU utilitami (*cron, wget* atd.), o jejichž existenci jsem neměl před zahájením prací ani tušení.

Při tvorbě samotné aplikací jsem nastudoval konfiguraci PHP serveru Apach a MySql serveru. Také jsem se zajímal o konfiguraci těchto serverů u poskytovatelů webhostingu a tudíž jsem se seznámil s možnostmi výběru nejvýhodnějšího poskytovatele pro umístění budoucích webových aplikací.

Seznámil jsem se také s pokročilými možnostmi rozesílání elektronické pošty a prostudoval možnosti zasílání SMS zpráv. Toto studium bylo velmi přínosné, protože bez rozesílání zpráv se dnes neobejde téměř žádná vyspělá webová aplikace. Pro tvorbu dalších aplikací toto téma bude jistě přínosem.

Samotné aplikace pak fungují tak, jak požadovalo zadání a disponují mnoha rozšířeními. Protože je celá práce založena na webových aplikacích, rozhodl jsem se implementovat aplikace za pomocí HTML, PHP, JavaScript, MySQL. Díky tomu jsem se velmi podrobně seznámil s těmito nástroji. Tato volba však přinesla i jiné výhody. Aplikace jsou umístěny na veřejně přístupném serveru a jsou přístupné přes webové rozhraní. Odpadá tedy nutnost instalací a překladu na různých architekturách a operačních systémech. Další výhodou je, že zatěžuje pouze jeden server a uživatel tak nemusí spouštět vlastní počítač vždy, kdy má být provedeno skenování. Nevýhodou je však nadměrné zatěžování webového serveru a uplatnění nastavených limitů. Limity jsou však individuální záležitostí každého serveru, a proto lze tento problém v případě potřeby vyřešit. Na chod aplikací, který je specifikován zadáním, limity nemají žádný vliv. Aplikace je možné dále rozvíjet. Například rozšířit je o multiuživatelský režim, který je v aplikacích již připraven. Další možností vývoje je zdokonalování filtrů.

Tuto bakalářskou práci jsem si vybral, protože se zajímám o webové aplikace a do budoucna se jim chci dále věnovat. Proto považuji za velký přínos vše, co jsem v rámci této práce nastudoval a implementoval.

Literatura

[1] Boška, M. *Historie HTML a WWW prohlížečů* [online]. [cit. 2007-04.05]. Dostupné na URL: < http://html-history.wz.cz >.

[2] *HTML - Wikipedie, otevřená encyklopedie* [online]. [cit. 2007-04.05]. Dostupné na URL: < http://cs.wikipedia.org/wiki/HTML >.

[3] *Trocha historie HTML* [online]. Poslední aktualizace 2006-03-31. [cit. 2007-04.05]. Dostupné na URL: < http://edys.blog.cz/rubriky/html >.

[4] *Manuál PHP:Historie projektů souvisejících s PHP* [online]. [cit. 2007-04.06]. Dostupné na URL: < http://www.cs.vsb.cz/amalka/doc/php/czech/history.html >.

[5] Zajíc, P. *Historie a Budoucnost.* V *PHP* [online]. Poslední aktualizace 2004-05-27. [cit. 2007-04.06].Dostupné na URL: < http://www.linuxsoft.cz/article.php?id_article=171 >.

[6] *PHP - Wikipedie, otevřená encyklopedie* [online]. [cit. 2007-04.06]. Dostupné na URL: < http://cs.wikipedia.org/wiki/php >.

[7] *JavaScript - Wikipedie, otevřená encyklopedie* [online]. [cit. 2007-04.06]. Dostupné na URL: < http://cs.wikipedia.org/wiki/Javascript >.

[8] Hruška, T. Burget, R. *Internetové aplikace(WAP) IV. část JavaScript a PHP.* Vysoké učení technické v Brně, Fakulta informačních technologií, Božetěchova 2, 612 66 Brno. © 2006.

[9] *MySQL - Wikipedie, otevřená encyklopedie* [online]. [cit. 2007-04.06]. Dostupné na URL: < http://encyklopedie.seznam.cz/heslo/482131-mysql >.

[10] *Wget - Wikipedie, otevřená encyklopedie* [online]. [cit. 2007-04.10]. Dostupné na URL: < http://wikipedia.infostar.cz/w/wg/wget.html >.

[11] Cetoraz, P. *Wget – Stahujeme kvalitně* [online]. Poslední aktualizace 2004-09-24. [cit. 2007-04.11]. Dostupné na URL: < http://www.cetoraz.info/pavel/wget.html >.

[12] *Fopen (PHP4, PHP5)* [online]. Poslední aktualizace 2007-04-15. [cit. 2007-04.16]. Dostupné na URL: < http://cz2.php.net/manual/cs/function.fopen.php >.

[13] Vícha , K. *PHP prakticky (1): funkce fopen* [online]. Poslední aktualizace 2005-12-25. [cit. 2007-04.16]. Dostupné na URL: < http://www.pc-politika.com/phprs/view.php >.

[14] Kocman , J. *Jak na démona Cron* [online]. Poslední aktualizace 2002-04-21. [cit. 2007-04.16]. Dostupné na URL: < http://interval.cz/clanky/jak-na-demona-cron >.

[15] Frank , T. *Cron - mohutný plánovač úloh* [online]. Poslední aktualizace 2001-06-23. [cit. 2007-04.16]. Dostupné na URL: < http://linux.juristic.cz/79129/clanek >.

[16] Hruška, T. *Značkovací jazyky –markup languages 2.díl.* Vysoké učení technické v Brně, Fakulta informačních technologií, Božetěchova 2, 612 66 Brno. © 2000 .

[17] Meduna, A., Lukáš, R. *Lexikální analýza.* Vysoké učení technické v Brně, Fakulta informačních technologií, Božetěchova 2, 612 66 Brno. © 2006.

[18] Meduna, A., Lukáš, R. *Speciální typy konečných automatů.* Vysoké učení technické v Brně, Fakulta informačních technologií, Božetěchova 2, 612 66 Brno. © 2006.

[19] Meduna, A., Lukáš, R. *Úvod do překladačů.* Vysoké učení technické v Brně, Fakulta informačních technologií, Božetěchova 2, 612 66 Brno. © 2006.

[20] Meduna, A., Lukáš, R. *Syntaktická analýza shora dolů.* Vysoké učení technické v Brně, Fakulta informačních technologií, Božetěchova 2, 612 66 Brno. © 2006.

[21] *HTML parser pro PHP-4* [online]. [cit. 2007-04.17]. Dostupné na URL: < http://php-html.sourceforge.net >.

[22] *XML - Wikipedie, otevřená encyklopedie* [online]. [cit. 2007-04.19]. Dostupné na URL: < http://cs.wikipedia.org/wiki/XML >.

[23] David, S. *PHP5 moduly, rozšíření a akcelerátory.* 1. vydání. Zoner Press, Brno, 2005. ISBN 80-86815-19-6.

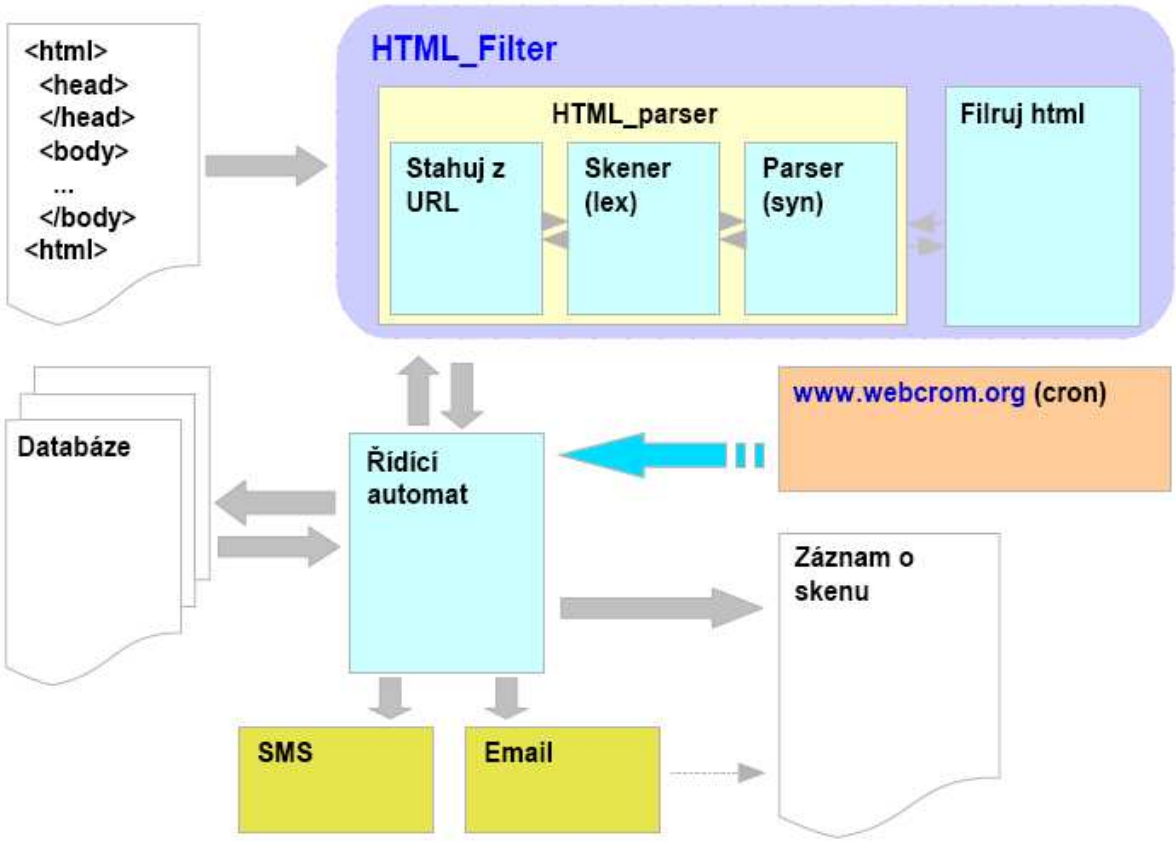
[24] Bárčík, F. *Co je služba SMS Midlet* [online]. [cit. 2007-04.19]. Dostupné na URL: < http://smsmidlet.com/sms_agent_lite.html >.

[25] Vrána, J. *Paralelní zpracování* [online]. Poslední aktualizace 2005-05-23. [cit. 2007-04.23]. Dostupné na URL: < http://www.root.cz/clanky/php-paralelni-zpracovani >.

Seznam příloh

Příloha 1. Dodatek ke kapitole 9

V aplikaci pro sledování sportovních výsledků není potřeba nic nastavovat. Stačí pouze přidat emailovou adresu, popřípadě mobilní telefonní číslo (podmínkou je, že číslo patří společnosti O2). Dále je možno přímo zobrazit protokol posledního skenování, data uložená v databázi a podrobnou nápovědu. Webové rozhraní aplikace je přístupné na *http://watch-dog.wz.cz*. Aplikace sleduje dvě HTML stránky s výsledky sportovních utkání. Stránky jsou dostupné na adrese *http://www.mkblansko.euweb.cz/mk%202006/tabulky4.htm* a tabulka výsledků jednotlivých zápasu na *http://www.mkblansko.euweb.cz/mk%202006/vysledky4.htm*. Pro představu o struktuře aplikace uvádím blokové schéma aplikace.



Obrázek Příloha.1.1. Blokové schéma aplikace.

Příloha 2. Manuál pro aplikaci s univerzálním filtrem

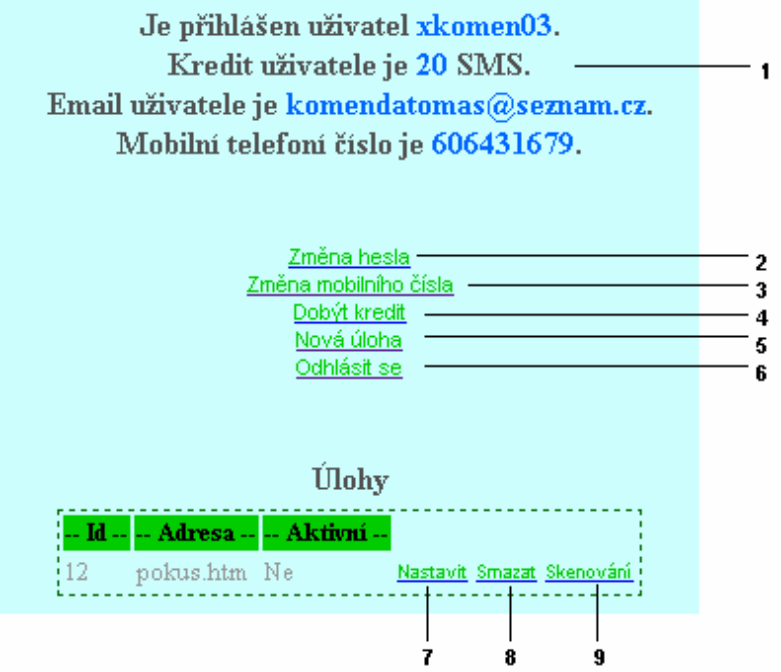
Aplikace je dostupná přes webové rozhraní jenž, naleznete na adrese *http://www.grantservice.eu/bp/index.php*. V následujících částech se pokusím popsat ovládání aplikace. Hlavním úkolem aplikace je sledování textových informací na libovolných HTML stránkách. Aplikace umí rozlišovat objekty tabulek, seznamů, divů, nadpisů, odkazů, spanů, odstavců, fontů a samotných textů. Je možno si libovolně zvolit část nebo části dokumentu, které budou sledovány. Sledována pak bude i samotná struktura zvolené části dokumentu. Libovolně je možné sledovat i atributy elementů.

Registrace nového uživatele

Při vstupu do rozhraní aplikace je po uživateli požadováno uživatelské jméno (login) a heslo (pass-word). Pro testovaní aplikace jsou vytvořeny dva účty. První účet je přístupný pod uživatelským jménem (login) *user* a heslem (pass-word) *user*. Uživatel má právo vytvořit jednu úlohu. Druhý účet je přístupný pod jménem (login) *root* a heslem (pass-word) *7M2yZtTs*. Tento účet patří superuživateli a je možné přes něj přistupovat a měnit údaje ostatních uživatelů. Nový účet je možné vytvořit z úvodní stránky přes odkaz *Nový účet*. Objeví se tabulka pro zadání požadovaných údajů. Systém pak sám vygeneruje uživateli heslo a zašle mu je na zadanou emailovou adresu. Pokud uživatel již má vytvořen účet a zapomněl heslo, může vyvolat jeho opětovné zaslání na email přes odkaz *Zapomněl jsem heslo.* Po přihlášení se uživatel dostane do centrálního menu.

Centrální menu

Přes centrální menu je možné přistupovat k jednotlivým úlohám a k nastavování údajů uživatele.



1. Zobrazuje informace o uživateli.
2. Spustí formulář pro změnu hesla.
3. Spustí formulář pro změnu telefonního čísla.
4. Zašle superuživateli zprávu s žádostí o navýšení kreditu pro odesílání SMS.
5. Spustí zadávání nové úlohy.
6. Odhlásí uživatele.
7. Spustí formulář pro nastavení konkrétní úlohy.
8. Smaže konkrétní úlohu.
9. Zobrazí stránku s historií skenování dané úlohy. Dále je zde možno spustit okamžité skenování dané úlohy.

Nová úloha

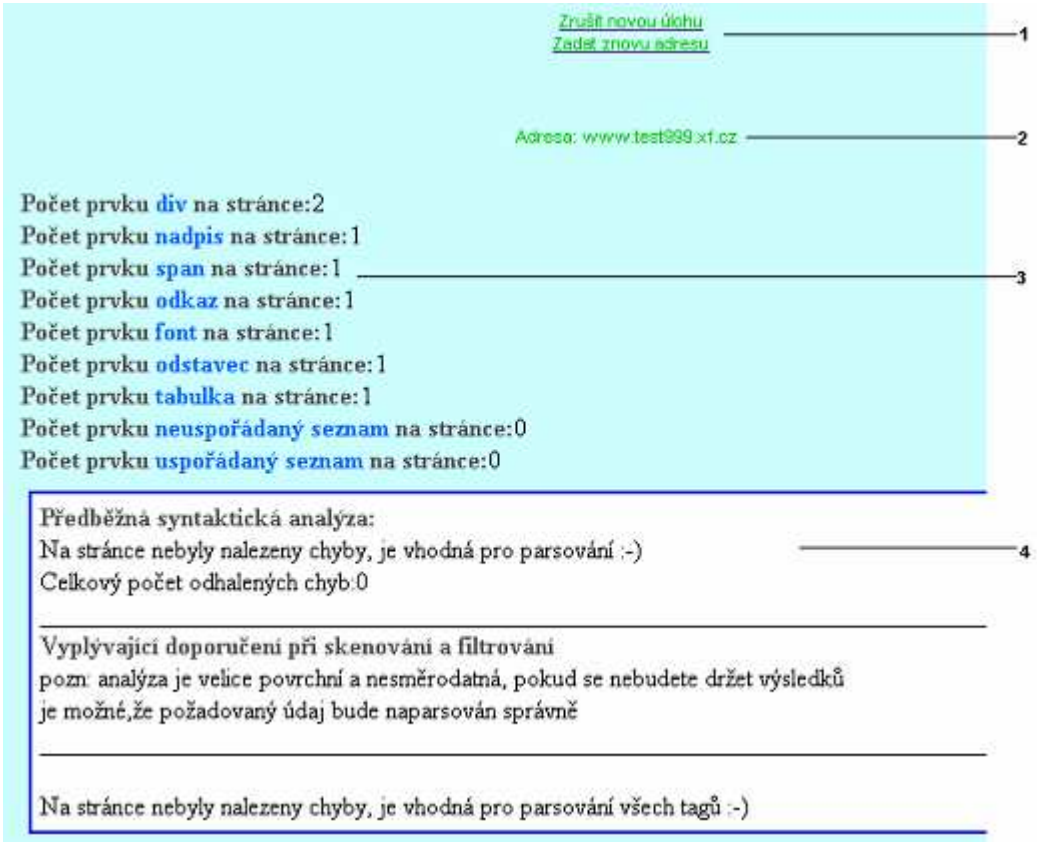
Novou úlohu je možno zadat z centrálního menu po kliknutí na odkaz *Nová úloha*. Pro testovací verzi jsem povolil pouze jednu úlohu, aby nebyl zbytečně vytěžován webový server. V následujících krocích popíšu zadání nové úlohy.

1. **Zadání adresy**

   Do kolonky pro adresu se zadá adresa skenované stránky. Chceme-li sledovat stránku na adrese *http://www.test999.xf.cz* zadáme do pole adresa pouze *www.test999.xf.cz*. Pro ověření správnosti můžete stisknout tlačítko s popiskem *náhled zadané stránky* a ve spodní části se vám objeví náhled zadané stránky. Pro pokračování v zadávání stiskněte tlačítko s popiskem *odeslat*.

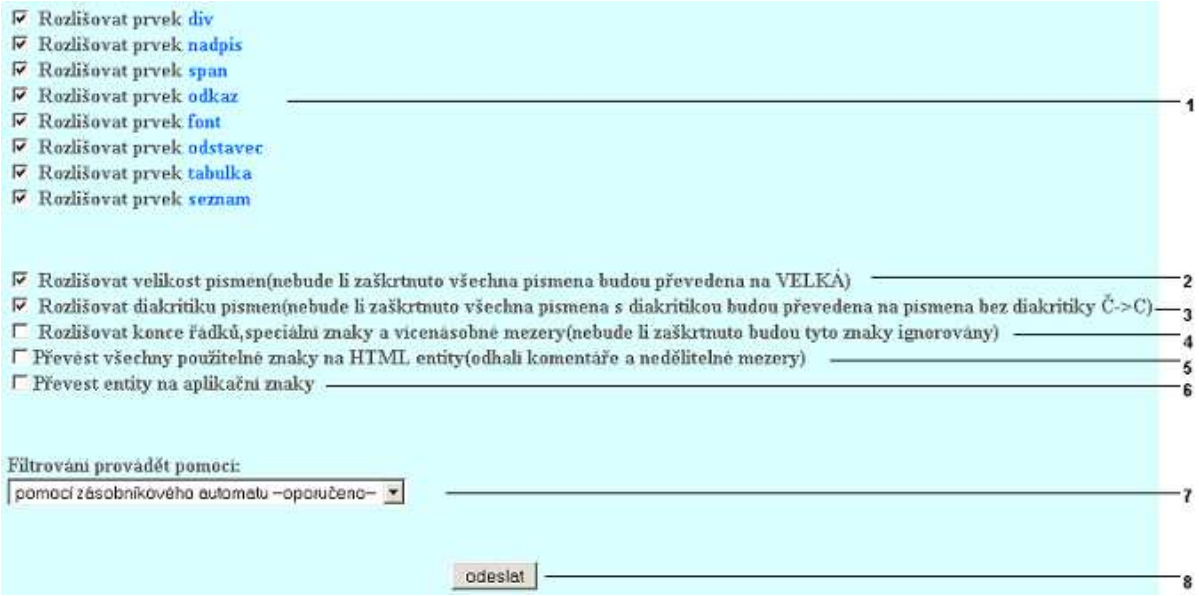
2. **Nastavení parametrů filtrace a předběžná analýza**

   Po odeslání adresy provede aplikace předběžnou syntaktickou analýzu obsahu HTML stránky. Zobrazí protokol o analýze a podle něj nastaví parametry pro filtraci. Tyto parametry je však možno nastavit dle vlastního uvážení.



1. Volby pro návrat k předchozím možnostem.
2. Zobrazuje adresu sledované stránky.
3. Počty sledovaných prvků na stránce odhalené předběžnou analýzou.
4. Protokol o předběžné syntaktické analýze.

Druhá část stránky pak nastavuje vlastnosti filtrování kódu a vytváření objektů.



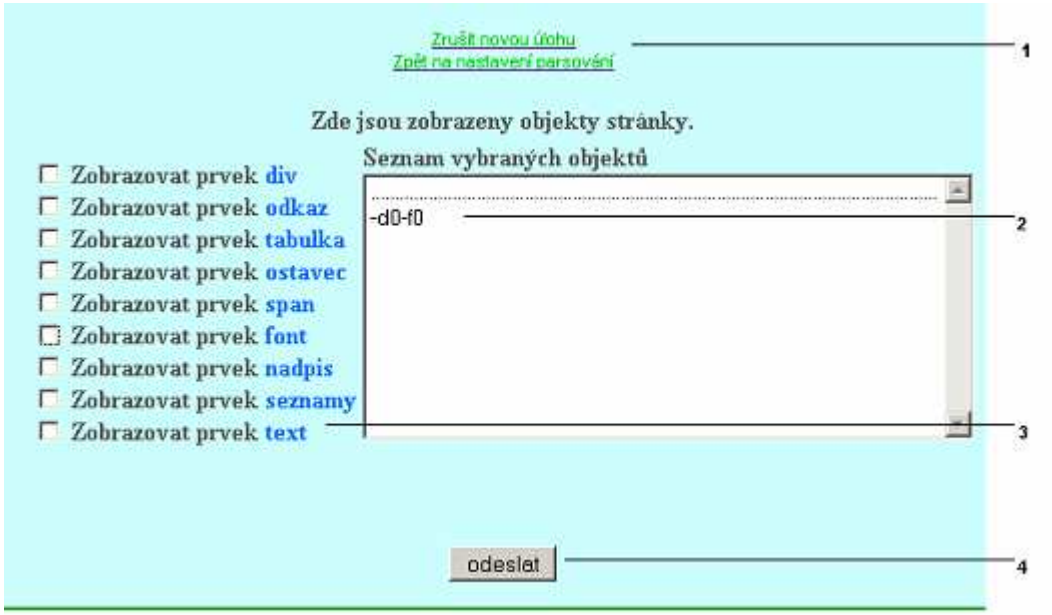
1. Volby elementů, jež budou rozlišovány jako samostatné objekty.
2. Možnost převodu všech písmen na velká.
3. Možnost zrušení diakritiky.
4. Možnost odstranění konců řádků a přebytečných bílých znaků z textu.
5. Možnost převodu všech značek HTML kódu, jež se neuplatňují v rozlišovaných elementech na HTML entity.
6. Možnost převodu všech HTML entit v kódu na jejich aplikační znaky.
7. Výběr způsobu filtrace.
8. Odešle zvoleného nastavení.

3. Kontrola filtrované stránky

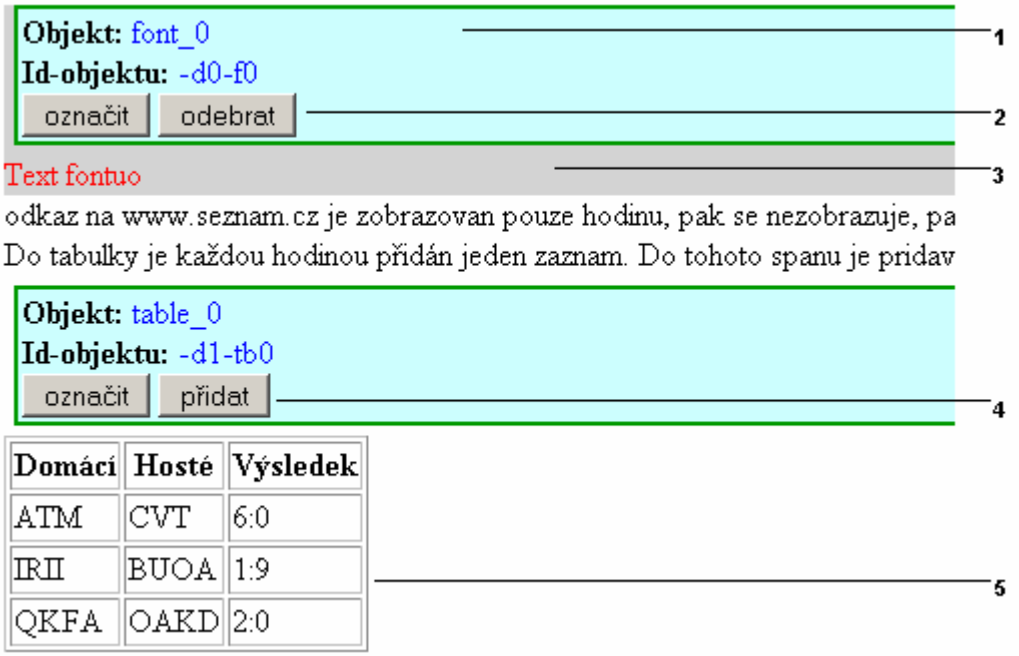
Po odeslání nastavení je stránka filtrována. Po filtraci se objeví nadhled stránky rozdělené na objekty. Pro výběr objektu pokračujeme stiskem tlačítka s popiskem *Odeslat* nebo je možné vrátit se zpět a přenastavit parametry.

4. Výběr objektů

Po odeslání je možno zvolit sledované objekty. Způsob výběru naznačuje obrázek. Pro uložení objektů do databáze pokračujeme stiskem tlačítka s popiskem *Odeslat* nebo je možné vrátit se zpět a přenastavit parametry.



1. Možnost návratu do centrálního menu nebo na nastavení parametrů.
2. Seznam vybraných objektů ke skenování.
3. V této části je možno zvolit objekty, u nichž se zobrazí ovládací panel. Nadhled stránky s ovládacími prvky je umístěn ve spodní části. Tak jak je to zobrazeno na dalším obrázku.
4. Odešle vybrané objekty pro uložení do databáze.

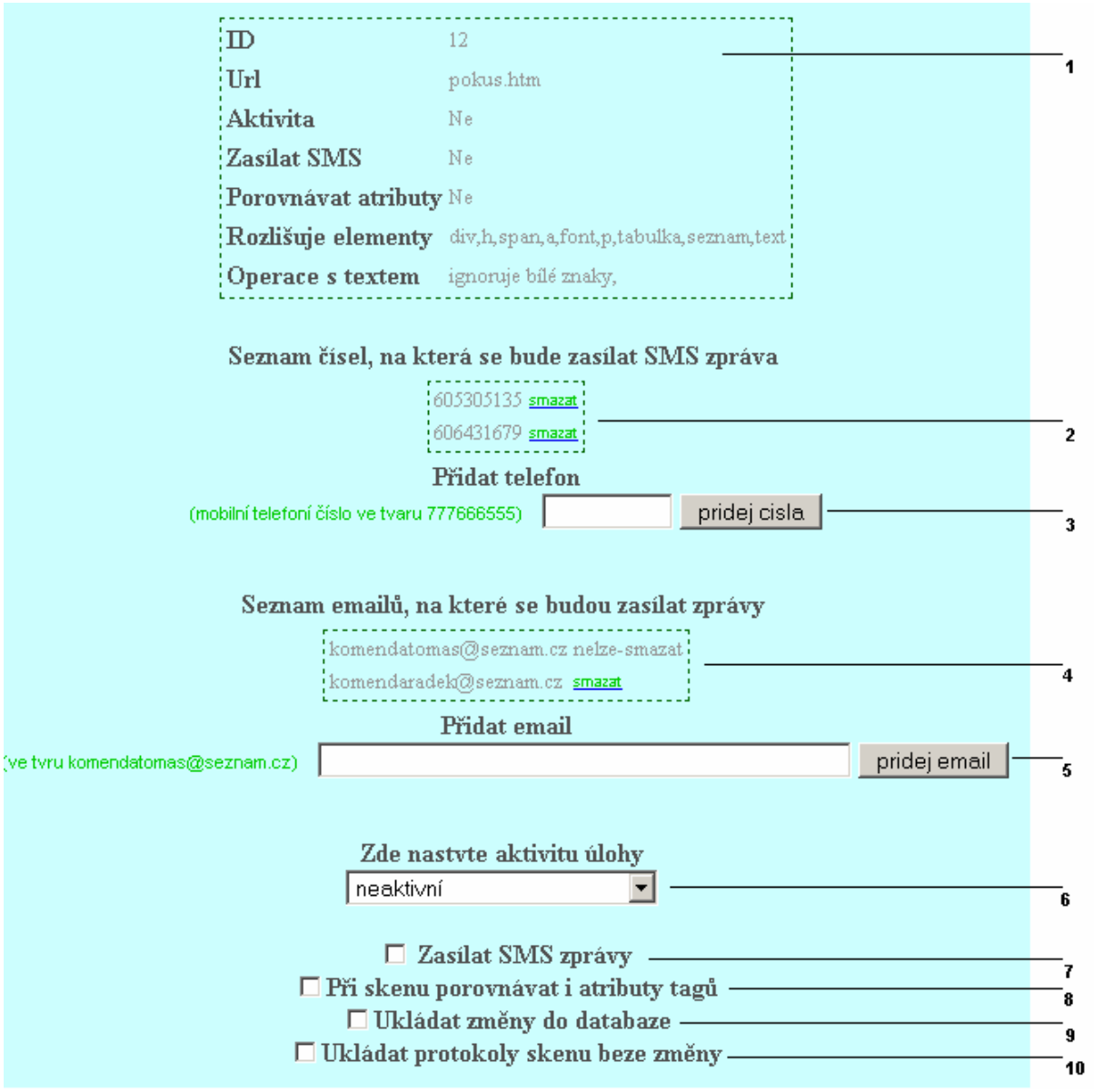


1. Označuje ovládací panel objektu *font_0*. Pokud však chceme ovládací panel pro konkrétní objekt, musíme jej povolit ve formuláři v horní části stránky.

2. Tlačítko *označit-odznačit* označí-odznačí objekt zelenou barvou. Tato volba nemá pro výběr objektu žádný význam. Slouží pouze pro představu uživatele, co vše daný objekt zahrnuje. Druhé tlačítko *přidat-odebrat* přidá nebo odebere objekty ze seznamu sledovaných objektů. Vybraný objekt je dále zvýrazněn šedou barvou. Pokud objekt obsahuje i další zanořené objekty, jsou také vybrány.
3. Označuje vybraný objekt fontu a textu, jež obsahuje element fontu.
4. Je podobné jako v bodě 2.
5. Označuje tabulku identifikovanou jako objekt *table_0*.

Nastavení úlohy

Poté co jsme zadali novou úlohu, je nutné nastavit její vlastnosti. Do nastavování vlastností se dostaneme s centrálního menu přes odkaz v tabulce úloh s názvem *Nastavit*. Možnosti nastavení popisuje následující obrázek.



1. Zde jsou zobrazeny informace o úloze.
2. Seznam mobilních telefonních čísel, na které budou zasílány informace o zjištěných změnách.
3. Přidá nové číslo do seznamu k dané úloze.
4. Seznam emailových adres, na které bude zasílán protokol o skenování. Vždy je povinný email uživatele jenž úlohu zadal. Ostatní emaily je možno libovolně mazat a přidávat.
5. Označuje pole pro zadání nové emailové adresy k dané úloze.
6. Zde je možno nastavit aktivitu úkolu (v jakých intervalech má být skenování prováděno).
7. Pokud je vybráno. Při změně sledovaných údajů bude zasílána SMS zpráva na uvedená čísla.
8. Pokud je vybráno boudou při porovnávání shody sledovaných dat porovnávány i atributy tagů.
9. Pokud je vybráno bude se obsah databáze se sledovanými údaji aktualizovat. To znamená, že pokud bude zjištěna změna sledovaných dat, uloží se nová podoba dat do databáze. Při dalším porovnání se pak bude porovnávat aktuální obsah s obsahem, u něhož byla naposledy zjištěna změna. Pokud není vybráno, bude se porovnávat vždy podle původního vzoru.
10. Pokud je vybráno budou se do databáze ukládat i protokoly o skenování, u nichž nebyla zjištěna změna ve sledovaných údajích.

Dále je možno vytvořit šablonu pro protokol. Ta se vytváří ve speciálním HTML editoru, který se spustí po kliknutí na odkaz *Vytvořit/změnit profil protokolu*. Editor je sám o sobě velice zajímavý, ale hlavně náročný na popis, proto jej dále podrobně nepopisuji.

Sledování údajů a historie sledování

Samotné sledování je prováděno automaticky v závislosti na nastavené aktivitě. Pokud však chceme provést sledování dané úlohy okamžitě, klikneme v centrálním menu v tabulce úloh na odkaz *Sledování.* Poté je možno spouštět sledování úlohy, jako procházet a spravovat historii skenování.

1. Způsobí návrat do centrálního menu.
2. Spustí skenování a vytvoří nový protokol (pokud je to v nastavení úlohy povoleno).
3. Zobrazí HTML protokol ke konkrétnímu skenování.

4. Zobrazí obsah SMS zprávy ke konkrétnímu skenování.
5. Smaže protokol, obsah SMS a záznam v tabulce konkrétního skenování.
6. Spustí filtr, jenž zobrazí pouze záznamy, u nichž byla zjištěna změna ve sledovaných údajích.
7. Smaže všechny záznamy v historii skenování.

Administrátorský režim

Pokud se přihlásíme jako superuživatel máme možnost přistupovat a měnit všechny údaje a úlohy uživatelů. Ovládání je podobné jako pro běžného uživatele.

Příloha 3. Test aplikace s univerzálním filtrem

Pro snadné testování aplikace jsem vytvořil testovací stránku, jenž obsahuje některé prvky, které je možno aplikací sledovat. Obsah stránky se pak každou hodinou mění. Na stránce je nadpis, jemuž je přidáno náhodně vygenerované písmeno a za hodinu je toto písmeno odebráno. Hodinu je nadpis bez přidaného písmene a poté je písmeno opět přidáno. Takto finguje i prvek odstavce a fontu, které jsou na stránce také přítomny. Objekt fontu také vždy jednou za hodinu změní svoji barvu, tedy atribut `color`. Dalším prvkem je element odkazu. Ten je však přítomen pouze hodinu a poté na hodinu zmizí. Tyto prvky jsou obsaženy v elementu divu. Dalším prvkem, jenž je uzavřen v samostatném divu je element spanu, který také mění jednou za hodinu svůj obsah (podobně jako nadpis). Posledním prvkem v druhém divu je tabulka se třemi sloupečky, jež představují název domácího mužstva, hostujícího mužstva a výsledek utkání. Každou hodinu je náhodně vygenerován nový řádek v této tabulce. Protože poskytovatel hostingu, na němž je tato stránka umístěna, neposkytuje přístup k periodickému časovači *cron*. Musím pro změnu údajů používat službu *WebCron*. Tato služba však není úplně spolehlivá (toto tvrzení je mým vlastním názorem, jenž vychází z tříměsíčního používání této služby) a muže se stát, že v určitou hodinu nebude skript aktivován a nedojde ke změně testovací stránky. Adresa stránky je *http://www.test999.xf.cz*. Pokud chcete provést změnu na testovací stránce okamžitě, je možné spustit skript na následující adrese je *http://www.test999.xf.cz/zmena.php*.

Při testech aplikace na výše uvedené stránce je možno vyzkoušet většinu funkcí. Pokud se však rozhodnete testovat jiné stránky, jejichž zdrojový kód obsahuje stovky nebo tisíce prvků, jež je možné rozdělit na samostatné objekty (například stránky, jejichž obsah sleduje první aplikace) a vybereme rozeznávání všech objektů (nedbáme doporučení předběžné analýzy), musíme počítat s časově náročnějším načítání a zobrazovaní stránky pro výběr objektů. Tato časová náročnost se v kritických případech může blížit i k jedné minutě. Tato doba však není způsobena neefektivní konstrukcí filtru či skriptu pro výběr objektu. Filtr je sám o sobě velice rychlí. Problémem je rychlost připojení a rychlost vykreslovaní všech prvků v samotném prohlížeči (klientská část). Samotný výběr objektů je již implementován pomocí klientského JavaScriptu. Po načtení a zobrazení všech prvků na stránce je proto práce již velice rychlá. Výše zmíněný problém je však aktuální pouze v případech skenování extremního množství objektu (řádově tisíce). Také velmi záleží na rychlosti připojení. Testoval jsem stránku na nichž bylo přibližně dva tisíce rozlišovaných objektů. K dispozici jsem měl sdílené připojení o rychlosti 256 Kb/s. Doba načtení všech objektů pro výběr (v kroku mezi zadáváním parametrů pro skenování a uložením do databáze) byla asi čtyřicet vteřin. Tato stejná akce trvala při rychlosti připojení garantovaného 6Mb/s deset vteřin a na *localhostu* je záležitostí necelé vteřiny. Ve

všech ostatních krocích výběru a v samotném skenování a porovnávání již problém s časovou náročností není.

Pokud se rozhodnete skenovat stránky, jež jsou sledovány první aplikací (http://www.mkblansko.euweb.cz/mk%202006/tabulky4.htm a stránka s výsledky utkání *http://www.mkblansko.euweb.cz/mk%202006/vysledky4.htm*), doporučuji ponechat nastavení skenování tak, jak jej přednastaví předběžná syntaktická analýza. Na stránce *http://www.mkblansko.euweb.cz/mk%202006/vysledky4.htm* je přítomna chyba, u jednoho elementu odstavce chybí uzavírací tag. Tato chyba je také odhalena předběžnou analýzou a je na ní upozorněno v protokolu. Tyto stránky již asi půl roku nezměnily svůj obsah, proto jsou vhodné akorát pro testování členění obsahu na objekty.

Pro testování změny obsahu, zasílání protokolů a SMS doporučuji testovací stránku *http://www.test999.xf.cz*. Ještě upozorním, že SMS zprávy jsou dvojího typu. První typ obsahuje textové obsahy původních, nových nebo změněných objektů. Pokud je však tato informace delší než je kapacitní norma pro SMS, je zasílána zpráva, která obsahuje pouze informace o počtu změněných, nových a ztracených (již neexistujících) objektů. Pro ujasnění formátu SMS zprávy uvádím následující ukázku.

```
[-d1-h0]n0,z1,o0[z]Nadpis>Nadpist|[-d2-tb0]n11,z0,o0[n]MEIW|QSAL|4:6|
```

Tato SMS byla zaslána, když jsem nechal na stránce *http://www.test999.xf.cz* sledovat nadpis a tabulku a poté jsem obsah testované stránky změnil pomocí připraveného skriptu *http://www.test999.xf.cz/zmena.php*. První část zprávy **`[-d1-h0]`** je identifikátorem objektu (první nadpis zanořený v prvním divu na stránce). Druhá část **`n0,z1,o0`** informuje, že v objektu nebyly nalezeny nové vnořené objekty, změněn byl jeden vnořený objekt (text) a nebyly odstraněny žádné vnořené objekty. Další část **`[z]Nadpis>Nadpist`** informuje jak se objekt změnil. Druhým objektem, o němž zpráva informuje, je první tabulka vnořená do druhého divu. V tabulce bylo zjištěno jedenáct nových objektů **`[-d2-tb0]n11,z0,o0[n]MEIW|QSAL|4:6|`**. Textového charakteru jsou však pouze tři a to název domácího mužstva, hostujícího mužstva a výsledek. Ostatní objekty jsou řádky a buňky tabulky. Ty však jsou pro protokol SMS nepodstatné, protože nepřináší žádnou hodnotnou textovou informaci.

Na závěr bych rád ještě upozornil, že pokud budete testovat aplikaci pomocí testovací stránky *http://www.test999.xf.cz* a při výběru sledovaných objektů (například tabulek) se budou zobrazovat prázdné objekty na začátku sledované stránky, nejde o chybu. Tyto objekty ve zdrojovém kódu skutečně jsou. Jedná se o součásti reklamy, kterou poskytovatel hostingu přidává na titulní stránku.

Celou aplikaci jsem testoval v prohlížeči Mozilla Firefox, ale je funkční i ve všech ostatních moderních prohlížečích.

Příloha 4. ER diagram databáze aplikace s univerzálním filtrem