SEVENTH FRAMEWORK PROGRAMME
FP7-ICT-2009-6

BlogForever
Grant agreement no.: 269963

BlogForever: D2.4 Weblog spider prototype and associated methodology

Editor:	**Morten Rynning**
Revision:	First version
Dissemination Level:	Public
Author(s):	Morten Rynning, Ioannis Manolopoulos, Ioannis Tselepidis, Michael Gulliksen, Mike Joy, Karen Stepanyan, Vangelis Banos
Due date of deliverable:	30th November 2011
Start date of project:	01 March 2011
Duration:	30 months
Lead Beneficiary name:	CyberWatcher

Abstract: The purpose of this document is to present the evaluation of different solutions for capturing blogs, established methodology and to describe the developed blog spider prototype.

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)

The **BlogForever** Consortium consists of:

Aristotle University of Thessaloniki (AUTH)	Greece
European Organization for Nuclear Research (CERN)	Switzerland
University of Glasgow (UG)	UK
The University of Warwick (UW)	UK
University of London (UL)	UK
Technische Universitat Berlin (TUB)	Germany
Cyberwatcher	Norway
SRDC Yazilim Arastrirma ve Gelistrirme ve Danismanlik Ticaret Limited Sirketi (SRDC)	Turkey
Tero Ltd (Tero)	Greece
Mokono GMBH	Germany
Phaistos SA (Phaistos)	Greece
Altec Software Development S.A. (Altec)	Greece

History

Version	*Date*	*Modification reason*	*Modified by*
0.1	20/08/11	Initial draft	Morten Rynning
0.2	12/09/11	Second draft	Morten Rynning and Michael Gulliksen
0.3	15/09/11	Feedback and revision	Vangelis Banos and Michael Gulliksen
0.4	19/09/11	Third draft	Morten Rynning
0.5	20/09/11	Fourth draft – distributed to partners	Morten Rynning
0.4	10/11/11	Fifth draft	Morten Rynning
0.5	20/11/11	Sixth draft	Karen Stepanyan, Vangelis Banos and I. Tselepidis
0.6	21/11/11	Seventh draft – distributed to partners	Morten Rynning
0.7	27/11/11	Including diagram to integrate OAIS to the spider (section 4.2)	Yunhyong Kim
0.8	27/11/11	Eighth draft	Mike Joy
1.0	29/11/11	Review and clarification	Ioannis Manolopoulos
1.0.1	15/02/12	Minor revisions	I. Tselepidis
1.1	29/02/12	Minor revisions	Morten Rynning

Table of Contents

TABLE OF CONTENTS 4

INDEX OF FIGURES 5

EXECUTIVE SUMMARY 6

1 INTRODUCTION 7

1.1 GOALS AND OBJECTIVES OF THE DELIVERABLE 7

2 APPROACHES AND TECHNOLOGIES FOR CRAWLING 8

2.1 PING SERVERS AND PINGBACKS 9
2.2 LINKBACKS, TRACKBACKS AND PINGBACKS 10
2.3 AVAILABLE BLOG PLATFORMS AND BLOG SEARCH ENGINE APIS 12

3 THE DIFFERENT APPROACHES TO CAPTURING AND INDEXING WEBLOGS 13

3.1 CAPTURING BLOG UPDATES 13
3.1.1 Compare text similarity using the Levenshtein distance 13
3.2 DEFINE CONTENT ELEMENTS AND LINK TYPES USING DECISION TREES 14
3.3 HTML-AWARE DATA EXTRACTION 15
3.3.1 HTML-aware data extraction prototypes using available open source libraries 15
3.4 ADVANCED INFORMATION RETRIEVAL FROM BLOG CONTENT 16
3.4.1 Semantic information in the web 16
3.4.2 State of the art on extraction of semantic information from blogs 21
3.4.3 Prototype semantic data extractor application 21
3.4.4 Future work 22
3.5 SPAM DETECTION 22
3.6 RANKING 23

4 THE SELECTED WEBLOG SPIDER ARCHITECTURE AND FUNCTIONALITY 25

4.1 HIGH LEVEL OVERVIEW 25
4.2 SINGLE-SERVER SOFTWARE ARCHITECTURE 29
4.2.1 Inputter and web interface 29
4.2.2 Host Analyzer 31
4.2.3 System Manager to operate the spider and capturing – including a blog-source library 32
4.2.4 The Worker for fetching 33
4.3 MULTIPLE-SERVER SOFTWARE ARCHITECTURE 35
4.4 THEORY/APPROACHES/METHODS USED 36
4.5 TECHNICAL PLATFORMS AND DEVELOPMENT TOOLS 37
4.6 CONNECTION WITH PREVIOUS BLOGFOREVER TASKS 38
4.6.1 Connections with the D2.1 Weblog survey 38
4.6.2 Connection with D2.2 Weblog Data Model 39
4.7 DEVELOPMENT CHALLENGES AND FUTURE WORK 40

5 CONCLUSIONS 42

6 REFERENCES 43

A. APPENDIX A – WEBLOG SPIDER PROTOTYPE CODE REFERENCE 44

B. APPENDIX B – WEBLOG SPIDER USE CASE 50

C. APPENDIX C – WEBLOG SPIDER PROTOTYPE INSTRUCTION MANUAL 54

D. APPENDIX D – PROTOTYPE SEMANTIC DATA EXTRACTOR APPLICATION CODE REFERENCE 63

E. APPENDIX E – HTML AWARE PROTOTYPE APPLICATIONS 66

F. APPENDIX F - GLOSSARY 70

Index of Figures

Figure 1 – Microformat example from http://johanramon.fr/ (class="adr") 17
Figure 2 – Event microdata from http://www.schema.org/Event 19
Figure 3 - Single-server Weblog Aggregation Prototype 28
Figure 4 - A single server BlogForever spider within the context of the OAIS (Open Archival Information System) model. 29
Figure 5 – Inputter view in the Spider Portal – a web interface for inserting and managing of selected blogs 30
Figure 7 - Host Analyzer, processing links to qualify blogs and establish link filters for monitoring 31
Figure 8 - Spider Portal interface – handling the System Manager User Interface 33
Figure 9 - Multi server architecture of the weblog spider 35
Figure 10 – Importance of blog elements to be preserved, according to D2.1 Weblog Survey 38
Figure 12 - While ping server is feeding the prototype spider with thousands of updates every minute – the user only wants a few selective updates. In the ping server section, two search filters are set. First requesting only blog hosts and blog posts from blogs 50
Figure 13 - In addition to science blogs and blogspotter blogs – the user would like the spider to capture all new updates regarding a qualified blog host: http://www.pharmacistplace.com 51
Figure 15 - After the blog host has been analysed and included in the Blog source database; all content elements are included and parsed – the output can be found in the Spider portal output section as shown above. 52
Figure 17 - Download and installation instruction file for the Blogforever Prototype Spider 54
Figure 18 - Spider portal for adding blogs manually; by pasting in URLs 57
Figure 20 - ping server section – automatically indexing new blogs according to filters defining what is relevant. 59
Figure 21 - Source list section where all latest blog sources included is listed 60
Figure 22 - Details linked from any of indexed sources – can display crawled XML, and link to actual HTML. 61
Figure 23 - Output section and link to FTP where all latest files with captured content, can be retrieved. 62
Figure 24 – HTML Database structure 66

Executive Summary

This report summarises the work done to evaluate available weblog data extraction methodologies and technologies. Additionally, this report presents a number of weblog data extraction tools implemented to test the aforementioned techniques and evaluate alternative ways to implement the BlogForever weblog spider in the following Work Package (WP4).

Weblog data extraction can be divided into two main steps, capturing weblog updates and extracting relevant information from updated blogs.

Section 2 evaluates technologies for capturing blog updates, namely notifications that a specific blog URL has been updated.

Section 3 evaluates technologies for capturing updated blog content, identifying new from old content, extracting structured information from semi-structured data as well as extracting semantic information such as microformats and microdata. Additionally, Section 3 evaluates spam filtering, an important aspect of blog data extraction.

Section 4 presents the selected approached for weblog data extraction including the software architecture and the methods applied, as well as the connections with other relevant BlogForever Tasks and Deliverables.

Finally, the report outlines the next steps to design and implement the BlogForever weblog aggregator component based on experience with the current work.

1 Introduction

Within the last five years, the amount of weblog crawling has increased rapidly along with the growth of social media. While for a long time the blogosphere was crawled and indexed by major search engines, more recently a large number of start-up agencies within the Social Media Monitoring area have launched their own spider platforms.

However, most spider platforms struggle with the vast volume of data within the blogosphere and are often limited to selected geographical locations. Others are limited to indexing RSS feed data received from weblog communities and ping servers. Another challenge which remains is that of enabling spiders to distinguish between different types of blogs. Currently, few weblog spiders can separate expert blogs from consumer blogs, or cover blogs from a selected domain, for instance scientific blogs. The challenges in differentiating blogs relate to the issues of spam and noise in blog search and analysis. The solutions developed as part the BlogForever project aim to address some of these challenges.

1.1 Goals and Objectives of the Deliverable

The BlogForever project aims to develop a weblog spider to capture weblogs, blog posts and additional blog content items to be indexed, analysed and preserved. This weblog spider is expected to crawl and capture a defined set of blogs, as well as identify and capture new blogs.

The primary objective of this deliverable is to present the design and development of the prototype BlogForever weblog spider and the associated methodologies. The prototype should take into account existing approaches and solutions to crawling. It should also be able to accommodate recent developments and recommendations in the area.

The developed prototype is expected to capture and populate a repository with weblog content using a combination of weblog specific technologies such as ping servers and RSS feed monitoring, as well as traditional web data extraction techniques such as scheduled crawls.

The weblog spider architecture, developed and presented in Section 4.2, defines the structure of the prototype. The prototype is expected to be the basis of a more comprehensive software solution throughout the next stages of the project in WP4, especially Task 4.2 (Design of the weblog spider component) and Task 4.3 (Implementation of the weblog spider component)[1]. The prototype will accommodate future requirements such as anticipated scalability and multi-server weblog crawling.

Prior to discussing the design and development of the prototype, this report describes some of the existing techniques for crawling and outlines their relevance for the BlogForever spider being developed. The report starts by outlining the existing approaches and technologies for crawling (Section 2). It then identifies techniques that are relevant for the development of the prototype (Section 3). This section also includes analysis and testing of various blog data extraction methodologies. Finally, the report presents the architecture and discusses the developed prototype (Section 4).

[1] For details see: Grant Agreement Annex I - Description of Work (DoW),

2 Approaches and Technologies for Crawling

This section describes the main technologies for capturing weblogs. It includes a number of additional tools and methods that are valuable for the purposes of filtering and parsing blogs as part of the digital preservation effort.

1. **Ping servers and Pingbacks**. In blogging, ping is an XML-RPC-based push mechanism by which a weblog notifies a server that its content has been updated. An XML-RPC signal is sent to one or more ping servers, which can then be used by those ping servers for generating a list of blogs that have published new material. Today, most blog authoring tools automatically ping one or more servers each time a blogger creates a new posts or updates earlier ones.

   Open ping servers allow other web services to subscribe to a list of blogs that have recently pinged them, and include, for instance, Weblogs.com. Subsequently, having a list of updated blogs, blog search engines can gain access to fresh results very quickly by polling only the newly-updated blogs. In addition to search engines, web feed aggregators use the results from ping servers to notify subscribers about the fresh material made available on their subscription lists.

   Unlike open ping servers, proprietary ping servers gather information primarily for their own applications. Proprietary ping servers are operated, for instance, by most of the major blog search engines.

2. **Blog aggregator software/APIs**. There are a number of tools that allow subscribing to web feeds and accessing content from various sources via a single interface. These aggregators can be proprietary or open, but in either case aggregators provide a mechanism for capturing blog RSS/Atom feeds. In addition to aggregators, there are also a number of plugins that utilise web feeds. These plugins are available for various blogging platforms, for instance, for WordPress[2]. Whilst aggregators provide mechanisms for accessing newly published blog content, they are not the only way of receiving the content. There are services, for instance, Feedburner[3] (now available via Google) that offer a range of middleware services between the publishers and consumers. Access to some of these services is available via API interfaces.

3. **Open weblog spider for identifying web feeds**. In addition to using aggregation and updates from a ping server, it is possible to identify blogs by searching blog service websites and directories. The use of directories in particular enables identification of new blogs, local blogs and niche blogs with greater efficiency.

4. **Full text weblog spider.** Full text weblog spiders enable gathering of content from a weblog site. Such a weblog spider may consist of two elements. Firstly, a spider component for finding blog sites and blog feeds and, secondly, a component that acquires the list of identified blogs and indexes them extensively to include all available content. This way of capturing is usually contrasted with RSS crawling that is limited to the data distributed via web feeds. Full text crawling provides a possibility of capturing content in full for blog posts, comments, links, attachments, graphics, author information and metadata.

Recent developments in the area of weblog crawling
The World Wide Web continues to grow and evolve. New technologies, services and standards emerge periodically, affecting the development of the blogosphere. Among the most recent

[2] http://wordpress.org/extend/plugins/search.php?q=feed&sort=
[3] http://feedburner.com

developments that may have direct implications for the development of the BlogForever project are the developments of Microdata, crawling of JavaScript by Google, and HTML5:

- Microdata (schema.org) is a standard for enriching web content. This standard is recognised and has been agreed by Microsoft, Google and Yahoo since July 2011. Microdata can be very valuable when crawling the web and weblogs for different items and types of content.

- Capturing comments on blog posts often requires running some JavaScript code. This forms one of the challenges when crawling social media websites. However, the recent announcement of Google highlighted the launch of a Google spider that is capable of executing AJAX/JavaScript code and capturing comments on social media sites, including Facebook [1]. The announcement raises a discussion on what the implications of this recent development may be in relation to the blogosphere. It is clear, however, that capturing content that has remained out of the reach of spiders is now becoming accessible.

- HTML5 is another substantial development in the area and constitutes a significant improvement on its predecessor version. Although earlier versions of HTML are still being used widely, the situation may change in the near future. The use of HTML5 can enable a more extensive extraction and processing of semantic entities.

2.1 Ping servers and pingbacks

Given the potential for identifying new blogs and monitoring updates, it appears beneficial to integrate BlogForever with a mechanism for working with ping servers. The spider can use openly available ping servers to gain information about new posts and weblogs.

1. New blog posts (updates). Information about updates acquired from the ping server can be collected by the BlogForever spider and used accordingly for crawling new posts.
2. New weblogs. Weblogs that have not been previously known to the BlogForever spider can be identified from the ping server and added to the list of crawled weblogs.

Using information about newly added or updated pages on already existing blogs can reduce the time lapse for crawling and indexing. For instance, the spider (more specifically, the integral component of the spider – the scheduler) may set a certain frequency for visiting/re-visiting weblogs for crawling new content. This frequency may be limited to daily visits, for example. However, crawling frequently updated weblogs may require more frequent visits. Having notifications from a ping server about new posts would allow fetching and indexing the posts immediately rather than with a set delay.

Additionally, a ping server may be a useful resource for discovering new weblogs and extending the repository of collected weblogs. For instance, when a new post is published by a previously unknown resource it can be used for adding a new weblog to the database. However, a common problem when working with ping servers is the presence of large amounts of spam. Spam sources may not necessarily be weblogs, and they may have irrelevant content or re-publish original content from elsewhere. All of those issues will need to be addressed at some point in the BlogForever project.

There are a large number of ping servers available today[4]. Not all of the ping servers are openly available, and some servers may not offer consistent or reliable services. Potentially suitable ping servers for the BlogForever include Blo.gs and Feedburner.

[4] Sample List of Ping Servers: http://www.blog-blog.de/blogs/7-Liste-der-Ping server-Pingservices.html

For the ping servers to be able to receive and distribute notifications about a new post, weblog users are expected to use a script to connect their weblog with a specified ping server. The addition of the script will associate the weblog with a ping server and enable sending a ping message once the content of the weblog is updated. As seen below in Table 1, the output from a ping server is very limited beyond telling that a URL has been updated.

Weblog tracking services, however, use ping servers differently. The BlogForever spider should use an appropriate interface for accessing the services provided by a ping server. More specifically, the BlogForever spider will need to connect to the ping server (sometimes called “hook up to a cloud”) and provide an IP address to be used for receiving the feed. It would also require connecting to a specific URL (e.g. ping.blo.gs) via specific port (e.g. port 29999, in the case of blo.gs), which will enable receiving a streaming version of the changes.xml files that contain information about the received ping messages.

Table 1 – Example of an output received from the blo.gs ping server

```
<?xml version="1.0" encoding="utf-8"?>
<weblogUpdates version="1" time="20041127T22:51:58Z">
<weblog name="blog one"
    url="http://example.com/blogone/"
    rss="http://example.com/blogone/index.rss"
    service="ping" ts="20041127T22:51:59Z"
/>
<weblog name="blog two"
    url="http://example.com/blogtwo/"
    service="ping" ts="20041127T22:52:06Z"
/>
<weblog name="blog three"
    url="http://example.com/blogthree/"
    service="blogger.com"
    polled="true" ts="20041127T22:47:38Z"
/>
</weblogUpdates>
```

2.2 Linkbacks, Trackbacks and Pingbacks

LinkBack (or linkback) is a mechanism that enables receiving notifications when posts are referenced in other weblogs. This mechanism provides information on how posts are being referenced and, therefore, are prized by many bloggers. Capturing linkback data from blogs can provide an insight into the network of blogs and the intricate inter-relations within the blogosphere. Linkback mechanisms are used by various search engines, like Google or Technorati, to rank blogs and their posts [2]. It is therefore important to ensure that the BlogForever spider captures and processes the information about linkbacks.

To enable effective capture of linkback data it is necessary to look into the available types and implementation of various linkback mechanisms. This section briefly outlines two of the linkback mechanisms: Trackbacks [3] and Pingbacks [4]. The use of RefBacks is not covered in this report due to the limitation in identifying the traces of using this type of linkback within the available source code.

Linkback mechanisms automate the exchange of links between bloggers. To enable the automation, the process of exchange consists of two parts: auto-discovery and notification. The auto-discovery mechanism requires each post to have an RDF fragment (see Table 2 for an example) or a <link> tag which specifies which page should be used by other blogs for submitting their linkback notifications. The notification page is used for collecting and processing the received linkback notifications. The protocols for sending notifications, on the other hand, differ from one type of linkback to another. For instance, trackback notification is sent as an HTTP POST request which

specifies the title, URL, and an excerpt of the posts as well as the name of the blog. On the other hand, pingback notifications require the use of XML-RPC calls. Regardless of the protocols used, the notified blog processes the received information and updates the blog by inserting a link to the resource that cites it.

Table 2 – **Example of RDF fragment enabling trackback auto-detection**

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:trackback="http://madskills.com/public/xml/rss/module/
trackback/"
xmlns:dc="http://purl.org/dc/elements/1.1/">
<rdf:Description
    rdf:about="http://www.orlandovacationer.com/archives/000076.html"
    trackback:ping="http://www.cadenhead.org/cgi-bin/mt-tb.cgi/76"
    dc:title="Dining in Orlando&apos;s Rainforest Cafes"
    dc:identifier="http://orlandovacationer.com/archives/000076.html"
    dc:subject=""
    dc:description="Theme  Park  Insider  reviews  the  Rainforest  Cafe
restaurants
at Disney World Animal Kingdom and Downtown Disney in Orlando: If
you’ve heard anything about the Rainforest Cafe, it’s likely been
the atmosphere, first and foremost. The entire restaurant is
themed to..."
    dc:creator="rcade"
    dc:date="2004-01-27T13:51:10-05:00" />
</rdf:RDF>
```

The approaches used by blogs for exploiting trackbacks may differ from one blog to another. Some blogging platforms, in addition to incorporating the links or RDF code for auto-detection, allow checking of links for potential trackback at the time of making a blog post. Because this check requires the blogging platform to access the included links and search for the trackback notification page, the process of making the posts may take longer than usual [5]. Hence, in some cases, bloggers may prefer to disable the automatic checks and manually control the trackback notifications by filling out a form that specifies the trackback page for sending a notification.

Capturing the relationships between blogs that are created by the linkbacks is an important part of the BlogForever project. Therefore the spider (if not at the prototyping stage than later in the project) will need to be able to identify and categorise linkbacks. This will be possible by parsing the content of the accessed weblog pages. However, a distinction between the general links and links added as a result of linkbacks is necessary. Although the inserted links usually appear as comments with an indication of representing a trackback, the implementation of inserting trackback links may vary from one blogging platform to another. The following tables (Table 3 and Table 4) demonstrate an example of client side code inserted into a blog for displaying a trackback.

Table 3 – **Example of an inserted trackback link (from [5])**

```
<MTEntryIfAllowPings>
   | <a href="<$MTCGIPath$><$MTTrackbackScript$>?__mode=view&amp;
entry_id=<$MTEntryID$>" onclick="OpenTrackback(this.href);
return false">TrackBack (<$MTEntryTrackbackCount$>)</a>
</MTEntryIfAllowPings>
```

Table 4 – **Example of JavaScript code for opening a trackback link in a pop-up window (from [5])**

```
<script language="javascript" type="text/javascript">
function OpenTrackback (c) {
  window.open(c,
```

```
    'trackback',
    'width=480,height=480,scrollbars=yes,status=yes');
}
</script>
```

An alternative representation of a trackback link can include the use of a rel="" attribute for the HTML <link> tag. For instance:

```
<a href="url" rel="trackback">Text</a>
```

An example of a pingback link inserted into a page can more commonly be:

```
<link rel="pingback" href="http://vbanos.gr/xmlrpc.php" />
```

Regardless of the mechanism for inserting and displaying linkbacks, the BlogForever spider aims to distinguish different types of trackbacks and capture these for preservation. Whether linkback insertions can be used to improve the process of identifying new blogs for crawling is an interesting question that requires evaluation and lies beyond the scope of this report.

2.3 Available blog platforms and blog search engine APIs

Searching and fetching web feeds is often made possible via APIs. Search services offered by Google, for instance, can be used to search or fetch feeds such as RSS. Similar functionality is often provided by blogging platforms. This section briefly outlines some of the weblog search engines and their use as part of the BlogForever spider.

There are a number of weblog search engines dedicated to searching though weblog content. Among those are Bloglines[5], BlogScope[6] and Technorati[7]. Technorati, which is amongst the earlier and most popular blog search engines, provides current information on popular weblogs and tags used to categorize blog postings. Some of these search engines, for instance BlogScope, offer a wider analysis of resources and extend beyond simple keyword search. Weblog search services process large amounts of data and can publish information about trending topics or blogs. Amatomu[8] and Icerocket[9] extend the list of similar tools with more specialised services.

In addition to freely available APIs, there are also commercial solutions that offer access to similar data or even crawling software itself as a hired platform. Among those are Twingly[10], Moreover and CyberWatcher[11].

Some of the blogging services like Blogger or WordPress offer APIs to their users for publishing or managing their blogs. These APIs are discussed in greater detail in the previously submitted D2.2 BlogForever report [6]. Although considered useful for working with individual blogs, the use of these APIs may not be relevant for identifying or crawling blog content. Blogger, similarly to Weblogs.com ping server, provides access to the changes XML file. However, it appears to be less common for blogging platforms to provide APIs or other distribution mechanisms for discovering and accessing blogs on a large scale. Therefore, the use of APIs for developing a spider prototype is not considered.

[5] http://www.bloglines.com/
[6] http://www.blogscope.net/
[7] http://technorati.com/
[8] http://www.amatomu.com/
[9] http://www.icerocket.com/
[10] http://www.twingly.com/
[11] http://www.cyberwatcher.com/

3 The different approaches to capturing and indexing weblogs

Capturing and indexing web content is an active research topic with several web data extraction methodologies available and a large volume of relevant tools already implemented. This section aims to present the techniques evaluated for capturing and indexing weblogs. The process can be divided into the following parts:

1. Capturing blog updates
2. Indexing weblog content
3. Advanced information retrieval from blog content
4. Spam detection (both in ping servers and blog content)
5. Blog ranking

3.1 Capturing blog updates

Section 2 (Approaches and technologies for crawling) discussed a number of approaches for defining which blog URLs have been updated. While this step is an important one, it is only the beginning in the task of capturing actual blog content. Specific techniques and algorithms are required to find the updated content and define the elements of the latest blog posts. The following subsections discuss some of them.

3.1.1 Compare text similarity using the Levenshtein distance

The Levenshtein distance [12] is a string metric for measuring the amount of difference between two character sequences. This can be used to compare text in RSS format with full text in HTML in order to perform approximate string matching and define the element location and attributed of the corresponding RSS element and full text from the HTML.

Computing the Levenshtein distance is based on the observation that if we reserve a matrix to hold the Levenshtein distances between all prefixes of the first string and all prefixes of the second, then we can compute the values in the matrix by flood filling the matrix, and thus find the distance between the two full strings as the last value computed.

This algorithm, an example of bottom-up dynamic programming, is discussed, with variants, in the 1974 article by Wagner and Fischer [13]. A straightforward implementation, as pseudo code for a function `LevenshteinDistance` (Table 5) that takes two strings, `s` of length `m`, and `t` of length `n`, and returns the Levenshtein distance between them is listed below:

Table 5 – **Pseudocode for the Levenstein distance algorithm.**

```
int LevenshteinDistance(char s[1..m], char t[1..n])
{
  // for all i and j, d[i,j] will hold the Levenshtein distance between
  // the first i characters of s and the first j characters of t;
  // note that d has (m+1)x(n+1) values
  declare int d[0..m, 0..n]

  for i from 0 to m
    // the distance of any first string to an empty second string
    d[i, 0] := i
  for j from 0 to n
    // the distance of any second string to an empty first string
    d[0, j] := j

  for j from 1 to n
  {
```

```
    for i from 1 to m
    {
      if s[i] = t[j] then
        d[i, j] := d[i-1, j-1]       // no operation required
      else
        d[i, j] := minimum
                   (
                     d[i-1, j] + 1,  // a deletion
                     d[i, j-1] + 1,  // an insertion
                     d[i-1, j-1] + 1 // a substitution
                   )
    }
  }
  return d[m,n]
```

3.2 Define content elements and link types using decision trees

A challenge when receiving input of blogs, e.g. from a ping server, is the volume of received URLs with no structure. Each URL or specifically, the associated page to the URL, can be a blog host, a blog post or a comment.

Machine learning using decision trees is a technique for defining which links are blog hosts, blog posts and comments. These algorithms utilize sample tests of multiple attributes to recognize a blog post link or a blog host link. This way the system learns the rules by attributes that relate to either type of link. The output of this URL analysis is a set of rules per source called link filters. Link filters separate valid links from invalid links and identify links that represent blog posts. Similarly, each blog post will be analyzed to extract full text content using the same set of machine learning algorithms that define content filters.

Machine learning using decision trees is also used to identify which part of the downloaded blog HTML corresponds to each blog element. Using this method, the application is able to identify blog elements such as main blog content, author name and publication date.

Decision trees are also known as classification trees or regression trees. In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. In data mining, a decision tree describes data but not decisions – rather, the resulting classification tree can be an input for decision making. ID3 and Neural Network algorithms were tested during the prototype application development in order to assess their performance in weblog data extraction.

ID3-algorithm

ID3 (Iterative Dichotomiser 3) is an algorithm used to generate a decision tree. The ID3 algorithm can be summarized as follows.

1. Take all unused attributes and count their entropy concerning test samples.
2. Choose attribute for which entropy is minimum (or, equivalently, information gain is maximum).
3. Make node containing that attribute.

The algorithm prefers smaller decision trees (simpler theories) over larger ones. However, it does not always produce the smallest tree, and is therefore a heuristic. It is flexible and can be used both for analyzing blog hosts as well as content per blog post and comments.

Neural network (NN)

A neural network consists of an interconnected group of artificial neurons, and it processes information using a connectionist approach to computation. Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs or to find patterns in data.

Neural network is more sophisticated than ID3 but not equally scalable. For this purpose, ID3 is the preferred algorithm as scalability is vital for a blog spider that is required to process large volumes of URLs from ping servers.

3.3 HTML-aware data extraction

HTML-aware data extraction methods are based on the inherent characteristics of the HTML mark-up language to infer extra information from documents. The HTML-aware method is the standard method to convert HTML into structured information. This is needed because HTML is not efficient for machine processing.

There are several challenges in converting HTML into text. The two basic requirements are (a) speed in processing and (b) accuracy in converting *all* the HTML elements into text. The latter is challenging due to many web technologies that obfuscate information and make it hard for the software to process it. Such technologies include JavaScript framed feeds from other URLs, Flash, etc. On the other hand, HTML5 and other open web technologies and standards promote open access to web information and better HTML-aware data extraction. A useful summary by Azagra covers the challenges as well as available tools [14].

When extracting data from a blog, the process can be based upon unwrapping the HTML on each page. In addition, the process can include downloading the entire site and analyzing the site structure using a DOM-tree. This also enables comparison of several webpages to be included when finding structure and patterns to convert into text. One early tool to include this was the RoadRunner, described in an early paper about extraction and analysis methods [11].

3.3.1 HTML-aware data extraction prototypes using available open source libraries

In order to evaluate HTML-aware data extraction, three libraries were evaluated through a basic prototype application that was developed for each of them, two of them well known and widely used (HTML Agility and Beautiful Soup) and another one that is normally used for web page testing but provides parsing capabilities too (Watin).

HTML Agility Pack[12] is an HTML parsing library for the .NET framework which is tolerant with HTML that is not well formed. The HTML tree can be queried with XPATH or LINQ [13]in newer versions (via a LINQ to XML interface).

Beautiful Soup[14] is an HTML parsing library written in the Python programming language. It's also tolerant in malformed HTML using a class that employs heuristics for getting a sensible tree even in the presence of HTML errors but it also provides a class for parsing XML, SGML or a domain-specific language that looks like XML.

---

[12] http://htmlagilitypack.codeplex.com/
[13] http://en.wikipedia.org/wiki/Language_Integrated_Query
[14] http://www.crummy.com/software/BeautifulSoup/

WatIn[15] is a .NET library primarily for Web testing that can be used for HTML parsing. It's not as well suited for parsing as HTML Agility or Beautifoul Soup but it is very good in interacting with web pages, executing commands (e.g. with auto clicks) etc., which gives it an advantage over the other two with regard to e.g. AJAX driven web sites.

After reading the libraries' documentation and conducting prototyping, the following conclusions were reached:

- HTML Agility and Beautiful Soup produced quite similar results, no definite advantage was observed for the one over the other.
- Watin is quite slow and heavy (normally it loads a browser object although it is possible to extract from web pages without loading a browser but this method is not well documented). It doesn't support the emulation of all browsers/all versions and its API has at least one considerable shortcoming: there is no attributes collection.
- A combination of Watin and HTML Agility has been used during the evaluation to overcome this problem (attributes were provided by HTML Agility to Watin). It is not adequate by itself for the purposes of the project's crawler but it might be very useful in combination with another library.
- The combination of HTML Agility and Watin seems promising as these two libraries are somehow complementary, one is good for parsing and the other for interacting with web pages. This combination is more straightforward than the combination of Beautiful Soup and Watin, since HTML Agility and Watin both target the same framework (.NET).

Technical details regarding the code implemented and tests conducted can be found at Appendix E.

3.4 Advanced information retrieval from blog content

BlogForever will develop robust digital preservation, management and dissemination facilities for blogs. An important part of BlogForever is the aggregation of blog content and more specifically the extraction of semantic elements omitted or ignored by current web archiving methods and solutions.

Semantic data embedded in web pages have been gaining ground considerably during the last few years in the form of many different schemas such as microformats, microdata and RDFa.

BlogForever aims to use information encoded in all these schemas in order to improve blog preservation, management and dissemination.

This subsection begins by presenting an overview of the state of semantic information in the web. Current schemas of semantic information embedded in web pages are also investigated and their importance, as well as the information they convey, are evaluated. Additionally, the work done to test semantic data extraction from blogs is illustrated. Finally, the open source library *any23*, which was selected to handle the task, is presented. The final part of this subsection provides a detailed description of the prototype and its use.

3.4.1 Semantic information in the web

The Semantic Web is a web of data that facilitates machines to understand the semantics, or meaning, of information on the World Wide Web. One of the current trends in the web today is the inclusion of semantic information within existing content on web pages. This way, special software such as search engines, web crawlers and even browsers can extract and process this information.
The main metadata schemas for this purpose are microformats, microdata and RDFa.

[15] http://watin.org/

Microformats

A microformat (sometimes abbreviated μF) is a web-based approach to semantic mark-up which seeks to re-use existing HTML/XHTML tags to convey metadata and other attributes in web pages and other contexts that support (X)HTML, such as RSS. This approach allows software to process information intended for end-users (such as contact information, geographic coordinates, calendar events, and the like) automatically.

Although the content of web pages is technically already capable of "automated processing", and has been since the inception of the web, such processing is difficult because the traditional mark-up tags used to display information on the web do not describe what the information means. Microformats can bridge this gap by attaching semantics, and thereby obviate other, more complicated, methods of automated processing, such as natural language processing or screen scraping (for example, Figure 1). The use, adoption and processing of microformats enable data items to be indexed, searched for, saved or cross-referenced, so that information can be reused or combined.

```
<p class="adr">
  <span class="street-address">7 bis, rue des Acres</span>
  <span class="postal-code">60200</span>
  <span class="locality">Compi&Atilde;&uml;gne</span> -
  <span class="country-name">France</span>
  <span class="tel">
    <abbr title="T&Atilde;&copy;l&Atilde;&copy;phone">T&Atilde;&copy;l.</abbr>
    :<span class="value">+33 (0)6 24 81 05 64</span>
  </span>
</p>
```

Figure 1 – Microformat example from http://johanramon.fr/ (class="adr")

Microformats allow the encoding and extraction of events, contact information, social relationships and so on. A brief list of microformats, both stable and draft is listed below.

Stable Microformats

- **hCalendar** is a simple, open, distributed calendaring and events format, using a 1:1 representation of standard iCalendar (RFC2445) VEVENT properties and values in semantic HTML or XHTML. http://microformats.org/wiki/hcalendar
- **hCard** is a simple, open, distributed format for representing people, companies, organisations, and places, using a 1:1 representation of vCard (RFC2426) properties and values in semantic HTML or XHTML. http://microformats.org/wiki/hcard
- **rel-license** is a simple, open, format for indicating content licenses which is embeddable in HTML or XHTML, Atom, RSS, and arbitrary XML. http://microformats.org/wiki/rel-license
- **rel-nofollow** is an elemental microformat, one of several open microformat standards. By adding rel="nofollow" to a hyperlink, a page indicates that the destination of that hyperlink should not be afforded any additional weight or ranking by user agents which perform link analysis upon web pages (e.g. search engines). http://microformats.org/wiki/rel-nofollow
- **rel-tag**. By adding rel="tag" to a hyperlink, a page indicates that the destination of that hyperlink is an author-designated "tag" (or keyword/subject) for the current page. http://microformats.org/wiki/rel-tag

- **VoteLinks.** Indexing and tracking applications treat all links as endorsements or expressions of support. This is a problem, as we need to link to those we disagree with as well, to discuss why. http://microformats.org/wiki/vote-links
- **XFN** (XHTML Friends Network) is a simple way to represent human relationships using hyperlinks. XFN enables web authors to indicate their relationship(s) to the people in their blogrolls simply by adding a 'rel' attribute to their <a href> tags. http://gmpg.org/xfn/
- **XMDP** (XHTML Meta Data Profiles) is a simple XHTML-based format for defining HTML meta data profiles easy to read and write by both humans and machines. The mark-up is a profile of XHTML. http://gmpg.org/xmdp/
- **XOXO** is a simple, open outline format written in standard XHTML and suitable for embedding in (X)HTML, Atom, RSS, and arbitrary XML. http://microformats.org/wiki/xoxo

Draft Microformats

- **adr** is a simple format for marking up address information, suitable for embedding in HTML, XHTML, Atom, RSS, and arbitrary XML. adr is a 1:1 representation of the *adr* property in the vCard standard (RFC2426) in HTML, one of several open microformat standards. It is also a property of hCard microformat. http://microformats.org/wiki/adr
- **geo** is a simple format for marking up WGS84 geographic coordinates (latitude; longitude), suitable for embedding in HTML or XHTML, Atom, RSS, and arbitrary XML. geo is a 1:1 representation of the "geo" property in the vCard standard (RFC2426) in HTML, one of several open microformat standards. http://microformats.org/wiki/geo
- **hAtom** is a microformat for content that can be syndicated, primarily but not exclusively weblog postings. hAtom is based on a subset of the Atom syndication format. http://microformats.org/wiki/hatom
- **hAudio** is a simple, open, distributed format, suitable for embedding information about audio recordings in (X)HTML, Atom, RSS and arbitrary XML. http://microformats.org/wiki/haudio
- **hListing** is a proposal for an open, distributed listings (small-ads; classifieds) format suitable for embedding in (X)HTML, Atom, RSS, and arbitrary XML. http://microformats.org/wiki/hlisting
- **hMedia** is a simple and open format for publishing Images Video and Audio. hMedia may be embedded in HTML or XHTML, Atom, RSS, and arbitrary XML. http://microformats.org/wiki/hmedia
- **hNews** is a microformat for news content. hNews extends hAtom, introducing a number of fields that more completely describe a journalistic work. hNews also introduces another data format, rel-principles, a format that describes the journalistic principles upheld by the journalist or news organization that has published the news item. http://microformats.org/wiki/hnews
- **hProduct** is a microformat suitable for publishing and embedding product data. hProduct is one of several open microformats standards suitable for embedding in HTML, XHTML, Atom, RSS, and arbitrary XML. http://microformats.org/wiki/hproduct
- **hRecipe** is a simple, open, distributed format, suitable for embedding information about recipes for cooking in (X)HTML, Atom, RSS, and arbitrary XML. http://microformats.org/wiki/hrecipe
- **hResume** is a microformat for publishing resumes and CVs. hResume is one of several open microformat standards suitable for embedding in HTML, XHTML, Atom, RSS and arbitrary XML. http://microformats.org/wiki/hresume
- **hReview** is a simple, open, distributed format, suitable for embedding reviews (of products, services, businesses, events, etc.) in HTML, XHTML, Atom, RSS and arbitrary XML. http://microformats.org/wiki/hreview

- **Rel-Directory** is one of several microformats. By adding 'directory' to the rel attribute of a hyperlink, a page indicates that the destination of the hyperlink is a directory listing that contains an entry for the current page. http://microformats.org/wiki/rel-directory
- **Rel-Enclosure** is a simple, open, format for indicating files to cache which is embeddable in (X)HTML, Atom, RSS and arbitrary XML. http://microformats.org/wiki/rel-enclosure
- **Rel-Home** is one of several microformats. By adding rel="home" to a hyperlink, a page indicates that the destination of that hyperlink is the homepage of the site in which the current page appears. http://microformats.org/wiki/rel-home
- **Rel-Payment** is a microformat for making exchanges of support (be it financial or otherwise) possible. By adding rel="payment" to a hyperlink a page indicates that the destination of that hyperlink provides a way to show or give support for the current page. http://microformats.org/wiki/rel-payment
- **Robot Exclusion Profile** is a reworking of the Robots META tag (and less-standard extensions) as a microformat. http://microformats.org/wiki/robots-exclusion
- **xFolk** (from "xFolksomony") is a simple and open format for publishing collections of bookmarks. It enables services for improving user experience and sharing data within web-based bookmarking software. xFolk may be embedded in HTML or XHTML, Atom, RSS, and arbitrary XML. http://microformats.org/wiki/xfolk

Microdata

Microdata are a WHATWG HTML5 specification used to nest semantics within existing content on web pages. Search engines, web crawlers and browsers can extract and process microdata from a web page, and use it to provide a richer browsing experience for users. Microdata use a supporting vocabulary to describe an item and name-value pairs to assign values to its properties (for example Figure 2). Microdata help technologies such as search engines and web crawlers to better understand what information is contained in a web page, hence contributing to improved search results. Microdata are an attempt to provide a simpler way of annotating HTML elements with machine readable tags than the similar approaches of using RDFa and microformats. A list of all microdata can be found at http://schema.org & http://data-vocabulary.org.

```
<div itemscope itemtype="http://schema.org/Event">
  <a itemprop="url" href="nba-miami-philidelphia-game3.html">
  NBA Eastern Conference First Round Playoff Tickets:
  <span itemprop="name"> Miami Heat at Philadelphia 76ers - Game 3 (Home Game 1) </span>
  </a>

  <meta itemprop="startDate" content="2016-04-21T20:00">
    Thu, 04/21/16
    8:00 p.m.

  <div itemprop="location" itemscope itemtype="http://schema.org/Place">
    <a itemprop="url" href="wells-fargo-center.html">
    Wells Fargo Center
    </a>
    <div itemprop="address" itemscope itemtype="http://schema.org/PostalAddress">
      <span itemprop="addressLocality">Philadelphia</span>,
      <span itemprop="addressRegion">PA</span>
    </div>
  </div>

  <div itemprop="offers" itemscope itemtype="http://schema.org/AggregateOffer">
    Priced from: <span itemprop="lowPrice">$35</span>
    <span itemprop="offerCount">1938</span> tickets left
  </div>
</div>
```

Figure 2 – Event microdata from http://www.schema.org/Event

Common Microdata Vocabularies

- **Creative works**:
  - Book http://www.schema.org/Book
  - Movie http://www.schema.org/Movie
  - Recipe http://www.schema.org/Recipe
- **Embedded non-text objects**:
  - AudioObject http://www.schema.org/AudioObject
  - ImageObject http://www.schema.org/ImageObject
  - VideoObject http://www.schema.org/VideoObject
- **Event:** An event happening at a certain time at a certain location. http://www.schema.org/Event
- **Organization:** An organization such as a school, NGO, corporation, club, etc. http://www.schema.org/Organization
- **Person:** http://www.schema.org/Person
- **Place** http://www.schema.org/Place
- **Product** http://www.schema.org/Product
- **Review** http://www.schema.org/Review

RDFa (or Resource Description Framework – in – attributes) is a W3C Recommendation that adds a set of attribute-level extensions to XHTML for embedding rich metadata within web documents. The RDF data model mapping enables its use for embedding RDF subject-predicate-object expressions within XHTML documents, and also enables the extraction of RDF model triples by compliant user agents.

The W3C RDF in XHTML Taskforce is also working on an implementation for non-XML versions of HTML [1]. The primary issue for the non-XML implementation is how to handle the lack of XML namespaces: http://www.w3.org/TR/xhtml-rdfa-primer/.

```
<div xmlns:foaf="http://xmlns.com/foaf/0.1/" about="#me" rel="foaf:knows">
   <ul>
      <li typeof="foaf:Person">
        <a property="foaf:name" rel="foaf:homepage" href="http://example.com/bob">Bob</a>
      </li>
      <li typeof="foaf:Person">
        <a property="foaf:name" rel="foaf:homepage" href="http://example.com/eve">Eve</a>
      </li>
      <li typeof="foaf:Person">
        <a property="foaf:name" rel="foaf:homepage" href="http://example.com/manu">Manu</a>
      </li>
   </ul>
</div>
```

Figure 3 – Sample FOAF RDFa elements

Vocabularies

- Friend-of-a-Friend (FOAF) - FOAF provides some basic machinery to help us tell the web about the connections between the things that matter to us.
- GoodRelations - GoodRelations is the most sophisticated and most widely used vocabulary for product, price, store, payment, and product feature information.
- Semantically Interlinked Online Communities - The SIOC initiative (Semantically-Interlinked Online Communities) aims to enable the integration of online community information.
- Description of a Career (DOAC) is a vocabulary to describe the professional capabilities of a worker.
- Media - An open standard for distributed media metadata.

- Audio - An open standard for distributed audio metadata such as audio recordings, albums, titles, and contributors.
- Video - An open standard for distributed video metadata such as movies, television, titles, contributors, and duration.
- Commerce - Another open standard for distributed commerce metadata such as price, currency, and amount.
- Music Ontology - Provides main concepts and properties for describing music (i.e. artists, albums, tracks, but also performances, arrangements, etc.)

3.4.2 State of the art on extraction of semantic information from blogs

There are many possible solutions to use in order to extract semantic information from web pages and particularly blogs. In essence, the process involves parsing the HTML source code of the web page and creating a DOM tree with all page elements and attributes. Afterwards, the software enumerates all elements trying to match known microformats, microdata & RDFa standards' patterns.

To avoid the difficult and error-prone task of HTML parsing and DOM tree creation, a number of open source libraries have emerged, giving the ability to extract semantic data transparently and hiding much of the complexity. Searching the web revealed multiple implementations and online services such as:

- PHP Microformats Parser http://www.phpclasses.org/package/3597-PHP-Extract-microformat-data-embedded-in-HTML.html
- http://www.alchemyapi.com/
- Perl Text::Microformat http://code.google.com/p/ufperl/
- Python library for extracting html5 microdata https://github.com/edsu/microdata
- Anything to Triples (Any23) library http://developers.any23.org/

Any23 was selected as the most suitable library to use in our prototype implementation for a number of reasons.

1. It supports a large number of formats:
   - RDF/XML, Turtle, Notation 3
   - RDFa with RDFa1.1 prefix mechanism
   - Microformats: Adr, Geo, hCalendar, hCard, hListing, hResume, hReview, License, XFN and Species
   - HTML5 Microdata: (such as Schema.org)
   - CSV: Comma Separated Values with separator autodetection.
2. It is currently used by a large number of applications and web services such as http://sindice.com/ and http://sig.ma/
3. It is actively developed and maintained but Digital Enterprise Research Institute (DERI) http://www.deri.ie/
4. It is written in Java and published under Apache License Version 2.0 http://www.apache.org/licenses/LICENSE-2.0

3.4.3 Prototype semantic data extractor application

The BlogForever semantic data extractor prototype application is a Java command line application capable of extracting semantic information from web pages. Application code references describing main classes and methods can be found at Appendix C.

Usage

The only requirement to use the prototype application is to have Java installed in your computer. Instructions:

1. Download software source code and binary from http://blogforever.eu/wp-content/uploads/2011/07/Blogforever-microformats.tar.gz
2. Unzip
3. Execute using the following command: ***./Extractor URL -o outputfile.xml***

Example

Execute the following command:

```
./Extractor http://rachaeljames.wordpress.com/2011/06/27/the-tree-
of-life/ -o outputfile.xml
```

Output file: outputfile.xml

```
<rdf:Description rdf:nodeID="node163o96nfgx12">
<rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
<mePage xmlns="http://vocab.sindice.com/xfn#"
rdf:resource="http://rikkicheri.blogspot.com/"/>
</rdf:Description>
```

3.4.4 Future work

The adoption of semantic information embedded in web pages is growing every day. Large companies like Microsoft, Google and Yahoo support microdata as well. Blogs also support some aspects of these protocols and their adoption is rising.

The investigation in microdata, microformats and RDFa as well as the prototype developed showcased the added value of supporting this technology in BlogForever. Our approach to semantic data extraction is to incorporate this functionality as an additional step in the data extraction process of the aggregator. The design and implementation of this feature will be part of Tasks 4.2 & 4.3, Design and implementation of the weblog spider component.

3.5 Spam detection

Spam detection is important for all blog crawling services. This is especially important when using ping servers or allowing access to an arbitrary list of weblogs beyond a defined list. As a separate process to fetching, spam filtering is about identifying and stopping blog posts that should not be further processed and stored in the repository.

Spam blogs (splogs) are an increasing problem when capturing blogs beyond a list of qualified weblogs. Splogs are generated with two often overlapping motives. The first motive is the creation of fake blogs, containing gibberish or hijacked content from other blogs and news sources with the sole purpose of hosting profitable context based advertisements. The second, and a better understood form, is to create false blogs that constitute a link farm intended to unjustifiably increase the ranking of affiliated sites [7].

Splog detection

There are several techniques for detecting Spam, and several freeware tools available such as blogspam.net. However most of these are too simple to be implemented in a weblog spider.

Another technique would be to implement our own Spam-blog Detection, and three different techniques are described in [7].

Given a blog profile, we present three (obviously non-exhaustive) scoring functions based on the heuristics stated below, denoted by *SF*1 to *SF*3. Each of them independently attempts to estimate the likelihood of a blog being a splog. For the ease of discussion, each state tuple in a given blog profile *b* is denoted as *ST*. A blog profile consists of the blog's URL and a sequence of blog state tuples, each of which is denoted as (t, *N, p.*spam_score*).*

*SF*1: *Inblog increment over time.* Based on a recent study on evolving graph data, links among legitimate blogs are expected to follow a densification law [1]. *SF*1 is therefore proposed to detect sudden shrink2 in inblog increment. Let *N'* be the estimated number of inblog links by a linear regression model during *W* time, *N* be the actual number, and N* be the average derived from *ST*, the likelihood of *b* being a splog is given as (*N'-N`*)*/N**

*SF*2: *Correlation between the number of posts appearing in top search results and inblog increment.* It is expected that a blog would attract more inblog links if its posts frequently appear in top-ranked search results, as those posts would expected to reach more readers. Hence, the blogs whose posts frequently appear in top-ranked results but do not attract inblog links are more likely to be splogs. *SF*2(*b*) = 1- C, where C is the correlation coefficient between the two values derived from *ST*.

*SF*3: *Average spam-post score.* As a legitimate blog often contains relatively fewer spam posts, the average spam-post score derived from *ST* can be used to detect splogs: *SF*3(*b*) = ∑ *p.*spam_score/ *(ST)*

An even more comprehensive insight to the problem and how to create a spam filter that can learn from new types of spam can be found in [9]. The 13 different features described there that can be used to identify splogs, and how these were tested and evaluated, are:

No.	Feature Description
1	Location Entity Ratio
2	Person Entity Ratio
3	Organization Entity Ratio
4	Female Pronoun Entity Ratio
5	Male Pronoun Entity Ratio
6	Text Compression Ratio
7	URL Compression Ratio
8	Anchor Compression Ratio
9	All URLs character size Ratio
10	All Anchors character size Ratio
11	Hyphens compared with number of URLs
12	Unique URLs by All URLs
13	Unique Anchors by All Anchors

Source: Table 1 in [9]

3.6 Ranking

Blog ranking is partly done to separate popular weblogs from irrelevant blogs or even spam. When searching through a blog search engine you might want the findings sorted by importance and relevance – for which ranking is used. Additionally, blog communities, directories and platforms also use blog ranking to structure blogs across popular topics. For example, French Wikio publishes, on the 5th of every month, its ranking of blogs in different categories.

To ensure a certain quality of the blogs' content, the ranking is created depending upon the number of tweets, and the weight and number of backlinks on a blog (the links that lead to that specific blog from different websites). As an example, for Wikio, generating a ranking is both for marketing as well as for securing an up to date and representative list of new blog posts. As for marketing**,** the bloggers can show their position in Wikio's ranking with a badge on their blog.

The method of ranking by links forms the basis for most search engines. Several web search ranking algorithms use link-based centrality metrics, including (in order of appearance) Marchiori's Hyper Search, Google's PageRank, Kleinberg's HITS algorithm, the CheiRank and TrustRank algorithms.

Link analysis is also conducted in information science and communication science in order to understand and extract information from the structure of collections of web pages. For example the analysis might be of the interlinking between politicians' web sites or blogs[16].

TrustRank is a link analysis technique described in a paper[17] by Stanford University and Yahoo! researchers for semi-automatically separating useful webpages from spam.

Another type of ranking is based on peer-to-peer (mutual sharing and linking between users e. g. peer networks). Although peer-to-peer systems available today (like Gnutella) are open, often anonymous and are lacking in accountability, some of those can still enable trustworthy relations. A research team from Stanford provides a reputation management system[18], where each peer in the system has a unique global trust value based on the peer's history of uploads. Any peer requesting resources will be able to access the trust value of a peer and avoid downloading files from untrusted peers.

[16] http://en.wikipedia.org/wiki/Network_theory
[17] Gyöngyi, Zoltán; Hector Garcia-Molina, Jan Pedersen, "Combating Web Spam with TrustRank", 2004
[18] http://en.wikipedia.org/wiki/EigenTrust

4 The selected weblog spider architecture and functionality

The main aim of this section is to present the selected approach for developing the weblog spider prototype application. First, a high level overview of the application processes and architecture is presented and discussed. Second, some key techniques and data extraction algorithms are elaborated. Third, the connections between the weblog spider prototype and existing work in D2.1 (Weblog Survey) as well as D2.2 (Weblog Data Model) are considered. Finally, future work-plan for developing D4.1 Weblog Spider Component Design) and D4.3 (Initial Weblog Spider Prototype) are presented.

4.1 High Level Overview

The weblog spider functionality can be divided into two main tasks: capture all available blog content elements and, then, continuously monitor and secure blog updates, including both blog posts and comments to each blog post.

The application processes can be divided into six main steps.

Step 1: Inputter

The Inputter is where all the blog URLs are inserted into the system. It is accessible through a web interface referred here as Spider Portal.

There are two different ways of inserting URLs. First option is for the end-user to manually insert a list of defined blog URLs (blog source – technically called blog hosts). The second option is to connect to a ping server, which receives blog URLs and blog post updates from a large number of blog authors and blog platforms. Blog authors post to certain ping servers to ensure their posts get to a wider reader audience. The spiders subscribe to the selected ping servers in order to receive an ongoing feed of URLs with new updates.

The main components of the Inputter are as follows.

- **Blog Host Addition.** “Add blog hosts” is the section in the Spider Portal where a list of blog URLs can be inserted, for the initial step of monitoring. This application just holds this list until it is received and handled by the source database.
- **Ping Server Management.** Ping servers are external to the spider. There is, however, a Ping Server Management section in the Spider Portal for handling the ping servers the spider subscribes to. When subscribing to a ping server the spider gets access to all new blog URLs received by the ping server from any blog author or blog platform, according to the rules set in the Spider Portal. Such rule can be: get Wordpress-URLs (see user guide) or a source name keyword.

Step 2: Source Database checking

Step 2 prequalifies the URLs received from the Inputter. The URLs are firstly checked by the Source Database in the System Manager. In the Source Database the URLs are sorted according to 3 categories: if the URL is previously marked as spam, it will be rejected; if the URL is a new blog post from a known blog host in the database, the URL will be processed by the Worker; if the URL is unknown, it will be processed by the Host Analyzer, Step 3.

The main component of this Step is the following.

- **The Source database** is a database that holds the entire list of all blog host sources that have been processed by the spider, and related blog post URLs, metadata and filtering rules for ongoing monitoring and extraction. It also holds the list of blacklisted blog hosts.

Step 3: Host Analyzer

The Host Analyzer interacts with the Fetcher in the Worker (also part of Step 4). The fetcher initially downloads the blog content from each unknown URL, in order to identify which blog host it relates to. When identifying the blog host, the Analyzer can identify the URL of the RSS of the blog host, and from this RSS the list of all URLs from each blog post.

The blog hosts URLs are now included into the source database of the System Manager, with a relating link filter describing the structure of this blog host. Each relating blog post, however, are resent to the Worker and fetcher for downloading content.

The main components involved in this step are as follows.

- **The Scheduler** is the unit that manages when the spider needs to check out certain URLs for any new updates. For most URLs the ping server delivers all updates automatically, except for three different areas where the spider needs to do frequent polling and downloading to look for updates: URLs inserted manually from the input application not covered by ping servers, blog comments from most URLs as long as the ping server does not push updates of this blog element, and thirdly, controlling some blog hosts that should be updated by a ping server to see if there are any omissions. Frequency of checking the URLs is rule-based.
- **The Fetcher** downloads the RSS and the entire HTML, matching them and analyzing which rules to apply to get the right URLs into the source database and right content including all blog elements.

Step 4: System Manager

This is a module rather than a step, as the System Manager will route all URLs of a blog host to the Worker, just storing input in the source database. However, the System Manager both handles the source database and the Scheduler. The Scheduler ensures that even blog content that is not offering RSS feeds is also covered and checked regularly.

Step 5: Worker

This step actually involves two parts. The first part includes the Fetcher, to download all content per URL sent from the System Manager. This includes firstly retrieving RSS, but more important, downloading all HTML. The captured RSS will be used initially to match the downloaded HTML, making sure the blog post has been identified correctly. The HMTL is then analyzed using machine learning techniques, such as ID3, in order to find the full text of the blog post, related comments and other relevant blog elements. This process may also include a spam detection, to filter out new types of spam.

The second part of the Worker is to parse all found HTML-based blog content and convert this into XML. This is handled through the leading parsing engine, the HTML Agility Pack. Also included in this part is the extraction of microdata with a semantic engine that extracts certain entities. The downloaded content is firstly compared with the RSS.

The main components involved in this step are as follows.

- **The Parser** converts all content in HTML into machine readable XML-format. Additionally, it extracts metadata and microformats to further enhance and enrich the XML.
- **The Semantic Engine** can enable entity extraction of the blog content from each web page. It is not implemented in the prototype as further investigation by BlogForever is necessary, which looks into the potential with upcoming HTML5 versus third party engines such as OpenCalais. Scaling and bottle neck is a major issue, so the project will also consider moving this module towards the repository.
- **The Spam Filter** module will include extra testing of links and content beyond what is done by host analyzer and fetcher. The more accurate the two others are, the less is needed by the spam filter.

Step 6: Exporter

The Exporter stores the parsed XML, including additional content. The storage is temporary, just enough to ensure the XML is available when the external Repository requests updates. The XML is then distributed onto whichever repository, search engine or database, that the end client is requesting from.

The only component of Step 6 is the Exporter.

- **The Exporter** stores the XML temporarily in an FTP server for repositories to come and fetch the latest data file of the XML captured by the spider. When the file is exported each data file is expected to be deleted. Any data file older than 24 hours on the FTP-server are deleted.

The steps and the components of the spider are presented in The BlogForever spider component forms part of the pre-ingest activity leading to the creation of the submission information package as prescribed by the Open Archival Information System (OAIS) model. The integration of the weblog spider into the OAIS model is illustrated in Figure 4.

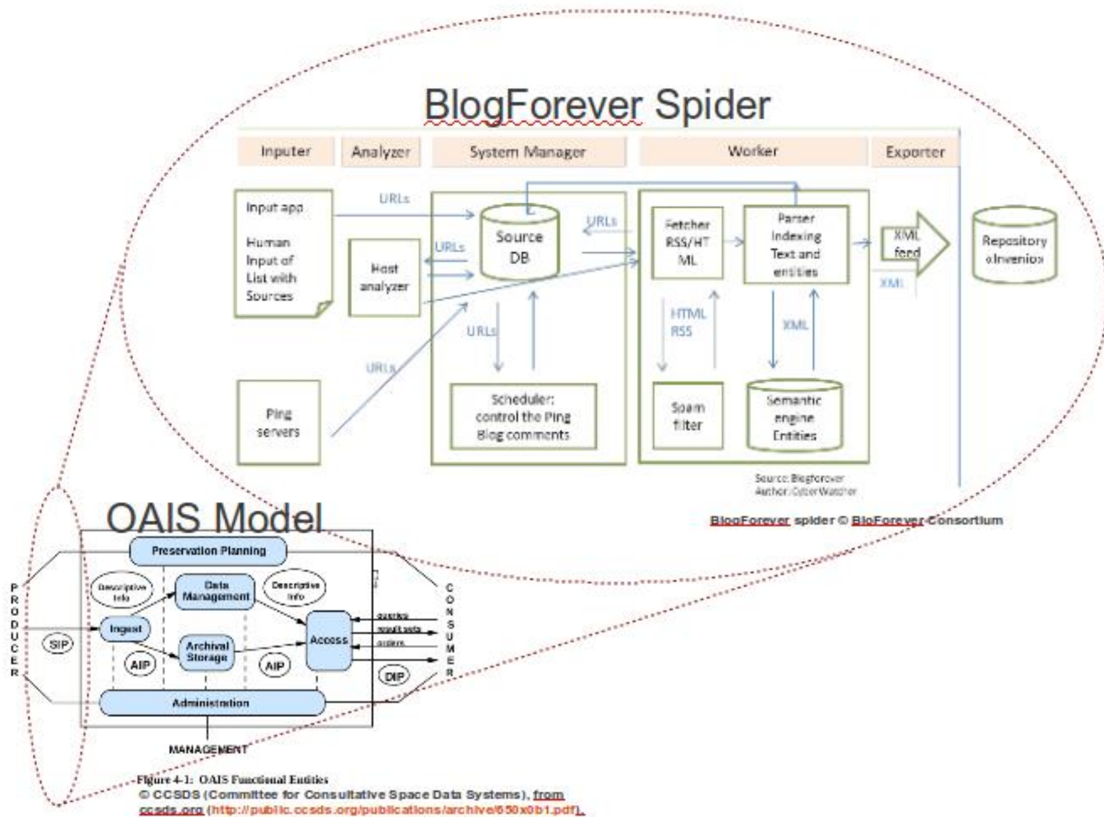

Figure 4 - A single server BlogForever spider within the context of the OAIS (Open Archival Information System) model.

.

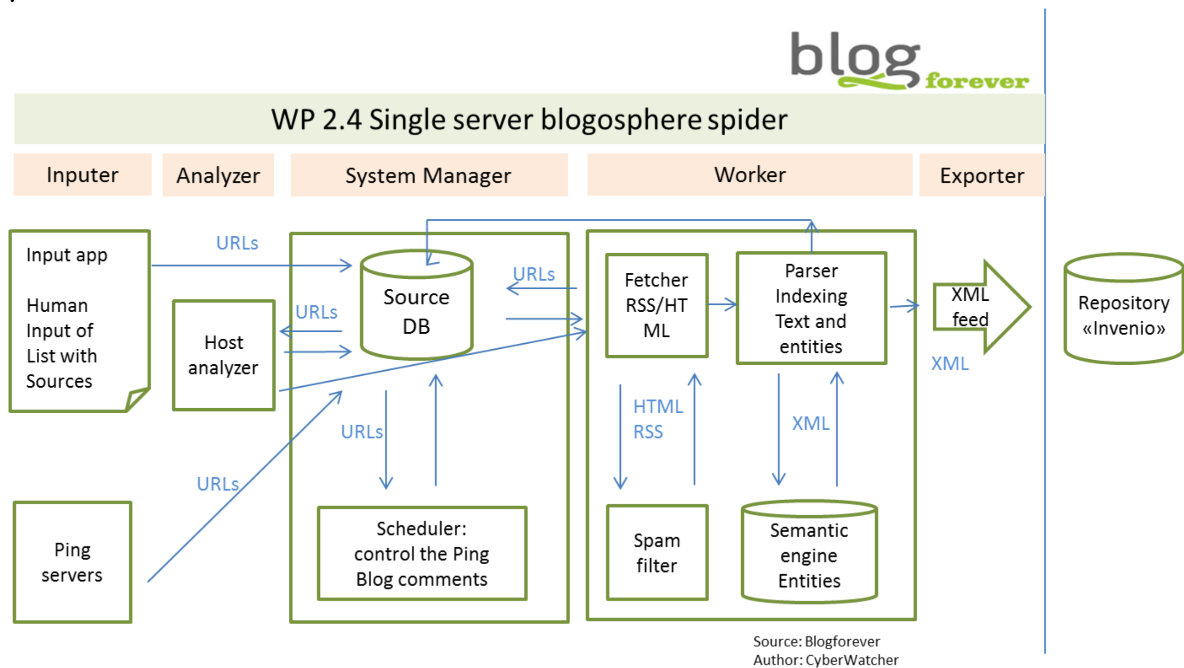

Figure 3 - Single-server Weblog Aggregation Prototype

The BlogForever spider component forms part of the pre-ingest activity leading to the creation of the submission information package as prescribed by the Open Archival Information System (OAIS) model[19]. The integration of the weblog spider into the OAIS model is illustrated in Figure 4.

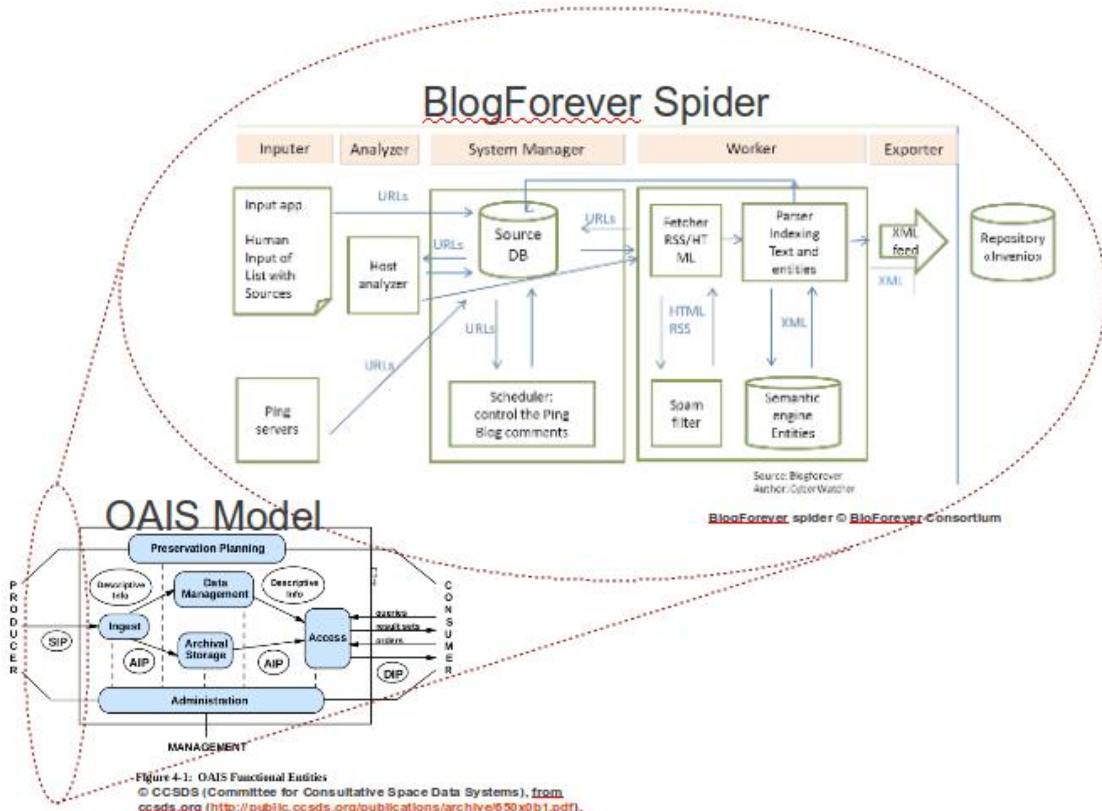

Figure 4 - A single server BlogForever spider within the context of the OAIS (Open Archival Information System) model.

4.2 Single-server software architecture

In the previous chapter, an overview of the weblog spider aggregator was presented. The main components of the application are presented in greater detail in this section. Detailed descriptions of the source code, structure and main classes can be found in Appendix A. Detailed instructions on installation and usage can be found in Appendix B.

4.2.1 Inputter and web interface

The Inputter is available through a web interface, Spider Portal. It enables blog monitoring in two ways:

a) manually, through user defined list of selected blog sites that can be maintained via the application interface; and
b) automatically, through subscribing to all new blog sites from a ping server.

The list of blog sources inserted manually will represent a limited volume of qualified blog sites while on the other hand the automatically generated blog sites through ping servers will represent a high volume.

[19] CCSDS (2002) "Reference Model for an Open Archival Information System (OAIS)", CCSDS 650.0-B-1 (2002): http://public.ccsds.org/publications/archive/650x0b1.pdf

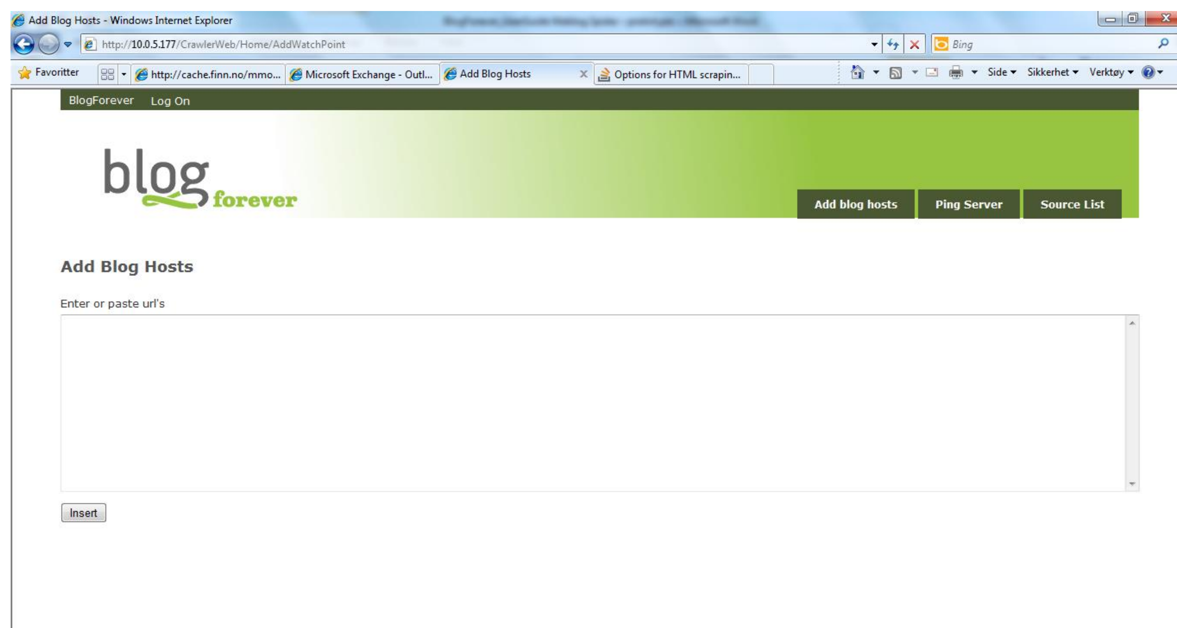

Figure 5 – Inputter view in the Spider Portal – a web interface for inserting and managing selected blogs

Manual monitor of user defined blogs

The input application enables the user to insert a list of blogs sites to be monitored and crawled. The list can be comprised of blog URLs and can be inserted via the application form or from a spreadsheet or comma separated file list.

The manual Inputter of the spider prototype currently supports only inserting the URLs of the blog. In WP4 the manual Inputter will allow insertion of more criteria such as language, location, topic or type of blog/author (e.g. academic, businessman), as shown in Figure 6.

Blog inserter				
List by 2011 - 11				
Blog URLs	**Language**	**Location**	**Topic**	**Author institution**
http://www.drugwonks.com/	English	Sweden	Drug research	Stockholm University
http://primeinc.org/pharmacistblog.html	English	UK	Cancer research	Cambridge
http://www.edrugsearch.com/edsblog	English	US	Cancer research	UCLA
http://pharmacytechnology.blogspot.com	English	US	Pharmacy Technology	Arizona State University
http://www.medhumanities.org	English	US	Cancer research	Stanford

Figure 6 - User defined blogs for the Inserter

The Inserter will send these blog URLs to the Source library to check if they already registered there. If they already exist in library, an output XML file will be generated and stored in the Exporter immediately. This will also create an FTP XML feed for the repository. If they are not registered in the Source library, they will be sent to the Host Analyzer for processing.

Automatic blog monitoring using ping servers

The Inputter module in the Spider Portal includes a manager for subscribing to one or several ping servers. An appropriate user interface allows manipulating the list of ping servers monitored by the

application. These ping servers will be a constant feed of blog URLs, including a mix of new blog hosts, blog posts, and spam blogs (often called splogs).

All the URLs coming from the ping server will be sent to the Source library to check whether they are already listed there and how to handle them accordingly. For instance, one handling rule could be to process only new blog posts from the blog hosts already listed in the source library and not anything else from unknown/unregistered blogs.

Another rule could be to process all URLs except the ones which are black listed in the Source library. The black list contains all URLs that are not recognized as blogs or contained problematic content which could not be processed by the spider.

All new URLs not identified as blog posts to identified blog hosts will be sent to the Host Analyzer, whereas blog posts to identified blog hosts will be handled by the Fetcher.

4.2.2 Host Analyzer

The Host Analyzer is the gateway for all new URLs. It qualifies or potentially blacklists new unknown URLs. Through a process of fetching HTML and analyzing content and URLs using machine learning processing, such as ID3, the Host Analyzer will accomplish the following steps in order to analyze a blog page.

1. Does this URL contain content likely from a blog or potential spam?
2. Does the URL relate to a feed, a specific comment, a post or a blog host? Identify the link of the blog host (or blog platform, when host is not a domain but part of a platform – e.g. Blogger.com is the platform but the blog host needed to find the blog posts is for example blogger.com/nokia).
3. Identify the link to the RSS for the blog host – which will deliver the latest blog posts?
4. Identify RSS for the comments.
5. Identify which link filter rules to apply for this blog host – to be added to the source library for future monitoring

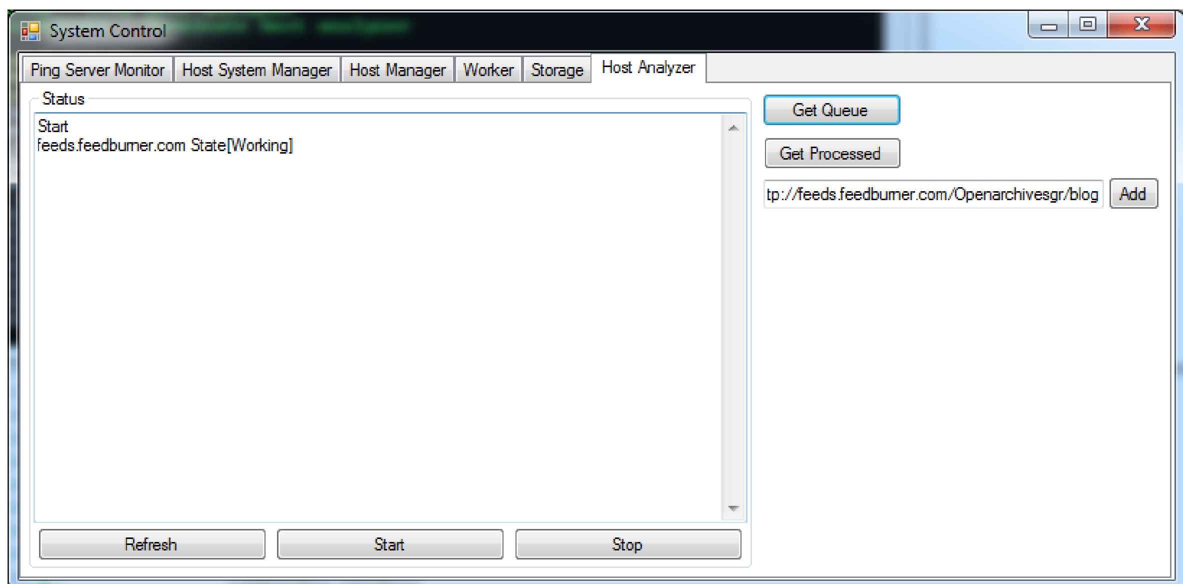

Figure 7 - Host Analyzer, processing links to qualify blogs and establish link filters for monitoring

4.2.3 System Manager to operate the spider and capturing – including a blog-source library

The System Manager is the centre of the entire spider. It communicates with all other modules in the spider. It also contains the main information and identifier of the Spider – e. g. which versions of the Spider should handle remote updates.

Source library

This is the database for all blogs to be monitored and processed towards the Exporter.

The source database includes a library of blog hosts, blog posts relating to it and how blog comments relate to each of the blog posts. The library also contains:

1. All information from the Input Application of manually inserted blog hosts
2. Link filters
3. Latest blog post and comment processed to exporter
4. Additional source information found from the processing – e. g. Parser, Semantic engine etc.
5. Communication with the Scheduler
6. List of black listed links – Spam list

The source library will also support future scaling through connecting to a centralized source library for distributed processing, exchanging qualified cluster of sources and updating to a new version of the Spider.

Scheduler

The Scheduler manages the monitoring process of the blog library.

Through the ping server the Spider will receive updates of blog posts from blog hosts that are connected to the actual ping server, so fetching will not normally be needed.

But the Scheduler will run control fetching of blog hosts with no updates within a defined time range – e. g. weekly. This will check if the blog is active or shut down, and if the ping server is missing out updates. Additionally, the Scheduler will handle blogs not connected to ping servers and comments which normally are not handled by the specified ping servers. The Scheduler will then request the RSS Comments and RSS for non ping server listed blog hosts for updates, and then let the Fetcher download the HTML.

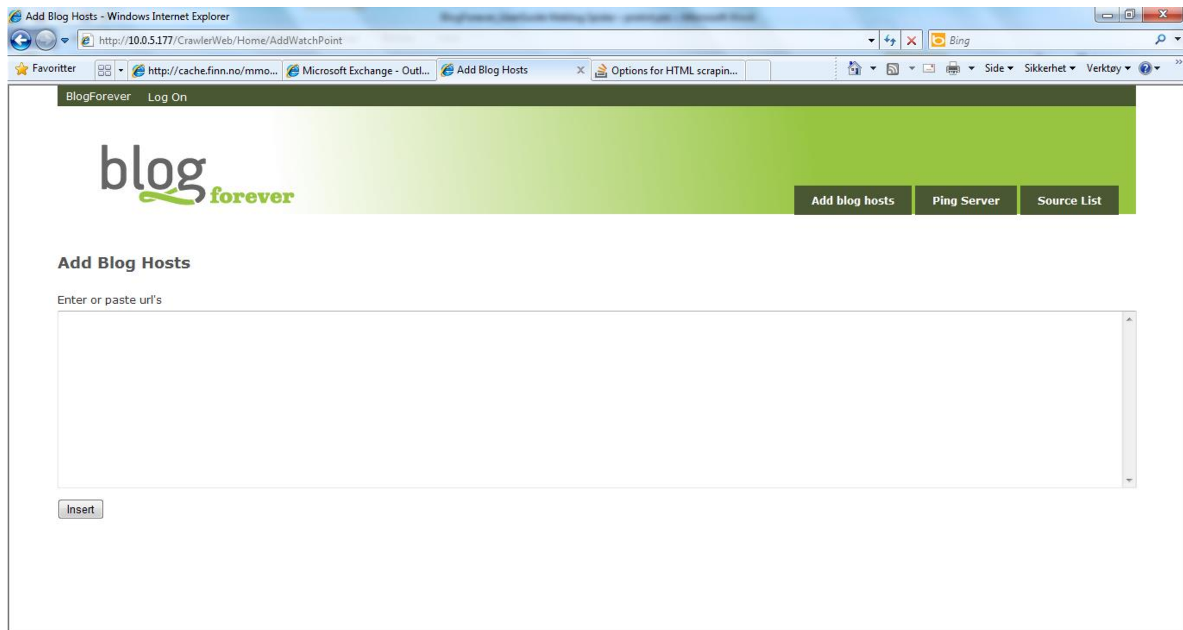

Figure 8 - Spider Portal interface – handling the System Manager User Interface

4.2.4 The Worker for fetching

The Worker handles the actual content from the Fetcher downloading HTML, to Spam filtering, Parser processing HTML to XML and adding additional blog elements, and finally to storage and the Exporter.

Fetcher

The task of the Fetcher is to download RSS, as well as HTML from the actual host. The Fetcher will be used for two different steps of the Spider: initially when Host Analyzer is qualifying URLs and then for downloading and analyzing content.

Fetcher for defining links and link filters

The Fetcher downloads HTML to find all links from the websites and identifies which of these constitute a blog host, RSS to the blog host, each blog post and RSS for blog comments. In this process the Spider will use machine learning and ID3 to create rules for recognizing the different types of links. It will firstly find the blog host, the RSS to the blog host, then the blog posts and the RSS to comments.

Fetcher for capturing full text and blog elements

In this process the Fetcher will download and compare RSS with the HTML of both the blog post and the blog host. The main steps of the process are listed below.

1. The Levenshtein distance will be used to detect which RSS posts relate to similar post in the HTML version.
2. Connecting to the Spam filter module will check whether the rest of the content, not included in RSS, is potentially relevant or spam.
3. Machine learning rules and platform specific rules (WordPress and Blogger) will be used to use the extract from RSS to define the entire full text of each blog post.
4. The rest of blog elements will also be used in similar machine learning processes, such as ID3, to find related comments, external links, dates, author pictures and attachments.

5. As some elements in the RSS may be additional to the full text – e. g. author and date – both parts of the RSS and full text are included in the Parser process and Exporter.

Based upon the filters such as those for Wordpress, or comparing RSS with HTML, the Worker can separate and download all different blog elements.

There are exception rules. A blogosphere spider development will be ongoing to be able to improve how well it captures all variations of blog posts. Sometimes RSS doesn't exist or is broken. Rules can still be used in that case, but with less control of the quality. Sometimes there is even information in the RSS that is not published in blog posts. We plan to establish a fall back rule for such a case. Author was only in RSS, not in the full text or the website. Initially we will not categorize such unknown RSS-content, but only index to be searchable. Extra RSS information can then be defined as one type of meta-data. This process also includes a spam-filter process, using available spam tools.

Spam-filtering

In connection with the fetching process, the removal of spam is handled in several steps.

The first line of defense is the Host Analyzer and Fetcher using the ID3 algorithm to identify which URLs match the link filters for blogs in acceptable formats. Unless the fetcher can identify a valid RSS for blog posts or blog comments, the source will be rejected or even blacklisted. Potentially, there will even be a pre-fetcher filtering, ruling out URLs that have characteristics of a splog. Among such characteristics can be an unreasonably high volume of new updates or links from the ping server.

The second involves matching RSS to the full text content from the HTML-site using Levenshtein distance. Without such a match there will be no further capturing of blog content.

Parser – that process the captured material and storage of XML-version

The Parser includes processing the captured HTML and converting all relevant captured blog elements into XML format storage for the Exporter. The process uses HTML-aware as an established technique to convert HTML into XML

Among the processes that the Parser may be doing in the final spider are the following.

1. Detecting further internal links to return to the System Manager for further work or fetching such as RSS comments and archived blog posts.
2. Detecting external links that can be categorized without having to be crawled separately – e. g. pictures, videos, blogrolls, Facebook, Twitter, LinkedIn and blog-networks.
3. Identifying attachment links in formats such as PDF and doc.
4. Utilizing microformats to expose more tags and meta info from the blogs.

The parser process may also include connecting to an external semantic engine, such as Open Calais, in order to identify elements within the text. However, such text mining in the crawling process is likely to become a bottle neck for parsing toward final XML-feed. This will affect scalability and stability.

In the final version, some light version of a semantic engine for entity extraction might be included. However, a better approach will be to implement such a semantic engine in front of repository. This can enable post processing of already stored data and batch jobs at any stage later.

The Parser is also a final step of processing text toward the repository. This includes converting all captured blog elements from HTML format into XML. Furthermore, it defines the format of the XML and additional data to be included in the XML of the blog post or comment.

Adding information into the final XML includes:

1. Comments to include the links to the blog post;
2. Semantic entities, meta tags or links to attachments;
3. Parts of the RSS to be merged or attached to the XML.

Even though the XML generated from the HTML is richer and more extensive than the RSS, there are some elements that make the RSS relevant for including in the final XML:

1. Date information is more structured in RSS than HTML;
2. For some blogs the author is in RSS but not in HTML;
3. A snippet of the text and XML is better for a scaled entity extraction than the complete XML.

4.3 Multiple-Server software architecture

The BlogForever Spider prototype is structured as a single server application to simplify local installation and maintenance. A single server Spider will however have scaling limitations as multiple processing is required when capturing and analyzing the structure of new blog sources.

The Spider prototype is not optimized, but scaling tests reveals the need for and benefits of a multiple server architecture. Such a structure enables scaling especially when handling new blog sources.

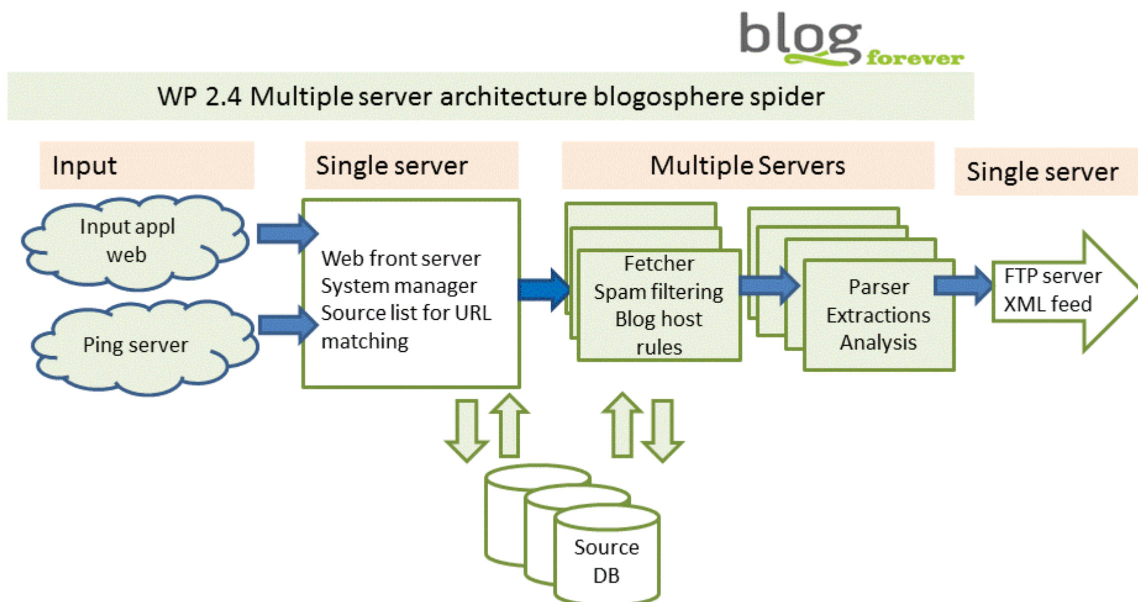

Figure 9 - Multi server architecture of the weblog spider

Front end web server

The front end server will mainly handle volumes of URLs from the ping server and input application. With a scaled volume of input URLs from one or several ping servers the front end server will sort which URLs are already detected in the source database and which are

new blog sources that need to be processed by one of the distributed servers handling fetching of the HTML and analysis of the host.

The front end server also hosts the Spider Portal where end users can insert new blogs and manage the spider towards the output XML feed.

The front end server may include the source database. However, this database can vary extensively with high volume input from ping servers, as well as serving all steps of the processing. It is therefore recommended to distribute the source database with just a small version kept in the front end server. This front end only holds URLs of already indexed sources, URLs of blog posts already indexed and URLs blacklisted as spam or splogs.

Fetcher and blog analyzer

The front end server will distribute all new URLs not found in the source database towards a number of distributed servers. These Fetcher-servers will download the HTML and analyze the host for structure and potential spam.

The analysis of the blog hosts includes matching patterns towards existing patterns in the source databases to automatically create correct processing rules for the new blog source.

The Fetcher-servers will also handle new blog posts from existing blog sources. The HTML of such blog posts and blog comments will be downloaded according to established rules in the source database.

All content fetched and processed will be handled by the Parser servers. The required number of distributed servers for the process mainly depends on what is included beyond converting the HTML content into XML.

Such extra processes, which require extra numbers of servers, analyze all blog elements, detecting meta tags and any entity extraction and semantic processes.

FTP server

The output from the parser will be stored on an FTP server, accessible for exporting onto any external application or repository.

The FTP server could potentially be the same server as the Front end server. However, the scaling will be much better for handling high volumes with a separate FTP server, and there will be no architecture issues with a separate FTP server.

4.4 Theory/Approaches/Methods used

The basic underlying approach used to capture blogs is subscription to ping servers. From ping servers a flow of URLs will include new blog sites, new blog posts and links to RSS feeds. With this approach a fetcher can capture high volume of blog content. This will be significant for parsing XML and exporting blog content. However, this approach needs additional methods or will otherwise lead to large amount of content without any identification, structure or removal of spam.

By the maturity of blog usage, with more dedicated, sophisticated and even professional users, it is imperative to segment blogs – initially by adding user required blogs that are tagged according to well defined criteria. This method is well established in news monitoring, research and market intelligence and enables an extensive source library. The spider will have an input application to insert lists of blogs with a few additional metadata.

The validation of input HTML and the identification of specific elements within the HTML document structure are done using a Modeling-based approach where HTML elements conforming to specific structure are searched inside the Document Object Model (DOM) of each blog page. Other HTML parsing methods are also used in order to qualify and structure the input so the output from the spider is as structured and enriched as possible, capturing all blog elements, updated and with marginal spam.

The approach is to find all blog elements from just a list of URLs, and to identify the RSS and use this to match with the fetched content of the entire blog site in HTML. This matching method will use a growing range of rules based upon decision trees and machine learning techniques. Such a method enables matching to improve constantly and handling the vast number of variations and break of standards caused by blog authors.

Scaling the semantic engine is not built-in to the spider. For the future, HTML5 may enable the parser itself to extract more entities. However, any analysis processing before storage will automatically become a bottleneck to the flow of new updates from ping servers. This may pile up and create instability. Therefore, unless the spider is set up to a limited number of sources, the semantic engine should be put onto the platform after storage and after being exported to a repository.

Spider Architecture

The spider architecture is based upon a freeware version which is able to be downloaded onto a single machine. This requires a small footprint and a balanced use of resources. For example, the volume of URLs from a ping server may cause bottleneck issues in the Fetcher.

However, the architecture also supports each server to be upgraded to a paid scalable version and even to connect to a central spider farm. This is based upon a current method and technology already developed for the CyberWatcher news spider. For a scaled version, the architecture supports adding more resources and server for parts of the spider, e.g. the Fetcher.

4.5 Technical platforms and development tools

The spider prototype is based upon the .NET Framework. Although it is developed by Microsoft, .NET supports several programming languages and allows language interoperability where each language can use code written in other languages.

One of the benefits of the .NET Framework is support for memory management. However, applications running in a managed environment tend to require more system resources than similar applications which access resources more directly.

The spider prototype is developed by experts in the .NET Framework, who have already created highly scale spiders using .NET. The .NET experience of the programming team and the support for memory handling are the main reasons for the choice of the framework.

The only consequence for the users will be that downloading the server requires a Microsoft server.

4.6 Connection with previous BlogForever tasks

The development of the weblog spider prototype is highly interconnected with other BlogForever project tasks. First of all, deliverable D2.1 Weblog survey was used as a basis to identify *software platforms*, *encoding standards*, *third party services* and *libraries* used in blogs. The results of this survey were used to define the environment and targets of the weblog spider prototype application.

4.6.1 Connections with the D2.1 Weblog survey

The BlogForever Weblog Survey (Task 2.1) was very successful in informing the project regarding: [i] the common practices of blogging and attitudes towards preservation of blogs; [ii] the use of technologies, standards and tools within blogs; and finally, [iii] the recent theoretical and technological advances for analysing blogs and their networks.

The survey listed 25 blog elements to potentially be captured by the spider and, then, to be preserved in the repository. This enabled us to focus on the most important elements first, improving the value of the aggregated data.

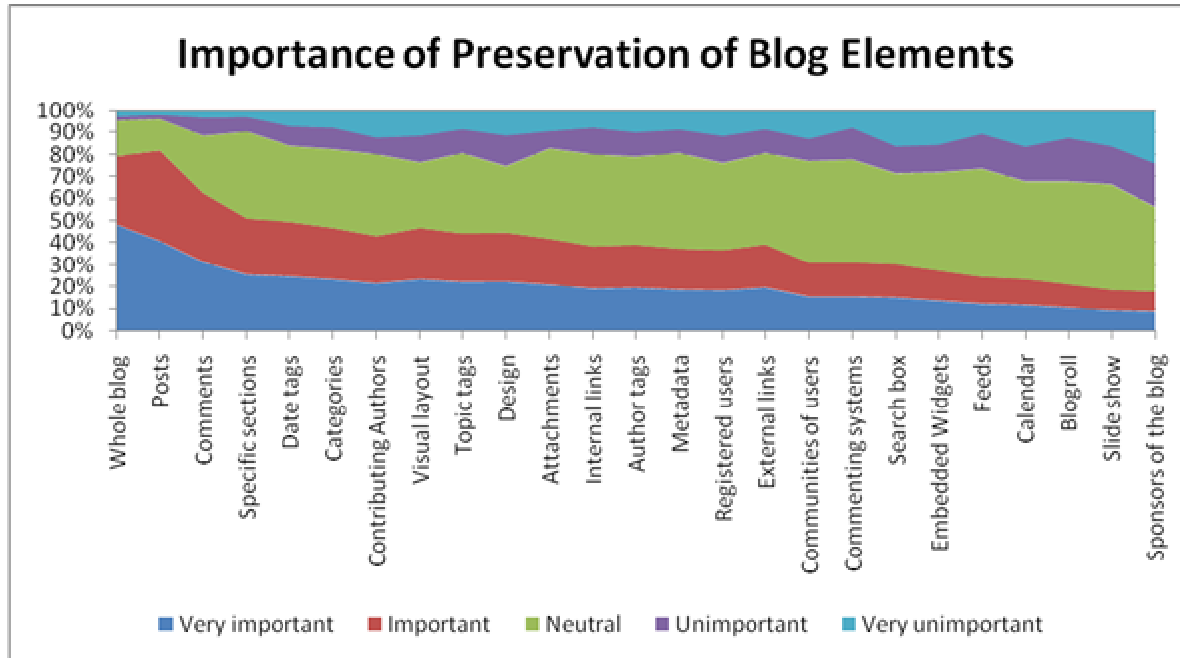

Figure 10 – Importance of blog elements to be preserved, according to D2.1 Weblog Survey

As seen in Figure 10, the weblog survey shows a natural path of important blog elements rated from the two main content elements – Posts and Comments – to the 9 least interesting elements being mostly functionalities with no content included. Accordingly, the weblog spider prototype will try to capture all the content, while other functionalities are not a priority.

Another important finding of the weblog technical survey conducted in Task 2.1 was the analysis and results of the blog publishing software use. Among the findings was that 50% of blogs use two main platforms, Wordpress[20] and Blogger[21]. Based upon these findings, the blog data extraction rules are required to facilitate the intricacies and special behaviors of these platforms.

[20] http://www.wordpress.org

One of the key characteristics of blogs is that they use some type of XML feed, RSS or Atom. This was confirmed in the weblog technical survey and the specific types of RSS and Atom versions and encodings were identified, enabling the weblog spider prototype to support them. Additionally, the weblog spider prototype will utilize XML feeds to analyze blog content and identify specific information elements. The spider will compare the information encoded in XML feed elements against the blog HTML in order to identify information such as the structure of the blog, the length of a blog post and the new blog content.

4.6.2 Connection with D2.2 Weblog Data Model

The BlogForever Weblog Semantics (Task 2.2) has already created D2.2 Weblog Data Model, describing in detail individual blog components and their interconnections. The entities comprising the weblog data model are listed in Table 6.

Table 6 – **Main blog data model entities**

Entity	*Attributes*	*Description*	*Prototype support*
Blog	title	*Title of the blog*	*Yes*
	subtitle	*Subtitles of the blog*	*No*
	URI	*URI of the blog*	*Yes*
	status_code	*Status defines whether the blog ceased to exist(if in RSS)*	*Yes*
	language	*Retrieved language field, as defined by the blog(if in RSS)*	*Yes*
	charset	*Retrieved charset field, as defined by the blog*	
	sitemap_uri	*URI of the blog sitemap if exists*	*No*
	platform	*Platform of the blog powering service, retrieved where available*	*No*
	platform_version	*Versioning information about the platform*	*Yes*
	webmaster	*Information about the webmaster where available*	*No*
	hosting_ip	*IP address of the blog*	*No*
	location_city	*Location city based on the hosting details*	*No*
	location_country	*Location country based on the hosting details*	*No*
	last_activity_date	*Date as retrieved from the blog*	*Yes*
	post_frequency	*As retrieved from the blog*	*Yes*
	update_frequency	*As retrieved from the blog*	*No*
	copyright	*Notes of copyright as retrieved from the blog*	*No*
	ownership_rights	*Notes of ownership rights as retrieved from the blog*	*No*
	distribution_rights	*Notes of distribution rights as retrieved from the blog*	*No*
	access_rights	*Notes of access rights as retrieved from the blog*	*No*
Entry	title	*Title of the entry*	*Yes*
	subtitle	*Subtitle of the entry if available*	*No*
	URI	*Entry URI*	*Yes*
	date_created	*Retrieved from the blog or obtained from the date/time crawling*	*Yes*
	date_modified	*Retrieved from the blog or obtained from the date/time crawling*	*No*
	version	*Auto-increment: derived version number (versioning support)*	*No*
	status_code	*Information about the state of the post: active, deleted, updated (versioning support)*	*No*
	geo_longitude	*Geographic positioning information*	*No*
	geo_latitude	*Geographic positioning information*	*No*
	visibility	*Information about accessibility of the post*	*No*

[21] http://www.blogger.com

	has_reply	*Derived property (also SIOC)*	*No*
	last_reply_date	*Derived property (also SIOC)*	*No*
	num_of_replies	*Derived property (also SIOC)*	*No*
	child_of	*ID of entry parent if available*	*No*
Page	template	*Information about the design template if available and if different from the general blog*	*No*
Post	type	*Type of the post if specified (e.g. WordPress): attachment, page/post or other custom type*	*No*
	posted_via	*Information about the service used for posting if specified*	*No*
	previous_URI	*URI to the previous post is available*	*Yes*
	next_URI	*URI to the next post if available*	*No*
Comment	subject	*Subject of the comment as retrieved*	*Yes*
	URI	*URI of the comment if available*	*Yes*
	status	*Information about the state of the comment: active, deleted, updated (versioning support)*	*No*
	date_added	*Date comment was added or retrieved*	*Yes*
	date_modified	*Date comment was modified or retrieved as modified*	*No*
	addressed_to_URI	*Implicit reference to a resource*	*Yes*
	geo_longitude	*Geographic positioning information*	*No*
	geo_latitude	*Geographic positioning information*	*No*
	has_reply	*Derived property (also SIOC)*	*No*
	num_replies	*Derived property (also SIOC)*	*No*
	is_child_of_post	*Indicates information about the parent post*	*Yes*
	is_child_of_comment	*Indicates information about the parent comment, from RSS*	*No*
Author	name_displayed	*Name of the poster as displayed*	*Yes*
	email_displayed	*Email address of the poster as displayed*	*Yes*
	is_anonymous	*Boolean property to indicate anonymity*	*No*
Content	full_content	*Content as extracted*	*Yes*
	full_content_format	*Content format (i.e. HTML, XML)*	*No*
	note	*Additional notes if available*	*No*
	encoding	*Information on encoding of the content*	*Yes*
	copyright	*Notes of copyright as retrieved from the blog*	*No*
	ownership_rights	*Notes of ownership rights as retrieved from the blog*	*No*
	distribution_rights	*Notes of distribution rights as retrieved from the blog*	*No*
	access_rights	*Notes of access rights as retrieved from the blog*	*No*

Since this is a prototype, not all entities are supported but most of the relevant entities available in the blog host will be included incrementally in the final BlogForever weblog aggregator which will be designed in Task 4.2 (Design of the weblog spider component) and implemented in Task 4.3 (Implementation of the weblog spider component).

4.7 Development challenges and future work

The spider prototype is the first version of the BlogForever spider. The platform is based upon well-known methods and modules. However, further tools, as well as tuning and scaling, are needed.

The developed prototype captures and populates a repository with weblog content using a combination of weblog specific technologies such as ping servers and RSS feed monitoring, as well as traditional web data extraction techniques such as scheduled crawls.

The key challenge of the objective is to capture and identify automatically all type of content elements and metadata across the variation of blog hosts. Analysing blog pages and extracting information is demanding both on resources as well as on handling all types of exceptions. For

instance, blog comments and some blog elements are not exposed to the spiders due to JavaScript coding.

Another related challenge is scaling. The more analysis is required for finding and identifying blog elements, the longer processing and more hardware is required. This can be handled through multiple servers, but the objective for the prototype has been to enable this on a single server. One of the key challenges of next development is the implementation of the multi-server architecture.

The prototype spider handles the above challenges, although further work is required to improve both analysis and scalability, as described in Section 4.7. This will be done in the next stages of the project in WP4, especially Task 4.2 (Design of the weblog spider component) and Task 4.3 (Implementation of the weblog spider component). Further work planned towards WP4 includes:

Expand the Host analyzer

The Host analyzer has used machine learning techniques to establish a few sets of rules detecting content and blog elements. Based on this experience with the prototype spider analysing a number of blogs more rules can be added to the platform. This can enable the prototype to handle increased volumes and different blog setups that were not handled in the initial prototype version.

Extend the source database

The prototype source database is a first version, based upon serialized lists of sources. The final BlogForever spider requires a better design for handling two parallel processes: URL matching to find new versus existing URLs and identifying and storing rules per blog host. The design should even be prepared for further scaling including multiple database servers.

Blog-ranking

For the techniques for blog ranking described in this report no tool has been implemented. Implementation of ranking requires a scaled, operational platform on which to test. The prototype will be used to test the listed techniques and additional ones to implement a blog ranking system in the BlogForever Spider.

Evaluating current spam filtering for further improvements

The host analyzer and filtering rules in the Fetcher will prevent most obvious spam. However, further testing is expected to demonstrate the need for more extensive spam filtering.

Extend parsing of additional blog elements

Whereas the prototype has fetched and downloaded most of the blog elements, future work will extend the parser to extract entities both from metadata as well as microformats and additional blog elements.

Testing semantic engines in the parser vs. the repository

The prototype spider has limited capacity to handle text mining and entity extraction when there is a high volume flow, and even in the final version the semantic engine will be a limited part of the entire Spider. However, some semantic engine might be connected and evaluated with a similar implementation in the repository.

5 Conclusions

The primary objective of this report has been to present the design and development of the BlogForever weblog spider prototype; as well as associated methodologies for blog monitoring and data extraction that have been considered in this development.

One key requirement having impact on the architecture and preferred technologies for the spider is that the entire spider should be able to download all the content onto one single server. In addition, the spider should have ability to scale onto multiple servers. This requires focus on resource efficiency.

The initial process of developing the spider included detecting which modern blogosphere technologies and protocols could be used to facilitate efficient blog crawling. This included considering ping servers, pingbacks, trackbacks & linkbacks as efficient methods of blog update notifications. Additionally, blog platforms and APIs were also researched.

The conclusion from this research was to combine Pingserver-feeds with a host analyzer and a blog crawler. The RSS-feeds from Pingserver secure timely updates and a standard formatted input used to analyze the blog page – with minimized resources spent on scheduled crawling.

The spider uses learning tree algorithms (such as ID3) to fetch and analyze blog sites efficiently. Learning tree algorithm is the preferred technology chosen to analysis of blog pages as it is not too resource-demanding. In addition, the efficiency of the learning tree algorithm increases over time with more usage.

The report outlines and evaluates different approaches to capturing and indexing weblogs. It must be noted that some of the most important methods were evaluated via prototyping. HTML-aware and semantic data extraction prototypes were implemented in order to gain experience in using these technologies and evaluate their characteristics, creating the foundation for further development in WP4.

The prototype spider based upon the chosen architecture and methods has demonstrated ability to scale (Appendix B) on a single server.

In addition, the prototype has demonstrated the ability to automatically capture blog content comprehensively, including additional entities from each site.

Connections with previous BlogForever work were also recorded and taken under consideration in the development of this prototype. The testing showed that the prototype has successfully fulfilled several of the requirements raised in D2.1 and D2.2, fully described in chapter 4.6.2, table 6.

Last but not least, current work has facilitated future development in the scope of the BlogForever Software Infrastructure (WP4) and especially Task 4.2 (Design of the weblog spider component) and Task 4.3 (Implementation of the weblog spider).

6 References

[1] E. Protalinski (2011, 10.11.2011). *Google starts indexing Facebook comments, AJAX/JavaScript content*. Available: http://www.zdnet.com/blog/facebook/google-starts-indexing-facebook-comments-ajaxjavascript-content/4978

[2] E. Bursztein, B. Gourdin and J. Mitchell (2011). "Reclaiming the blogosphere, talkback: a secure linkback protocol for weblogs," *Computer Security–ESORICS 2011,* pp. 133-149.

[3] S. Apart (2011, 10.11.2011). *Events and Callbacks in Movable Type: Table of Contents*. Available: http://www.movabletype.org/documentation/developer/callbacks/

[4] S. Langridge and I. Hickson (2002, 27.11.2011). *"Pingback 1.0"*, Available: http://www.hixie.ch/specs/pingback/pingback

[5] R. Cadenhead (2004). *Movable Type 3 Bible*. London: Wiley.

[6] K. Stepanyan, M. Joy, A. Cristea, Y. Kim, E. Pinsent and S. Kopidaki (2011, 27.11.2011). *"D2.2 Report: BlogForever Data Model"*. Available: http://blogforever.eu/

[7] B. Wu and B.D. Davison (2005). "Identifying link farm spam pages", *Proceedings of the 14th World-Wide Web Conference*, pp. 820-829.

[8] L. Zhu, A. Sun and B. Choi (2008). "Online spam-blog detection through blog search", *Proceedings of CIKM 2008*, pp. 1347-1348.

[9] P. Kolari, A. Java, T. Finin, T. Oates and A. Joshi (2006). "Detecting spam blogs: A machine learning approach", *Proceedings of the 21st National Conference on Artificial Intelligence*, p. 1351.

[10] X.A. Boronat (2008). *"A comparison of HTML-aware tools for Web Data extraction"*, Master's thesis, Universität Leipzig, Fakultät für Mathematik und Informatik, Abteilung Datenbanken.

[11] A.H.F. Laender, B.A. Ribeiro-Neto, A.S. da Silva and J. S. Teixeira (2002). "A brief survey of web data extraction tools", *ACM Sigmod Record,* vol. 31, pp. 84-93.

[12] V.I. Levenshtein (1966). "Binary codes capable of correcting deletions, insertions, and reversals". *Soviet Physics Doklady,* vol. 10, pp. 707–710.

[13] R.A. Wagner and M.J. Fischer (1974). *"The String-to-String Correction Problem"*, Journal of the ACM, vol. 21 no. 1, pp. 168-173.

[14] P. Kolari, A. Java, T. Finin, T. Oates and A. Josh (2006). *"Detecting Spam Blogs: A Machine Learning Approach"***,** University of Maryland Baltimore County.

[15] X. Azagra (2011, 27/11/11). *"HTML aware tools available for data extraction"*, Universität Leipzig. Available: http://dbs.uni-leipzig.de/file/Xavi_Extraction.pdf

[16] N. Agarwal and H. Liu (2009). *"Modeling and Data Mining in Blogosphere"*, San Rafael, CA: Morgan & Claypool Publishers.

[17] Gyöngyi, Zoltán; Hector Garcia-Molina, Jan Pedersen, *"Combating Web Spam with TrustRank"*, 2004

A. Appendix A – Weblog Spider Prototype Code Reference

A.1 Weblog Spider Prototype Documentation

RuleGenerator
RuleGenerator
* *Summary:* Class for creating a ruleset by matching wath is defined in a rss feed with the content located on a webpage. It uses unsupervised learning by combining Levensthein string match and a html dom structure (HtmlAgilityPack) to get the correct XPaths
RuleGenerator
GenerateRules(CW.Crawler.Library.RssData,CW.Crawler.Library.WebDocument)
* *Summary:* This is the first version of the rule generator
* *Param:* **rssData**: One rss item representing in this case a blogpost or a blog comment
* *Param:* **webDocument**: Webpage of the given rss data
* *Returns:* Returns a rule for extracting structured data from a webpage
Serialization
Serialization
* *Summary:* A helper class for xml serializing data to stream/file/string
Contracts
IPingServerMonitorHostContract
* *Summary:* Interface for the ping monitor service
Contracts
IPingServerMonitorHostContract.GetRecords(System.DateTime,System.DateTime)
* *Summary:* Returns statistic
* *Param:* **min**: Min date
* *Param:* **max**: Max date
* *Returns:*
Contracts
IPingServerMonitorHostContract.GetFilterByIds(System.Int32[])
* *Summary:* Returns an array of filters
* *Param:* **id**: Filter ids
* *Returns:* Array of filters
Contracts
IPingServerMonitorHostContract.GetFilterByParentId(System.Nullable{System.Int32})
* *Summary:* Returns filters where parent Id is predefined. If Id == null, all root filters will be returned
* *Param:* **id**: Parent Id
* *Returns:* Array of filters
Contracts
IPingServerMonitorHostContract.UpdateFilter(CW.Crawler.Library.Core.Filter)
* *Summary:* Update a given filter
* *Param:* **filter**: Filter class as input
* *Returns:* If successfull, return true else false
Contracts
IPingServerMonitorHostContract.DeleteFilter(CW.Crawler.Library.Core.Filter)
* *Summary:* Deletes a filter
* *Param:* **filter**: Filter class as input
* *Returns:* If successfull, return true else false
Contracts
IPingServerMonitorHostContract.AddFilter(CW.Crawler.Library.Core.Filter)
* *Summary:* Adds a new filter to the system
* *Param:* **filter**: Filter class as input
Rule
Rule

* *Summary:* Rule class represents how information should be extracted fromt the webpage using xpath related structure (more compatible with html dom structures)

HtmlPath

HtmlPath

* *Summary:* Html path represents a object describing how to navigate a dom structure

HtmlPathItem

HtmlPathItem

* *Summary:* This class represents one element of the html path. E.g. div , span, id, class etc.

ID3

DecisionTreeNode

* *Summary:* ID3 Decisiontree, it uses entropy score to create the decisiontree

ID3

DecisionTreeNode.Match(System.Collections.Generic.IEnumerable{CW.Crawler.Library.ID3.DataElement})

* *Summary:* A Function for matching data against the decision tree
* *Param:* **dataSet:** Dataset
* *Returns:* Returns either true, false or not found

ID3

DecisionTreeNode.CreateDecisionTreeNode(System.Collections.Generic.IEnumerable{CW.Crawler.Library.ID3.DataSet},System.Int32[])

* *Summary:* Creates a decision tree from a dataset, where it is possible to force part of the selection in the algorithm (Which columns should be analyzed first in the dataset)
* *Param:* **sets:** Dataset
* *Param:* **columnOrder:** Column order to analyze, can be null
* *Returns:* Returns the root tree node

ID3

DecisionTreeNode.RuleReduction(CW.Crawler.Library.ID3.DecisionTreeNode)

* *Summary:* A method for reducing the size of the descision tree
* *Param:* **root:**

Contracts

ICrawlerSystemContract

* *Summary:* The servicecontract between server and client

Contracts

ICrawlerSystemContract.AddWatchPoints(System.String[])

* *Summary:* Adds an array of urls which to generate rules and extract data from
* *Param:* **watchPoints:** An array of string url's

Contracts

ICrawlerSystemContract.GetHosts(System.String,System.Int32,System.Int32)

* *Summary:* Method for returning av array of hosts, using search filter and paging
* *Param:* **filter:** part of the host name (dns name)
* *Param:* **pageSize:** Default pagesize 10
* *Param:* **pageId:** Page id
* *Returns:* Returns an array of host descriptions

Contracts

ICrawlerSystemContract.GetWatchPoints(System.Int32)

* *Summary:* Returns descriptions of all the urls from a given host
* *Param:* **hostId:** Host id
* *Returns:* Array of descriptions

Contracts

ICrawlerSystemContract.GetEntity(System.Int32)

* *Summary:* Returns a blog entity, comment or blogpost
* *Param:* **entityId:** Entity id
* *Returns:* Returns a blogentity

Contracts

ICrawlerSystemContract.GetEntities(System.Int32,System.Int32)

* *Summary:* Returns an array of blog entities from a given url/blog
* *Param:* **watchPointId:** Url id/blog id
* *Param:* **takeCount:** How many should be returned
* *Returns:* Array of blog entities

Contracts
ICrawlerSystemContract.GetLatestEntities(System.Int32)
* *Summary:* Returns the latest enteties found.
* *Param:* **takeCount:** How many items shold be returned
* *Returns:* Returns an array of blog entities

Core
Filter
* *Summary:* A Regular expression filter for removing unwanted noise. It is WCF compatible

Core
PingServerLog
* *Summary:* Log class containing data of a ping server data package

Core
PingServer
* *Summary:* Class representing a ping server

Core
PingServerMonitor
* *Summary:* This class keeps track of which ping server to download and process

Core
PingServerMonitor.ProcessPingServer(CW.Crawler.Library.Core.PingServer,CW.Crawler.Library.Core.PingServerLog)
* *Summary:* Downloads data from a given ping server and extracts the ping information
* *Param:* **pingServer:**
* *Param:* **log:**

Core
TextBlock
* *Summary:* Represents a block of text with data mapped onto it by using start and stop indexes

Core
HostAnalyzer
* *Summary:* The host analyzer is a server component that creates a ruleset for a given rss feed by comparing the content of the rss feed items with the original

Core
HostAnalyzer.AnalyzeHost(CW.Crawler.Library.Host,System.String[],CW.WCF.Communication.RemoteEndpoint)
* *Summary:* Adds a job for the host analyzer
* *Param:* **host:** A class representing the host/domain
* *Param:* **uris:** Pages that should be analyzed
* *Param:* **remoteEndpoint:** Not in use
* *Returns:* Returns a job ID

Core
HostAnalyzer.ProcessData()
* *Summary:* Process method, is processed all the jobs listed in the queue
* *Returns:* If there is no more data, it will return false

Core
HostAnalyzer.DequeueProcessedItem()
* *Summary:* A method for dequeueing finished jobs
* *Returns:* Returns null if there are no data exists, or it returns the result of an host analysis

EntityElement
EntityElement
* *Summary:* Base class representing value, type, parent and start/stop index in the data

Entity
Entity

* *Summary:* The rule sets creates entities with data extracted from a webpage, it contains start and stop indexes from where the data has been extracted as well as the value
Entity
Identity
* *Summary:* If this property exists, it's because this entity is a root entity
EntityResult
EntityResult
* *Summary:* When structured data is extracted from page where there are dependencies / connections to each other, we no longer work on a tree structure but instead a graph structure. This class keeps track of all enteties defined as a root entity
Levenstein
StringToType(System.String)
* *Summary:* Converts a string into symbols
* *Param:* **s:** The string
* *Returns:* Array of symbols representing the string
Levenstein
StringToType2(System.String)
* *Summary:* Converts a string into symbols
* *Param:* **s:** The string
* *Returns:* Array of symbols representing the string
Levenstein
StringToType3(System.String)
* *Summary:* Converts a string into symbols
* *Param:* **s:** The string
* *Returns:* Array of symbols representing the string
Levenstein
ComputeDistance2D(System.String,System.String)
* *Summary:* A levensthein algoritm for matching data with several different formats. Exact, type, type dependent of prev
* *Param:* **s:** First string
* *Param:* **t:** Second string
* *Returns:* Returns an array of distances
Levenstein
ComputeDistance(System.String,System.String)
* *Summary:* Returns the minimum distance between the two strings
* *Param:* **s:** First string
* *Param:* **t:** Second string
* *Returns:* Minimum distance
Levenstein
ComputeDistance(System.String,System.String,System.Int32)
* *Summary:* Computes the distance between two strings, if a max value has been reached, it will return int.max
* *Param:* **s:** String s
* *Param:* **t:** string t
* *Param:* **max:** The threshold for error before it returns
* *Returns:* Returns the minimum distance between strings
RssDocument
RssDocument
* *Summary:* Class for mapping data from RSS and atom into one format
Core
EntityStorage
* *Summary:* A storage class for storing data onto the filesystem
Core
EntityStorage.GetEntity(System.Int32)
* *Summary:* Returns a specific blog entity

* *Param:* **entityId:** Entity id
* *Returns:* Returns null or one blog entity

Core
EntityStorage.GetEntities(System.Int32,System.Int32)
* *Summary:* Returns entities from a given blog host
* *Param:* **watchPointId:** The owner of the blog post
* *Param:* **takeCount:** How many items should be returned
* *Returns:* Returns an array of blog entities

Core
EntityStorage.GetLatestEntities(System.Int32)
* *Summary:* Returns the last set of data found
* *Param:* **takeCount:** How may items should be returned
* *Returns:* Returns an array of blog entities

Core
HostStorage
* *Summary:* A storage class for storing host data onto the file system

Core
CrawlerSystem
* *Summary:* The main core performs several different tasks such as Queue for downloading new content, Unsupervised learning of rules and Output storage

Core
CrawlerSystem.GetWatchPoints(System.Int32)
* *Summary:* Returns all the urs's associated to a given host, it is used by the WCF communication
* *Param:* **hostId:**
* *Returns:* Array of descriptions

Core
CrawlerSystem.GetEntity(System.Int32)
* *Summary:* Returns a blog entity of a given entityid, it is used by the WCF communication
* *Param:* **entityId:** Id of the entity
* *Returns:* Returns a blogentity

Core
CrawlerSystem.GetEntities(System.Int32,System.Int32)
* *Summary:* Returns blog entities from a give url, the method is used by the WCF communication
* *Param:* **watchPointId:**
* *Param:* **takeCount:**
* *Returns:*

Core
CrawlerSystem.AddWatchPoint(System.Uri[])
* *Summary:* Adds an array of uri's to be processed. It is used by the WCF communication
* *Param:* **uris:** Array of uri's

Core
CrawlerSystem.ProcessCreateRules()
* *Summary:* For each new url found and needs to be analyzed this method will call the hostanalyzer to create a rule set for the given rss feed, either it's a post feed or a comment feed.

Core
CrawlerSystem.ProcessPollWatchPoints()
* *Summary:* This method retreives a list of url's to download and extract the content and finally storing it on the file system

Core
CrawlerSystem.ProcessUnknowns()
* *Summary:* For each url that is unknown and is not assosiated to any rules and blogs, it will be downloaded to see what it is. Either a blog where the rss feed and rss comment feed is stored in the header or it is the rss itself. When the url for the blog post and/or comments are found they will be pushed to a host analyzer component for creating a rule set for extracting the structured data.

Core

WatchPointDescription

* *Summary:* A description class representing a url that has a rule associated to it. It is used by the WCF communication

Core

SystemHost

* *Summary:* Host representing a domain with several watchpoints/urls to be monitored

Core

SystemHostDescription

* *Summary:* A description class representing a host / dns name. It is used byte the WCF communication

Core

BlogEntity

* *Summary:* Blog entity is the base class that represents data extracted from the blog.

Core

BaseBlogElement

* *Summary:* Blog base class representing a single dataelement assosiated with blogdata

B. Appendix B – Weblog Spider Use Case

B.1 Use case

In order to evaluate the prototype, a dedicated server was setup in AUTH data center with the following specifications:

- Operating System: Windows 2008 Server 64bit
- CPU: 2 Xeon CPUs 2.5Ghz
- RAM: 4 GB
- Hard Disk: 20GB (SAS) plus 60GB (SATA)

Remote access and networking were also established and the prototype application was installed and tested continuously during the last stages of development from September 2011 to November 2011. The final test was conducted from the 22nd to the 24th of November 2011. A ping server with high traffic was monitored during this time and 2 URL filters were set a) ***blogspot.com/2011*** and b) ***science***.

The resulting application statistics showed that 8 million pings were evaluated during this time and 36.000 distinct blogs were selected for crawling based on the URL filters. Additionally, 25.000 posts were already processed; resulting in the creation of the corresponding XML files and another 197.000 were queued for processing. For the record, the size of the XML files containing extracted blog data was approximately 1 GB.

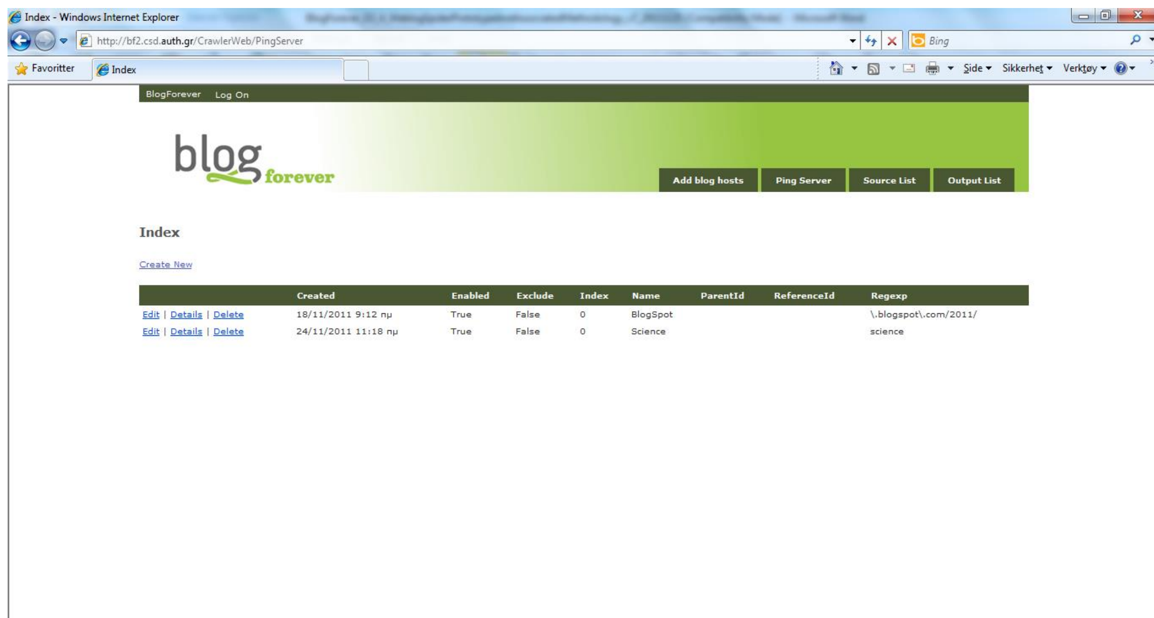

Figure 11 - While ping server is feeding the prototype spider with thousands of updates every minute – the user only wants a few selective updates. In the ping server section, two search filters are set. First requesting only blog hosts and blog posts from blogs

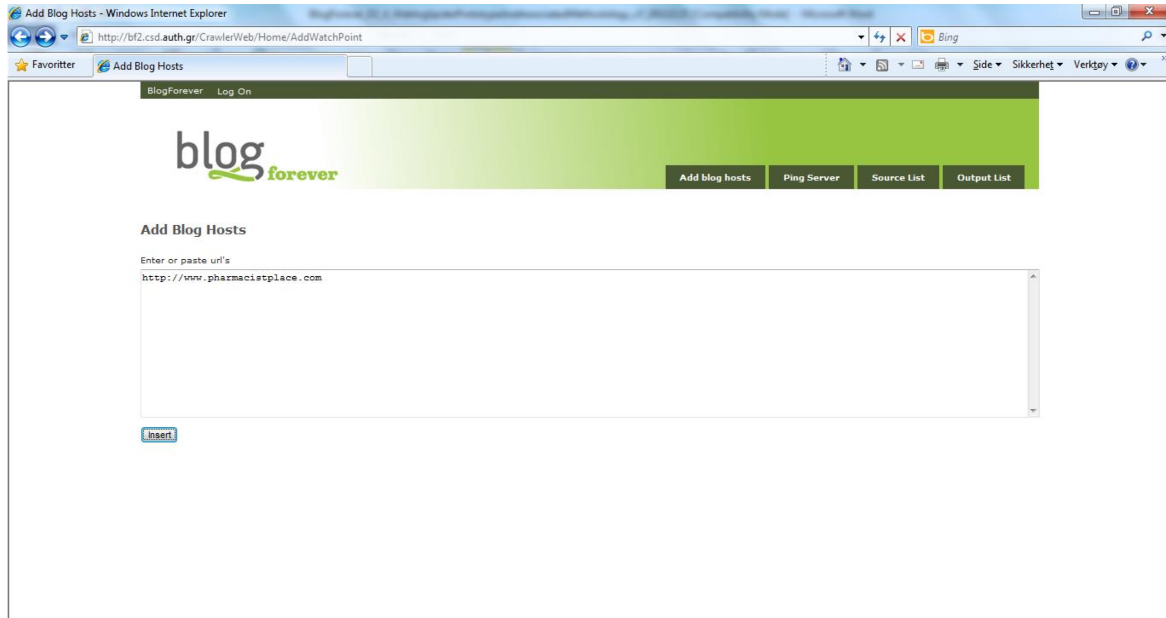

Figure 12 - In addition to science blogs and blogspotter blogs – the user would like the spider to capture all new updates regarding a qualified blog host: http://www.pharmacistplace.com

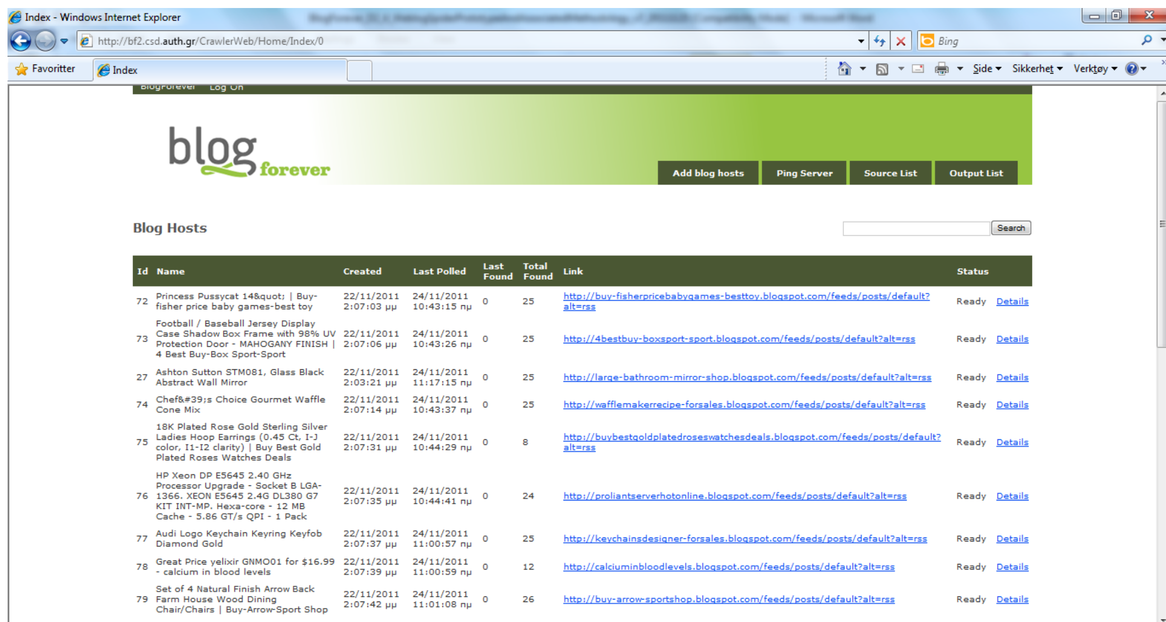

Figure 13 - The spider processes all URLs received from the ping server, matching it with the requested blog hosts – blogspotter 2011 and sources named "science". In addition the scheduler crawls the manual added blog host Pharmacistplace to process this accordingly. As soon as they are analysed and blog posts are captured, the blog host will appear in the blog database – and chronologically be listed in the Source List section of the Spider Portal.

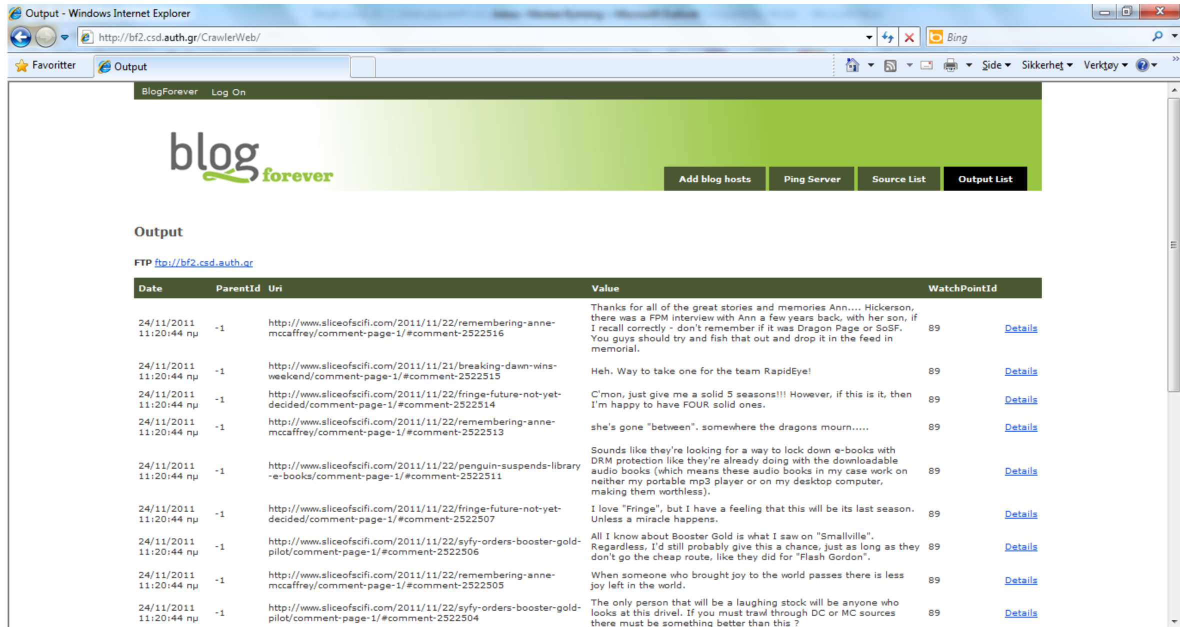

Figure 14 - After the blog host has been analysed and included in the Blog source database; all content elements are included and parsed – the output can be found in the Spider portal output section as shown above.

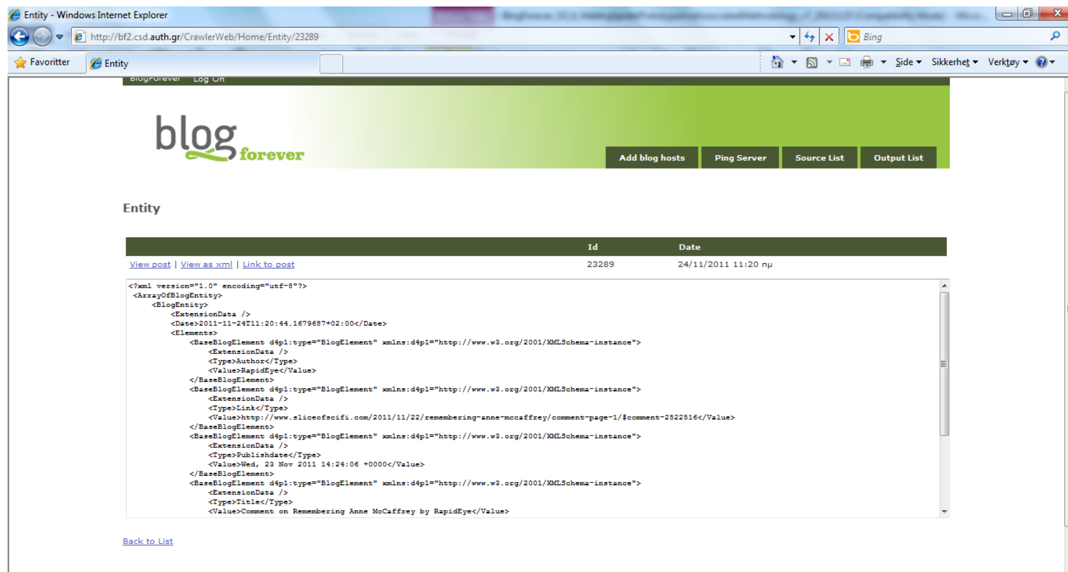

Figure 15 - Through the output section – all captured data becomes available in XML, or through the linked FTP server. The XML files from FTP looks as below.

Example of XML file from the FTP-server


```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfBlogEntity>
  <BlogEntity>
    <ExtensionData />
    <Date>2011-11-24T11:20:44.1679687+02:00</Date>
    <Elements>
      <BaseBlogElement d4p1:type="BlogElement" xmlns:d4p1="http://www.w3.org/2001/XMLSchema-instance">
        <ExtensionData />
```

<Type>Author</Type>
<Value>RapidEye</Value>
</BaseBlogElement>
<BaseBlogElement d4p1:type="BlogElement" xmlns:d4p1="http://www.w3.org/2001/XMLSchema-instance">
<ExtensionData />
<Type>Link</Type>
<Value>http://www.sliceofscifi.com/2011/11/22/remembering-anne-mccaffrey/comment-page-1/#comment-2522516</Value>
</BaseBlogElement>
<BaseBlogElement d4p1:type="BlogElement" xmlns:d4p1="http://www.w3.org/2001/XMLSchema-instance">
<ExtensionData />
<Type>Publishdate</Type>
<Value>Wed, 23 Nov 2011 14:24:06 +0000</Value>
</BaseBlogElement>
<BaseBlogElement d4p1:type="BlogElement" xmlns:d4p1="http://www.w3.org/2001/XMLSchema-instance">
<ExtensionData />
<Type>Title</Type>
<Value>Comment on Remembering Anne McCaffrey by RapidEye</Value>
</BaseBlogElement>
</Elements>
<Id>23289</Id>
<ParentId>-1</ParentId>
<Type>Comment</Type>
<Uri>http://www.sliceofscifi.com/2011/11/22/remembering-anne-mccaffrey/comment-page-1/#comment-2522516</Uri>
<Value>Thanks for all of the great stories and memories Ann....

Hickerson, there was a FPM interview with Ann a few years back, with her son, if I recall correctly - don't remember if it was Dragon Page or SoSF. You guys should try and fish that out and drop it in the feed in memorial.</Value>
<WatchPointId>89</WatchPointId>
</BlogEntity>
</ArrayOfBlogEntity>

C. Appendix C – Weblog Spider Prototype Instruction Manual

This section contains weblog spider prototype installation and usage instructions.

C.1 Installation

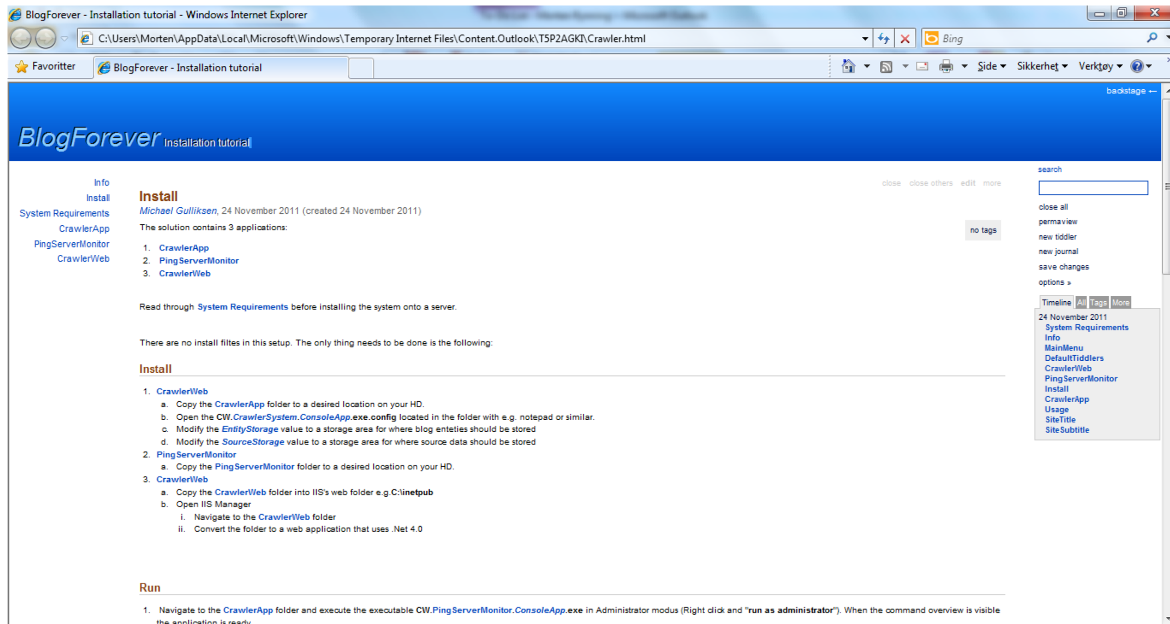

Figure 16 - Download and installation instruction file for the Blogforever Prototype Spider

The prototype spider can be downloaded and run from a single server at any location. The current download location is http://bf2.csd.auth.gr/BFCrawler.rar.

The download solution contains 3 applications:
1. CrawlerApp
2. Ping ServerMonitor
3. CrawlerWeb
Read through System Requirements before installing the system onto a server.

Installation

There are no install files in this setup. The only thing needs to be done is the following:
1.CrawlerWeb
a. Copy the CrawlerApp folder to a desired location on your HD.
b. Open the CW.CrawlerSystem.ConsoleApp.exe.config located in the folder with e.g. notepad or similar.
c. Modify the EntityStorage value to a storage area for where blog enteties should be stored
d. Modify the SourceStorage value to a storage area for where source data should be stored
2.Ping ServerMonitor
a. Copy the PingServerMonitor folder to a desired location on your HD.
3.CrawlerWeb
a. Copy the CrawlerWeb folder into IIS's web folder e.g.C:\inetpub
b. Open IIS Manager
i. Navigate to the CrawlerWeb folder
ii. Convert the folder to a web application that uses .Net 4.0

Run

1. Navigate to the CrawlerApp folder and execute the executable CW.PingServerMonitor.ConsoleApp.exe in Administrator modus (Right click and "run as administrator"). When the command overview is visible the application is ready.
2. Navigate to the PingServerMonitor folder and execute the executable CW.PingServerMonitor.ConsoleApp.exe in Administrator modus (Right click and "run as administrator").
    a. Enter "start" in the console window and hit enter to execute
    b. Enter "enable" in the console window and hit enter to execute to start downloading from pingserver(s)
    c. Enter "disable" in the console window and hit enter to execute to stop downloading from pingserver(s)

Reset

1.CrawlerApp
    a. Terminate the CrawlerApp by typing "q" in the console and the press "y".
    b. When the application has terminated navigate to the storage folder defined earlier and delete everything
    c. Start the application as described in the Run section

1.PingServerMonitor
    a. This application seldom needs to reset because it is accessible through the web interface.

To reset the application terminate the console window either by typing "q" in the console and the press "y" or clicking on the close button located upper right corner of the window.

System Requirements

• Microsoft .NET Framework Version 4.0 is required.
• Minimum Hardware Requirements:
    ◦ 2 GB of memory,
    ◦ Dual core CPU 1.6 GHz
    ◦ 10 GB of free hard disk space
• Recommended Hardware Requirements
    ◦ 8 GB of memory,
    ◦ Dual core CPU 2.4 GHz
    ◦ 10 GB of free hard disk space

CrawlerApp

The CrawlerApp is the core server application for crawling and processing html blog data. While executing it is using 2 TCP ports for communication:
    • 47001 is used for application management (start, stop etc)
    • 47000 is used for communication between CrawlerApp and CrawlerWeb and between CrawlerApp and PingServerMonitor
This process has up to 5 threads that will consume as much CPU as possible for:
• Creating rules
• Identifying page type
• Downloading content
• Scheduling

PingServerMonitor

The main purpose of the PingServerMonitor is to download xml feeds from ping servers and delegate the result back to the CrawlerApp. It is opening TCP/IP port 47002 which can be used for application management (start, stop etc).

The system uses regular expressions for filtering the feed. When the system is up and running for the first time, the user should not enable the server in the console until the right filter has been set. A ping server can produce 2-3 million records pr 5 min and therefore it is important in the beginning to set up a strict filter.

CrawlerWeb

The web interface of the application contains the following tabs:

- Add blog hosts
  - The user can manually add blog's for monitoring. Please Remember to add http:// in front of the URL
- Ping Server
  - The user can add/remove and update regular expressions for filtering the ping server data
- Source List
  - All the sources that the system has registered.
- Output List
  - Lists the latest blog entities stored to disk.

Downloading requests: technical@cyberwatcher.com

C.2 Usage instructions

Spider Portal – web interface overview

The Spider Portal is the web interface of the Spider, enabling easy inserting of new blogs onto the Spider - for monitoring. There are two different ways of including blogs into the Spider

1. Manually inserted – pasting a list of relevant blog URLs
2. Automatically through ping server – write a filter keyword which will be used to find relevant blog URLs and automatically add this into the source list of monitored blogs

All found blogs (blog hosts) through these two Input-processes, will be added to the Source List.

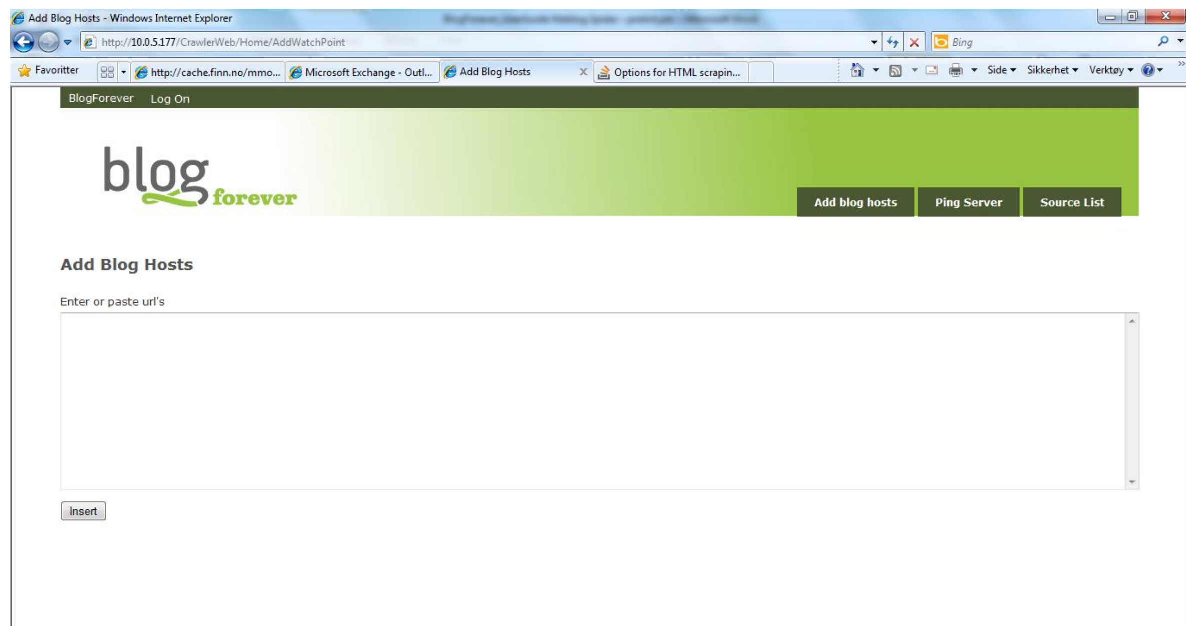

Figure 17 - Spider portal for adding blogs manually; by pasting in URLs

Add blog hosts – manual inserting blogs

This method enables the user to paste a list of blogs sites to be crawled. The list shall be URLs to blog sites with listed blog posts. The list can be inserted from a spreadsheet, or comma separated list.

The blog hosts URLs shall be inserted into this form and then submit insert:

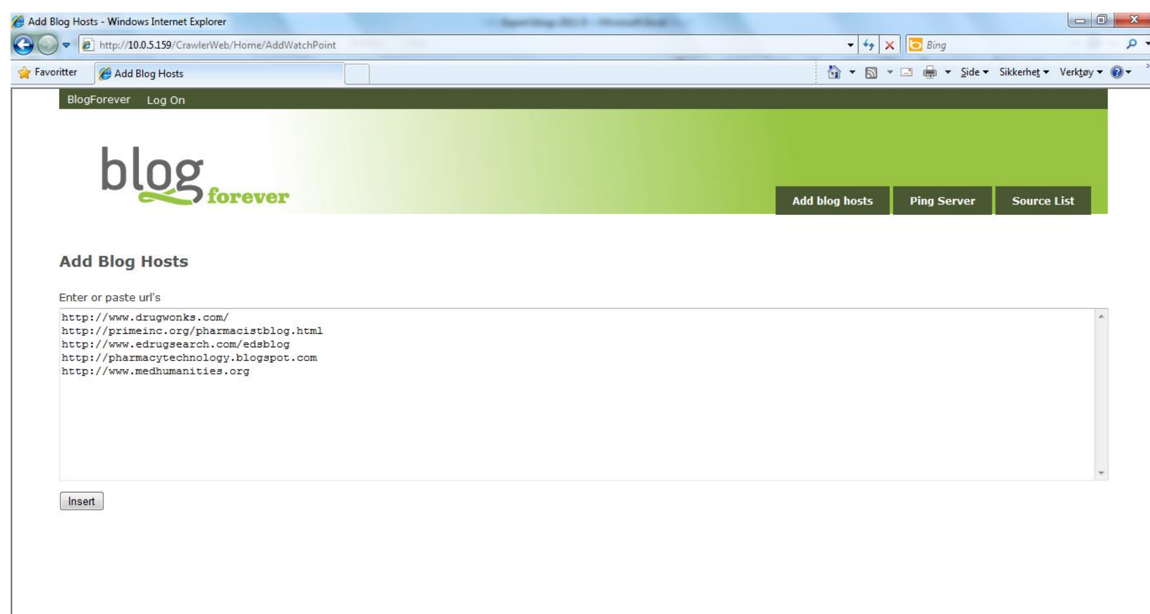

Figure 18 - Add blogs manually – through inserting URLs as above

Steps:

1. Put together a list of URLs separated with semicolons, or one line each or a row in Excel
2. Make sure all URLs includes http://

3. Paste the list into the field shown in the “add Blog hosts” section
4. Click *insert*

In order to know if the blog hosts are being successfully processed – a status can be tracked at the section “Source list”.

Ping server – adding new blogs automatically

This alternative enables the end user to insert potentially relevant blogs – even though the user doesn’t have a specific list of blogs.

This function enables to utilize ping servers which collect and allow distribution of millions of URLs with recent blog updates. The Spider subscribes to a named ping server giving such access 24/7. Even though the Spider has no knowledge about content on each of these blogs – it can filter upon information in the named URL.

In the Spider Portal tab – “Ping server” – the end user can insert keywords that are relevant to types of blogs wanted to be captured.

Create new

Insert keywords to tracking for new blogs

1. Write words consisting of minimum 3 letters. (Too long words are neither likely to be matched)
2. Words should be fairly generic in order to get matches
3. English Words have by far most chance to match
4. Max 2 words – each word will be checked individually (as ANY in Boolean query)
5. Insert end of URL-string – as of getting blogs related to the first part of the URL (see example \blogspot\.com/2011/)
6. Save the filters by clicking *Save*

View the list of ping server filters that is used to find new blogs to be included by the Spider

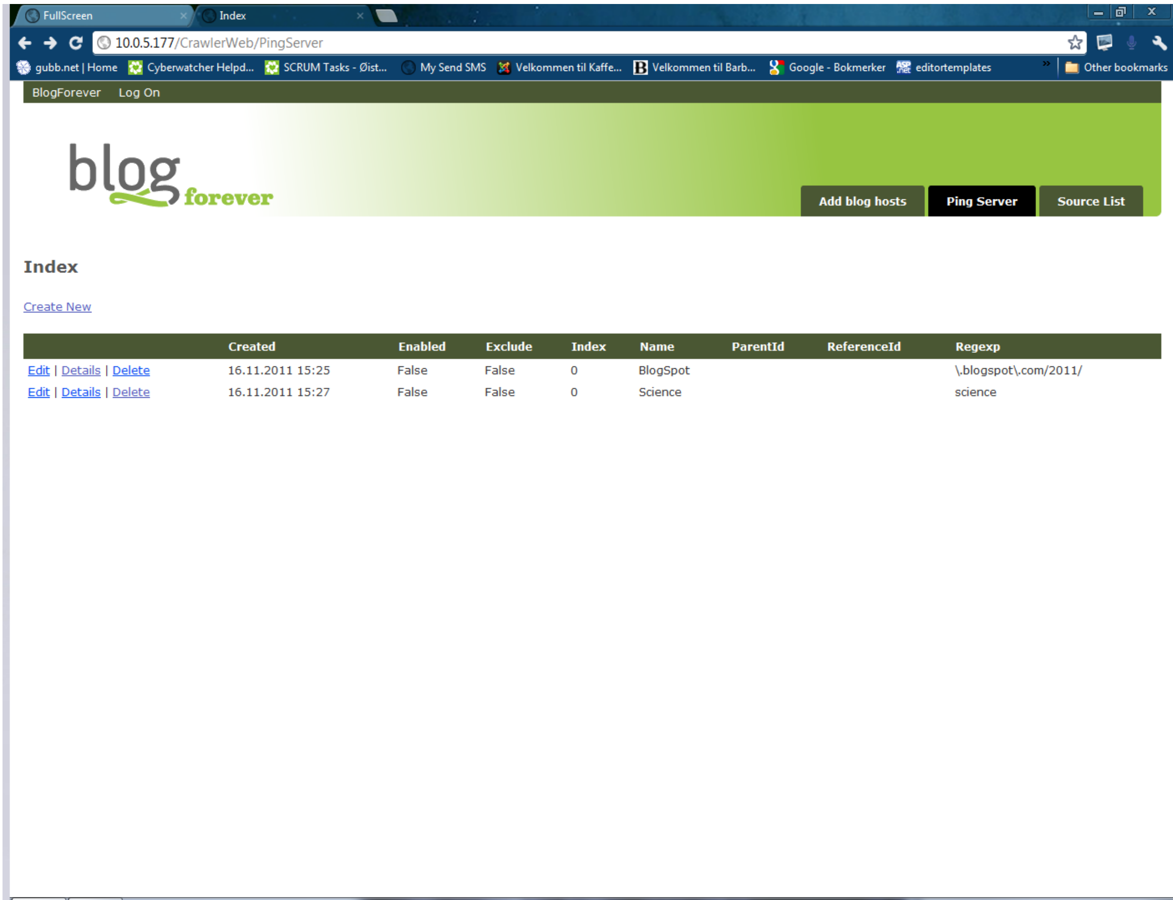

Figure 19 - ping server section – automatically indexing new blogs according to filters defining what is relevant.

The Spider is now actively and ongoing track the named ping server – to detect and insert relevant blogs.

To pause tracking for new blogs, click *limit ping server* to source list only. To reactivate please click *"tracking for new blogs"*.

These options define how the Spider monitors for new blog posts or blog comments from blog hosts. By limit ping server to source list – there will be no new sources added and only updates from listed blog sources will be processed as an XML output – in the Exporter, FTP section.

Source database – listing captured blogs

This part of the Spider portal lists latest blog host found and processed.

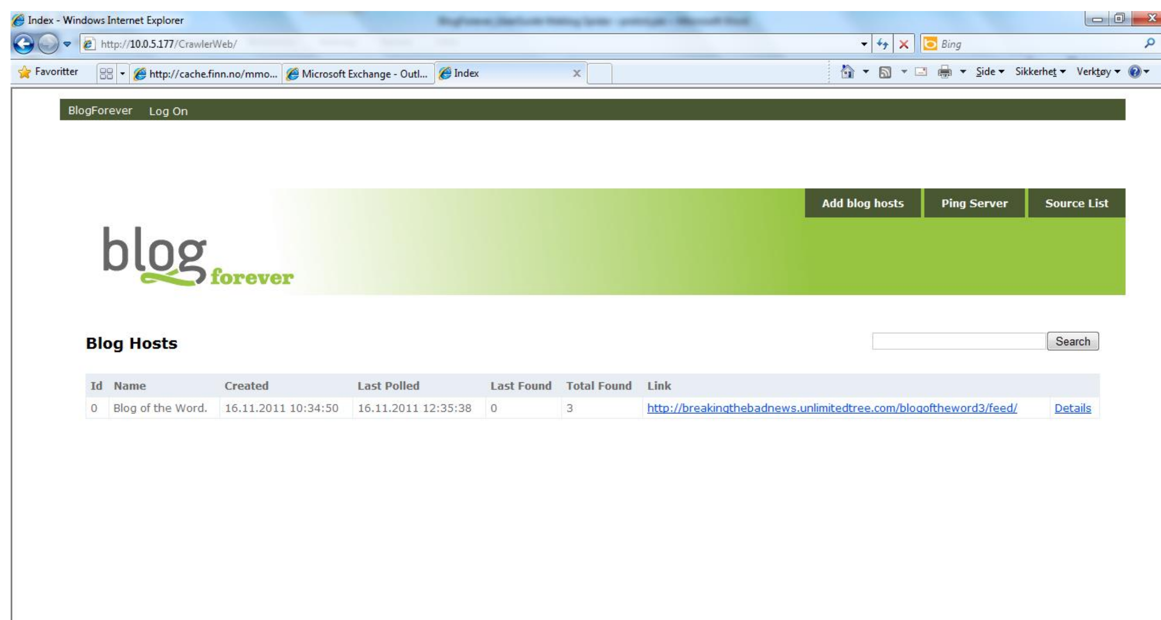

Figure 20 - Source list section where all latest blog sources included is listed

This table show:

- Latest blog post added is listed with automatic detected Blog host name, and an internal ID.
- Created date show when the source was first included into the Spider and added to the source list
- Last polled data shows recent update. The blog on top of the list should be todays date – as long as there are ongoing activities in the Spider.
- Last found – would normally be one, the latest new blog post or blog comments.
- Total found shows volume of all blog posts or blog comments captured by the Spider since it was included in the Spider and the source list.
- Link is the URL link to the RSS from where the blog host is feeding updates to the Spider.
- *Detail* is a link to explore further about content captured

A search function enables searching through the source list to list already included blogs.

Details – content captured per each blog host

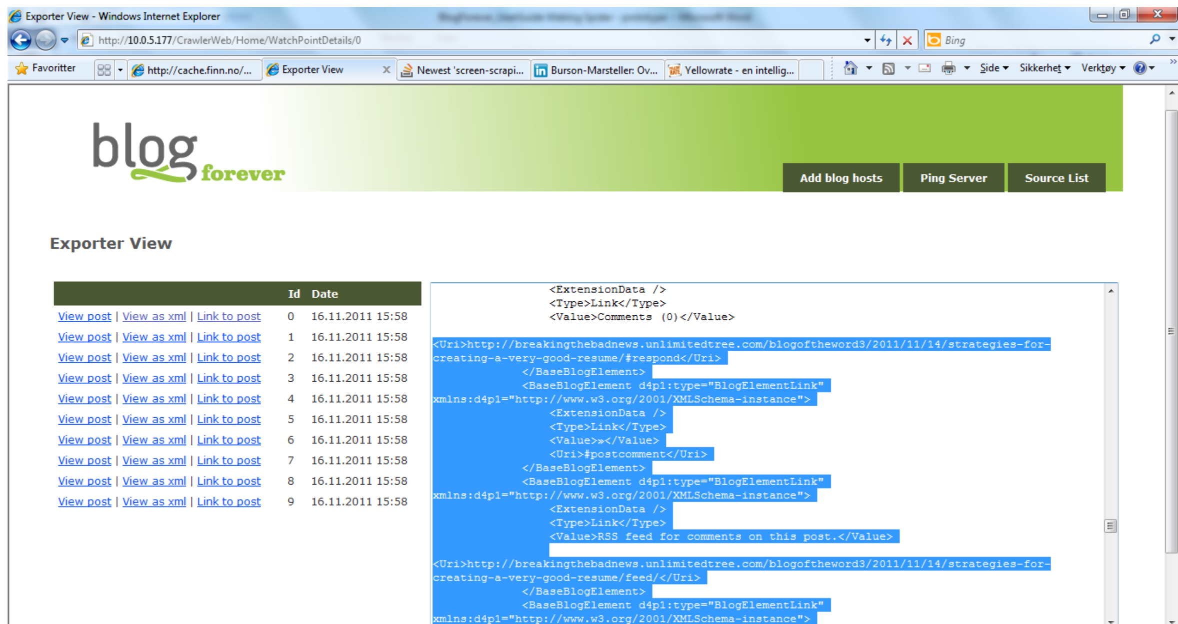

Figure 21 - Details linked from any of indexed sources – can display crawled XML, and link to actual HTML.

Details – enables to see fulltext of the actual captured content, from a list of all the latest blog posts or comments captured.

Three options are available to view the content:

A tagged text version of the content	View Post
XML fulltext	View as XML
The live HTML version	Link to post

Exporter – FTP output with stored files of XML

Output is a view in the Spider Portal, where all indexed content is listed chronological.
The output view found in top right corner of the portal also link to the FTP-section for exporting all captured blog content.File Transfer Protocol (FTP) is a standard network protocol used to transfer files from one storage to another.

FTP holds all the latest captured content, stored for the user to export it and integrate it with another application or repository. From the FTP the user shall download latest files and delete the copied file, which is a standard FTP procedure.

The Spider will hold the file for 24 hours – and then delete older files not removed by user. The delete procedure is necessary to avoid storage overload.

FTP login (if needed):
User name: bf
Password: bfever

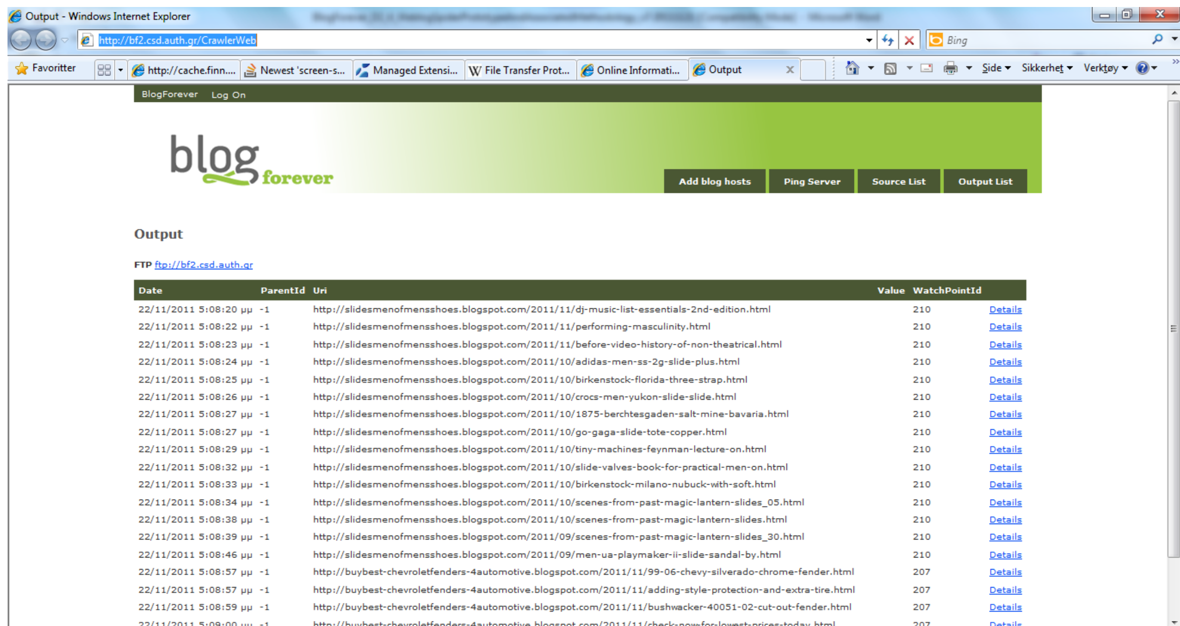

Figure 22 - Output section and link to FTP where all latest files with captured content, can be retrieved.

D. Appendix D – Prototype Semantic Data Extractor Application Code Reference

The prototype application consists of three main classes ExtractionInput, Microformat Extractor & Extract. the application are described in the following section.

Class ExtractionInput

This class represents the input of the extraction process in an abstract sense. Internally keeps the html documents as a DOM tree. This DOM tree used to extract semantic data by running sequentially Any23 extractors for each supported format. Also keeps information about the document, such as document's uri, charset, mime type, etc, and provides convenience access to them.

The constructor of ExtactionInput class just creates an instance of it without any associated html document. To associate an instance with a specific html document use one of the **setInput()** methods. There are five setInput() methods to support the following types of html input: URL, String, byte array, File and Document.

- **setInput(URL input)**
  This method uses the URL of a document. First fetch it from the web using the Apache HttpClient library. Second parse the fetched document to obtain the dom tree and various information about it.

- **setInput(String input, String documentURI)**
  Use this method when the html code of a document is available as a String. The method also parses the html to get the DOM tree representation and information associated with it. Except the string representation of the html code, the URI of the document must be passed to the method. The URI it's very important to be an absolute URI (not null, empty or relative) as it's used at the extraction process.

- **setInput(byte[] input, String documentURI)**
  This method is similar to setInput(String input, String documentURI) but in this case the html document is available as a byte array.

- **setInput(File input, String documentURI)**
  Use this method when the html document is stored as file in the local disk.

- **setInput(Document input, String documentURI)**
  Use this method when the DOM representation of the html document is already available. Using this method helps us **avoid the extra cost of parsing the input**. The input is passed as org.w3c.dom.Document.

ExtactionInput class has also a set of methods used to get various information of the underlying html document.

- **Document getDom()**
  Returns the documents as org.w3c.dom.Document

- **InputStream getInputStream()**
  Returns an InputStream associated with the content of the html document.

- **String getEncoding()**

Returns the document encoding (i.e. UTF-8) if available.

- **MIMEType getMIMEType()**
  Returns the document mime type as an org.deri.any23.MIMEType. It's used in the extraction process and it's always text/html.

- **String getLanguage()** Returns the document language (i.e en or el) for documents in English or Greek respectively.

- **URI getDocumentURI()**
  Returns the document URI.

Example of ExtractionInput class:

```
String  documentURI = "http://www.flixster.com/user/flixsterman/friends";

// Parse input ang get the dom tree
DOMParser parser = new DOMParser();
parser.parse(new URL(documentURI ).openStream());
Document dom = parser.getDocument();

// Create an ExtractionInput instance and connect in with the dom tree
ExtractionInput input = new ExtractionInput();
input.setInput(dom, documentURI);
```

Class MicroformatExtractor
This class acts as a facade of the extraction process over an ExtractionInput input format. It is thread-safe as the same instance can used from multiple threads. It Supports various RDF formats as output, such as Turtle[11] , RDF/XML[12] and Ntriples[14]. There are three methods, one for each supported output format.

- **extractToRDFXML(ExtractionInput input, OutputStream output)**
  This method extracts all semantic data from the input document in rdf/xml and sends them to the specific OutputStream.
- **extractToTurtle(ExtractionInput input, OutputStream output)**
  This method extracts all semantic data from the input document in Turtle format and sends them to the OutputStream.
- **extractToNTriples(ExtractionInput input, OutputStream output)**
  This method extracts all semantic data from the input document as a set of triples and sends them to the OutputStream.

Example of MicroformatExtractor class:

```
// Create a MicroformatExtractor instance and run extreactToRDFXML()
// method over an ExtractionInput.
MicroformatExtractor extractor = new MicroformatExtractor();
extractor.extractToRDFXML(input, System.out);
```

Sample output of the extraction result for http://www.flixster.com/user/flixsterman/friends

Class Extract
This class offers a simple command line tool used to extract microformats from a html document. The input is a local file or an url. The extraction result can be stored as a local file or can be printed

on stdout. The default out format is rdf/xml. A sample out of the extraction results for http://www.flixster.com/user/flixsterman/friends is listed below.

```
<rdf:RDF
    xmlns:dcterms="http://purl.org/dc/terms/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

    <rdf:Description rdf:about="http://www.flixster.com/user/flixsterman/friends">
        <dcterms:title>Randy Tippy's Friends (Page 1)</dcterms:title>
    </rdf:Description>

    <rdf:Description rdf:nodeID="node1612vdag2x1">
        <friend xmlns="http://vocab.sindice.com/xfn#" rdf:nodeID="node1612vdag2x26"/>
    </rdf:Description>

    <rdf:Description rdf:about="http://www.flixster.com/user/flixsterman/friends">
        <friend-hyperlink
            xmlns="http://vocab.sindice.com/xfn#"
            rdf:resource="http://www.flixster.com/user/366weirdmovies"/>
    </rdf:Description>

    <rdf:Description rdf:nodeID="node1612vdag2x26">
        <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
        <mePage
            xmlns="http://vocab.sindice.com/xfn#"
            rdf:resource="http://www.flixster.com/user/366weirdmovies"/>
    </rdf:Description>

    ....

    <rdf:Description rdf:nodeID="node1612vdag2x1">
        <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
        <mePage
            xmlns="http://vocab.sindice.com/xfn#"
            rdf:resource="http://www.flixster.com/user/flixsterman/friends"/>
    </rdf:Description>

</rdf:RDF>
```

E. Appendix E – HTML Aware prototype applications

For the purposes of evaluating HTML Agility Pack and Beautiful Soup, two small prototypes were developed in order to learn the specifics of implementation and gain experience in using these two libraries. First of all, an Access database was created with the following tables:

- Links: This table holds the links that are used for the evaluation.
- Elements: Which elements to use for the evaluation, one row per element.
- Results: This table holds the results of the evaluation (Library used, Link, Element, Attribute, Attribute value, InnerHtml) with a separate row for each change in any of Link/Element/Attribute.

Figure 23 – HTML Database structure

A small application has been developed for each library that does the following:

- Reads the links from the Link table in the MsAccess database.
- Reads the elements (tags) form the Elements table in the MsAccess database.
- For each link it sends an HTTP request against this link and for each element that was read from the database it loops through this element's attributes collection and writes the value of each attribute, along with the name of the attribute, in the Results table. It also gets the InnerHtml of the element and writes this too.

Queries to group the results by Library/Element/Attribute and return the count have also been created in the database in order to facilitate the comparison of the results.

Both of these applications were tested with various blogs and their results were evaluated without defining a clear winner.

Table 6 - HTML Agility Pack prototype

```
Imports HtmlAgilityPack
Imports System.Data
Imports System.Data.OleDb
Partial Class _Default
    Inherits System.Web.UI.Page
    Function getCon() As OleDbConnection
        Dim connectionString As String
        connectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" & HttpContext.Current.Server.MapPath("~/App_Data") & "\WebDataExtractionEval.accdb;"
        Dim con As New OleDbConnection(connectionString)
        con.Open()
        Return con
```

```
    End Function
    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Button1.Click

        'On Error Resume Next
        Dim webGet As New HtmlWeb
        Dim document As HtmlDocument
        Dim vElementName As String
        Dim con As OleDbConnection
        con = getCon()

        Dim CmdSelectLinks As New OleDbCommand("SELECT Link FROM Links", con)
        Dim vLinksReader As OleDbDataReader = CmdSelectLinks.ExecuteReader()

        If vLinksReader.HasRows Then
            While vLinksReader.Read()
                'MsgBox(vLinksReader.GetString(0))

                document = webGet.Load(vLinksReader.GetString(0)) ' "F:\Data\DevIT\b\tmp\vm4.html"
                'Dim metaTags As HtmlNodeCollection =
document.DocumentNode.SelectNodes("//meta")

                Dim CmdSelectElements As New OleDbCommand("SELECT Element FROM
HtmlElements", con)
                Dim vElementReader As OleDbDataReader = CmdSelectElements.ExecuteReader()

                While vElementReader.Read()

                    vElementName = vElementReader.GetString(0)
                    'MsgBox(vLinksReader.GetString(0))
                    Dim vElements As HtmlNodeCollection = document.DocumentNode.SelectNodes("//"
& vElementName)
                    If vElements Is Nothing Then
                        'MsgBox(vLinksReader.GetString(0))
                        Exit While
                    End If
                    For Each vElement As HtmlNode In vElements
                        If vElement.Attributes.Count > 0 Then
                            For Each vAttribute As HtmlAttribute In vElement.Attributes
                                vAttribute.Name = Replace(vAttribute.Name, "'", "''")
                                vAttribute.Value = Replace(vAttribute.Value, "'", "''")
                                Dim CmdSelectResults As New OleDbCommand("INSERT INTO
Results(Tool,Link,Element,Attribute,AValue) VALUES('HtmlAgility','" &
vLinksReader.GetString(0) & "','" & vElementName & "','" & vAttribute.Name & "','" &
vAttribute.Value & "')", con)
                                CmdSelectResults.ExecuteNonQuery()
                            Next vAttribute
                        Else
                            If String.IsNullOrEmpty(vElement.InnerHtml) = False Then
                                vElement.InnerHtml = Replace(vElement.InnerHtml, "'", "''")
                                Dim CmdSelectResults As New OleDbCommand("INSERT INTO
Results(Tool,Link,Element,Attribute,InnerHtml) VALUES('HtmlAgility','" &
vLinksReader.GetString(0) & "','" & vElementName & "','','" & vElement.InnerHtml & "')", con)
                                CmdSelectResults.ExecuteNonQuery()
                            End If
```

```
                End If

              Next vElement

            End While

          End While

        End If
        MsgBox("End")
        con.Close()
    End Sub
End Class
```

Table 7, BeautifulSoup prototype application

```
import urllib
import win32com.client

from BeautifulSoup import BeautifulSoup

def main():

    soup = BeautifulSoup(urllib.urlopen("http://blog.asmartbear.com/"))
    #soup = BeautifulSoup(urllib.urlopen("http://www.huffingtonpost.com/rep-bernie-sanders/why-i-
voted-no-on-the-def_b_919461.html?ir=Yahoo"))

    conn= win32com.client.Dispatch("ADODB.Connection")

    db = r"C:\Users\John\eclipse\workspace\BeautiSoupEval\WebDataExtractionEval.accdb"
    print db

    #DSN="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + db
    DSN="Driver={Microsoft Access Driver (*.mdb, *.accdb)};DBQ=" + db
    conn.Open(DSN)

    rsLinks = win32com.client.Dispatch("ADODB.Recordset")
    rsLinks.Open( "Links", conn, 1, 3 )

    rsElements = win32com.client.Dispatch("ADODB.Recordset")

    rsResults = win32com.client.Dispatch("ADODB.Recordset")

    rsLinks.MoveFirst

    while not rsLinks.EOF:

        soup = BeautifulSoup(urllib.urlopen(rsLinks.Fields.Item(1).Value))
        rsElements.Open( "HtmlElements", conn, 1, 3 )
        while not rsElements.EOF:
            for element in soup.findAll(rsElements.Fields.Item(1).Value):
                try:
                    if element.attrs:
                        for attribute, value in element.attrs:
                            #print attribute , " - ", value
```

```
                    vValue=value.replace("'","''")
                    rsResults.Open( "INSERT INTO Results(Tool,Link, Element, Attribute, Avalue) VALUES('Beautiful Soup','" + rsLinks.Fields.Item(1).Value + "','" + rsElements.Fields.Item(1).Value + "','" + attribute + "','" + vValue + "')", conn,1,3)
                else:
                    innerHtml = "".join([str(x) for x in element.contents])
                    vInnerHtml =innerHtml.replace("'","''")
                    rsResults.Open( "INSERT INTO Results(Tool,Link, Element, Attribute, InnerHtml) VALUES('Beautiful Soup','" + rsLinks.Fields.Item(1).Value + "','" + rsElements.Fields.Item(1).Value + "','" + attribute + "','" + vInnerHtml + "')", conn,1,3)
            except Exception:
                continue
          rsElements.MoveNext()
        rsElements.Close()
        rsLinks.MoveNext()

    print "The end"


if __name__ == '__main__':
    main()
```

F. Appendix F - Glossary

Anything To Triples (any23) is a library, a web service and a command line tool that extracts structured data in RDF format from a variety of Web documents. Currently it supports the following input formats:

•RDF/XML, Turtle, Notation 3
•RDFa with RDFa1.1 prefix mechanism
•Microformats: Adr, Geo, hCalendar, hCard, hListing, hResume, hReview, License, XFN and Species
•HTML5 Microdata: (such as Schema.org)
•CSV: Comma Separated Values with separator autodetection.

Document Object Model (DOM) is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents. Objects in the DOM tree may be addressed and manipulated by using methods on the objects. The public interface of a DOM is specified in its application programming interface (API).

File Transfer Protocol (FTP) is a standard network protocol used to transfer files from one host to another host over a TCP-based network, such as the Internet. FTP

Microformats are small patterns of HTML to represent commonly published things like people, events, blog posts, reviews and tags in web pages. Microformats enable the publishing of higher fidelity information on the web; and make it easier to capture and index the information for semantic search and analysis.

Ping server is an XML-RPC-based push mechanism by which a weblog notifies a server that its content has been updated. An XML-RPC signal is sent to one or more "ping servers", which can then generate a list of blogs that have new material. Today, most blog authoring tools automatically ping one or more servers each time the blogger creates a new post or updates an old one.

RDFa (Resource Description Framework – in – attributes) is a W3C Recommendation that adds a set of attribute-level extensions to XHTML for embedding rich metadata within web documents. The RDF data model mapping enables its use for embedding RDF subject-predicate-object expressions within XHTML documents, and also enables the extraction of RDF model triples by compliant user agents.

RSS (Really Simple Syndication) is a family of web feed formats used to publish frequently updated works—such as blog entries, news headlines, audio, and video—in a standardized format. An RSS document includes full or summarized text, plus metadata such as publishing dates and authorship

Splogs are blogs where the articles are fake, and are only created for search engine spamming. To spam in blogs, conversely, is to include random comments on the blogs of innocent bystanders, in which spammers take advantage of a site's ability to allow visitors to post comments that may include links. In fact, one of the earliest uses of the term "splog" referred to the latter.

XML (Extensible Markup Language) is a set of rules for encoding documents in machine-readable form.

XML-RPC is a spec and a set of implementations that allow software running on disparate operating systems, running in different environments to make procedure calls over the Internet. It's remote procedure calling using HTTP as the transport and XML as the encoding. XML-RPC is designed to be as simple as possible, while allowing complex data structures to be transmitted processed and returned.

Web spider is a computer program that browses the World Wide Web in a methodical, automated manner. Other terms for web spiders are crawlers, ants, automatic indexers, bots. Many sites, in particular search engines, use crawling as a means of providing up-to-date data. Web crawlers are mainly used to create a copy of all the visited pages.

WHATWG (Web Hypertext Application Working Group) is a community of people interested in evolving HTML and related technologies. The WHATWG was founded by individuals from Apple, the Mozilla Foundation and Opera Software in 2004. Since then, the editor of the WHATWG specifications, Ian Hickson, has moved to Google. Chris Wilson of Microsoft was invited but did not join, citing the lack of a patent policy to ensure all specifications can be implemented on a royalty-free basis.