SEVENTH FRAMEWORK PROGRAMME
FP7-ICT-2009-6

BlogForever
Grant agreement no.: 269963

D2.6 BlogForever Report: Data Extraction Methodology

Editor:	A. Cristea, M. Joy
Revision:	0.1
Dissemination Level:	Public
Author(s):	K. Stepanyan, G. Gkotsis, E. Pincent, V. Banos, R. Davis
Due date of deliverable:	31 May 2012
Actual submission date:	31 May 2012
Start date of project:	01 March 2011
Duration:	30 months
Lead Beneficiary name:	University of Warwick (UW)

Abstract: This report outlines an inquiry into the area of web data extraction, conducted within the context of blog preservation. The report reviews theoretical advances and practical developments for implementing data extraction. The inquiry is extended through an experiment that demonstrates the effectiveness and feasibility of implementing some of the suggested approaches. More specifically, the report discusses an approach based on unsupervised machine learning that employs the RSS feeds and HTML representations of blogs. It outlines the possibilities of extracting semantics available in blogs and demonstrates the benefits of exploiting available standards such as microformats and microdata. The report proceeds to propose a methodology for extracting and processing blog data to further inform the design and development of the BlogForever platform.

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

Revision History

Version	*Date*	*Modification reason*	*Modified by*
0.1	01.12.2011	Report structure and role distribution	K. Stepanyan
0.2	10.12.2012	Working draft	K. Stepanyan
0.3	12.01.2012	Addition of received approaches to parsing	T. Farrell, K. Stepanyan
0.4	30.03.2012	Review of approaches for data extraction	K.Stepanyan, G. Gkotsis
0.5	25.04.2012	Review and feedback on the draft	V. Banos
0.6	20.05.2012	Substantial extension: release for internal review	G. Gkotsis, K. Stepanyan, M.Joy, A. Cristea
0.7	21.05.2012	Discussion of earlier work on the use of RSS	R. Davis
0.8	25.05.2012	Extension on the topic of NER	V. Banos
0.9	28.05.2012	Review and corrections	V. Banos, M. Joy, A. Cristea
0.95	29.05.2012	Review and corrections	S. Arango-Docio
0.96	11.09.2012	Internal Review	H. Kalb, L. Paraskevi, M. Rynning
1.0	08.10.2012	Addressing the points of the Internal Review	G. Gkotsis, K. Stepanyan

Table of Contents

List of Figures

List of Tables

Executive Summary

This report proposes a data extraction methodology with respect to the BlogForever project. It introduces a workflow that describes the process of extracting data from blogs in line with the earlier conducted work on developing a spider prototype. The proposed methodology integrates a set of relevant approaches that utilise the use of web feeds for automatic generation of data extraction rules and the use of semantic annotation in blogs. Prior to discussing the proposed methodology, the report outlines the requirements of the project and the background of the earlier conducted work.

The background and related work included in the report, firstly, describes the nature of blogs as subjects of data extraction and positions them in relation to other web resources. The purpose of this description is to identify the relevance of various data extraction approaches that can be employed and successfully applied within the domain of blogs. Secondly, it proceeds to discuss the body of knowledge that provides foundations for data extraction and clarifies nomenclature, terms or working definitions adopted within the report (including the distinction between pre/post processing from data extraction). Thirdly, it briefly outlines the relevance of machine learning approaches and proceeds to discuss the use of wrappers for the task set in this report. The report highlights the need for developing automatic approaches to data extraction and proceeds to discuss the subject as part of the empirical inquiry.

The empirical work includes two use cases. First, based on the subject of data extraction from wikis (Section 5) draws conclusions relevant for de-duplication and optimisation of pre-processing resources, through reducing operating, bandwidth and storage costs. The second use case (Section 6.1) demonstrates the benefits and as well as the limitations of archiving blogs by using web feeds. The proposed methodology attempts to address the limitations of the described use case and is described in sections 6.2-6.8.

The report includes a detailed account of prototyping and evaluating a data extraction approach that employs web feeds. The results demonstrate a great potential for developing an automated data extraction technique by generating a wrapper. The performance of the data extraction prototype is shown to be consistent and relevant for adoption. In addition to employing automatic wrapper-based data extraction, the report discusses the trends and potential of employing semantic mark-up information. The idea is to support the data extraction and control mechanisms discussed as part of the data extraction system architecture in order to correct or further improve the performance of data extraction components where necessary. The report provides a set of recommendations and diagrammatic descriptions of the proposed methodology, data extraction workflows and possible data extraction system architecture to be considered for the design and development of the BlogForever spider component. Finally, the report looks into the concepts of Named Entity Recognition (NER) and discusses the prototyped software that enabled commenting on the potential of NER within the BlogForever context. Consequently, this report provides a summary of fundamental technologies and informs directions for implementing data extraction from blogs.

1 Introduction

The BlogForever project aims to develop solutions for preserving, managing and disseminating blogs. Capturing blog data is paramount for developing effective preservation solutions. The task for extracting blog data is far from being trivial, since it requires processing large collections of heterogeneous, distributed, time-variant and semi-structured documents. This document outlines an inquiry into the recent theoretical advances and practical solutions relevant for developing an effective data extraction solution within the domain of the Blogosphere. Additionally, it determines a set of best practices for efficient data extraction from weblogs. Prior to that, however, we clarify the aim and the rationale of the conducted inquiry. We establish operational boundaries considering established research areas such as, information retrieval, data mining and knowledge discovery, before proceeding with the selection and review of relevant approaches.

1.1 BlogForever: Background

The main aim of the BlogForever project is to develop robust digital preservation, management and dissemination facilities for blogs. Addressing the project aims requires development of new, considerably improved solutions that capture the dynamic and continuously evolving nature of blogs, their networks and social structure. The developed solutions should enable tracing the exchange of concepts within blogs and the ideas that they foster. The solutions should also ensure authenticity, integrity, completeness, usability and long term accessibility of blogs as a valuable cultural, social and intellectual resource.

The outcomes of the project are expected to benefit a number of stakeholders, in particular, libraries, information centres, museums, universities, research institutes, businesses, as well as blog authors and readers in general (for more details see [1], p. 11). The solutions, however, should not be restricted to institutional use only. Achieving the aims requires an investigation into the structure of blogs and their semantics. It requires understanding of blog networks and their dynamics. The investigation should inform the design and development of the BlogForever solutions and the development of the data model in particular.

1.2 Main Goals of Work Package Two

Work Package Two (WP2) is one of the six work packages of the BlogForever project that has three primary and sequential tasks[1]:

> *Task 2.1: Weblogs Survey.* Conduct user survey into: blogging practices, preservation of blogs, technologies and structural patterns (Deliverable D2.1 [2]).
> *Task 2.2: Weblog Semantics.* Conduct semantic analysis of blogs to inform the development of the data model and the ontological representation of the domain (Deliverables D2.2 [3] and D2.3).
> *Task 2.3: Weblog Data Extraction.* Review existing web data extraction methodologies and tools, assess their potential use for blog data extraction, review spam filtering methods and develop a prototype application (Deliverables D2.4 [4], D2.5 [5] and D2.6).

The first two tasks (T2.1 and T2.2) inform the completion of the Task 2.3. Task (2.3), subsequently, plays an important role for informing the design and development of the spider component of the BlogForever platform. A prototype application for crawling blogs has already been developed. The deliverable D2.4 includes the spider prototype and the report that accompanies the software. Deliverable D2.6 is expected to progress on the earlier deliverables and propose a methodology for data extraction from blogs.

[1] *For details see: Grant Agreement Annex I - Description of Work (DoW), page 7.*

1.3 Objectives and Rationale of the Report

The main objective behind this report is to determine the best practices for efficient blog data extraction in general and in particular to specify the optimal methods for the BlogForever platform. To achieve this aim the following objectives have been selected:

- Introduce the task of Weblog Data extraction (section 2)
- Evaluate existing theoretical solutions for data extraction (section 3)
- Evaluate existing applications, tools and libraries for data extraction (section 3)
- Identify BlogForever scenarios for data extraction, analyse the requirements, procedures and outcomes in order to define similarities and differences (section 4)
- Investigate available methods and libraries for parsing web resources (section 4)
- Identify suitable approaches to data extraction (sections 5 & 6)
- Integrate the identified approaches into a unified flow diagram describing the process and the method of data extraction (section 7)
- Investigate further opportunities for post-processing extracted weblog data (section 8)

1.4 Background to Web Data Extraction

The task of data extraction from the Web is challenging yet fascinating. The Web, and the Blogosphere as its constituent part, corresponds to a massive, publicly accessible data source. However, the scale is not the only challenge for capturing web resources. The heterogeneous nature of these resources, the large numbers of third party elements and advertisements, the rapid changes, the propagation of user-generated content and the diversity of inter-relations across the resources are among the common characteristics of the Web. These characteristics amplify the complexity of processing and capturing Web resources.

Unlike static web pages, blogs are commonly perceived as dynamic and versatile web spaces. They promote social connectedness, sharing, content creation and collaboration. Blogs vary in their choice of subject, writing style, purpose, media use, platform and presentation. They can represent individuals, organisations or companies and connect to different audiences. Blogs can assemble into specialised communities and interweave into sparse networks. This investigation focuses on data extraction techniques suitable for blogs. Detailed coverage of solutions for data extraction that are less appropriate in the context of the Blogosphere remains beyond the scope of this report. Prior to proceeding to the subject of data extraction, it is necessary to outline the ‘blog’ as the object of data extraction.

2 Blogs as Subject of Data Extraction

The definition of the term blog varies widely within the reviewed literature (e.g. [6, 7]). Furthermore, with the increasing number of services that encourage the propagation of user-generated content, the notion of a blog is becoming increasingly blurred [8]. However, there are a range of commonly perceived characteristics and widely acknowledged components of blogs. The consideration of these characteristics and components is critical for developing an appropriate data extraction methodology. The earlier inquiry into the structure and semantics of blogs [3] provides an insight into some of their prominent characteristics. Blogs are commonly seen as personal sites that contain time-stamped entries (posts) and comments. They are often inter-connected via various types of links and offer web feeds, such as an RSS or Atom, which enable distribution of the published content. While this description may be common, blogs and their structure vary widely, and therefore may deviate from this description. Yet, to keep the project within manageable realms, a working definition of a blogging platform has been proposed [3, p. 16]:

> *A blog is a web service that encompasses an accessible and widely accepted mechanism for creating, maintaining and automatically distributing chronologically published material on the Web, along with the feedback and user domain associated with it*[2].

Such web services, as well as the platforms that power these services, are numerous. For instance, Saddington [9] in his blog reports 118 blogging services and platforms. Custom built platforms that are not widely used [10] is another common issue that contributes to the challenge of working with blogs. However, whether standard or not, these platforms and services are being used widely. The outcome of using a blogging platform becomes a blog.

Blogs are the objects of preservation for BlogForever defined as:

> *Blog (as an object for preservation) is a chronological stream of user-generated content and associated activities to be represented with a level of granularity enabling its use by humans and computer systems.*

The graphical representation of this perspective is shown in Figure 1. The stream contains posts and comments, usually distributed via web feeds, but is not limited to the feed content. Unlike static HTML resources, the content of the blog resides in (typically) a relational database, though, alternative solutions for managing and maintaining the data are becoming available [11]. At the time of accessing a blog (that is, requesting the resource via HTTP protocol), it is being dynamically compiled by the blogging platform and sent to the web browser for rendering. The automation supported by the blogging platform provides a common structure that can be observed across the various blog pages.

[2] The term 'web service' is used here in a generic way. The term does not infer the use of specific Web Service technologies such as SOAP or REST.

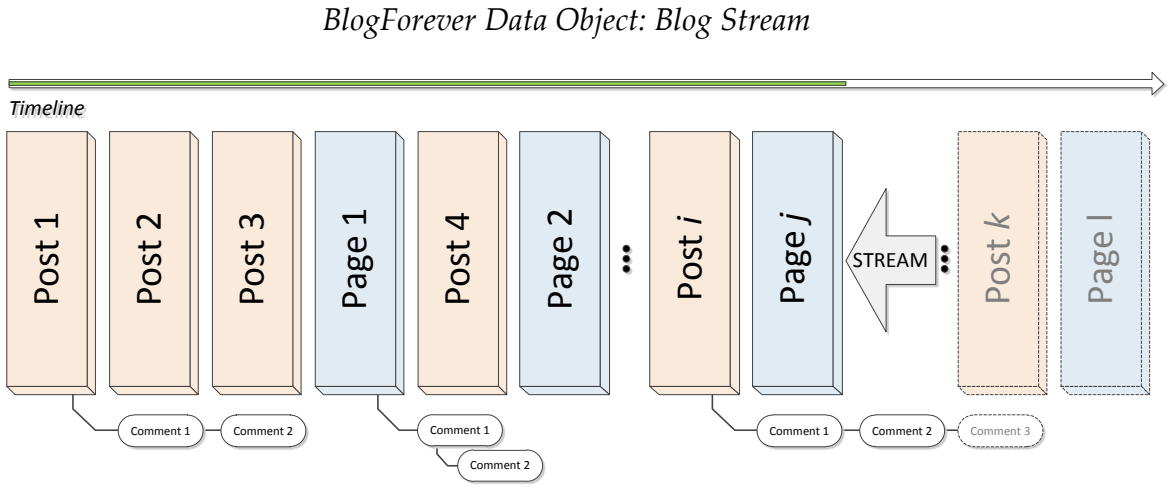

Figure 1: High level representation of a blog (from [3]).

2.1 Blogs as Semi-Structured Web Resources

Blogging platforms, as many other web applications, dynamically compile blogs by using the content stored in the relevant databases. As a result, blogs constitute semi-structured resources that can be crawled[3], analysed, aggregated and managed according to their structure.

Strategies for data extraction differ according to the types of the resources that they are applied to. It is therefore important to acknowledge the common structures as well as the idiosyncrasies that blogs exhibit. Among the most common characteristics, which demonstrate the presence of a certain structure behind the resource, are the following descriptive accounts of blogs:

- The *home page* of the blogs lists titles and summaries of blog posts.
- Blogs contain *descriptive web pages* (that do not appear as part of the listed summaries, blog archives or feeds).
- Each blog post is (usually) published via a unique link, so called *permalink*.
- Blog posts may have associated *comments*. The structures of comment sections may vary widely.
- Blogs exhibit XML based *web feeds* (e.g. RSS, Atom). The number of feeds may be greater than one. The content distributed via web feeds may include recent posts, comments, most popular posts or categorised/aggregated content.

The inquiry into the structure and semantics of blogs enabled identification of the most prominent constituent elements of blogs. This inquiry enabled developing the data model for accommodating blog data. Among the most common elements (Figure 2) are Blog Entries (identified as being either Posts or Pages) that may have Comments and are associated with an Author. Both Entries as well as Comments exhibit certain Content. These entries are analysed further and (where relevant) broken down into fewer compound objects. For instance Content, as one of the most complex elements is further described by simpler objects, for instance, Text, Tags, Links or Multimedia.

[3] Crawling is a process performed by a software tool (i.e. spider) that traverses the hypertext structure of web pages and retrieves accessible documents [12] M. D. Dikaiakos, A. Stassopoulou, and L. Papageorgiou, "An investigation of web crawler behavior: characterization and metrics," *Computer Communications,* vol. 28, pp. 880-897, 2005.. More information about the process of crawling is available in D2.4 Spider Prototype. [4] M. Rynning, V. Banos, K. Stepanyan, M. Joy, and M. Gulliksen, "BlogForever: D2.4 Weblog spider prototype and associated methodology", 2011, Available: http://blogforever.eu/wp-content/uploads/2011/11/BlogForever_D2_4_WeblogSpiderPrototypeAndAssociatedMethodology.pdf..

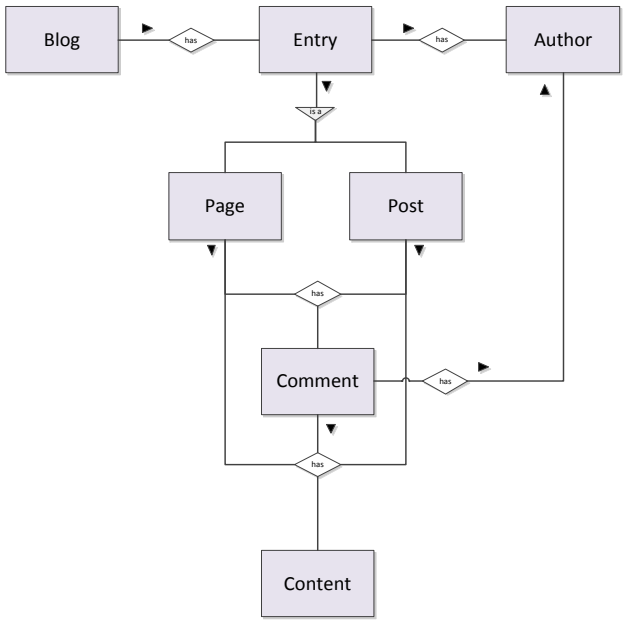

Figure 2: Core elements of blogs and their relationship (from [3]).

These structured characteristics and constituent elements, commonly exhibited in the Blogosphere, can be exploited for extracting and aggregating blog data. However, despite the commonalities, blogs remain highly heterogeneous. Firstly, bloggers can apply idiosyncratic templates to modify the layout and presentation of their blogs. These templates can be easily altered at any time, further complicating the task of data extraction. Secondly, blogs commonly publish unstructured text, include third-party content and can include dynamically added components. Hence, blogs constitute a 'noisy' (from the web data extraction point of view) resource. Noise can pose additional challenges in identifying certain structural components of the page. Navigation links, advertisements, widgets and third party data, can contribute to the level of noise within a particular resource. Finally, the interrelation between the various structural elements of the blogs can further extend the complexity of the data extraction task. Conceptual elements may exhibit a hierarchy or graph-like relationships that are difficult to identify and capture. For instance, blog comments can develop into threaded discussions where the commenter addresses the response to another commenter rather than the author of the post, or published content may link to internal as well as external blogs. These examples highlight the complexity of the structure that may be difficult to identify and extract.

The advances in the area of data extraction resulted in a range of approaches that can be applied to semi-structured resources such as blogs. They are categorised as supervised, unsupervised and partially supervised methods and are discussed further in section 3.2.

2.2 Semantics in Blogs

The Blogosphere contains vast amounts of user generated content. While identification of structured and semi-structured elements within blogs is undoubtedly useful, there are more subtle elements that require understanding of the meaning embedded into the posted content, also known as semantics. In the area of computer science, the term *semantics* (from Greek sēmantiká – the study of meaning) is often contrasted to the term *syntax* (from Greek syntaxis – putting in order). Understanding the semantics and extracting relevant data is important for ensuring successful blog preservation.

This report discusses relevant approaches that include, for instance, Natural Language Processing (NLP), the use of Microdata, Microformats or RDFa. Some of the approaches, known for semantic

analysis are inherently complex. Yet, practical use of approaches such as, sentiment analysis or opinion mining, is gaining momentum. Some approaches to semantic analysis, particularly those under the umbrella term of NLP, are computationally expensive and non-deterministic. However, their application as part of the data extraction methodology is discussed in this report. Recent theoretical and technological advances in the area of information retrieval are outlined where relevant.

3 Basic Concepts and Approaches to Data Extraction

The problem of web information extraction dates back from the early days of the web and is fascinating and genuinely hard. The web, and the Blogosphere as a constituent part, correspond to a massive, publicly accessible unstructured data source. However, the scale is not the only challenge for capturing web resources. The heterogeneous nature of these resources, the large numbers of third party elements and advertisements, the rapid changes, the propagation of user-generated content and the diversity of inter-relations across the resources are among the common characteristics of the web. These characteristics amplify the complexity of capturing, processing and transformation of these web resources into structured data. The successful extraction of weblog properties into structured data is of paramount importance for improving the quality of analysis, searching and preservation of such resources.

This section discusses the various concepts of data extraction as well as the methodologies used to achieve it. A short introduction to data mining and machine learning techniques is provided in order to familiarise the reader to the state-of-the-art practices of data extraction. A short survey on the tools that facilitate data extraction (wrappers and wrapper generation) is carried out in order to identify the requirements and challenges for the case of weblog data extraction.

3.1 Data Mining and Extraction

Data mining is an interdisciplinary field in computer science, also known as knowledge discovery in databases (KDD), that focuses on "extraction of implicit, previously unknown, and potentially useful information from data" [13, p. xxiii]. The process of Data Mining includes discovery of new patterns from large datasets by using methods from artificial intelligence, machine learning, database systems and statistics. The application of data mining spans widely from forecasting by using historical data to discovery of a structure within data. The sheer amount of data generated nowadays requires application of effective data mining solutions for interpreting, understanding and acting upon.

Within the broad area of Data Mining there are a number of sub-fields and research directions such as Text/Web Mining, Information Extraction, Document Classification, Language Identification, Metadata Extraction and Entity Extraction. Web mining is a relatively new and intrinsically complex field. At its current state, the literature has no real consensus about the coverage of this filed [13]. The definitions of the term *data extraction* remain rather general. For example, Baumgartner *et al.* [14, p. 1] define a web data extraction system as *"a software system that automatically and repeatedly extracts data from web pages with changing content and delivers the extracted data to a database or some other application*". The lack of clarity opens up a question, that is: what approaches should be included into the data extraction methodology and covered by this report? For instance, should the approaches positioned under the broad term of Natural Language processing be included into the data extraction methodology?

We refer to the DoW and earlier submitted deliverables D2.2, D2.4 and D2.5 for gaining necessary insight for defining the boundaries of this report.

The DoW suggests the outcomes of this report should inform the development of the spider component that will extract necessary data. The requirements of spider include effective and efficient data extraction. However, the design of the spider prototype (D2.4) and the requirements for spam filtering (D2.5) indicate a possibility of an overhead. Data extraction, as a resource-expensive process, will further contribute to the anticipated overhead. To avoid unnecessary overload at crawl-time, some processing of the retrieved blogs and documents can, therefore, be conducted separately from the spider or at a later stage. Later processing can be made possible subject to retaining access to the full content of the retrieved document. The data model (D2.2) provides the necessary structures to enable content analysis at a later stage. Therefore, within the boundaries of this report we distinguish the notions of data extraction and post-processing to the following:

The notion of Data Extraction is confined to processing of HTML or XML-based resources for extracting data objects. Processing of extracted data or documents that are not annotated (i.e. Simple Text) is referred to here as post-processing.

This report focuses on data extraction only as defined above. Approaches of post-processing (e.g. linguistic analysis or textual mining) are only outlined where necessary.

3.2 Machine Learning: Supervised, Unsupervised and Partially Supervised

Data mining processes can be broadly categorised as automatic and, more usually, semi-automatic. These processes are designed for discovery of patterns in datasets [13]. Data extraction can then be performed after a set of rules are produced according to the discovered patterns. Manual intervention in generating the rules often pays off in terms of performance. However, in many data mining situations producing rules manually can be too labour intensive. Application of *machine learning* techniques can significantly reduce or eliminate human intervention.

Machine learning is a burgeoning approach for mining knowledge from data. It is about "*algorithms for inferring structure from data and ways of validating that structure*"[13]. Application of machine learning allows capturing of structural descriptions inductively from the given examples, studied as part of the larger collected data cohort. Machine learning forms a technical basis of data mining. It has been successfully used for web data mining with a range of readily-available algorithms that can be applied to the tasks identifying the patterns that are relevant for data extraction [15].

The process of discovering patterns is commonly categorised as [16]:

- Supervised learning
- Unsupervised learning
- Partially supervised learning

3.2.1 Supervised Learning

Supervised learning has been successfully applied in many domains, including Web mining. It is also called inductive learning, and includes some of the most widely used data mining techniques.

Supervised learning (also known as classification or inductive learning) is the process of inferring a function (also referred to as classifier) from labelled data (with pre-defined classes), which can be applied for processing new data. Applying the derived function on new/future data enables to classify data instances into the known classes. For example, the task of making a decision (i.e. *class value*) on a credit application form based on a set of attributes, such as Age, Employment or Residence, is a classic task of a supervised learning. Once the rules are inferred from the *training dataset*, which includes the known class values, the rules can be applied on the *test/unseen dataset*, where the class value is unknown [16].

The use of supervised learning techniques presupposes the existence of a separate training dataset before it is applied on the test/actual dataset. Additionally, the performance of this approach is generally dependent on: [a] expert opinion specifying and on specifying most informative/relevant attributes; [b] pre-processing for handling missing data; and [c] feature selection (i.e. removal of less relevant attributes for optimising the process of classification) [17]. Among the most widely used techniques for classification is the *Decision Tree* learning, where the tree with attributes is traversed top-down according to the given instances until reaching the leaf node with the classifier. The use of *Support Vector Machines* (SVMs) is another popular technique for learning. SVMs have a solid theoretical foundation and perform classification more accurately, especially when very high

dimensional data is used. Other approaches used to devise rules from the training data include *Naïve Bayesian Classification* and *Class Association Rules* (CAR) [16].

.

The use of supervised learning for web data mining is common. Among possible applications within the context of blog preservation may include the:

- Recommendation of blogs on a specific topic or according to a set of attributes,
- Identification of blogs from other resources given the patterns of URLs
- Recognition and extraction of named entities, such as organisations, places or dates

The use of supervised learning can, therefore, investigated further for the purposes of data extraction.

3.2.2 Unsupervised Learning

The primary difference of the supervised learning from the unsupervised one is the prerequisite of having a known/training dataset. In some cases, the data may not have specific class attributes. Instead, it may be necessary to explore the data to find some intrinsic structures. Unsupervised learning is one of the approaches for finding such structures (for instance, clustering) [16]. Due to the diversity across blogs and web documents in general, unsupervised learning is another potentially relevant approach.

Unsupervised learning is the process of organising data into groups, where the instances are seen as similar. This process results in an identification of distinct clusters that contain "similar" data instances. Hence, the grouping is based on the principle of "maximizing the intraclass similarity and minimizing the interclass similarity" [18, p. 26]. A simple example of clustering is presented in Figure 3. While supervised learning included a set of known class labels, unsupervised learning (i.e. the process of clustering) is used for generating such labels. Hence, clustering can facilitate development of taxonomies by informing the process of organising the observations into groups or a hierarchy of similar classes *(ibid.)*.

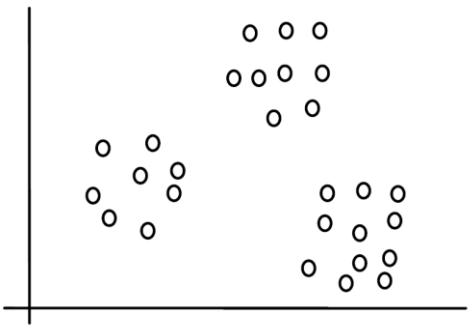

Figure 3: Graphical representation of clustering (from [16]).

Most clustering algorithms work on numeric data. K-means is one of those algorithms. The simplicity and efficiency of this algorithm, as well as its availability in various software packages, put K-means among the most widely applied techniques. The algorithm partitions the given dataset into a specified (k) number of clusters based on the measure of distance between the instances in the given dataset [16]. Alternatives to K-means include the density-based clustering algorithm [19] that allows finding clusters of arbitrary shapes, a grid-based clustering technique [20], or a neural network clustering algorithm such as Self-Organizing Map (SOM) [21]. Approaches for clustering categorical data have also been developed (e.g. k-modes). Within the Web domain, unsupervised learning is used primarily for clustering Web pages using text-based similarity or clustering of search engine results according

to different topics. This approach can be used for automating the process of finding patterns and extracting data from semi-structured resources accordingly [16].

3.2.3 Partially Supervised Learning

Partially supervised learning techniques have been developed to overcome one of the drawbacks of supervised learning, that is the requirement of having (often large amount of) labelled training data. Given the fact that the development of training data requires manual intervention and can be resource intensive and time consuming, the rationale for alternative approaches can be easily justified.

Partially supervised approaches, as the name suggests, combine elements of supervised and unsupervised learning. They do not require full supervision and, therefore, reduce the efforts for manual labelling. There are two primary learning tasks:

- L and U learning: Learning from Labelled and unlabelled examples
- P and U Learning: Learning from Positive and unlabelled examples

While the former task includes a smaller set of labelled instances of each class and a larger set of unlabelled instances, the latter one includes a set of only positively labelled and a set of unlabelled instances.

In partially supervised approaches, the unlabelled data can be helpful. Under certain circumstances, using unlabelled datasets along with a small labelled one can be better than using the small labelled dataset on its own. When used for text classification, unlabelled data can be cross-correlated with labelled data for identifying other potential labels. Classes from a small labelled dataset can be inferred using Naïve Bayes, and then extended to a larger dataset. For instance, given the fact that the word “homework” belongs to a particular class, unlabelled data can be classified similarly if a correlation between the word “homework” and some other word (e.g. “lecture”) from unclassified data is found. Implementation of L and U approaches includes, for instance, co-training or the use of the Expectation–Maximization algorithm. Implementation of P and U approaches, on the other hand, requires taking into account the missing labels (a set of unlabelled instances) within the labelled dataset. Theoretical and experimental studies suggest that training datasets are to be substantial for the P and U approaches to work effectively [13, 16].

3.2.4 Summary

The task of blog data extraction can benefit from supervised, unsupervised and partially supervised machine learning approaches. The survey by Qu and Davison [22] demonstrates a potential range where unsupervised learning can be applied. Their list includes learning for distinguishing web resources (e.g. blogs vs. other web documents), classification of web blogs into directories and subject hierarchies, and measurement of mood or sentiment in blogs. Classification of blogs according to their genre, such as news, commentary or journal, is another task that can benefit from machine learning. Unsupervised learning approaches can also be used within the context of weblog data extraction. Similar to the more general example discussed by Alvarez [23] or [24], blog DOM trees/sub-trees can be processed to find sets of comments or a collection of blog entries as listed on the home page. These, and other relevant techniques, are discussed in the context of the BlogForever data extraction task in section 6. Additionally, as Qu and Davison [22] suggest, introduction of other supporting measures, such as Semantic Web technologies, can complement and further improve the task of data extraction. We devote a separate section (6.6) to discuss the use of Semantic Web technologies and Ontologies in data extraction.

3.3 Wrappers

Data extraction from structured or semi-structured resources often includes the use of *wrappers*. A wrapper is a software tool that allows extraction of data records represented on websites with fixed-term templates [16]. The data records usually represent underlying databases, such as products, financial data, academic publications, or in the context of BlogForever, blogs. Extracted data can then be aggregated, stored, analysed or reused.

Aggregation and analysis of extracted data becomes increasingly important and attracts companies and organisations. There are many companies that extract data and offer various commercial services based on the data. Among the companies that focus on blogs and social media are Spinn3r[4], Radian6[5] and Sysomos[6]. The approaches that are common in the area of data extraction can be classified as [16] follows.

1. Manual Approach, where the programmers find patterns and develop scripts for extracting data according to that pattern.
2. Wrapper Induction, a semi-automatic approach which requires the use of supervised learning. Wrapper induction is conducted by manually labelling data records and inferring certain data extraction rules that can be applied to other, similarly formatted pages.
3. Automatic Extraction, where an unsupervised approach is used for discovering patterns and inferring extraction rules automatically. This approach becomes highly scalable by eliminating the need for manual labelling.

BlogForever solutions should enable data extraction from a large number of blogs. Manual data extraction approaches would require development of a custom data extraction script for each of the blogs. Therefore, manual approaches will not be suitable in the context of BlogForever. On the other hand, Automatic Extraction and Wrapper Induction are potentially suitable and are discussed further in this report.

Liu [16] defines mainly two types of structured or semi-structured pages. These are:

1. List Pages – pages that exhibit several objects, where each of the objects is structured using a specific template.
2. Detail Pages – pages that describe a single object in greater detail.

In the context of blogs, a typical home page that displays a set of brief excerpts from (chiefly recent) blog entries can be considered a list page. Detail pages are those that exhibit the entre content of a blog entry and associated comments. Each of the detail pages is usually assigned a unique URL called a permalink. An example of a list page and a detail page is shown in Figure 4. Detail pages are referred to as blog entry pages from this point forward in the report.

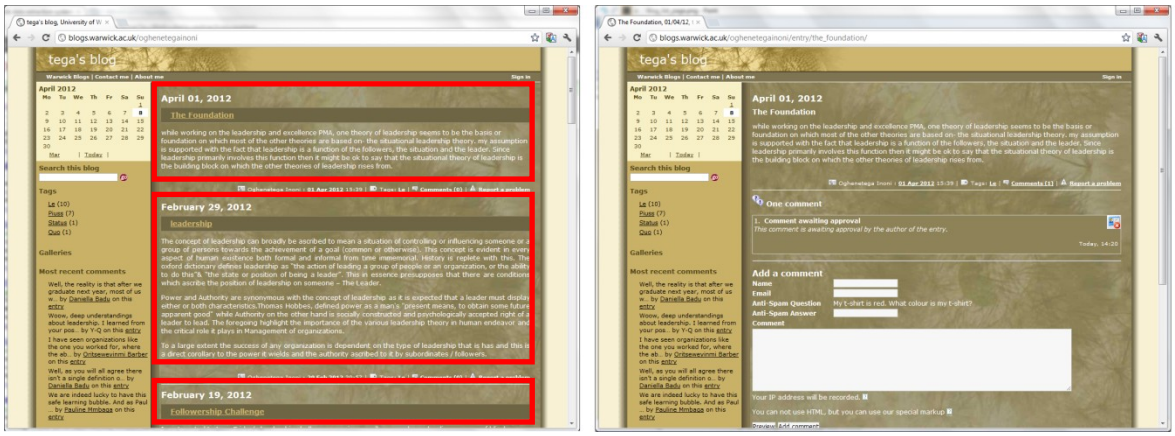

Figure 4: Example of a list page (left) and a blog entry page (right) of a blog.

[4] http://spinn3r.com/
[5] http://www.radian6.com/
[6] http://www.sysomos.com/

The BlogForever data extraction should be primarily performed on the blog entry pages, to ensure that the full content is captured. However, processing the list pages can also be necessary, for instance, when locating the blog entry pages is necessary.

3.3.1 Wrappers

The work on using wrappers for data extraction started around 1995-1996. Traditionally, a wrapper is considered a semi-automatic approach due to the requirement for labelling a collection of pages manually prior to applying the rules for extracting target data from other similarly formatted pages [16]. However, the developments in the area, suggest the possibility of employing automatic, unsupervised approaches. Hence, we adopt the general definition by Ferrara, who defines the term wrapper as follows:

> "A wrapper is a procedure that implements a family of algorithms, that *seek* and *find* the information the user needs to extract from an unstructured source, and *transform* them into structured data, *merging* and *unifying* these information for *future processing*" [25, p. 13, original emphasis].

Baumgartner [14, p. 3], describes a wrapper using a functional perspective, where "a web wrapper is seen as a mapping *f* from the parse or DOM tree *T* of a web page to a set *f(T)* of sub-trees of *T*, where each sub-tree corresponds to an extracted data object". The development of a wrapper is called *wrapper generation* or *induction*. Developing a wrapper can take different forms that use wrapper generation languages or text pattern matching. The term wrapper induction is often used to describe an inductive process of wrapper generation; it may, for instance, involve application of supervised learning. Therefore, the term wrapper generation is more general.

However, the wrappers that define the process of data extraction require adjustment with the changes of the website. Frequent changes of templates or website structures make *wrapper maintenance* an important aspect of using wrappers for data extraction. We discuss these issues in greater detail in the following sections 3.3.2, 3.3.3 and 3.3.4.

3.3.2 Wrapper Induction

Wrapper induction techniques derive delimiter-based extraction rules from a set of training examples. These techniques rely on the formatting features that define the structure of the resource. Hence, they are especially useful for data extraction from HTML pages [26]. Laender *(ibid.)* discusses some of the early and commonly used wrapper induction tools that include WIEN [27], STALKER [28] and SoftMealy [29]. These wrappers differ from each other according to the approach taken to generate extraction rules. However, all of them require a training example that requires manual intervention. More recent developments in the area of wrapper inductions include approaches such as $(LP)^2$ [30] and EVIUS [31]. We briefly outline the benefits and limitations of these approaches.

WIEN [27] is one of the first wrappers that appeared on the market. It requires a set of labelled input pages to generate a specific wrapper. WIEN is using a specific heuristic for developing the wrappers that relies on the pre-defined structure of the web page. The biggest disadvantage of WIEN is its inability to work with nested data objects [26]. SoftMealy [29] uses finite-state automata[7] for the inductive generation of rules. It breaks the string output of the HTML resource into tokens and generates the rules searching for adjacency of tokens in the training data. The process of tokenising is not limited to HTML documents and can be used with text [26]. Unlike SoftMealy or WIEN, STALKER [28] is able to work hierarchical data. It uses both a sequence of tokens representing as

[7] http://en.wikipedia.org/wiki/Finite-state_machine

well as page structure called Embedded Catalogue Tree. It is the use of the tree that allows STALKER to work with hierarchical data [26]. The main disadvantage of these wrappers was their requirement of having large amounts of manually labelled training data.

The later developments, as outlined by Turmo *et al.* [32] in their survey, add a level of automation in labelling the training data. EVIUS [31] was one of the developments that proposed a mechanism for inferring rules form from text and semi-structured documents. The process of inferring the rules in EVIUS takes into account the order in which the concepts have to be learnt. It assumes the existence of certain dependencies among concepts that have to be addressed while learning. This assumption enables inference of certain rules about the dependent concept using the information about the known concept. $(LP)^2$ is another approach that tried to combine the NLP for inducing labelling rules. It requires manual intervention for correcting inferred labels. While this approach does not automate data extraction completely, it demonstrates the potential of employing NLP techniques (discussed in details by Sarawagi [33]) for reducing the efforts for developing training sets. However, it appears that the potential for using semi-automatic wrapper induction tools within the context of BlogForever remains limited. These tools have limitations in working with complex objects, may not be easy to use and will require an overhead on developing and maintaining the wrappers [26].

We devote special attention to another recent advancement, that is OXPath [34], developed as part of the DIADEM[8] project. OXPath *(op. cit.)* aims to address the limitations of the available solutions. At this stage of the development (see GitHub Project[9] for details), OXPath allows user interaction with, the so called, 'deep web' that is made possible by introducing functionality for filling out HTML forms or navigating through a sequence of specific hyperlink paths.

The functionality provided by OXPath can be particularly useful within the context of BlogForever. Firstly, generating rules for navigating through the archive of the blog or through the sequence of blog entries can enable crawling and extracting the necessary content without performing a full domain-level crawling. Secondly, the possibility of interacting with HTML forms can, if necessary, enable customised extraction of the content; for instance, by performing a blog-search on a specific topic and extracting only the entries returned as a result of the search. Furthermore, future developments following this direction of research may soon offer solutions for automating or further reducing manual input.

3.3.3 Wrapper Maintenance

Once the wrapper is generated, it can be used for extracting data. However, websites do change, and in some cases, fairly regularly. Using the earlier generated wrapper on a modified website may no longer be possible. Hence, updating the wrappers according to the change and maintaining their performance remains an important issue that needs to be addressed when planning to extract data periodically. If wrappers are to be used for extracting data from blogs, than solutions for wrapper maintenance should be explored.

One of the solutions highlighted in the literature suggests ensuring that the tools for generating wrappers are easy to use. User friendly tools may reduce the workload for generating and re-generating wrappers. Ideally, however, a mechanism should be in place to detect the change and be able to repair the problem automatically. We discuss some of the wrappers that were developed with user interaction in mind and proceed to outline some of the possible mechanism for automating the task of wrapper maintenance.

Among the wrappers that were designed with human interaction in mind is Thresher [35], which enables non-technical users to generate wrappers by highlighting certain sections of the web

[8] http://diadem.cs.ox.ac.uk
[9] https://github.com/diadem/OXPath

document within a browser. User input is then analysed by using *tree edit distance* between DOM sub-trees for generating a wrapper. User interface undoubtedly simplifies the process of wrapper induction. However, Thresher remains limited in a number of ways, for instance, in its ability to specify negative example or actively interacting with the document.

In fact, Thresher is not the only solution that offers graphical user interface options. A number of active companies on the market offer a range of customisable data extraction-powered services [14]. Denodo[10], Lixto[11], Krapowtech[12] and Deixto[13] are some of those ventures.

Developments in the area of wrapper automation can support resolving the two main problems of wrapper maintenance [16, 36]:

1. *Wrapper verification problem:* automatic identification of changes within the target document that requires wrapper modification, and
2. *Wrapper repair problem:* automatic repair of the wrapper.

Both of the problems can be resolved by introducing mechanisms that enabling learning the characteristic patterns of the target document, which are then checked across the characteristics of the extracted data to monitor its correctness. If comparison fails, the same characteristics can be used to identify the correct items (also called re-labelling). However, the process of re-labelling may not be suitable if the target document changes substantially (i.e. beyond minor formatting changes) [16].

Dalvi *et al.* [37] propose a solution for the wrapper maintenance problem. They particularly focus on websites that can be (repeatedly) changed by their scripts. They take temporal snapshots of the DOM tree of the target page for developing a tree-edit model and then use it for improving the wrapper construction. This approach employs a probabilistic approach and uses historic data, which allows them to report improvements of robustness measures from 40% to 86%. Other solutions in the direction of wrapper verification and repair include the work of Chidlovskii *et al.* [38], who suggest using grammatical inference for wrapper recovery and report a considerable improvement in scaling wrapper maintenance. Alternative approaches include [39], [40] and [41].

Wrapper maintenance within the context of BlogForever can be considered a simpler task due to the specific data model and access to the historical data. However, the solutions outlined above may also be considered useful.

3.3.4 Automatic Wrapper Generation

The main limitations for using data extraction wrappers are the prerequisite of having a labelled training dataset and the requirements for having maintenance mechanisms. However, various techniques have been developed to automate or reduce the effort in generating wrappers. Instance-based learning is one of those techniques.

Although not fully automatic, instance-based learning techniques are used to identify examples that can significantly reduce the workload of labelling. They use only a single example of a labelled page (i.e. instance) comparing the items from the new page with the suffixes and prefixes of the labelled example. If the unlabelled item cannot be identified it is then marked for manual labelling. This process is called active learning [16].

Zhai and Liu [42] discuss their implementation of an instance-based learning technique by means of the IDE (Instance-Based Data Extraction) algorithm. They suggest a measure of similarity/distance

[10] http://www.denodo.com
[11] http://www.lixto.com/index.php/products/middleware-tools/
[12] http://kapowsoftware.com/solutions/web-intelligence/web-data-extraction.php
[13] http://www.deixto.com

that is used for comparing the labelled items to those from the target document. The authors report good results (99.9% recall/precision and 99.5% accuracy) after evaluating their approach on 24 e-commerce websites. Its application on blogs should theoretically be possible, however, given the requirement for preserving large number of blogs, labelling even a single page may still be resource expensive. Hence, we explore this area further for an alternative, fully automated solution.

Among the fully automated wrapper generation approaches that deserve our attention are the work by Zhao *et al.* [43] and Bronzi et al. [44]. Zhao *et al. (ibid.)* identified the possibility of generating wrappers automatically for extracting a list of search results returned by various search engines or search-enabled tools. While focusing on a specific domain, their approach may also be useful within the context of BlogForever. The wrapper generation employs both, visual information, such as coordinate system to describe the location of rendering boxes, as well as tag structure, such as content line separators. The process continues by grouping extracted blocks according to various measures of visual and structural similarity and building the wrapper accordingly.
The approach taken by Bronzi *et al.* [44] can be potentially useful for extracting data from blogs. Bronzi *et al.* explore the possibilities of exploiting redundant data present in web documents (e.g. detail pages). The primary difference of the approach is in using multiple pages as input as a means of validation. The inferred rules become more resilient to work on the other pages and, according authors' evaluation improve the precision and recall of the data extraction. Given the fact that blog entry pages are most common and of interest for data extraction, a similar approach to Bronzi *et al.* is suggested here for consideration. In summary, wrapper generation and automatic wrapper generation approaches in particular can be suitable for the BlogForever data extraction methodology.

3.3.5 Automatic Data Extraction

This section discusses the possibilities of automating the process of data extraction and further minimising or eliminating the need for human intervention. While automatic wrapper generation offers a viable solution for the data extraction from blogs, it is necessary to consider alternative automatic approaches.

The main alternatives to wrapper induction are the automatic data extraction approaches that do not require generation of wrappers. Automatic extraction may not involve generation of wrappers, because some of the data records may be present as a list in one particular page of a website. For instance, a list of publications exhibited by a blogging scientist on one of the blog's pages, or a list of conferences that the author published on his/her blog. Blog entries and pages that exhibit similar content may appear on a range of blogs. However, generating a wrapper across these blogs or for a specific page within a single page is not suitable. This subtle distinction is important when considering the methodology and choosing most appropriate approach.

Despite the subtle differences, the general progress in data extraction research, may feed both of the approaches, since they both rely on common methods used in machine learning, natural language processing or analysis of semi-structured documents. Hence, some intersections that can inform both of the approaches are expected. Both approaches can benefit from research into: Web Block Classification, Website Exploration, Understanding Web Tables, Understanding Forms or Semantic Annotation. For instance, solutions for automatic data extraction may involve identification of data records and their boundaries based on a given list page. Heuristic rules, domain ontologies or other techniques for automating the process of identifying data records can be used here. There is a large number of proposed solutions, evaluations and tools developed for implementing the task of data extraction. Some of these are discussed in [45], [46, 47], [48], [42], [49], [50] and many others. RoadRunner [51, 52] is among the freely available tools, and is frequently referred for comparison and benchmarking. We discuss most relevant of these approaches further down this report in line with blog data extraction (Section 6).

Scalability of automatic approaches that enable data extraction from large numbers of websites and reduced maintenance costs are the main benefit of automatic data extraction. However, their disadvantages include extraction of large amounts of unwanted or inability to coherently integrate the data extracted from divergent resources. Techniques that involve some human intervention remain more accurate in general [16].

3.3.6 Other Approaches

Laender *et al.* [26], in their survey of data extraction tools, identify a group of tools that rely on the inherent structural features of HTML documents. This group is referred to as *HTML-aware*. HTML aware tools rely on parsers and their ability to generate the tree of web documents. Once the web documents are parsed, these tools rely on the tag hierarchy of the document tree for performing a semi-automatic or automatic data extraction as discussed earlier in section 3. Many HTML-aware tools produce wrappers. Some of the HTML-aware tools have already been mentioned to describe the processes of wrapper generation and maintenance. Among these tools are RoadRunner [52], XWRAP [53] or W4F [54]. An extended introduction to HTML-aware tools is discussed as part of the BlogForever spider prototype report [4]. They are considered suitable and largely relevant for consideration for performing data extraction from the domain of blogs. A thesis that discusses various HTML-aware tools has been written by Boronat [55]. Some of the tools described by Boronat *(ibid.)* are discussed later in this report.

3.3.7 Summary

The available approaches to data extraction, as reviewed in this section, are numerous. The selection of appropriate data extraction techniques should consider the level of human intervention requires, the trade-offs of precision and recall. Table 1 **and** Figure 5 **outline the possibilities** for data extraction for blogs.

Table 1: Comparison of general approaches to data extraction

	Can it be used for the entire BF blogosphere	Extract all data desired (recall)	Quality of extracted data (precision)	Automatic
Use of RSS	YES	Limited	Very High	YES
Use of API	limited	Limited	High	YES
Database Access	NO	YES	Very High	Semi-automatic
HTML processing	YES	YES	Average	Semi-automatic

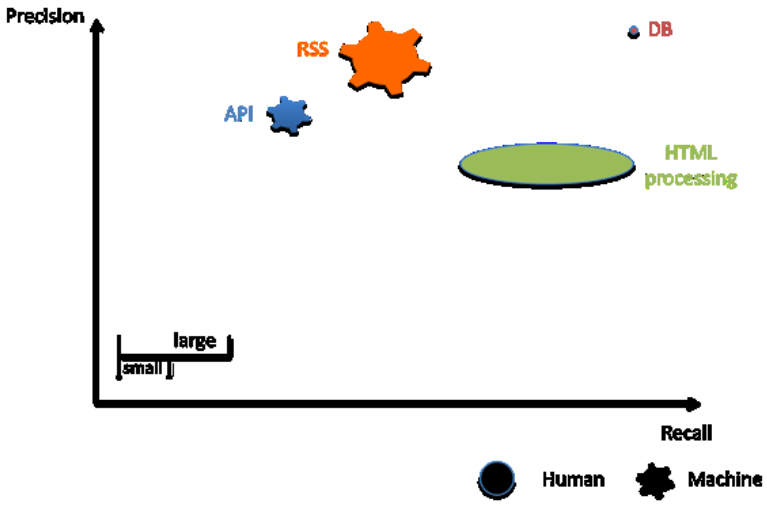

Figure 5: Graphical representation of graphical approaches to data extraction.

Despite the fact that web feeds constitute a useful resource for extracting information from weblogs, they remain insufficient for ensuring robust information extraction. Hence, the consideration of HTML documents that render weblog data remains paramount. In the context of BlogForever, processing RSS feeds along with the generated HTML code should be adopted. The selection of the method for extracting data from HTML should also be scalable. Hence the tools and methods that require human intervention are less likely to be suitable for in the context of BlogForever.

4 Prerequisites of Blog Data Extraction

The approaches to web data extraction are often borrowed from the wider field of data mining. The primary differences between the approaches are driven by the methods of data collection. Unlike traditional data mining, processing of web data often requires to first crawl and collect the data [16]. The methods for processing crawled data can therefore depend on the ways the data are being collected. This section discusses some of the aspects of crawling that can have direct implications on the process of data extraction. It aims to highlight the types of pre-processing of the collected material that are necessary for performing data extraction.

We start by summarising the plans and progress for developing the spider component of BlogForever and progress to highlight some of the aspects of crawling directly related to the task of data extraction. Among those tasks are the following.

a. Examination of web resources prior to crawling and exclusion of irrelevant resources from crawling. For instance, it is important to ensure that crawling of a generic website or an online shop is avoided (4.1.1). While BlogForever solutions assume intervention of experts when choosing blogs for preservation, it remains necessary to ensure that irrelevant resources are excluded due to use of ping servers or similar methods for extending the collections.
b. Identification and classification of relevant resources published in blogs. For instance, it is important to locate web feeds of a blog and select most appropriate ones to support the process of data extraction (4.1.3)
c. Optimising the process of crawling by avoiding collection of duplicate content. For instance, it is important to manage the computational resources and to avoid collection of multiple representations of the same blog content that is rearranged by certain filters such as blogs. This issue is discussed in detail as part of an empirical evaluation discussed in detail in section 5.1.

4.1 Blog Spider

One of the key components of the anticipated BlogForever platform is the Spider. Spiders, also known as crawlers or robots, are software systems that access and download large numbers of web pages [56]. The downloaded pages can be aggregated, indexed, searched through, preserved, analysed and processed in many other ways.

The data extraction task is reliant on the existence of a specialised blog spider. Unlike common spiders, the blog spider should be designed to handle timely blog updates, fetching the content and preparing it for further processing. We highlight some of the prerequisite custom features of the blog spider that are essential for data extraction.

4.1.1 Filtering of Blogs from Other Resources

Since, the focus of BlogForever is on blogs, filtering blogs from other resources and preventing their ingestion into blog repositories is important. If non-blog resources are not eliminated from crawling, the data extraction process may suffer. If machine learning techniques are used, for instance, non-blog resources may affect the process of data extraction. It is therefore, important to introduce mechanisms that filter blogs from the rest of the Web. This is particularly important when a large corpus of blogs is being prepared. When working with large datasets that exceeds, for instance, half a million blogs, manual intervention in checking resources will be resource intensive. While expert intervention in small cases will be most appropriate, use of automated techniques to reduce the workload of the manual input for filtering out irrelevant resources would be beneficial.

We suggest some approaches for filtering blogs out, but assume that the filtering is a prerequisite to data extraction. The task of filtering can be addressed by using classification. We discuss its relevance below.

The problem of classification can be approached by performing a binary or multiclass classification. Given the possibility of classifying pages according to their functions, genre or subject it may be necessary to assign more than one label to a resource. For example, when performing a subject classification, a resource can be labelled according to one or more subjects, such as 'Business', 'Science' or 'Sports'. At the same time, the same resource can also be classified according to its functional characteristics or role, such as a personal website, organisational website or course website [22]. Depending on the chosen perspective on blogs, the classification may require consideration of single or multiple sets of classes as shown in Figure 6.

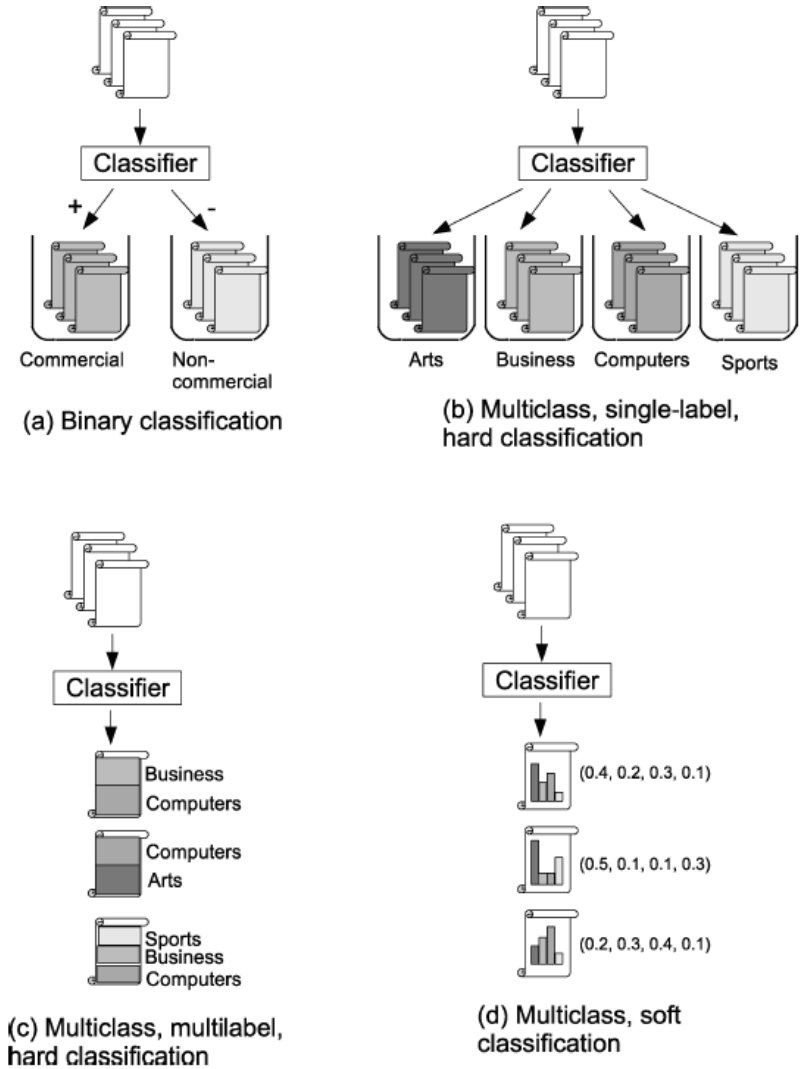

Figure 6: Types of classification (from Qi and Davison [22])

The problem is not trivial, since it may involve various dimensions, such as subject classification, functional classification or sentiment classification. However, some approaches can be applied to the problem of filtering blogs from other resources.

Binary Classification

Classification that defines whether a web resource is a blog is usually considered as binary. Nanno *et al.* [57] identify blogs based on a set of simple heuristics. The choice of their heuristics is not justified and appears incomprehensive. They attach greater importance to the temporal aspects of blogs (i.e. date/time, chronological order and frequency of publication no less than a month). They consider resources to be non-blogs if they use tags (e.g. 'bbs','chat', 'reply' and 're:') associated to other tools, such as forums or chat. An alternative to the temporal approach includes domain and URL analysis. Since the majority of blogs are hosted by blogging service providers, such as Blogspot[14], WordPress[15] or LiveJournal[16], a simple process of URL pattern matching should provide a sufficiently good level of accuracy [58]. A simple Python-based implementation of URL-based blog detection is available [59]. The URL-based techniques do not require parsing the page and are therefore less computationally expensive. However, since a weblog may be hosted under any hostname and its URLs may be rendered arbitrarily, these techniques are not sufficient for all cases.

Alternative approaches that employ machine learning have been evaluated by Elgersma and De Rijke [60]. They suggest that off-the shelf tools and available algorithms can be applied with a sufficient level of accuracy (i.e. around 90%) when using a combination of human-selected attributes related to blog features along with URL match. A machine learning approach is also patented by Google [61]. It uses a sequential approach in analysing the URL patterns, then moving parsing the resources and searching for existence of RSS or other blog-like features. The identification of web feeds considered to be a well-explored domain. The discovery of feeds for RSS, for instance, is based on exploiting the <link> tag and its attributes such as *rel*, *type*, *title* and *href* [62].

Multiclass Classification of Blogs

Among the multiclass classification techniques one may include classification of blog genre, where more than two classes are available for classification. For instance, Nowson [63] and Qu *et al.* [64] propose classifications by genre where blogs are categorised as personal diaries, political or news. The selection or development of a well-defined taxonomy however is important for the multiclass classification Qi [22]. As part of the data extraction methodology, binary classification and identification of blogs from other resources dominates the agenda. The review of Qi *(ibid.)* provides a useful overview of relevant approaches, suggesting the use of textual and visual elements that are directly included in web pages, and extending this approach by including features from neighbouring pages. Appropriate mechanisms for classification are necessary prior to initiating data extraction.

4.1.2 Distinguishing between Blog Portals, Blogs, Posts and Comments

Prior to performing extraction from a given URL, it is necessary to identify whether the given URL is a blog hosting platform, a blog or a single post. URLs, received from a ping server are not classified or marked. Each URL can represent various resources such as a blog host, a blog post or a comment. Depending on the type of the resource, the data extraction approach should differ.

The use of machine learning can be useful in this direction too. Decision tree algorithms can be implemented based on various attributes for distinguishing between the URIs. The output of this URI 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. The content can also be analysed using the similar approach for extracting full text and may be referred to as *content filter*. Machine learning using decision trees can also be used to identify conceptual blog elements from HTML markup. More specifically, ID3 and Neural Network algorithms were tested during the Spider Prototype application development in order to assess their performance in blog data extraction (for details see [4]). The ID3 algorithm has preference for smaller decision trees (simpler theories). It is flexible and can be used both for analysing blog hosts as well as content per blog post and comments. Neural networks consist

[14] http://www.blogspot.com
[15] http://www.wordpress.com
[16] http://www.livejournal.com

of connected nodes - artificial neurons. Neural networks are used for modelling complex systems and identifying patterns in data. Neural network approach is more complex than ID3 and not equally scalable. The use of neural networks may not be suitable for working with vast quantities of URIs received from ping servers. Hence, ID3 is a more likely candidate for implementing the prerequisite of distinguishing the resources due to the requirements of scalability.

4.1.3 Classification of Resources and Identification of Relevant Blog/Comment Feeds

Web feeds, like RSS and Atom, can be essential for data extraction from blogs. They are represented in a machine readable format and are most commonly used to provide content syndication and notification of updates from multiple websites into a single application [65]. However, while use of a web feed for data extraction can be very useful as Davis *et al.* [66] suggest, the task has its challenges.

Blogs may exhibit more than one web feed. A typical WordPress blog has two feeds: one for posts and another for comments. Less frequently, but still considerably often, blogs can exhibit feeds that distribute content selectively. For instance, feeds for Most Popular posts or Collections by Certain Categories are also possible. Intelligent decision-making on selecting appropriate web feeds is necessary. The use of heuristics of web feeds along with identification of generating blogging platforms (e.g. WordPress) can be most appropriate within the context of BlogForever.

4.1.4 Support for User-Click Simulation and AJAX

Since blogs are generally hosted by a web application and are rendered dynamically, some of these applications may include use of AJAX[17] for updating pages asynchronously. Therefore, prior to initiating the task of data extraction it is necessary to ensure that the necessary content is accessible. Furthermore, it may also be necessary to simulate the user click stream for displaying some of the elements of the page [25]. For instance, the full list of comments on a post may not always be displayed and may require user interaction via the browser for obtaining the missing content for extraction. A potential solution for enabling interaction with dynamic HTML and AJAX comes from SeleniumHQ[18].

4.1.5 Eliminating Spam

Given that the data extraction methodology is likely to rely on machine learning techniques, it is necessary to ensure that the employed datasets are free from spam. Spam, that may include splogs or irrelevant comments may distort the outcomes of machine learning and result in unexpected irregularities. Suggestions for spam detection and removal were discussed elsewhere [5]. However, it is necessary to emphasise that spam detection mechanisms should be put in place prior to proceeding to the data extraction stage.

4.2 Parsing: HTML and XML Parsers

In addition to a capable web spider, the resources need to be parsed before any data extraction takes place. However, the task of parsing is far from trivial. Obstacles to parsing involve session identifiers, client-side JavaScript code, missing or invalid formats. A solid data validation and error

[17] http://en.wikipedia.org/wiki/Ajax_(programming)
[18] http://seleniumhq.org/

recovery mechanisms are necessary to ensure effective handling of poorly structured web resources such as blogs.

XML and HTML have been used for different purposes. Although both of them constitute a language for representing semi-structured data, HTML is mainly presentation-oriented whereas XML, being more pedantic, is suitable for database applications [67]. Thus, the current status is that browsers mainly consume HTML, while developers and programs work with XML data.
Similar to the above observations concerning their usage, parsing XML and HTML share certain resemblances, but are not the same processes. The main reason originates from the fact that HTML is written in SGML (Standard Generalized Markup Language), which in turn was used to introduce XML a few years after HTML was invented[19] (see Figure 7). To be more precise, SGML is a meta-language, rather than a language like HTML or XML[20]. This common – to a certain extent – yet different history of HTML and XML can be observed after a quick survey on the corresponding parsers. Generally, HTML is *not* considered to be a "de facto" valid XML document. When an HTML document conforms to XML, this is an XHTML document (eXtensible HyperText Markup Language) and can therefore be valid input for an XML parser. Similarly, when an HTML document is not an XHTML one, a special purpose parser is needed. In fact, a frequent practice that attempts to unify the parsing of XML and HTML documents is to use HTML to XHTML converters.

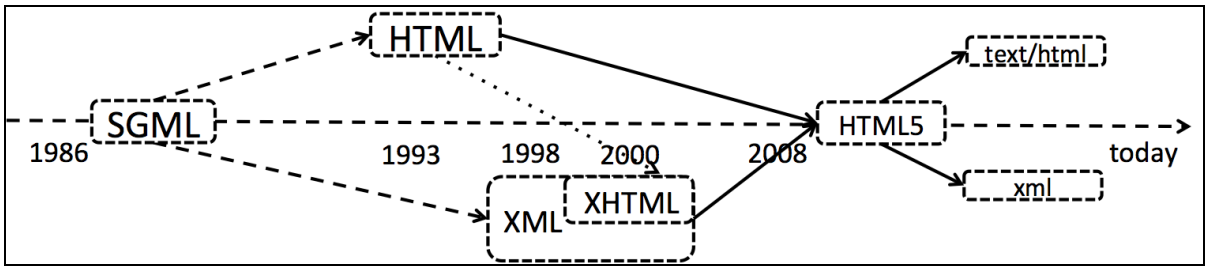

Figure 7: Timeline of HTML and XML

In addition to the inherent differences between an HTML and an XHTML document, a document may be malformed (e.g. not adhering to proper nesting and closure syntax). Malformed documents need to be fixed prior to parsing, since handling irregularities is not considered a built-in feature of a parser. A collection of libraries and tools are available for detecting and fixing these types of errors (e.g. HTML Tidy[21], xmlint[22], HTML Purifier[23] or W3C Markup Validator[24]).

After the HTML/XML document has been checked and fixed, the parser is able to read the text, analyse it into a sequence of tokens and eventually derive its formal structure. The available list of tools and libraries that perform the above operation is rather extensive. The following subsections discuss some of the available parsers.

4.2.1 Parsing

Since pre-processing and the performance of data extraction tools are heavily dependent on parsers we discuss some of the available open source libraries that can be considered for parsing blogs.

[19] A detailed review on the differences between SGML and XML can be found at http://www.w3.org/TR/NOTE-sgml-xml-971215.
[20] XML AND HTML are "applications" of SGML [http://validator.w3.org/docs/sgml.html].
[21] http://tidy.sourceforge.net/
[22] http://xmlsoft.org/xmllint.html
[23] http://htmlpurifier.org/
[24] http://validator.w3.org/

4.2.1.1 JSoup

JSoup is "a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods"[25]. JSoup is an open source project distributed under the MIT licence. Moreover, JSoup is compliant to the latest version of HTML5 and parses a document the same way a modern browser does. This means that JSoup allows processing an HTML document either by traversing the DOM tree, or by using CSS selectors (also known as jquery-like). Table 2 presents an example of how to use JSoup.

Table 2: Example of how to use JSoup using DOM or CSS selectors.

`File input = new File("/tmp/input.html");`
`Document doc = Jsoup.parse(input, "UTF-8",`
`"http://example.com/");`
 	
`Element content =`
`doc.getElementById("content");`
`Elements links =`
`content.getElementsByTag("a");`
	`File input = new File("/tmp/input.html");`
	`Document doc = Jsoup.parse(input, "UTF-8",`
	`"http://example.com/");`
	
	`Elements links = doc.select("a[href]"); // a`
	`with href`
	`Elements pngs =`
	`doc.select("img[src$=.png]");`
	`// img with src ending .png`

DOM method	CSS selector method

4.2.1.2 ROME Tools

ROME Tools suit is an open-source (Apache licence 2.0) Java library that allows for the parsing and reading of syndication formats like RSS and Atom. One of the biggest strengths of the library is its compliance with the big collection of various "flavours" of syndication formats[26]:

- RSS 0.90
- RSS 0.91 Netscape
- RSS 0.91 Userland
- RSS 0.92
- RSS 0.93
- RSS 0.94
- RSS 1.0
- RSS 2.0
- Atom 0.3
- Atom 1.0

Moreover, ROME is built on top of JDOM[27], which is one of the most popular Java representations of an XML document. ROME is using JDOM's XMLReader SAX processing methodology in order to parse and read a feed.

4.2.1.3 Beautiful Soup

Beautiful Soup[28] is an HTML parsing library written in the Python programming language. It is also tolerant to 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.

[25] http://jsoup.org/

[26] A comprehensive review of the various RSS formats can be found at http://www.rssboard.org/. More information about the Atom format can be found at the Internet Engineering Task Force web site (http://tools.ietf.org/html/rfc4287).

[27] http://www.jdom.org

[28] http://www.crummy.com/software/BeautifulSoup/

4.2.1.4 HTML Agility Pack

HTML Agility Pack[29] 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[30] in newer versions (via a LINQ to XML interface).

4.2.1.5 WatIn

WatIn[31] is a .NET library primarily for Web testing that can be used for HTML parsing. It is not as well suited for parsing as HTML Agility or Beautiful 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.

The evaluation of the three parsers, namely HTML Agility, Beautiful Soup and Watin has been performed and reported as part of an earlier deliverable (see [4]). The results indicated largely similar performance results for all three of the libraries. Technical details regarding the code implemented and tests conducted can be found at Appendix E.

The characteristics of the overviewed parsers is presented in Table 3. The decisions on use of specific libraries as part of this report were based on the selection of a Java programming language for developing and evaluating a data extraction prototype. Hence, the prototype has been based on the use of Rome tools in order to support the processing of various formats of web feeds. For the purposes of HTML parsing the JSoup library was selected due to the affordances it offers for working both with CSS selectors and the traditional DOM tree access.

Table 3: Characteristics of the reviewed parsers

	XML	Web Feeds	X/HTML	AJAX	CSS Selectors	Open Source	Language
JSoup	×	×	✓	×	✓	✓	Java
ROME Tools	×	✓	×	×	×	✓	Java
Beautiful Soup	✓	×	✓	×	×	✓	Python
HTML Agility Pack	×	×	✓	×	×	✓	C#
WatIn	×	×	✓	✓	✓	✓	C#

4.3 Summary

This section introduced the notion of pre-processing as a set of prerequisites for performing data extraction. Separating this processing from the rest of the data extraction can lead to a more optimised use of resources. These prerequisites include general tasks of classifying, filtering and parsing web resources, preparing these for data extraction. In addition to clarifying the boundaries to the notion of data extraction discussed in this report, this section reviews some of the literature making specific suggestions for pre-processing the web data.

The discussion in this section is conducted in relation to the BlogForever Spider component, as well as the general processes such as parsing. Parsing is one of the fundamental processes required for working with Web data. There is a large number of Open Source and commercial parsing libraries

[29] http://htmlagilitypack.codeplex.com/
[30] http://en.wikipedia.org/wiki/Language_Integrated_Query
[31] http://watin.org/

and tools that can be employed for reading blog content. This section overviews some of the frequently used parsers and appropriate tools in line with the types of content exhibited in blogs (e.g. RSS/Atom feeds, X/HTML). Furthermore, this section highlighted the processes that are necessary to avoid redundant processing of irrelevant resources and proposed potential solutions referring to literature.

5 Case Study on Pre-processing: Lessons Learnt

In the following section, a case study of data extraction from wikis using the Heritrix crawler (used to refer to the spider in this section) is presented. The purpose of this case study is to demonstrate web spider performance in specific cases and identify their implications to blog data extraction.

5.1 Use of Heritrix with Wikis

This case study describes the work of the University of London (UL) collecting web content for the Joint Information Systems Committee (JISC). The aim is to gain insight into the operation of another spider and draw lessons related to pre-processing and data extraction. The study is based on an exercise of collecting copies of UK University project websites and adding them to the UK Web Archive[32]. The websites were harvested using the Heritrix[33] crawler and rendered on the public archive site using a British Library version of the Wayback Machine[34]. In this section describe the context of the study along with the outcomes that are relevant for the aims of this document.

Around 2006-2007 it became common for these JISC projects to start using wikis to record and deliver their project work. Popular platforms in use were MediaWiki[35] and Twiki. The study was aimed to capture and archive these wiki-based websites. We encountered some issues associated with crawling the websites and tried resolving these as discussed below.

The Web Curator Tool[36], which is the user-friendly interface for Heritrix, was used to look into the harvest data and size of all incoming crawl. The harvests of wikis took a long time to complete, and were exceptionally large when they arrived. This was a matter of some concern to the archiving team. Harvests that take a long time consume a lot of bandwidth, which in the UK Web Archive is a shared resource. Large harvests also consume a lot of storage, which is another cost, and these websites are being kept permanently. Since these websites would be crawled every quarter, these issues could escalate.

5.1.1 Behaviour of Heritrix with Wikis

The Heritrix crawler works on a recursive process. Gordon Mohr's *Introduction To Heritrix*[37] describes it as follows:

> *Executing a crawl repeats the following recursive process, common to all web crawlers, with the specific components chosen:*
>
> *1. Choose a URI from among all those scheduled*
> *2. Fetch that URI*
> *3. Analyze or archive the results*
> *4. Select discovered URIs of interest, and add to those scheduled*
> *5. Note that the URI is done and repeat*

The assumption was that the Heritrix crawler was requesting too many pages from the wiki. Considering the fact that a wiki is database-driven it should not be surprising that it is creating a lot

[32] http://www.webarchive.org.uk/ukwa/
[33] https://webarchive.jira.com/wiki/display/Heritrix/Heritrix
[34] http://archive-access.sourceforge.net/projects/wayback/
[35] http://www.mediawiki.org/wiki/MediaWiki
[36] http://webcurator.sourceforge.net/
[37] https://webarchive.jira.com/wiki/download/attachments/5441/Mohr-et-al-2004.pdf

of its pages "on the fly". In one sense, a wiki is a form of content management system. There are four main reasons why MediaWiki generates so much crawlable content:

1. **Edit history**. Since a wiki is editable by multiple contributors, which is part of its core function, it means there are numerous past versions of pages stored in the wiki database (see Figure 8). These can be retrieved from the wiki database by a number of values, including version numbers, dates of edit, and names of editors. In short, the wiki holds a retrievable version of its own edit history. This can be seen if one looks at the "History" tab of our test wiki. It displays radio buttons and a search function that enables sophisticated interrogation and searching of the edit history. It also allows retrieval of the earliest or latest versions, and allows the user to retrieve batches of pages, from between 20 to 500 at a time.

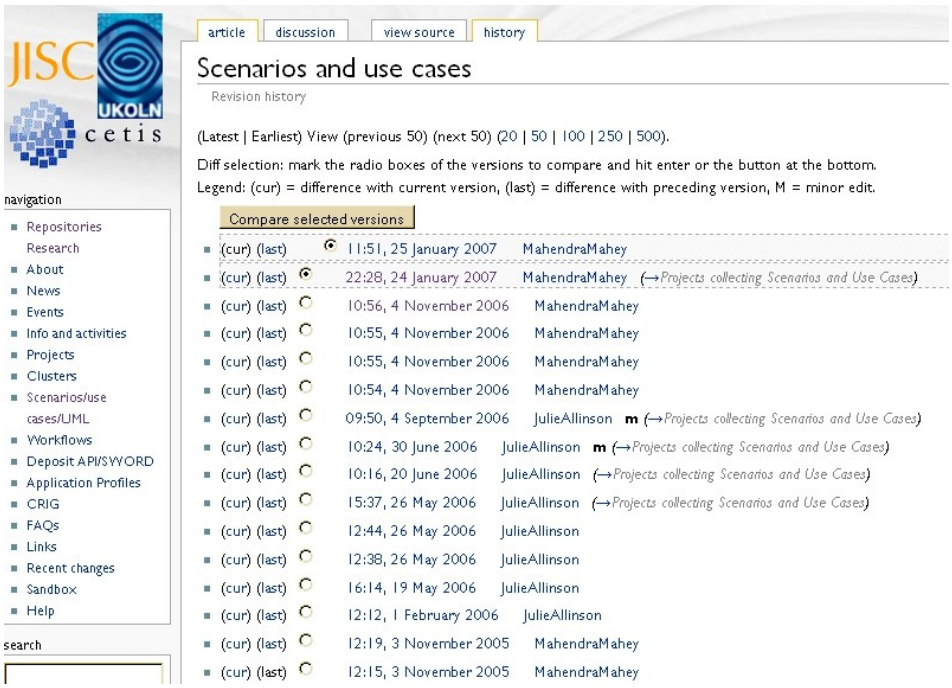

Figure 8: Example of tracking the history of edits in wikis

2. **Discussion**. MediaWiki offers a "discussion" function, where changes and edits can be discussed by the team of contributors, adding yet more textual richness to the edit history.

3. **Print versions**. A MediaWiki page can also be rendered as a printable version. This simple function recasts the data as printer-friendly text.

4. **Logins and other wiki actions.** In order to contribute to the wiki, editors have to log in to edit pages. The step creates more pages, depending on the action that the editor wishes to perform.

All of these functions create content which, as far as the Heritrix web crawler is concerned, constitute "different" pages - because they all have their own number ID stored in the database, or are identifiable by a specific code. It's possible that, when performing a crawl of a wiki, Heritrix does not cease gathering until it has requested and copied every single one of these pages.

5.1.2 Exclusion Filters for Wikis

UL's solution to this "bloat" issue was to experiment with exclusion filters. This is a simple way of programming Heritrix such that it can be trained to ignore certain things from the gather. It can

ignore types of file format, entire folders in a web structure, or even strings of code. This last feature was used to program Heritrix effectively.

An operation like this starts with the assumption that the archived version would not have all the functionality of the original wiki. This involves making a selection decision. The expectation was that the discussion / edit / history functions of the wiki in the archive copy would be lost. It would also be possible to lose the edit history, the audit trail that showed how the wiki content was written. Generally, this is no great loss for the purposes of the UK Web Archive, as scholars who browse the archived copy of a wiki are not expecting to be able to edit pages, nor join in the discussions, nor browse the history of stored versions of pages. Indeed in a lot of cases, they would require a login to do so. The users of the UK Web Archive simply want to see the results of a project team's work.38

The second step was to understand each section of the code and outcome it produced in the wiki. There are two ways of finding this out, and one method is to examine the log files. The log files created by Web Curator Tool, complete records of every single client-server request, show patterns to the crawl.

Another method is to hover over or follow links in the live target wiki. For example: `http://www.ukoln.ac.uk/repositories/digirep/index.php?title=Talk:JISC_Digital_Repository_Wiki&oldid=2185`. This is a History query retrieving an older version of a page. The part of the code that interests us is `"&oldid=2185"`, showing the old ID of that particular page title.

Lastly, these exclusions were expressed in Heritrix. This is a matter of adding the extracts of wiki code to the exclusion function, using regular expressions. The basic principle is using the code `'.*'` for exclusions; they must appear either side of the code that needs to be excluded. Note also the addition of the backslash on `'.*?title=Special.*'` to 'escape' the `'?'` character.

The list of exclusion codes used for wikis is shown in Table 4.

Table 4: List of exclusion codes used for crawling wikis.

Exclusion Codes
.*&oldid.*
.*&diff.*
.*&limit.*
.*&direction.*
.*Recentchanges.*
.*/Special.*
.*\?title=Special.*
.*&action=edit.*
.*&action=history.*
.*&section.*
.*&redlink.*
.*&printable=yes.*
.*&redirect=no.*

These exclusions have the effect of telling Heritrix to exclude certain pages and actions from the harvesting action.

[38] This selection decision was queried by one web archivist, who said "I agree insofar as the edit pages are concerned, but there's an argument for capturing the history pages as an audit trail of who contributed what and when. This is interesting and valuable stuff – a particular (or should that be significant?) characteristic of wikis is that they are collaborative tools. The history pages are evidence of this and I imagine there will be people who want to know what was done by whom and when. You could make the same argument for the discussion pages, but in my experience the discussion pages are not widely used."

The results of these trials produced archive copies of wikis which were considerably smaller in size, and took a fraction of the time to gather. However, as expected, the results lost functionality. As well as losing the discussion / edit / history functions, it also eliminated the Toolbox functions, the 'printable' views of the content, and the login pages.

5.1.3 Data Extraction

The process described above resembles a form of data extraction that applies to wikis. However, there is one key area where this process overlaps with blog content. The exclusion technique might also work for blogs, where (depending on the platform) the same content can appear in different places and can be harvested many times over. The comment from Gordon Paynter describes this below. Note his third paragraph.

> *"We've encountered pretty the same Wiki issue as Ed describes at the National Library of New Zealand. The curators used exclusion filters much as Ed describes to limit the harvester (I will also be sending them this article in case they need to harvest mediawiki).*
>
> *"The biggest problem is not the edit history, but the differences between versions of a page. A page with 10 versions will have dozens of "diff" pages (Ed's .*&diff.* pattern presumably takes care of this). So even if we want to capture the edit history, we don't want the differences.*
>
> *"We have similar but more frequent issues with blogs. On some platforms each post appears on the homepage and on its own page, but also separately on the archive pages for the relevant year, month and day, and also on the page for any tags. So you can wind up harvesting the same content many times over. Again, crafting exclusion filters is the solution."* [39]

5.1.4 Exclusion Filters for Blogs

An experiment was carried out by UL to discover whether a similar filtering method could work successfully with blogs that are harvested by Heritrix. A test gather was run on a blog called ENABLE[40], hosted by Google.

Without filters, the gather took two hours to harvest and produced a gather 1.1 GB in size. With Web Curator Tool, it is possible to perform Quality Assurance by browsing the folder tree of the harvest itself (WCT calls this the "Graphical view of harvested data"), and also examine the crawl log, which is a complete record of all client-server requests, saved as a text file.

When the folder tree was browsed, it was noted that the "core content" of the blog had been successfully harvested as expected, and stored in a folder hierarchy divided into years and months, as partially illustrated in Figure 9.

[39] Comment posted by Gordon Paynter on 14th March 2009, on http://dablog.ulcc.ac.uk/2009/03/10/working-with-web-curator-tool-part-2-wikis/
[40] http://jiscenable.blogspot.co.uk/

Figure 9: Folder hierarchy of harvested blog content

It was also apparent however that large numbers of additional HTML pages had been created, with file names such as this:

```
http://jiscenable.blogspot.co.uk/search?updated-max/0752008-10-
01T23:59:00-07:00/46max-results/0757/46reverse-
paginate/75true/46start/07521/46by-date/75false
```

When the crawl log was examined, similar strings that correspond to these pages were found. For example:

```
http://jiscenable.blogspot.co.uk/search?updated-max=2009-05-
18T13:21:00+01:00&max-results=7&reverse-paginate=true&start=14&by-
date=false text/html #001 20120328104505026+148
```

Broadly, what was happening here was that Heritrix was creating duplicate copies of the same content, by accessing the content through search requests and through many different URLs. In other words, it was making too many requests of the blog, in the same way that it makes too many requests of a wiki.

Every time Heritrix performs this action, it adds a page of HTML to the archive (see Figure 10). Presumably these pages are simply the result of an "on-the-fly" query of the blog. But as far as the archive copy is concerned, it is simply duplicated content. There is also, in this example, a large amount of it, causing concern for resources such as bandwidth and storage.

Figure 10: 'Duplicated' blog content as harvested by Heritrix

To test this theory, it was decided to perform a filtered gather of the blog and find whether de-duplication and elimination of unwanted content was possible. The first step was to identify what needed to be filtered out. From examining the folder tree and the crawl log, there were three main query-type strings that stood out as being prime targets for filtering.

1. Strings beginning http://jiscenable.blogspot.co.uk/search?updated-max, which as noted above represent queries of the blog database and which produced a page of HTML for each query.
2. Strings beginning http://jiscenable.blogspot.co.uk/search/label. These represent queries of the tags in the blog database. Such strings produced additional pages of HTML, not only for the tag itself but also for queries based on maximum numbers and updates. In the unfiltered gather, these resulted in a folder 93.85 MB in size.
3. Strings beginning http://jiscenable.blogspot.co.uk/flvurl=http. These are evidence of Heritrix attempting to locate streaming video content, usually from googlevideo. 4201 such requests were found in the crawl log, all of them resulting in 404 (Not Found) returns, meaning they would be visible in the archived gather in any case. Unwanted HTML pages were created. In the unfiltered gather, these resulted in a folder 253.38 MB in size.

The following codes for exclusion filters were used.

5.1.4.1 Example One

```
.*search\?updated-max.*
.*search\?updated-min.*
```

These filters were to address the problem of duplicated page content. The crawl log has many examples of patterns like this:

```
http://jiscenable.blogspot.co.uk/search?updated-max=2009-03-
27T13:36:00Z&max-results=7&reverse-paginate=true
http://jiscenable.blogspot.com/search?updated-max/0752009-03-27T06:36:00-
07:00/46max-results/0757/46reverse-paginate/75true
http://jiscenable.blogspot.com/search?updated-max=2009-03-27T06:36:00-
07:00&max-results=7&reverse-paginate=true
http://jiscenable.blogspot.co.uk/search?updated-max/x3d2009-03-
27T06:36:00-07:00/x26max-results/x3d7/x26reverse-paginate/x3dtrue
http://jiscenable.blogspot.co.uk/search?updated-max/0752009-03-
27T06:36:00-07:00/46max-results/0757/46reverse-paginate/75true
http://jiscenable.blogspot.co.uk/search?updated-max=2009-02-
06T10:19:00Z&max-results=7
http://jiscenable.blogspot.com/search?updated-max/0752009-02-06T02:19:00-
08:00/46max-results/0757
http://jiscenable.blogspot.com/search?updated-max=2009-02-06T02:19:00-
08:00&max-results=7
http://jiscenable.blogspot.co.uk/search?updated-max/x3d2009-02-
06T02:19:00-08:00/x26max-results/x3d7
http://jiscenable.blogspot.co.uk/search?updated-max/0752009-02-
06T02:19:00-08:00/46max-results/0757
http://jiscenable.blogspot.co.uk/search?updated-max=2009-02-06T02:19:00-
08:00&max-results=7
```

The filters use regular expressions, and the application of .* at the start and end of the phrase means the filter applied to all instances of the code, regardless of where it appeared in a string. The use of

the \ character was needed to "escape" the question mark. See 5.1.4.2 below for notes on regular expressions.

5.1.4.2 Example Two

```
.*\?max-results.*
.*\?updated-max.*
```

These filters addressed the problem of duplicated tag content. The problem here was that Heritrix was creating numerous pages for each named tag, as evidenced in crawl log patterns like these:

```
http://jiscenable.blogspot.co.uk/search/label/structure?updated-max=2008-10-28T10:26:00Z&max-results=20&start=8&by-date=false
http://jiscenable.blogspot.com/search/label/structure?updated-max=2008-10-28T03:26:00-07:00&max-results=20&start=8&by-date=false
http://jiscenable.blogspot.com/search/label/structure?updated-max/0752008-10-28T03:26:00-07:00/46max-results/07520/46start/758/46by-date/75false
http://jiscenable.blogspot.co.uk/search/label/structure?updated-max/0752008-10-28T03:26:00-07:00/46max-results/07520/46start/758/46by-date/75false
http://jiscenable.blogspot.co.uk/search/label/structure?updated-max/x3d2008-10-28T03:26:00-07:00/x26max-results/x3d20/x26start/x3d8/x26by-date/x3dfalse
http://jiscenable.blogspot.co.uk/search/label/structure?max-results=20
http://jiscenable.blogspot.com/search/label/structure?max-results=20
http://jiscenable.blogspot.com/search/label/structure?max-results/07520
http://jiscenable.blogspot.co.uk/search/label/structure?max-results/x3d20
http://jiscenable.blogspot.co.uk/search/label/structure?max-results/07520
```

The exclusion filter simply focuses on those parts of the string that are causing these returns. Filtering out /search/label/ would result in having broken tags.

5.1.4.3 Example Three

```
.*flvurl=http.*
```

This was to exclude the streaming video requests.

5.1.5 Results

A gather filtered as above created a harvest only 32.1 MB in size, compared with the 1.1 GB unfiltered gather. The /search/label/ folder was 6.78 MB in size, and on the fly harvesting problem had been eliminated altogether. This filtered version was harvested in only five minutes, rather than two hours.

When browsed and tested, the archived copy behaved as expected, with no loss of content, nor any navigational issues.

5.1.6 Conclusions

The above case study highlights two matters of interest:

1. The importance of de-duplication (via filtering the crawl)
2. Optimisation of resources, through reducing operating costs, bandwidth and storage costs

The descriptions of filtering in this section are mainly applicable to the behaviour of the Heritrix crawler. Heritrix may not behave in exactly the same way as the CyberWatcher spider. However, the patterns found in these experiments indicate what can happen when a crawler is making too many searches and requests within a resource, and also show that de-duplication is possible.

As far as the wikis are concerned, the downside of de-duplication is that it restricts or even eliminates some of the functionality in the archived version. However, this is because wikis can be as complicated as content management systems; this problem was not found with the case study blog experiment.

The case study demonstrates that a web archivist needs to recognise and identify the core content that needs to be crawled for success, and also needs to "train" the crawler, with exclusions, to avoid making multiple requests of the content within the target website. This strategy would optimise the crawler and avoid duplication.

The implication of the case study for data extraction is that the BlogForever project needs to identify these types of links that may mislead the crawler. Once it becomes possible to recognise and analyse certain strings that indicate they are duplicate or unwanted requests, it may be possible to train the crawler to avoid following them. This makes the issue directly relevant to the problem of data extraction.

6 Blog Data Extraction

A survey on solutions related to web information extraction has been presented (section 3). This survey includes methodologies, tools and technologies that consider web pages and attempt to analyse the content of a webpage accordingly. The study has shown that web pages follow an arbitrary structure and heuristics are applied in order to parse the web page. Hence, adopting traditional (web information extraction) techniques for addressing the task of weblog data extraction is not sufficient for providing a complete solution. This section takes into account the existence of web feeds as a core feature of blogs. The main motivation is that through the exploitation of web feeds, extraction of data from blogs may be improved. Initially, a case study of web feeds as the sole archiving method is presented. Afterwards, a novel approach that follows a mixed strategy of exploiting both feeds and web content is presented and proposed. Finally, a discussion concerning some semantically annotated information residing in blogs takes place. This information is found through microformats and microdata and is an extra source of blog data information.

6.1 Use of Web Feeds for Archiving: a Case Study

In 2009, the University of London and the British Library undertook a JISC-funded project, ArchivePress[41], which examined the feasibility of using web feeds (RSS and Atom) as the basis for archiving blog content, and created a demonstration system based on the open source WordPress blog software platform [68].

6.1.1 The ArchivePress Project

The project was based on the premise that higher education institutions were seeing a growing number of their information resources expressed in blogs, often distributed over diverse platforms and hosts, and that institutions have a need to preserve integral and authentic copies of such valuable primary information resources, as a record of business, publication, intellectual activity or institutional history. The project cited prominent commentators on the evolution of scholarly communications in the internet era, such as Michael Nielsen, Peter Murray Rust and Heather Morrison, who all drew attention to the growing importance of blogging in academia. It also referenced the work of Carolyn Hank on bloggers' attitudes to preservation was also cited by the project.

Although blogging and RSS emerged separately, their rapid convergence had the effect of making blogs and web feeds largely synonymous. By the time Google acquired Blogger in 2003 [69], it could be argued that the availability of an RSS or Atom newsfeed was a defining feature of a blog: all fully functional blogs should expose a newsfeed of posts, and many also expose a newsfeed of comments - either a single feed of all comments on a blog, or one feed per post.

The optimal use case for the ArchivePress approach, therefore, was for the harvester to be seeded with the URL of a blog's feed, from which the harvester would collect new content as it was published, and also the URLs of comment feeds, which be automatically added to the scheduler stack for future polling.

The scope of the project explicitly excluded the harvesting of the full browser rendering of blog contents (headers, sidebars, advertising and widgets), focusing solely on collecting the marked-up text of blog posts and blog comments (including embedded media). The approach was suggested by the observation that blog content is frequently consumed through automated syndication and aggregation in news reader applications, rather than by navigation of blog websites themselves. Chris

[41] http://archivepress.ulcc.ac.uk/

Rusbridge, then Director of the Digital Curation Centre[42] at Edinburgh University, observed, with reason, that "blogs represent an area where the content is primary and design secondary" [70].

The ArchivePress project was based on using the WordPress blogging application itself as the repository for managing blog content (posts and comments) harvested by RSS. WordPress already provided a robust data model and database implementation for storing and managing blog content. The work of the project was in automating the seeding and harvesting of feeds, and importing blog posts and comments, and associated metadata, from newsfeed content. The project was agnostic about the type of feeds (supporting RSS 0.x, RSS 1.x, RSS 2.x and Atom), since all these feed formats are commonly encountered in the blogosphere (Google's Blogger application, in particular, favours Atom feeds). The plugins developed for the project were partly based on an existing Open Source, third-party WordPress plugin, FeedWordPress[43], which, in turn, used the MagpieRSS[44] PHP feed-processing library. MagpieRSS supports processing of Atom and the most commonly-used versions of RSS.

6.1.2 Benefits of Harvesting Web Feeds

The project perceived the main benefits of harvesting web feeds to be:

- That they are a native and integral format for blog data content
- That they are ubiquitous in blogging applications, and platform-independent (any blog platform - WordPress, Blogger, Typepad, among others - will expose its data as RSS or Atom feeds)
- That they contained rich and easily accessible metadata about blogs, posts and comments
- That the harvested blog content would be much "cleaner" than that harvested by standard web crawlers such as Heritrix, omitting website content other than the content of posts and comments
- That they provide a robust, regular XML-based data format, more structured, adaptable and reusable than the data returned by standard web crawlers
- That use of web feeds uses long-established native Web functionality to improve efficiency in harvesting, saving bandwidth and avoiding retrieval of duplicate or redundant information, for example, by making use of the HTTP response 304, "Not modified", to indicate that cached copies of Web resources are unchanged since last retrieved.

With reference to the OAIS framework for archives, these features offered potential for improved management of archived blog content.

6.1.3 Metadata in Web Feeds

One output of the project was an analysis of the metadata explicitly embedded by default in the web feeds of three major blog platforms (WordPress, Blogger and Typepad). This indicates the extent of rich metadata and emergence of linked data applications; however it also highlights inconsistencies across the platforms.

Table 5: Metadata commonly expressed in web feeds.

	WordPress	**WordPress**	**Blogger**	**Blogger**	**Typepad**	**Typepad**
Metadata description	*Element*	*Attribute*	*Element*	*Attribute*	*Element*	*Attribute*
Type of feed + version	rss	version	feed		feed	xmlns

[42] http://www.dcc.ac.uk/
[43] http://wordpress.org/extend/plugins/feedwordpress/
[44] http://magpierss.sourceforge.net/

Namespace for content	rss	xmlns:content				
Namespace for wellformedweb	rss	xmlns:wfw				
Namespace for Dublin Core elements	rss	xmlns:dc			feed	xmlns:dc
Namespace for Atom syndication format	rss	xmlns:atom				
Namespace for syndication	rss	xmlns:sy				
Namespace for syndication (2)	rss	xmlns:slash				
Namespace for syndication (3)	rss	xmlns:media				
Namespace for thread					feed	xmlns:thr
Unique identifier for the blog	channel		id		id	
Title of the blog	title		title	type	title	
URL of the blog	atom:link	href, rel, type	link	rel, type, href	link	rel, type, href
URL of the blog (2)	link					
Description of the blog	description		subtitle	type	subtitle	
Date blog updated	lastBuildDate		updated		updated	
Blogging software name	generator		generator	version, uri	generator	uri
Language of the blog	language					
Period over which channel format is updated	sy:updatePeriod					
Frequency of updates	sy:updateFrequency					
Gravatar wrapper	image					
URL of gravatar	url					
Title of gravatar	title					
Gravatar links here	link					

6.1.4 Limitations of Harvesting Web Feeds

The project successfully developed plugins for the WordPress blogging software that could harvest content from web feeds and re-render them in a repository setting. However, during the course of the project a number of limitations of harvesting web feeds were also noted:

- It excludes any content of the blog site other than posts and comments (this was acknowledged in the project's original scope)
- Feeds do not usually deliver all posts in a blog, just an arbitray number of the most recent posts. This means that "bootstrapping" an existing blog into the ArchivePress system might involve requesting the blog owner to temporarily enable a feed with all items (if possible). Since the ArchivePress harvesting system was designed to be non-invasive of the target blogs, this was clearly an undesirable limitation.
- Feeds did not necessarily deliver the full post, sometimes containing only an extract or the first paragraphs. WordPress (for example) supports a "continuation" mark-up element: text above the mark will appear in feeds and summary Web pages, while text below the mark only appears when the full post page is requested. These behaviours are all under blog author control. Nevertheless, feed-based harvesting is dependent on blog authors using the features in a way that supports publishing of the full post text in the feeds.

- Versioning: blog platforms were not consistent in the way they dealt with feeds of posts that had been edited after initial publication. Post content might, therefore, be edited by the author after the harvested from the first-published version, and this might not be signalled in the published feed.
- Performance: because the project was fairly small, explicit performance testing was not conducted. However, as the number of feeds to poll grew (including a post feed for each selected blog, and a comments feed for each harvested post), the scripts consumed more resources and took longer to run. It was clear that for harvesting more than a few blogs more sophisticated scheduling and processing was necessary than it was possible to achieve with developing a small-scale PHP plugin for WordPress.

6.1.5 Conclusions

A number of workarounds were devised which mitigated some of the limitations of using web feeds. The problems of incomplete post syndication, versioning and the "bootstrapping" problem were addressed by retrieving and comparing post content directly from the blog, utilsing the URL specified in the feed, and the semantic structure of the blog's HTML. It was not considered that this invalidated the feed-harvesting approach, since the feeds still provided effective and efficient signalling to the harvester. Furthermore, the web feeds represent an aspect of blog content as it is experienced by significant proportion of blog users. The conclusion of the project, therefore, was that web feeds should be utilised in blog archiving applications, but only in conjunction with processing of the standard HTML website view.

6.2 Web Feeds and HTML Content Matching

The review of the related methodologies has shown an inadequacy in achieving efficient blog data extraction (Sections 3 & 4). The following subsections propose a novel approach that is exploiting the web feeds found in blogs. Through the cross-matching of information found in semi-structured documents (HTML) and strictly structured documents (i.e. web feeds), the approach establishes an automatic, more complete and precise solution. Our approach is divided into three steps that are discussed below (Sections 6.2.1, 6.2.2 and 6.2.3).Following these three steps, this subsection presents and discusses the problem of text matching (Section 6.2.4) that emerged as part of the approach and an example (Section 6.2.5) of the approach is provided.

6.2.1 Step 1: Feed Processing and Capturing of Post Properties

At this stage, the only prerequisite is for the URL of a feed to be known. The first action is to fetch the content of the feed and process it through an XML parsing library.
Generally, a feed contains a section of entries that point to the list of weblog posts. For each entry, the following attributes are typically found and stored.

- *Title*: An optional, object of type string. A post can contain no more than 1 title.
- *Author*: An optional, object of type string.
- *Date Published*: A required object of type date. A post contains exactly 1 publication date.
- *Summary*: An optional HTML encoded part of the weblog post's content (usually the beginning of the post's content).
- *Permalink*: A required, URL that points to the actual weblog post.
- *Categories*: An optional collection of words/phrases describing the topic of the post.

6.2.2 Step 2: Generation of Filters

The second step includes the generation of filters. The term filter is coined and used the same way it is used by other works related to web information extraction. Baumgartner et al.[67] used the concept of filter as the building block that constructs a pattern, which in turn describes a generalised tree path in the HTML parse tree. Adding a filter to a pattern extends the set of extracted targets, whereas imposing a condition to a filter restricts the set of targets. The same concept is used by XWRAP [71] in order to describe the so-called extraction rules. XWrap's extraction rules are described in XPath-like expressions and point to regions of the HTML document that are of interest. The following paragraphs describe and discuss the HTML features to be exploited through the filters.

XPath

XPath[45] (XML Path Language) is a W3C-definedlanguage initially introduced in 1999 that is used to address parts of an XML document. Version 2.0 of XPath is the current version since 2007 and aims at increasing its processing capabilities while remaining backwards compatible. An element can be addressed in several ways in XPath. There are roughly two basic approaches when addressing an element: the absolute and relative path. Some examples based on the XML document of Table 6 will be provided. However, it is worth noting that XPath is a powerful language that exceeds the rather simple examples provided and allows the creation of dense and meaningful expressions[46].

Table 6: an example of an XML document

```
<computer>
  <desktop>
        <company>LG</company>
        <item>21</item>
  </desktop>
  <laptop>
        <company>DELL</company>
        <item>17</item>
  </laptop>
</computer>
```

Absolute Path Identify

An example of absolute path expressions is provided at Table 7:

Table 7: Absolute path examples

XPath expressions	Returned values
`/computer/desktop/item`	21
`/computer/laptop/item`	17

Notice that in the beginning of the expressions a slash (/) is used in order to describe a path that starts from the root element of the document.

Relative Path Identify

XPath allows addressing elements in a non-absolute way, as described in the example of Table 8.

Table 8: Relative path examples

XPath expressions	Description	Returned values
`item`	All <item> elements will be returned.	21 17

[45] http://www.w3.org/TR/xpath/

[46] A long list of XPath expression examples and their explanation can be found at http://msdn.microsoft.com/en-us/library/ms256086.aspx

`//laptop[company="DELL"]`	All <laptop> elements that contain a <company> child element with value "DELL".	`<laptop>`
		`<company>DELL</company>`
		`<item>17</item>`
		`</laptop>`

CSS classes

CSS (Cascading Style Sheets) "is a simple mechanism for adding style (e.g., fonts, colours, spacing) to Web documents"[47], first introduced in 1996. It allows the separation of document content from document presentation through the definition of a set of rules. The rules contain information about the *selectors* and a *declaration block*, as in the following example:

Table 9: CSS example

CSS code	Description
`h1 { color: black; background-color: red !important; }`	<h1> elements (*selector*) contain text in black colour and red background overriding other declarations (*declaration block*)

Moreover, CSS allows extending simple elements through *classes*, depending on the presentation layer, as shown in the example in Table 10.

Table 10: CSS class example[48]

CSS Code	HTML code	Display
`p.first{ color: blue; }` `p.second{ color: red; }`	<html> <body> This is a normal paragraph. This is a paragraph that uses the p.first CSS code! This is a paragraph that uses the p.second CSS code! …	This is a normal paragraph. This is a paragraph that uses the p.first CSS code! This is a paragraph that uses the p.second CSS code!

HTML ID

The id attribute specifies a unique id for an HTML element of a document. It is commonly used in cases where CSS code needs to address one specific, unique element (e.g. the title of a post) or run JavaScript. An example of id usage is provided at Table 11.

Table 11: HTML id attribute example

HTML code
`<div id="header">Hello World!</div>`

Following related work, we introduce the concept of filter in order to identify and describe specific data elements of an HTML document. In our approach, a filter is described through three basic attributes: the *absolute path*, the *CSS classes* applied and the *identifiers* of the HTML element.

[47] http://www.w3.org/Style/CSS/
[48] The example is taken from http://www.tizag.com/cssT/class.php

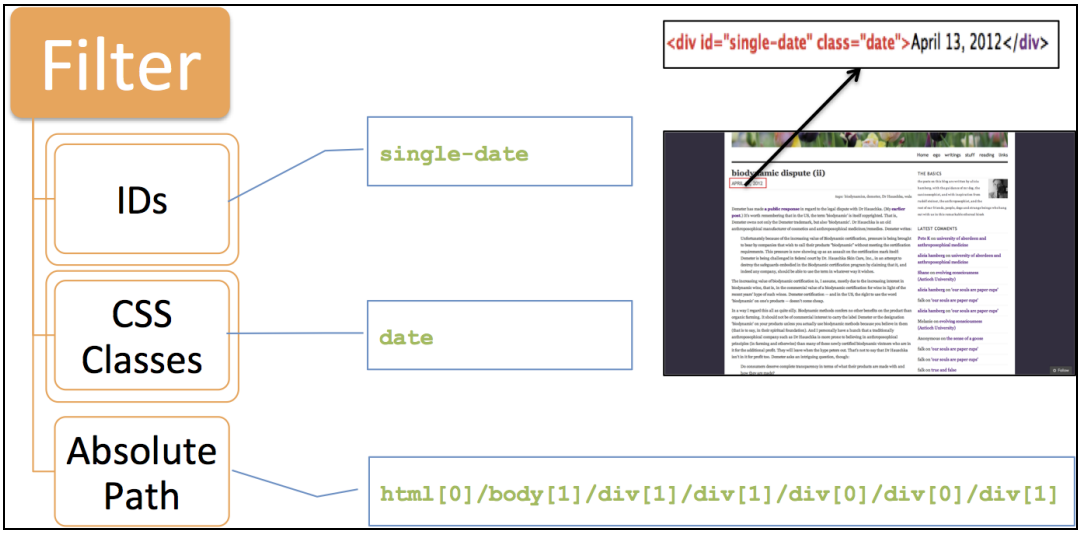

Figure 11: The structure of a filter. An example is annotated for the case of an element containing a date value.

Figure 11 shows the structure of a filter. When pointing at a specific element, a set of ID values, CSS Classes and an absolute path are used to describe the element. More specifically, when an element is identified, the text of the element is extracted and any ID or CSS class applied is added to the filter. Afterwards, an iterative selection of the parent element continues as long as the value of the parent element remains the same. For every selection of a parent element, and while the above condition is met, the IDs and CSS classes found are added to the filter, resulting to the collection of a set of values accordingly. The absolute path is the path from the root of an HTML document to the desired element. This element may either be a node of the DOM tree or simply an end-node (leaf). Thus, the absolute path is described as a sequence of edges, where every edge is defined as the *name* of the element and the positional information of the element (*index*)[49]. This sequence of edges starts from the root of the document and ends with the element containing the property we are interested on. For the example of Figure 11, the values for the IDs attribute is a single value *single-date*, for the CSS Classes the value *date* and the Absolute Path is *html[0]/body[1]/div[1]/div[1]/div[0]/div[0]/div[1]*.

6.2.3 Step 3: Induction of Rules and Blog Data Extraction

After the completion of step 2, a collection of filters is populated for every property. The values of these filters are such that when applied to the blog posts from which they were extracted, they link back to the HTML element containing the value of the property matched. However, due to information noise, the filters contain information that needs further processing. This "filter pollution" is due to one or more of the following reasons.

- The visual theme of each blog is different. This directly affects the values of the filter attributes.
- Even for a single blog, the blogging platform may choose to render differently the property to be found across different posts. For example, as the size of the content of the post changes, the absolute path is affected. This value of the path, if reused elsewhere, may be pointing to different or null elements.

[49] The positional information of an HTML element is crucial in HTML documents, since this affects its visual representation. This is one of the reasons that HTML DOM trees are viewed as labelled ordered trees in literature (e.g. [72] P. Geibel, O. Pustylnikov, A. Mehler, H. Gust, and K. U. Kühnberger, "Classification of documents based on the structure of their dom trees", 2008, pp. 779-788.).

- The value to be matched may be repeated more than once in a single web page. For instance, the title of a blog post may be included at the sidebar under the section "recent posts" or the name of the author may be referred several times.
- There are cases where a *part* of a filter should be applied in order to extract the desired property. For example, based on a set of instances, there might be one case where the value of an absolute path is changing for every instance. At the same time, an identical CSS class value across posts might identify the desired property. In that case, only the CSS class value should be used to extract the desired property.

The goal of this stage is to address the above issues. More specifically, the aim is to exploit the knowledge residing in the filters, in order to generate *a set of rules* that describe how to extract the properties for a blog automatically. Research fields like machine learning and artificial intelligence have shared similar objectives. For example in AI, expert systems are typically divided into two parts, the knowledge base and the reasoning engine [73]. Expert systems achieve induction through a set of rules that run in the engine and refer to a knowledge base. However, these rules are created by the designer of the system prior to the execution of the expert system. Machine learning approaches (discussed more extensively in Section 3) attempt to generate rules through a *learning by example* methodology, i.e. a general rule is extracted through the observation of a set of instances. The output of this approach may be a classification, usually calculated in models like neural networks, support vector machines, decision trees, or more simply a set of rules etc.

Consider a decision tree, as the one described on Figure 12. The nodes of this tree contain the names of the attributes. The edges of the tree contain the value of the parent attribute-node. Finally, the leaves contain the expected outcome. For the example provided, the expected outcome is a decision on whether to play tennis or not (example taken from [74]).

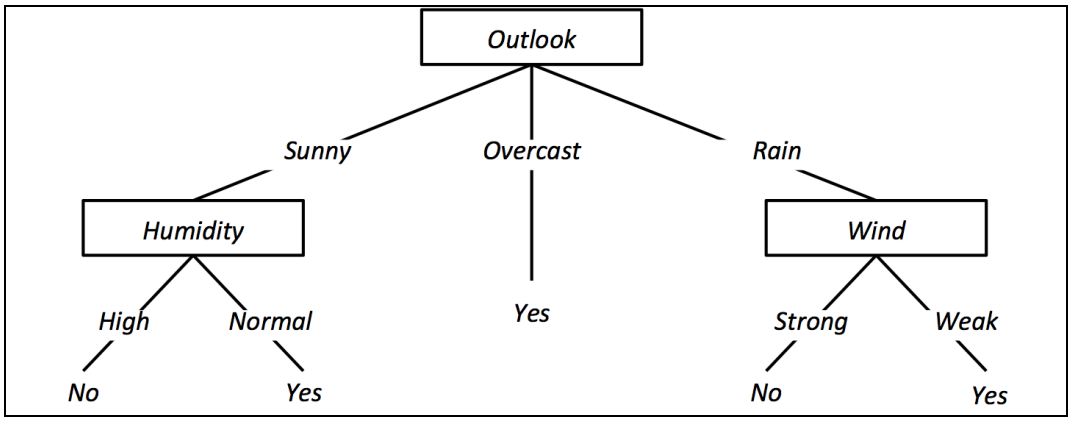

Figure 12: A simple decision tree (taken from [74]).

For the provided example, the decision tree may be described as a set of rules, as follows:

1. $If\ (Outlook = "Sunny") \wedge (Humidity = "High")$
   $THEN\ PlayBall = No$
2. $If\ (Outlook = "Sunny") \wedge (Humidity = "Normal")$
   $THEN\ PlayBall = Yes$
3. $If\ (Outlook = "Overcast")$
   $THEN\ PlayBall = Yes$
4. $If\ (Outlook = "Rain") \wedge (Wind = "Strong")$
   $THEN\ PlayBall = No$
5. $If\ (Outlook = "Rain") \wedge (Wind = "Weak")$
   $THEN\ PlayBall = Yes$

Decision trees are the outcome of the processing of already *acquired knowledge* through induction mechanisms. These trees are constructed beginning with the root of the tree and proceeding down to its leaves [74]. Quinlan's research has been a pioneer in the subject of decision tree generation. Initially, through the ID3 algorithm [75] and later on with follow up systems like C4.5 [76], the approach applied provides a simple and elegant solution on classification problems. Consider the decision tree of Figure 12: the decision tree is the product of the following cases:

Table 12: A data collection example (taken from [74]).

Day	Outlook	Temperature	Humidity	Wind	Play Ball
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

In order to generate the decision tree, the ID3 algorithm is using the concepts of *entropy* and *information gain*. Entropy is defined as a measure of "uncertainty" or "randomness" of a random phenomenon [77]. $Entropy(S)$ of the set S for a given attribute is defined as:

$$E(S) = -\sum_{j=1}^{n} f_s(j) \log_2 f_s(j)$$

where:

- $E(S)$ is the entropy of set S
- n is the number of different values of the attribute in S
- $f_s(j)$ is the frequency of the value j in the set S
- $\log_2$ is the binary logarithm

For the example provided, there are 9 cases where PlayBall is Yes and 5 cases where PlayBall is No. Therefore the entropy is:

$$E(S) = -\frac{9}{14} \log_2 \left(\frac{9}{14}\right) - \frac{5}{14} \log_2 \left(\frac{5}{14}\right) = 0.940 \quad (1)$$

Let's consider the attribute Wind. There are 8 occurrences where the Wind is weak and 6 occurrences where the wind is strong. Gain of an attribute, which is calculated based on the entropy, is a measure of how well the attribute separates the training examples into different targeted classes. More specifically, gain for an attribute A of a set S, $G(S,A)$ is defined as:

$$G(S,A) = E(S) - \sum_{i=1}^{m} f_s(A_i) E(S_{A_i}) \quad (2)$$

where:

- $E(S)$ is the entropy of a set S
- m is the number of different values of the attribute A in S
- $f_s(A_i)$ is the proportion of items possessing A_i as value for A in S
- A_i is the i[th] possible value of A
- S_{A_i} is a subset of S containing all items where the value of A is A_i

Therefore, gain for attribute Wind would be:

$$G(S, Wind) = E(S) - \frac{8}{14} * E(S_{weak}) - \frac{6}{14} * E(S_{strong})$$
$$= 0.940 - \frac{8}{14} \times 0.811 - \frac{6}{14} \times 1 = 0.048$$

Similarly, we find that:

$G(S, Outlook) = 0.246$
$G(S, Temperature) = 0.029$
$G(S, Humidity) = 0.151$

The Outlook attribute has the highest gain and is therefore selected as the root of the decision tree. From this node, an edge is created to its children for every distinct value. The next attribute, which is placed at the second level, is selected by calculating its gain for the case of every value of the Wind attribute. More specifically, for the case of a sunny Outlook, the remaining attributes have the following gain:

$G(S_{sunny}, Humidity) = 0.970$
$G(S_{sunny}, Temperature) = 0.570$
$G(S_{sunny}, Wind) = 0.019$

The Humidity attribute has the highest gain and is therefore placed as the second attribute to be examined for the case of a sunny Outlook. This process is repeated for every node, until all data are classified or all attributes have been used. The result for the example discussed is the decision tree already illustrated in Figure 12.

The example described above presents several properties similar to the identification of extraction rules. It highlights the attributes to be selected based on the measurements of Entropy and Gain. However, in the case of blog data extraction, a decision tree would not be appropriate, since – as discussed in the beginning – there are cases where:

- Having to pick a value for *every* attribute of a rule could potentially eliminate the desired element, and
- Some values are inherently *unique* for every post.

For the above reasons, our approach adopts the concepts of entropy-based measurements [78] with the aim to generate a set of disjoint rules, rather than create a decision tree.

As discussed above for the case of ID3 algorithm, Entropy is a measurement that expresses the uncertainty of a certain phenomenon (e.g. whether to play ball). For the case of filters and blog data extraction, the result is always – by definition – the successful extraction of the property for the given instance. Therefore, Entropy equals 0. Moreover, Gain is now defined in relation to every value i of its attribute A over its set:

$$G(S, A_i) = -f_s(A_i) E(S_{A_i}) \quad (3)$$

Entropy $E(S_{A_i})$ is the entropy of an attribute A over its value i. For example, consider the example in Table 13:

Table 13: A simple example of different filters.

ID	IDs	CSS Classes	Absolute path
1	¢	cssValue1	path1
2	id1	cssValue1	path1
3	id1	cssValue1	path2
4	¢	cssValue2	path2
5	¢	cssValue2	path3

For this table, we have:

$$E_{IDs_{null}} = -\frac{1}{5}log_2\left(\frac{1}{5}\right) = -0,46$$
$$G(S, IDs_{null}) = -\frac{1}{5}(-0,46) = 0,09$$
$$E_{IDs_{id1}} = -\frac{2}{5}log_2\left(\frac{2}{5}\right) = -0,53$$
$$G(S, IDs_{id1}) = -\frac{2}{5}(-0,53) = 0,21$$
$$E_{CSS_{cssValue1}} = -\frac{3}{5}log_2\left(\frac{3}{5}\right) = -0,44$$
$$G(S, IDs_{id1}) = -\frac{3}{5}(-0,44) = 0,27$$
$$E_{CSS_{cssValue2}} = -\frac{2}{5}log_2\left(\frac{2}{5}\right) = -0,53$$
$$G(S, CSS_{cssValue2}) = -\frac{2}{5}(-0,53) = 0,21$$
$$E_{Path_{path1}} = -\frac{2}{5}log_2\left(\frac{2}{5}\right) = -0,53$$
$$G(S, Path_{path1}) = -\frac{2}{5}(-0,53) = 0,21$$
$$E_{Path_{path2}} = -\frac{2}{5}log_2\left(\frac{2}{5}\right) = -0,53$$
$$G(S, Path_{path2}) = -\frac{2}{5}(-0,53) = 0,21$$
$$E_{Path_{path3}} = -\frac{1}{5}log_2\left(\frac{1}{5}\right) = -0,46$$
$$G(S, Path_{path3}) = -\frac{1}{5}(-2,72) = 0,09$$

Based on the above calculations, the rules are extracted and ordered by their gain. Finally, if the applied rule fails (i.e. when zero or more than one elements are identified while a single value property is expected, e.g. for the case of a title), the next rule is selected. For the example above, the following rules are extracted in the following order[50]:

1. CSS class for value *cssValue1*
2. CSS class for value *cssValue2*
3. ID for value *id1*
4. Path for *path1*
5. Path for *path2*
6. Path for *path3*

Figure 13 presents an overview of the methodology described. As already discussed in detail, the proposed solution involves the execution of three steps. The first step includes the rather trivial task of reading and storing the blog data properties found in the Web feed. The second step includes the training of the wrapper through the cross matching of information found on the web feed and the HTML documents. This step leads to the generation of information, captured through the filters, which describes where the blog data reside. The final step deals with the processing of the filters, in order to generate the rules that point to the desired properties in the general case of a blog.

[50] Based on some empirical evaluation, for the cases where the information gain for an attribute of CSS class or IDs equals to the gain of an absolute path attribute, the latter is demoted.

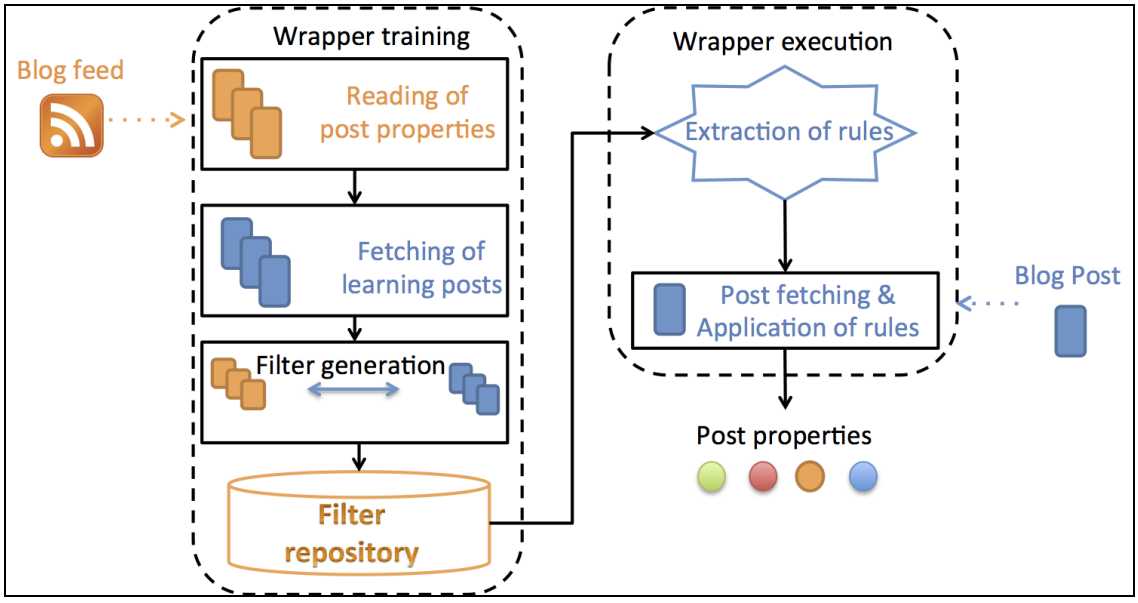

Figure 13: Overview of the blog data extraction methodology

6.2.4 Text Matching: Complete or Partial – Approximate or Absolute

As already described, the proposed methodology relies on the fact that the identification of an HTML element against a specific value is achieved. Generally, text matching is not a trivial task and can be classified into the following string matching cases.

- *Complete* matching: Every character of a string has to match every character of another string and vice versa.
- *Partial* matching: Every character of one string matches a substring of another string. Insertion is needed to have a complete matching.
- *Absolute* matching: A sequence of characters matches completely another one.
- *Approximate* matching: A sequence of characters is aligned to another sequence optimally; i.e. the least possible changes take place in order to have an absolute matching. Substitution is needed to achieve absolute matching.

Table 14: An example of different cases of text matching[51]

	Absolute	*Approximate*
Complete	kitten kitten	**k**itten **s**itten
Partial	**kitt**en **kitt**	**kitt**en s**itt**i**n**g

Whilst absolute matching is a fast and trivial task, this type of matching is of limited interest and applicability. On the contrary, approximate matching has been a field of study in several problem domains. The main reason for the popularity of approximate string matching is that inherent data distortion appears in a plethora of problems. For the case under examination (identifying specific HTML elements) the case of approximate, partial text matching[52] is needed, since two different, semi-structured data sources (the HTML document and the web feed) are cross-matched.

[51] The example provided is taken and extended from the one provided by Wikipedia (http://en.wikipedia.org/wiki/Levenshtein_distance).
[52] The term is also used as "optimal string alignment".

Matching a text or a sequence against another is a problem appearing in several domains. In record linkage, object identification, plagiarism detection, protein sequence alignment, spam filtering. Navarro [79] provides an extensive survey on algorithms that deal with the problem of string matching. Furthermore, the same work discusses the complexity aspects of every approach and evaluates the algorithms implemented. Winkler [80] gives specific examples of string comparison based on various algorithms and assesses the limitations of every approach. In any case, every algorithm introduces the concept of distance in order to quantify the similarity between two strings.

When trying to perform matching, the concept of *distance* between two strings emerges, in order to express the (complement of the) level of similarity between the two elements. The most popular measurement is the edit distance, also known as the Levenshtein distance (from now on denoted as $d_E(x,y)$ for the case of matching strings x, y) by the Russian scientist Levenshtein who was the first who described this measurement. The Levenshtein distance is defined as the minimum number of edits required to transform one string to another. The types of edits considered are the insertion, deletion or substitution of a single character. As an example, the values of the Levenshtein distance for the pairs of strings of Table 14 are the following:

Table 15: Levenshtein distance values for the example of Table 14

Pairs of strings	Levenshtein distance
(kitten, kitten)	0
(kitten, sitten)	1
(kitten, kitt)	2
(kitten, sitting)	3

The Levenshtein distance is quite popular because it presents several advantages. First of all, it is a simple concept that can be easily understood and its return value can be effortlessly assessed. Secondly, the algorithm that calculates this distance is very efficient computationally (time and space complexity are low) and finally, this distance establishes a *metric space*[53]. However, the Levenshtein distance value expresses the number of character edits and therefore is not unitless. Thus, having an absolute value as a result of the matching between two strings does not give much information about the "degree of matching" between them. For example, comparing one string of 200 characters against another of 1000 characters would lead at the best case scenario (complete, partial matching) to a Levenshtein distance value of 800. At the same time, comparing two strings of 200 bytes would yield a result no more than 200 (worst case scenario, no match for any character). The above remarks have lead to the further development of text distance measurements. Typically, these metrics adopt and extend the Levenshtein distance while at the same time are attempting to diminish the difference of the length of the strings. The most common variations are the ones presented in Table 16.

Table 16: Variations of the Levenshtein distance[54]

Name	**Expression**
Normalised by the sum	$d_{sum}(x,y) = \frac{d_E(x,y)}{\lvert x\rvert - \lvert y\rvert}$
Normalised by the maximum of the length of the strings	$d_{max}(x,y) = \frac{d_E(x,y)}{\max\{\lvert x\rvert, \lvert y\rvert\}}$
Normalised by the minimum of the length of the strings	$d_{min}(x,y) = \frac{d_E(x,y)}{\min\{\lvert x\rvert, \lvert y\rvert\}}$

[53] Establishing a metric space based on a distance function leads to a very powerful mathematical model. This is because a metric space presents important properties (e.g. the triangular inequality, identity of indiscernibles, symmetry) that can be used to extract meaningful knowledge between objects belonging to a large collection, without the need to calculate the distance for every pair of objects.

[54] $|x|$, $|y|$ are the lengths of the strings x, y, respectively.

Normalisation by Yujian and Bo [81]	$d_{YB}(x,y) = \frac{2d_L(x,y)}{\lvert x\rvert - \lvert y\rvert + d_L(x,y)}$

It is worth noting that each variation presents some advantages over the original Levenshtein distance. For example, both d_{nsm} and d_{max} have normalised values to the scale of 0 and 1 and are unitless (i.e. do not express number of characters but percentage). The same applies for distance d_{YB}, which has the extra feature to establish a metric space. On the other hand, d_{min} is not normalised and therefore maintains the basic limitation of the original Levenshtein distance. However, according to our evaluation shown below, all 4 distance metrics fail to quantify partial matching of blog properties adequately.

Taking into account the above, another distance metric that is not based on the Levenshtein metric was considered; the Jaro-Winkler, which is introduced by Winkler [82] in 1990 and extends the metric by Jaro [83]. The initial, Jaro distance, denoted as d_j, is as follows:

$$d_j(x,y) = \frac{1}{3}\left(\frac{m}{|x|} + \frac{m}{|y|} + \frac{m-t}{m}\right) \quad (4)$$

where:

- m is the number of matching characters
- t is half the number of transpositions

The Jaro-Winkler distance (denoted as d_{jw}), which does not establish a metric space, extends the above function in order to act in favour of cases where the beginning of one string matches to the beginning of another. The distance is as follows:

$$d_{jw}(x,y) = d_j(x,y) + \left(\ell p\left(1 - d_j(x,y)\right)\right) \quad (5)$$

where:

- d_j is the Jaro distance
- ℓ is the length of common prefix at the start of the string up to a maximum of 4 characters
- p is a constant scaling factor for how much the score is adjusted upwards for having common prefixes

Based on the above survey, an experiment was conducted in order to evaluate which distance metric would be more suitable for our case. The input for the experiment was a list of 8 different, real-world strings found on the web, scaling from small (13 characters) to big (2,848 characters) with partial or no overlap extracted from 2 webpages. Since the Levenshtein and the d_{min} distance returned values higher than 1 (unnormalised distances)[55], these distances were not processed any further. The result for the rest of the distance functions is shown on Figure 14.

[55] More specifically, the lowest and highest value for the Levenshtein distance was 14 and 2839 respectively. The corresponding values for d_{max} were 0,038 and 218,385.

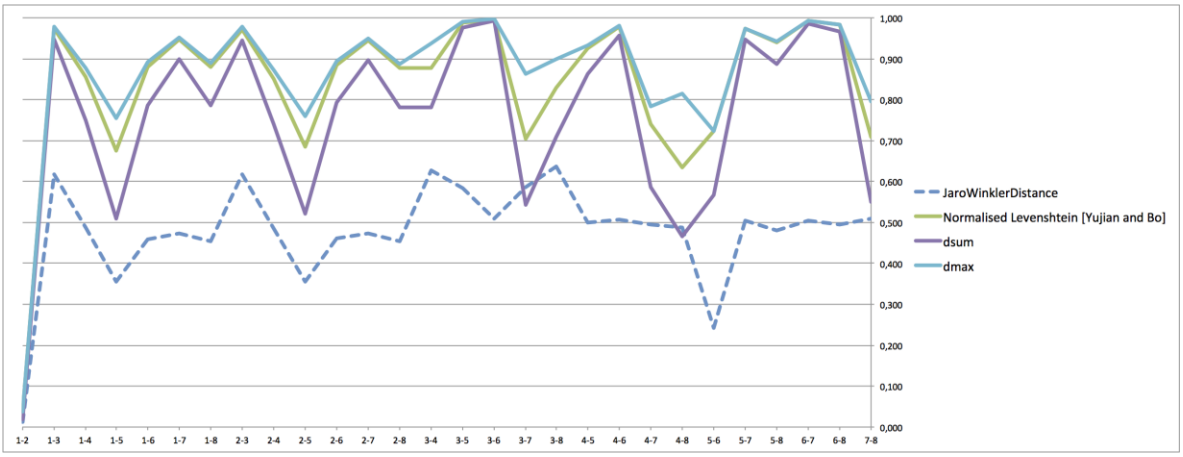

Figure 14: Distance for 8 different strings for 4 different distance functions. Low values denote better matching.

The horizontal axis of Figure 14 corresponds to the pair of strings (e.g. 1-2 is the value for matching string1 against string2) measured, for a total of 28 matches. The vertical axis corresponds to the value of the distance measurement. Since every function is symmetric, the result for the reverse pair of strings was omitted (i.e. $d(x, y) = d(y, x)$). The graph shows that the *JaroWinkler* distance is more salient, or else is the most *tolerant* distance for cases of partial matching between strings of different lengths[56].

6.2.5 Example

In order to demonstrate how the proposed methodology works, we considered a specific example of a blog. The blog shown here is hosted on http://blogs.warwick.ac.uk, which is powered by the blogging platform BlogBuilder, developed by Warwick IT services[57]. The blog of the example is http://blogs.warwick.ac.uk/johnsmith, owned by a user by the name "John Smith"[58]. For this example, the property to be extracted will be the author of the blog. The steps are as follows:

- *Step 1: Feed processing and capturing of post properties*

At this step, the Web feed was found and processed. More specifically, the feed is an Atom feed (http://blogs.warwick.ac.uk/johnsmith?atom=atom) containing entries for the latest 20 posts published by the user. However, in order to keep the example as general possible, the latest 10 posts were considered as being part of the Web Feed and the other 10 were discarded, which is the most typical number of entries found in web feeds [84]. For this feed, the value of the author is always the same (John Smith); however, even if more than one authors were blogging, the step would be the same, since the value to be matched is extracted on a per-post, "case by case" basis.

- *Step 2: Generation of filters*

At this step, the value of the author found in the feed is being matched to the HTML document of the blog post. For every match, a filter is generated, described in the attributes IDs, CSS Classes and Absolute Path accordingly. The filters extracted are shown on Table 17.

[56] The full details of the strings to be matched, as well as the result for all distance functions are provided at Appendix 11.2.
[57] http://www2.warwick.ac.uk/services/its/
[58] The name of the user has been anonymised for privacy reasons.

Table 17: Filters generated for a single Blog, for the case of author

Post Id	IDs	CSS Classes	Absolute Path
18110	¢	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[8]/span[0]
18111	¢	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[13]/span[0]
18112	¢	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[7]/span[0]
18113	¢	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[9]/span[0]
18113	¢		/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[7]
18114	¢	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[2]/span[0]
18115	¢	commentauthor	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[1]/div[0]/ol[2]/li[0]/h4[5]
18115	¢	commentauthor	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[1]/div[0]/ol[2]/li[0]/h4[5]/a[0]
18115	¢	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[2]/span[0]
18116	¢	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[7]/span[0]
18117	¢	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[7]/span[0]
18118	¢	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[6]/span[0]
18119	¢	author	/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[7]/span[0]

The first column of Table 17 refers to the Id of every post. The table shows that for each of the 10 posts, at least one filter was generated, which means that the value of the author found in the Web feed was successfully matched for every case. Moreover, we notice that there are cases (more precisely, post ids 18113 and 18115[59]) where the author appears more than one time (2 and 3 times, respectively). This is happening because the text "John Smith" either appears in a separate paragraph (as a "signature" to the content of the post 18113) or as a comment author (post 18115). However, for the majority of the cases the author name appears as following:

```
<span class="author”>
        <img src="/blogbuilder/media/images/tiny_author.gif" title="2nd Year Undergraduate
        Student: Statistics" alt="">
        John Smith
</span>
```

After a short survey on the attributes of the filters, we notice that the IDs are not used to identify the HTML element at all (values are null), while the CSS class value "author" is used with high frequency. Finally, the Absolute path, while it follows a common path to a certain extent, its value is typically variant.

- *Step 3: Induction of rules and weblog data extraction*

At this step, the gain for every value of every attribute is calculated, as described in section 6.2.3. The result of the calculation shows that the first rule is the one where value "*author*" is matched for the case of *CSS Classes*. Moreover, if we attempt to evaluate this rule, the result will show that the author is extracted successfully for every blog post (a more thorough discussion on evaluation issues follows at Section 6.4). More specifically, we have the following values:

[59] Post 18113 and 18115 refer to http://blogs.warwick.ac.uk/harunahasan/entry/p10_final_entry/ and http://blogs.warwick.ac.uk/harunahasan/entry/p7_reflective_entry_1/, respectively (online, accessed May 10, 2012).

Table 18: Gain for every attribute value of the filters of Table 17. Selecting the rule described by the highest gain (CSS class with value "author") will result in the successful extraction of the desired property.

Attribute value	# occurrences	Gain
author	10	0,22
/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[7]/span[0]	4	0,16
commentauthor	2	0,06
/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[2]/span[0]	2	0,06
/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[8]/span[0]	1	0,02
/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[13]/span[0]	1	0,02
/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[9]/span[0]	1	0,02
/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[7]	1	0,02
/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[1]/div[0]/ol[2]/li[0]/h4[5]/a[0]	1	0,02
/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[1]/div[0]/ol[2]/li[0]/h4[5]	1	0,02
/html[0]/body[1]/div[0]/div[0]/div[0]/div[1]/div[0]/div[0]/div[1]/div[1]/div[0]/p[6]/span[0]	1	0,02

6.3 Blog data Extraction Prototype Development

In order to assess and evaluate the described methodology, UW has developed a software prototype that implements the described methodology. The prototype was programmed in Java™ SE 6 Update 29[60]. Moreover, the following freely available Java libraries were used.

- **Jsoup** 1.6.2[61]: an HTML Parser. A presentation of this library is provided at section 4.2.1.1.
- **LingPipe** 4.1.0[62]: a toolkit for processing text using computational linguistics. This library was used in order to run the Jaro-Winkler distance algorithm, which is necessary during the text matching process.
- **ROME** 1.0[63]: a set of RSS and Atom Utilities. The library was used in order to read and process the web feed passed in the beginning of the data extraction process. A short demo of the library is provided at section 4.2.1.2.
- **JDOM** 1.1.3[64]: a general purpose XML parsing library. Used by ROME.
- **Commons Lang** 3.1[65]: a popular library with helper utilities, including String manipulation methods. This library was used in order to assess the Levenshtein distance.
- **MySQL Connector/J** 5.1.18[66]: the official JDBC driver for MySQL[67]. All information extracted, both from the RSS and the Web (blog post properties, filters, extracted blog post properties), was stored and retrieved through a MySQL database.

Development was done in IntelliJ IDEA 11[68], a Java IDE. A UML Class diagram was generated through reverse engineering from the source code by IntelliJ (see Figure 15[69]). The class

[60] http://www.oracle.com/technetwork/java/javase/downloads/index.html
[61] http://jsoup.org/
[62] http://alias-i.com/lingpipe/
[63] The ROME project website is http://rometools.org/. The latest version is available for download at http://repo1.maven.org/maven2/rome/rome/
[64] http://www.jdom.org/
[65] http://commons.apache.org/lang/
[66] http://dev.mysql.com/downloads/connector/j/
[67] http://www.mysql.com/
[68] http://www.jetbrains.com/idea/
[69] A more extended UML Class diagram is shown at section 11.3 (Appendix C: Software prototype).

DataExtractionUtils is a helper class that provides a set of public, static methods used mostly during the text matching process. The most basic, public class (subject to be consumed by any third-party in a future public release) is the class *Extractor*. The class Extractor contains all the information required to perform data extraction for a single blog, while providing an interface that uses the lower level Classes and functions. More specifically, Extractor stores information about one Blog (the most important attribute of the Class Blog is the Feed URL) and the filters acquired during the matching. The filters acquired are extracted through an iterative matching of the Posts fetched and the entries of the Web feed. The Class Post contains information about the permalink URL of the blog post, the Document as represented in Jsoup and the properties found on the Web feed. Invoking methods of the Class Post (e.g. the method getFiltersByText) returns a collection of Filters. A Filter is a simple data structure, composed by the triple of attributes, namely IDs, CSS Classes (collections of String) and Absolute Path (ArrayList). Finally, the Class MySQLAccess performs simple operations that allow storing and fetching content.

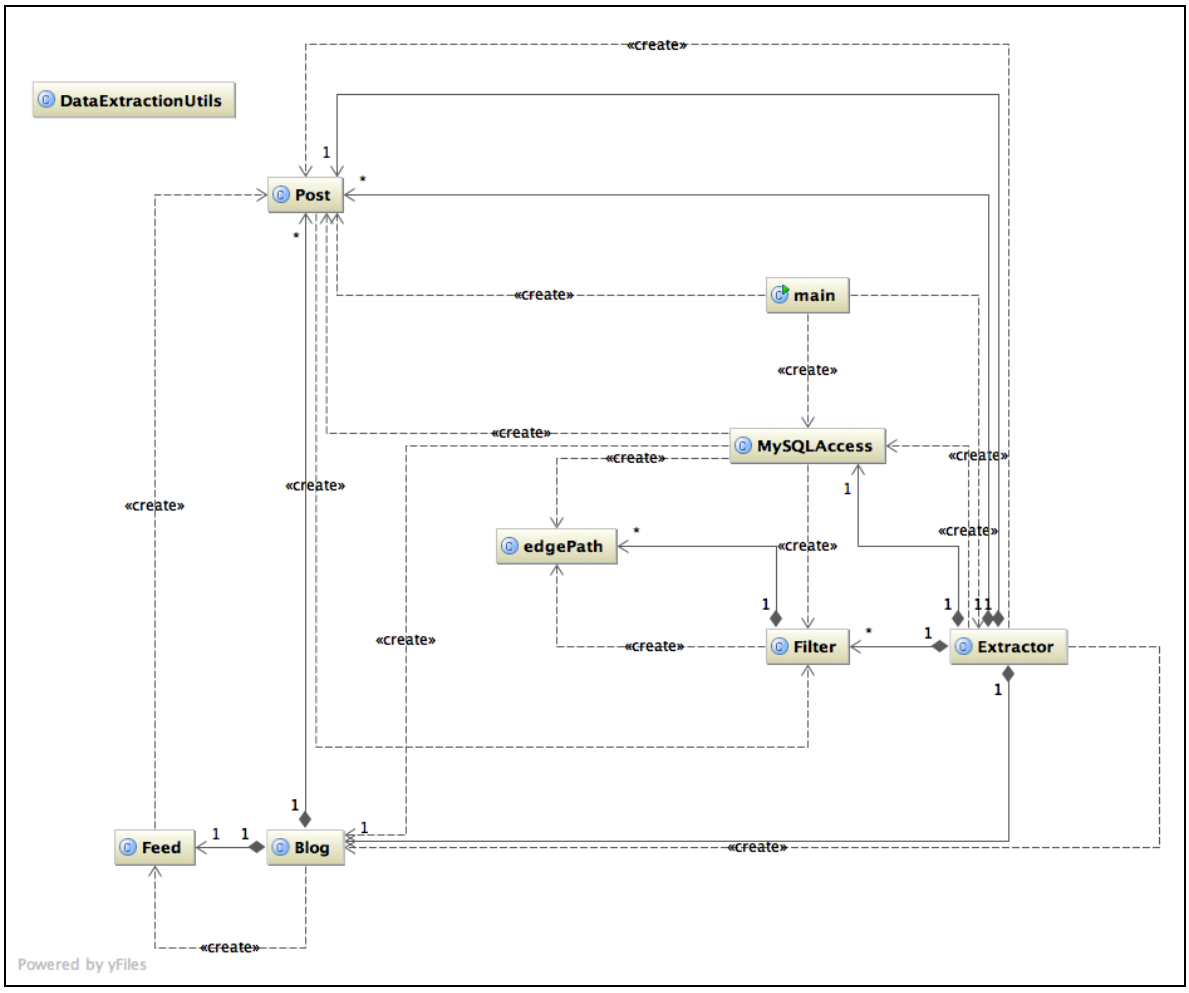

Figure 15: UML Class diagram of the data extraction prototype.

During the development, some issues surfaced that shaped the final proposed extraction methodology, which have not been discussed yet. These issues, as well a suggestion for further improvement, are:

- Matching a date (e.g. publication date) is a complex task, compared to the simpler text matching (e.g. the blog post title). The main reason for this is the fact that every blog may style the date in different ways. For instance, a date might appear as “Friday, 11 May 12”, “Friday, 11 May 2012” or even 11/05/2012. In order to address the above issue, the prototype is currently casting the date value in the most popular styles (as well as some variations of them) available, such as:

  - SHORT (e.g. 05.11.12)
  - MEDIUM (e.g. May 11, 2012)
  - LONG (e.g. May 12, 2012)
  - FULL (e.g. Friday, May 11, 2012 AD)

  Moreover, currently the date is rendered in English language only, which is usually not sufficient for matching the date in blogs written different languages. An improvement would be to identify the language of the document (e.g. with Tika[70]) and style the date following the locale of the language identified.
- Initially, the rules are ordered by their information gain. When a rule fails to return a valid value (e.g. a title should be exactly one HTML element), the next rule is applied until the algorithm runs out of rules. This is not always the best practice, since there are cases where searching for a property should actually return no value (e.g. a title being blank). Instead, the current prototype exhausts all possible rules, leading to false values. An improvement would be to select rules that present information gain higher than a certain threshold. We are currently experimenting with various configurations and data in order to further study the above settings.
- For the case of the content of the blog post, a web feed often contains a small snippet of the actual post. This is the most important reason why the Jaro-Winkler distance measurement was selected over the variations of the Levenshtein distance. Therefore, after the summary is identified in the HTML document, the issue of extending the summary and describing the full content is open. The software prototype is addressing this issue as following:
  - While the parent element does not contain special tag names (e.g. div), the algorithm considers the parent element as the placeholder for the content recursively.
  - When a special tag name is found (e.g. div), the previously selected HTML element is the final element used to generate the filter.

6.4 Performance and Evaluation

Performance

The prototype was developed and ran in a MacBook Pro, with a 2.66 GHz Intel Core 2 Duo CPU and 8 GBs of memory. While running the prototype for evaluation purposes, the CPU usage was approximately around 10%, with some peaks of 50%, as seen on Figure 16.

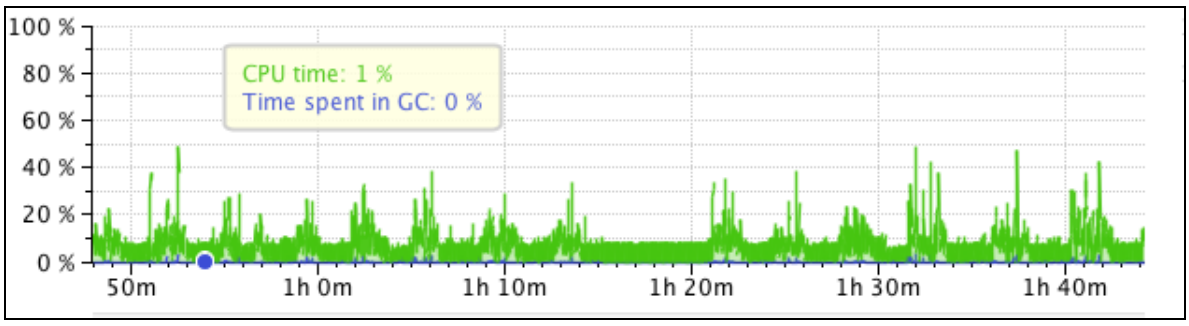

Figure 16: CPU usage of the prototype

Moreover, the process was profiled[71] with YourKit Java Profiler 11[72]. The analysis shows that most of the CPU time spent was during the fetching of a resource from the Internet (see Figure 17).

[70] http://tika.apache.org/
[71] Profiling a program includes the analysis of the resources spent, like memory and CPU usage, as well as the frequency of usage of methods invoked during its runtime.
[72] http://www.yourkit.com/

Therefore, the CPU time is expected to be significantly lower, as fetching a resource is generally assigned to other components of a weblog crawler (i.e. distributed workers with fetching role).

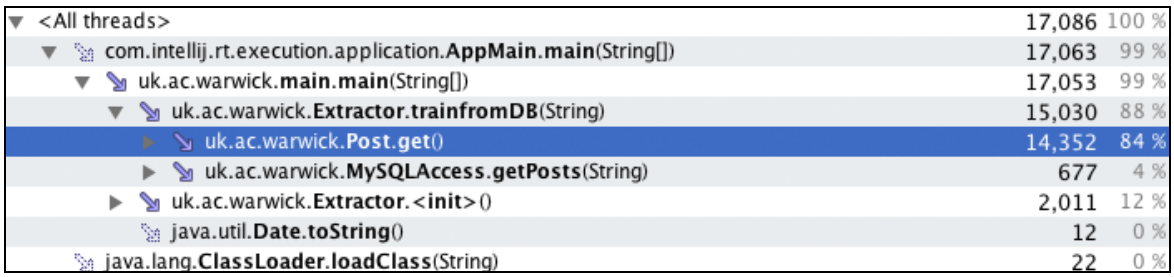

Figure 17: CPU usage by methods invocation. The number on the left of the usage percentage is milliseconds.

Concerning the memory usage, the prototype was assigned to process blogs in a batch mode of 100 blogs. Therefore, there were occurrences where hundreds of posts and thousands of filters were stored in the central memory. This is clearly shown on Figure 18, where some peaks around 100 Mbytes were observed. After the batch of 100 blogs is finished and a new cycle starts, the memory is no longer addressed, and the garbage collector of the Java Virtual Machine is called in order to free the unaddressed memory.

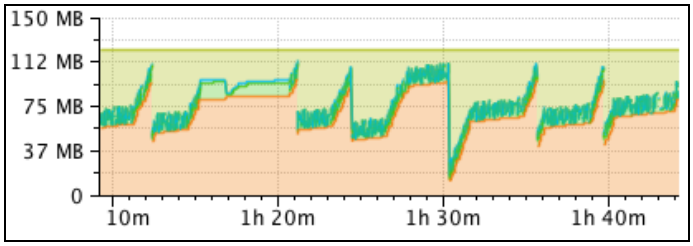

Figure 18: Memory usage of the prototype.

Overall, the performance analysis shows that the prototype is running fast (during a long run, the average rate for producing filters is *2,5 filters/second*), while keeping the necessary resources like memory and CPU usage low. Furthermore, incorporating the described methodology into the BlogForever infrastructure and more specifically the crawling process will definitely accomplish the efficient reuse of the majority of the resources spent.

Evaluation

The above measurements were taken during the *evaluation* of the prototype. The subject of this evaluation was not only to assess the performance aspects of the prototype, but also to evaluate the accuracy of the actual blog data extraction. In order to achieve the above, a use case was set up. Details about the use case including the following:

- A big collection of diverse blogs was populated. For every blog, the feed containing a list of posts was acquired. The total number of blogs was 325 blogs.
- Every blog feed contained at least 10 entries. If more entries were included, the remaining entries were discarded and were not used for training or evaluating the wrapper.
- The blog post properties title, author, publication date and post content were evaluated.

Evaluation was conducted following the *10-fold validation* method. This method is very popular when evaluating the quality of a classification approach. The aim of this method is to assess the expected accuracy on future examples, a measurement known as *prediction error*. In order to achieve

the above, the training data, which is the collection of 325 blogs as mentioned earlier, is divided in 10 disjoint folds. Training takes place for the 9 out of the 10 blocks and testing of the model occurs for the remaining fold. This task is repeated 10 times, in order to test all blocks (see Figure 19). In our case, one block corresponds to one entry found on a web feed. Finally, the prediction error is extracted by calculating the average errors for all 10 cases.

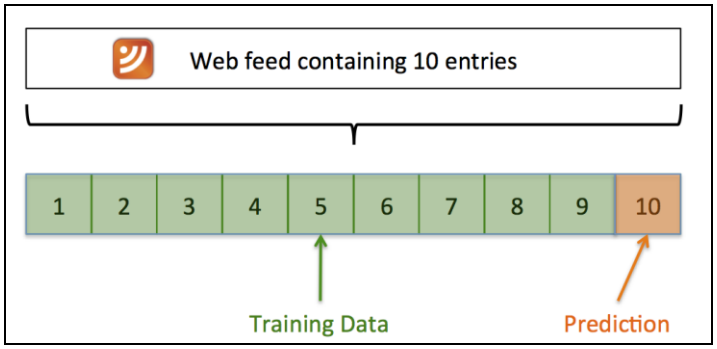

Figure 19: A visualisation of the 10-fold validation approach applied.

The results of the evaluation were made on the author and the title of a blog post. While the web feed of the blog contains more information like the content and the publication date of the post, validating the correct extraction of the values was not possible. For the case of the content, the feed contains less information, namely the summary. For the case of the date, the styling of the date during the training was not captured and was therefore not easy to automatically validate the extraction. A future version of the prototype will include more information concerning the date matching in order to be included in future validations. Given the above, a manual validation on a small sample of the use case has shown that the proposed approach captures the correct data in most of the cases and is equally effective to the properties been 10-fold validated. The results of the evaluation are shown in Table 19.

Table 19: Accuracy for the evaluation study

	Author mismatch	Title mismatch	Accuracy
Total	493/2547	43/2547	89%

Results of Table 19 show that accuracy is high (*89%*). For the case of the title, the accuracy is as high as *98.3%*. For the case of the author, the accuracy drops to *80.6%*. A survey on the errors of the author attribute shows that these errors appear mostly due to the following reasons:

- There are cases where a blog does not show the name of the author at all. Instead the author name might appear as an author of comment, leading to the false production of filters.
- Similarly to the above, there are a lot of cases where the author name appears as a comment in addition to the post author. This leads to the generation of several filters that point to comment blocks, leading to an increased gain of elements found in comments.
- Because of the time difference between the fetching of the web feed and of the blog post, there are cases where blog posts have changed.

In order to address the issues discussed above, the next version of the prototype will attempt to be more flexible during the matching of the properties to be found. For example, there are cases where the name of the author appears with some preceding text such as “by <author name>”. The current implementation is only looking for a complete and absolute match on the author and title attribute. In addition to the above, the analysis on the total collection of the filters and the attributes (IDs and CSS classes) of the filters more specifically has revealed some patterns that can be used in order to train the model faster and increase its accuracy. The patterns show that the most common values for the

IDs of the author are "*header*" and "*site-title,branding,header*". For the CSS Classes the values appearing most are shown in Table 20.

Table 20: Some CSS Classes values ordered by the number of occurrences. The first column corresponds to the case of the author and the second to the title of a blog post

Author	Title
fn	entry-title
author,vcard	post-title
url,fn,n,author,vcard	title
url,fn	storytitle
url	posttitle
comment-author	post-format-icon
url,comment-author	pagetitle
author	entrytitle
url,fn,n	single-title
comment-author,vcard	rsswidget
fn,comment-author,vcard	single
url,fn,comment-author,vcard	photo-title

The values in the Table 20 can be used to our model in order to promote several well-known values and discard others (e.g. the value comment-author is used to describe a comment and not the author of a post, even if there are cases where the values are identical).

6.5 Extraction of Extended Blog Data

The earlier sections of this report addressed data extraction of basic elements directly associated with the core of the blog data model [3]. However, the blog data model is extended through a set of components that enable storage of additional data elements of a blog. This section describes the process of data extraction that goes beyond the core and includes data that fits the components of the extended blog data model.

The purpose of data extraction is to also populate the additional components of the extended data model (see [3] for details) which do not require post-processing of crawled data or already stored in the database. Components that are addressed as part of data extraction are: Network and Linked Data, Community, Categorised Content, Crawling Info, External Widgets, Web Feed and Blog Context. Data extraction in relation to each of the components is addressed in this section.

Network and Linked Data
As a prerequisite for extracting network data, the types of these networks must be defined. Networks can denote the relation (also ties) between blogs, their pages, authors or various other concepts. Similarly, the relationships between the concepts can be defined in a number of ways. For example, a relation (directed/indirect) between authors can be considered as existent if comments are being exchanged on authors' blogs. A more detailed account of possible network structures within the Blogosphere is discussed elsewhere [3]. Figure 20 describes the relevant part of the model. However, identification of some of the network structures remains beyond the task of data extraction and must be addressed as part of post-processing. Here we highlight some of the common network structures directly related to the process of data extraction.

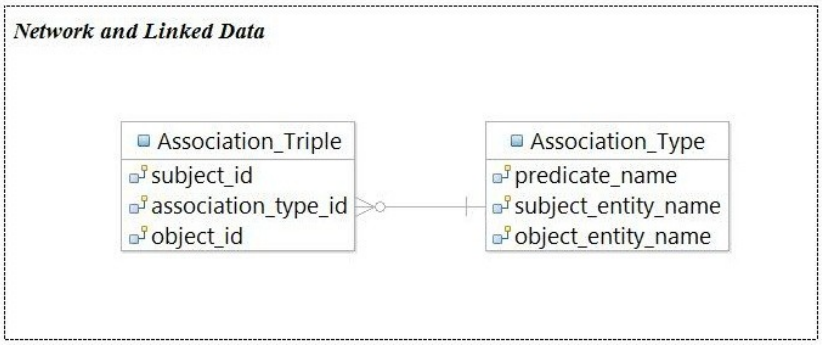

Figure 20: Excerpts from the data model to capture network structures

Among the most frequently network structures discussed on the Web and the Blogosphere are hyperlink networks. Capturing hyperlink network structures relies on successful extraction and analysis of hyperlinks (e.g. [85], [86]). This procedure is relatively easy and unambiguous given the existing HTML mark-up. Hyperlink extraction is a common procedure in crawling. Some crawlers may have a dedicated component for extracting and storing hyperlinks [56]. This commonly used component can be adopted for the purpose of extracting hyperlink networks. The process of extracting hyperlink networks may require additional steps of: distinguishing between external and internal hyperlinks or excluding hyperlinks to non-blog resources.

The increasing take up of semantic mark-up and linked data opens up opportunities of defining specific types of hyperlink data. It becomes possible to identify what type of resources the hyperlinks point to. Processing of the semantic notation is discussed in section 6.6.

Community

The Community component (Figure 21) aims to capture a list of attributes associated with the authors and readers of a blog. The extraction of these properties requires combined use of embedded structured content and post-processing. The priority for identification of elements should be given to known ontologies and vocabularies. The use of FOAF mark-up should prevail when aiming to capture the attributes of this component. Information about post-processing of the data for extracting attributes, such as affiliations, names or places should be conducted as part of post-processing.

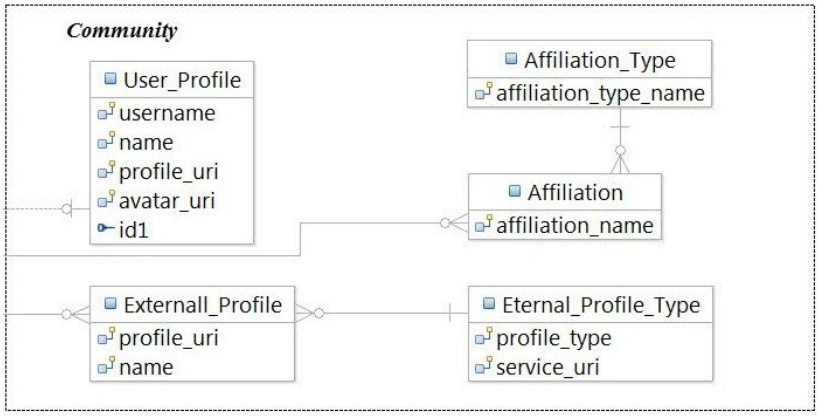

Figure 21: Excerpt from the data model to capture data around blog communities

Categorised Content

The Categorised Content (Figure 22) component aims to distinguish various media and types of data elements. The methods for identifying most of the attributes will require HTML processing. The HTML tags and attributes are to be analysed to identify the files, their formats and paths as linked

within the blog content. The use of marked MIME types associated with audio, video, image and document are to be processed and distinguished within the repository.

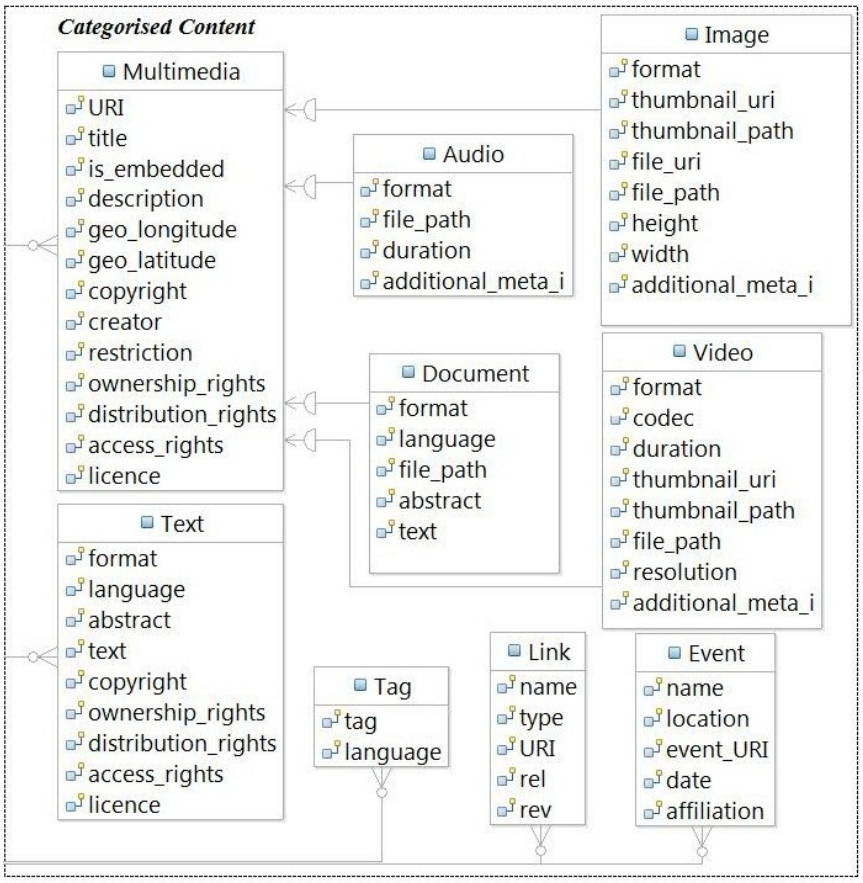

Figure 22: Excerpt from the data model to capture categorised content

The textual content (cleared from HTML mark-up) need to be captured and stored separately. This process also referred to as 'stripping' the HTML tags is a frequent procedure in many applications. The function of stripping tags can be implemented by using regular expressions. JSoup or similar libraries (see Section 4.2 for details) for parsing and processing HTML content are already equipped with the functionality of clearing the HTML-mark-up. The decision on using specific library can be made along with selecting the libraries for parsing. Capturing textual content also requires detection of the language used. The process of language detection can be addressed by the use of Tika[73] library.

Crawling Info and Web Feed

Information about the crawling performed to acquire blog data and the web feeds used are passed directly from the crawler. The type of the feed (identified as part of the pre-processing for data extraction) along with the parameters of crawling (e.g. software version used, dates, authentication) are not considered part of the data extraction process (Figure 23).

[73] http://tika.apache.org/

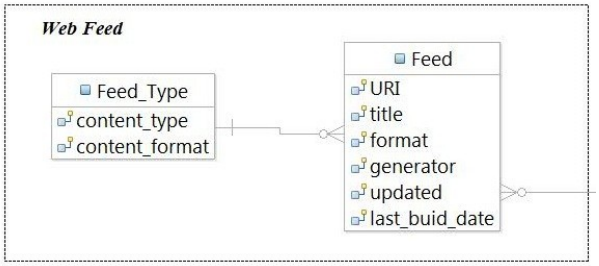

Figure 23: Excerpt from the data model to capture data about blog feeds

External Widgets

External widgets (Figure 24), viewed as embedded blog components that display external content, are to be captured within the blog. Widgets are frequently used within the blogosphere to display the content of externally published content fed by RSS. The standardisation in developing widgets is limited. Hence, the task of identifying widgets within a blog is not trivial. However, widgets are usually displayed alongside the main content area and contain references to the external services feeding the data. Identification of the <aside> containers within the HTML pages can support the task of extracting information about the widgets. The analysis of the links to the external resources can provide further information about the type of the widget. Given the diversity of the widgets and the standards, a focused approach in working with specific types of widgets would be most relevant. Among available examples in detecting a specific type of a widget is a service offered by BuzzStream[74].

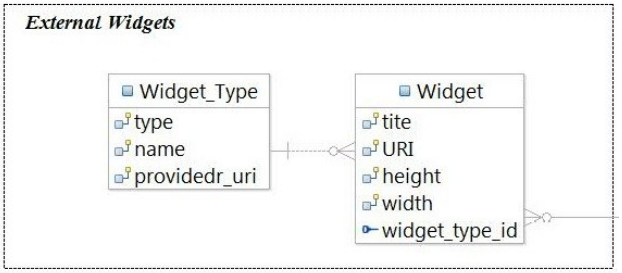

Figure 24: Excerpt from the data model to capture data about widgets

Blog Context

Blog Context (Figure 25) provides descriptive information about the blog and its elements. It enables capturing the view of the blog as a graphical snapshot. Capturing the view can be achieved by using one of the freely available tools such as wkhtmltopdf[75]. Openly available services such as BlogBooker[76] can also be considered, however, inbuilt services should be given preference.

The entities within the component enable capturing the metadata provided by the blogger to describe the keywords and descriptions of the blog via the <meta> tag. It also enables storing the structured data as marked within the blog. Hence, all the relevant <meta> tags should be captured and stored once the blog data is parsed. This should also include semantic annotation. Last, but not least, mechanisms responsible for the presentation layer of the blog should be captured where possible. These include information about templates where possible and/or the CSS files along with associated graphics. This process can exploit HTML mark-up that provides information about the blogging platform to support the process of extracting locating the rendering and presentation related files and data.

[74] http://tools.buzzstream.com/blogroll-list-builder
[75] http://code.google.com/p/wkhtmltopdf/
[76] http://www.blogbooker.com/index.php

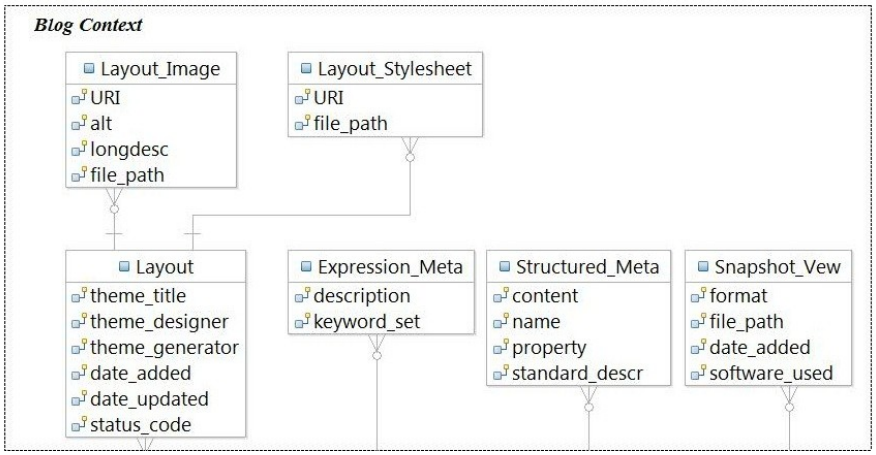

Figure 25: Excerpt from the data model to capture blog context

6.6 Extraction of Structured Data

Semantic notation that describes the concepts (e.g. people, products, organisations, places or events) within web pages is increasing rapidly. Semantic mark-up of web resources makes their processing more robust by enabling higher levels of automation. The processes of data extraction can also benefit from exploiting available semantic notation.

The standards of adopted notations vary widely. Web Data Commons[77], an organisation that provides open access to crawled web data, highlights eleven formats used for semantically marking web data (see Table 21).

Table 21: Semantic data types as referred to by Web Data Commons.

Format	Description
RDFa	A specification to express structured data in any markup language (e.g HTML). It uses RDF for mapping subject-predicate-object expressions within web documents.
HTML Microdata	Nested groups of name-value pairs added to HTML documents along with the existing web content.
hCalendar Microformat	A format for events and calendaring. It uses the representation of the iCalendar standard.
hCard Microformat	A format for representing people, companies, organizations and places. It uses the representation of the vCard (RFC2426) standard.
Geo Microformat	A representation of the "geo" property from the vCard standard. Geo is commonly used to mark the latitude/longitude coordinates.
hListing Microformat	A proposal for a listings/small-ads format for embedding in HTML.
hResume Microformat	A format for representing fields common in resumes as published on the Web.
hReview Microformat	A format for embedding reviews and can be suitable for products, services, businesses, events and so on.
hRecipe Microformat	A format for denoting cooking recipes within HTML.
Species Microformat	A proposal to enable describing taxonomies of species in web documents.

[77] http://webdatacommons.org

XFN Microformat	A format (XHTML Friends Network) for representing human relationships using hyperlinks.

The process of semantic data extraction requires parsing a page and detection of specific formats (e.g. microdata, microformats). The ready-made commercial and openly available solutions for extracting semantic data are provided via APIs or as a suite of software tools. Among the available APIs are the Alchemy API[78] that enables extraction of microformats or structured data amongst other services. Among the software tools, Any23[79] is an openly available library that is frequently used and is (currently) being actively developed. Among other smaller libraries and scripts are: PHP Microformats Parser[80], Text::Microformat[81] and many others[82] on source code repositories.

The following sections 6.6.1 and 6.6.2 describe the potential use of microformats as part of the data extraction process.

6.6.1 Extraction of Microformats

The main goal of microformats is to use existing HTML/XHTML tags to embed machine readable metadata into web documents. The use of Microformats has been studied and tested in the earlier submitted deliverable (see [4]for details). We highlight some of the results of this work and comment on their adoption for data extraction.

Microformats have been a fairly recent introduction to the Web. The use of microformats across the Web grew rapidly shortly after their introduction. Over the last years, their use of microformats accelerated across the web due to support from search engines like Google. However, after the joined proposition of Microdata the uptake in using microformats have slowed down. The use of hCard and XFN within the Common Crawl corpus has grown by less than 1% between the 2009/2010 and early 2012. The use of GEO, hCalendar and hReview remained approximately unchanged [87]. Similar trends of slight update of some microformats and reduction of others is reported by Yahoo! Research [88]. However, the availability of microformats still takes a reasonable portion of the semantically annotated web documents. According to Bizer [89], microformats represent around a half of the semantically annotated web documents. Their consideration as part of the data extraction can be considered relevant.

Given the availability of semantically annotated content, the questions are:

- What Microformats are to be used when extracting data from blogs?
- What tools/infrastructure can be employed for extracting microformats?

The decision on using specific microformats can be based on their relevance to the aims and objectives of the BlogForever and the maturity of the particular standard.
From the following list of stable Microformats [4], we identify Microformats that are more likely to provide relevant information.

Stable Microformats: Considered for Extraction

- **hCalendar**[83]: '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'.

[78] http://www.alchemyapi.com/
[79] http://incubator.apache.org/any23/
[80] http://www.phpclasses.org/package/3597-PHP-Extract-microformat-data-embedded-in-HTML.html
[81] http://code.google.com/p/ufperl/
[82] https://github.com/search?q=microdata&type=Everything&repo=&langOverride=&start_value=1
[83] http://microformats.org/wiki/hcalendar

- **hCard**[84]**:** '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'.
- **rel-license**[85]: 'a simple, open, format for indicating content licenses which is embeddable in HTML or XHTML, Atom, RSS, and arbitrary XML'.
- **rel-tag**[86]: a format for 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.
- **XFN**[87] **(**XHTML Friends Network): '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.'
- **XMDP**[88] (XHTML Meta Data Profiles): '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'.

Stable Microformats: Less Relevant for Extraction

- **rel-nofollow**[89]: '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)'.
- **VoteLinks**[90]**: '**a format for 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'.
- **XOXO**[91]**:** 'a simple, open outline format written in standard XHTML and suitable for embedding in (X)HTML, Atom, RSS, and arbitrary XML'.

Draft Microformats

Many of the draft microformats are highly relevant for the data extraction. For instance the following:

- **adr**[92]**:** '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'.
- **geo**[93]: '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'.

Other microformats may be relevant when performing data extraction on specific subjects or within the boundaries of a certain domain. For instance:

- **hListing**[94]**: '**a proposal for an open, distributed listings (small-ads; classifieds) format suitable for embedding in (X)HTML, Atom, RSS, and arbitrary XML'.
- **hNews**[95]: '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

[84] http://microformats.org/wiki/hcard
[85] http://microformats.org/wiki/rel-license
[86] http://microformats.org/wiki/rel-tag
[87] http://gmpg.org/xfn/
[88] http://gmpg.org/xmdp/
[89] http://microformats.org/wiki/rel-nofollow
[90] http://microformats.org/wiki/vote-links
[91] http://microformats.org/wiki/xoxo
[92] http://microformats.org/wiki/adr
[93] http://microformats.org/wiki/geo
[94] http://microformats.org/wiki/hlisting
[95] http://microformats.org/wiki/hnews

format, rel-principles, a format that describes the journalistic principles upheld by the journalist or news organization that has published the news item'.

- **hProduct**[96]:'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'.
- **hRecipe**[97]: ' a simple, open, distributed format, suitable for embedding information about recipes for cooking in (X)HTML, Atom, RSS, and arbitrary XML'.

Any23 for Microformat Extraction

The extraction of Microformats has been evaluated and documented as part of the earlier submitted deliverable D2.4 [4]. The evaluation included the use of Any23 library which was customised to extract semantic content from blogs. The reader is directed to read the details of the evaluation from the submitted report *(op. cit.)*. The outcomes of the report suggest that using the Any23[98] library is appropriate for the task. The extensive list of supported additional formats of the library (e.g. RDF/XML, RDFa, Microdata and CSV) enables its use for extracting a wider range of semantic data. Other advantages include its being Open Source and wide developer/user community. The use of Any23 is, therefore, recommended for the task of Microformat extraction.

6.6.2 Microdata and RDFa

Microdata constitute a WHATWG HTML5 specification for nesting semantics into web documents. It can be used by web crawlers, search engines and browsers for improving and understanding the information within the web documents and for enhancing user experiences (and relevance of search results in particular). Unlike RDFa or Microformats, Microdata attempts to simplify the process of annotating HTML elements with machine readable tags. Microdata use a supporting vocabulary for describing an item and name-value pairs to assign values to its properties. Each name-value pair of Microdata is called a property, while groups are called items. Items and properties are represented by regular elements. For instance, 'itemscope' attribute is used to create an item. Adding a property to an item requires the use of the 'itemprop' attribute [90].

Recent development in the use of Microdata is schema.org[99]. Schema.org is a joint initiative by major search engines that provides recommendations on the use of Microdata. The recommendations revolve around schemas, which describe various concepts and their properties. The provided schemas describe concepts such as: Book, Movie, Recipe, Event, Person, Organisation and Product.

Among the concepts described by schema.org is 'Blog'. The schema describing the concept of blog includes properties that are of direct interest for BlogForever. If the blog is annotated by using the recommended notation, the process of data extraction can be conducted automatically without the need for generating and executing a wrapper. The properties of the annotated blog, such as 'author', 'blogPost' or 'datePublished' can be extracted without the use of web feeds. However, given that the number of annotated web documents and blogs in particular is still small, wrapper-based approaches should not be excluded from the methodology. Furthermore, given the fact that web documents remain poorly annotated, the use of wrapper based approaches can be used for identifying the components of blogs and republishing the them as Linked Data or offering enhanced search experiences as part of the BlogForever repository.

Yet another approach for semantic mark-up is RDFa. RDFa was proposed in 2004 for inserting metadata into XML – a step towards a machine-readable Web. RDFa has been specified for XHTML, however, the attributes of RDFa are recognised in any version of HTML [91]. It provides a

[96] http://microformats.org/wiki/hproduct
[97] http://microformats.org/wiki/hrecipe
[98] http://incubator.apache.org/any23/
[99] http://schema.org

vocabulary-agnostic syntax for describing and annotating resources. It is used for creating links to resources on other pages using custom HTML attributes. As one of the earlier developed standards, the adoption of RDFa is more common. The adoption of RDFa, however, increased rapidly in the last couple of years reaching the frequency mark of around 4-5% [87, 89].

The use of RDFa is particularly relevant for the task of data extraction from blogs. Frequently used along with ontologies such as FOAF[100], GD[101], SKOS[102], SIOC[103] and MO[104] (see [87]), extraction of blog data can benefit from utilising this format. FOAF, SIOC and SKOS are particularly relevant for extracting data from blogs [92]. The rationale for adopting RDFa is similar to that of Microdata, with the additional benefits of the reported rapid uptake. Similarly to microformats, Any23 currently remains among the most relevant tools for exploiting the use of Microdata and RDFa within Web documents for data extraction.

6.6.3 Semantics of HTML Mark-up and Page Layout

The use of semantics embedded within the HTML mark-up itself, is equally important as the use of ontologies and controlled vocabularies as discussed in earlier sections. Widely adopted notation within HTML can provide insight into the published data and support its extraction. For instance, the use of <ol> tag, commonly used to denote ordered lists, can provide clues about the data being published. The semantics of the mark-up becomes further emphasised within the context of HTML5[105].

The most recent specification of HTML5 (as published by W3C [93]) includes mark-up that denotes, for instance, *page sections*, *content groups* or *text-level semantics*. Among the elements that denote various page sections are: <section>, <nav>, <article>, <aside>, headings elements (<h1-6>), <hgroup>, <header>, <footer> and <address>. Content groups are marked with elements such as: <blockquote>, <dl> (description list), <figure>, <figcaption>, <hr> (thematic break), <pre> (preformatted text) and so on. Similarly, text level semantics are used to convey meaningful information associated with the text. HTML elements such as <cite>, , , <small> etc. are among tags widely used to embed additional semantics. The adoption of HTML5 appears to be gaining momentum. Within the Blogosphere, 25% of resources were identified as using HTML5 [94].

Therefore, data extraction can be supported where HTML embedded semantics is evident. More specifically, understanding the layout of the page referring to the page section notation can help identifying sections that are of interest to data extraction such as navigation elements, articles or non-primary content (<aside>). Additional granularity can be achieved by processing extracted data to identify figure captions or citations. Yet, despite their potential value in supporting data extraction, it remains insufficient to extract the data based on the semantics.

6.7 Ontology-Based Data Extraction

The term ‘ontology’ originates from philosophy to describe the nature and structure of reality. The way in which this term is being used differs across communities. The knowledge engineering community, more recently, has adopted the term and started using it to define explicit specifications of conceptualisations [95].

[100] http://xmlns.com/foaf/spec/
[101] https://developers.google.com/gdata/docs/2.0/elements#gdReference
[102] http://www.w3.org/2004/02/skos/
[103] http://sioc-project.org/
[104] http://musicontology.com/
[105] http://dev.w3.org/html5/spec/

Data extraction communities attempt to exploit the inherent capabilities of ontologies to provide insight. The use of semantics encapsulated within the data can support and offer levels of automation for data extraction. An ontology-based data extraction was proposed by Embley *et al.* [96]. They proposed extracting the data records from list pages by using an ontology defined by experts. Laender [26], in his review, categorises tools that employ ontologies for data extraction as ontology-based. These tools use the data itself, rather than relying on the page structure.

In addition to the semantic mark-up (as discussed in Section 6.6) that uses certain ontologies, such as DC[106], FOAF[107] or SIOC[108], the task of data extraction can also be informed by existing *domain knowledge*. Domain knowledge, represented as an ontology (including a vocabulary of concepts and relationships between them) can then be used to support the process of data extraction. Approaches of using domain ontologies have been described in the earlier works, such as social networks and communities [97], bio-informatics (as described in [25]), or more generally, with web tables [98].

The benefits of using ontologies for data extraction become clear given the statistics for the semantically annotated web pages. Only 13-30% of all the HTML pages contain *some* structured data [99]. Data structured according to specific ontologies may have a smaller representation. While the recent introduction of Microdata and Microformats contribute to the uptake of semantic mark-up, many websites remain reluctant to invest into publishing high quality structured data using most suitable ontologies. Therefore, rather than waiting for the websites to provide annotated content, alternative solutions for identifying concepts would greatly support the task of data extraction.

As part of the BlogForever project, data extraction is limited to blogs. The Blogosphere, sharing across the blogs some of its features, provides common grounds for developing domain specific knowledge. The multitude of forms that blogs may take and subject domains that blogs may cover remains a significant challenge. Ontologies describing the domain of blogs can help by

- Identifying, extracting and labelling conceptual objects from blogs, and
- Annotating and republishing content as linked data.

The DIADEM project, as discussed by Furche *et al.*[100, 101], is among the initiatives to use domain knowledge and ontological constraints for improving the performance of data extraction. Use of ontologies, in this case, has been demonstrated for understanding the HTML forms of real-estate agency websites. Ontologies, as shown on this example, can formally model web forms that can be used to check the coherence of the rules identified from the different web sites. While the use of ontologies is found useful, the process of creating the ontologies remains manual.

Automatic approaches for developing ontologies have also been proposed. Su *et al.* [102] propose ODE – Ontology-assisted Data Extraction – that builds on approaches of OntoBuilder [103] and TANGO [104]. OntoBuilder identifies web form labels and fields from different websites and integrate these into common domain knowledge. TANGO, on the other hand, attempts to understand tabular data of web pages (focusing primarily on list pages) and automatically construct, so-called mini-ontologies that can be further mapped and integrated. The use of lexical databases such as WordNet[109], plays an important role in integration. ODE, therefore, can integrate a range of forms and use the results they produce for automating data extraction.

Unlike DIADEM or ODE, the use of web forms is less relevant in the Blogosphere. Most common forms are used for adding comments or searching through blog archives. Furthermore, for performing a generic, as opposed to domain specific, data extraction automatic generation of ontologies may not be relevant. Instead, a manually developed ontology aligned with the blog data model [3] developed

[106] http://dublincore.org/
[107] http://www.foaf-project.org/
[108] http://sioc-project.org/
[109] http://wordnet.princeton.edu/

earlier as part of the BlogForever project may be considered as a better alternative. Additionally, the use of other expert-created and tested (high level) ontologies such as SIOC, FOAF and SKOS[110] can also be employed when necessary.

Even when manually created ontologies are used, the issue of extracting and, most importantly, integrating the data remains open. For example, within the context of BlogForever, the problem of identification and extraction of author data has been identified as important. The task of capturing author networks is more challenging compared to extraction to HTML link networks. In addition to the challenge of identifying concepts the data extraction task requires their meaningful integration. The use of ontologies for integrating the data remains central. However, being in its early days, this area remains largely unexplored. Some seminal works that discuss such integration include, for example, the so-called tripartite model of integrating actors, concepts and instances as proposed by Mika [97], or, Social identity Schema Mapping (SISM) [105] that employs SKOS for mapping concepts represented in FOAF, vCard, XFN and other similar ontologies. These or similar novel approaches can be particularly useful for achieving successful data extraction.

6.8 Discussion and Related Work

A large variety of technologies have addressed the issue of extracting information from web pages. Concerning blogs, several services can be found like Bloglines[111], BlogScope[112], Technorati[113] and until recently BlogPulse[114]. The main aim of these services is to perform analytics, including trends, link analysis, as well as more sophisticated semantic processing of data like opinion mining and sentiment analysis. Furthermore, this functionality is provided through APIs and web services. For example, Bloglines provides an RSS interface that allows aggregating web feeds from selected blogs or create more dynamic views based on geographical information found on the content of the posts. BlogScope and its channelled commercial product sysomos[115] offer analytics services through social media monitoring. However, to our knowledge the above systems exploit the web services provided by the media provider (i.e. web feeds of the blogs, pings, pingback, etc.) and therefore do not attempt to identify and extract properties residing in blog posts.

Probably the closest approach to the one proposed in this report is the one by the Webdam project[116], a project aiming at archiving web objects through web feeds. The paper by Oita and Sellenart [84] presents a study that follows a strategy similar to ours. The work aims at analysing a web page while matching it to the semantic information found in a web feed. The authors present results concerning polling and update interval estimation of feeds, as well as some discussion on the amount of information found in these feeds. Our work shares similar goals and extends the Webdam approach. More specifically, while both approaches rely on knowledge acquired during the cross-matching of a web feed against a web page, our approach devises general extraction rules for a weblog and may therefore be applied to content not included by a web feed (i.e. old posts). This is one of the reasons that the approach by Oita and Sellenart is achieving good results for the main content of a web page and delivers poor results on properties like title, date etc [84].

ArchivePress[117], similarly to BlogForever, is a blog archiving project that has developed a plugin[118] that allows archiving blogs powered by WordPress blogging platform only, making the solution

[110] http://www.w3.org/2004/02/skos/
[111] http://www.bloglines.com/
[112] http://www.blogscope.net/
[113] http://technorati.com/
[114] Since January 2012 the website http://www.blogpulse.com/ is no longer available.
[115] http://www.sysomos.com/
[116] http://webdam.inria.fr/
[117] http://archivepress.ulcc.ac.uk/
[118] http://code.google.com/p/archivepress/downloads/list

proposed relatively limited. Furthermore, in order to use the plugin, the user must be the owner of the blog and have the technical knowledge to install and run it. In the case of WordPress, due to its popularity and its open source access, several plugins have been developed that allow backing up a blog and therefore achieve the same functionality[119].

From a more general point of view, Heritrix[120], the open source archive crawler used by Internet Archive[121], was used in order to revisit webpages and identify important changes through heuristics [106]. This method is similar to the methodology proposed by the generic wrapper induction approach, discussed earlier in section 3.3. The above solutions share a common strategy, where the attempt is to pairwise compare webpages in order to automatically identify the template that renders the webpage. However, these approaches are not tailored to address blog data extraction.

Finally, other approaches exist that aim at identifying the main article of a web page. Boilperpipe[122] is the state-of-the-art and most successful tool in analysing the content of a web page [107]. Boilerpipe makes use of structural features such as HTML tags or sequences of tags forming subtrees and employs functions found in quantitative linguistics. Through these functions, such as average word length and average sentence length, every web page segment is analysed and the candidate with the highest score is selected as the main article. Even though the result of this approach delivers relatively good results (an evaluation by Oita and Sellenart [84] shows that the precision of Boilerpipe is 62.5%, even though our experiments have shown that in the case of blogs Boilerpipe is performing better), the application of this approach to blog data extraction is relatively limited, since it can only extract the main content of a blog post.

6.9 Conclusions

A method for inducing a fully automated weblog wrapper generation was presented. Initially, a case study where web feeds were used for archiving is provided. This case study has revealed that web feeds contain useful information that can be exploited efficiently. However, relying on web feeds may limit the amount of weblog content accessible. In order to remedy the above, a wrapper that is trained through information found on web feeds is proposed.

The generated wrapper exhibits increased granularity, since it manages to identify and extract several weblog properties, such as the title, author, publication date and main content of the post. This is accomplished through the induction of rules, which are selected through the adoption of a probabilistic approach based on information entropy and gain. The devising of these rules is based on the generation of filters. The filters constitute a structure that when applied to a web document singles out an HTML element. They are described in triples where each of its element-attributes describes the HTML element in different forms (absolute path, CSS Classes and HTML identifiers). The overall approach is evaluated against a real-world collection of weblogs and the results show that the wrappers generated are robust and efficient in handling different types of weblogs.

[119] For example:
- http://wordpress.org/extend/plugins/backupwordpress/
- http://pluginbuddy.com/purchase/backupbuddy/
- http://www.idrive.com/wordpress.htm
- http://wordpress.org/extend/plugins/lh-rdf/

[120] http://crawler.archive.org/

[121] http://www.archive.org/

[122] http://code.google.com/p/boilerpipe/

7 Data Extraction Methodology

This section outlines a blog data extraction methodology, considered relevant for application within the context of the BlogForever project. It consolidates the earlier discussions (Sections 3-6) and makes suggestions directly related to the development of the BlogForever platform. It abstracts to discuss the methodology at a higher level by providing examples of using the BlogForever platform and highlighting possible data extraction processes associated with the use. Unlike other known approaches this section focuses exclusively on data extraction from blogs only. This section first introduces the examples of using the system and discusses the relevant data extraction methodology accordingly. Section 7.1 describes the process of crawling and data extraction for capturing and updating blog entries. It is followed by suggestions for data storage (Section 7.2) and a workflow of the wrapper-based extraction flow (Section 7.3).

7.1 Crawling and Data Extraction Actions

Extracting data from blogs can be viewed as an essential process that leads to the creation or maintenance of a blog repository. The users of the BlogForever platform, unlike the users of the repository supported by the BlogForever platform, are directly involved with the task of ingesting blogs into the repository, monitoring and maintaining it. Abstracting the discussion of data extraction to the level of platform use, enables clarifying possible ways and differences in employing data extraction.

Data extraction and ingest into the repository as a functional requirement of the BlogForever platform is directly related to the following broadly defined actions:

- Populate a Repository
- Update and Maintain a Repository

Each of these actions may include a set of operations associated with data extraction:

- Crawl and capture the entire blog given the URL of the blog
- Add/Update a new post given the permalink (most likely received from the ping server)

The sequence of operations for performing data extraction will differ depending on the action. We describe each of these operations as workflow diagrams that describe the data extraction methodology.

7.1.1 Crawl and Capture Entire Blog

Capturing the entire blog from a given URL involves various data extraction related tasks. According to the earlier discussion, the use of web feeds for generating wrappers has been shown to be an effective method for extracting data. Hence, we propose a workflow that describes the process of data extraction. The operation of capturing the entire blog described here combines two workflows: extraction of all blog posts and extraction of all blog pages.

Extraction of All Blog Posts: We present the flow diagram to describe the process of capturing blog posts as follows:

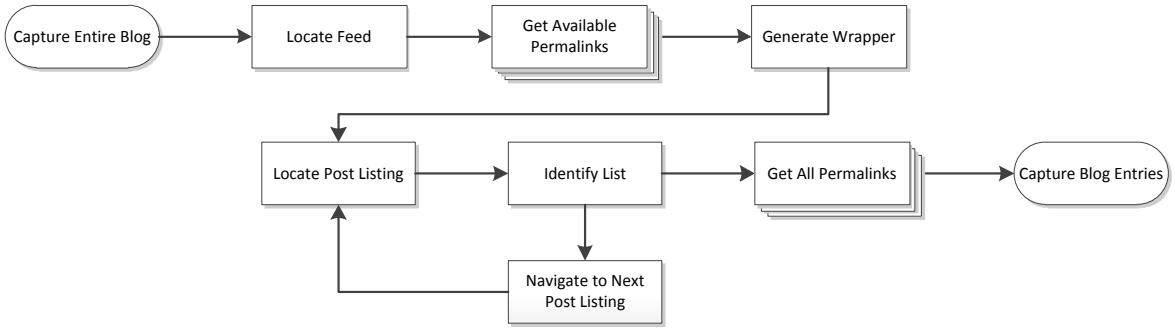

Figure 26: Workflow for capturing blog posts from the given blog URL.

Each of the steps presented in Figure 26 are discussed below.

1. *Locate Feed:* identify the appropriate feed/feeds. The feed enables the automation of the wrapper generation process. The task of identifying feeds must be divided into subtasks of: identifying feeds from HTML and selecting the appropriate feeds if more than one feed is available. Identification of feeds can be achieved by parsing the HTML and searching for '<link>' tag with type "application/atom+xml" for Atom feeds or "application/rss +xml"' for RSS. Choosing one feed over another is less simple and may require some heuristics of the URL of the feed.
2. *Get Available Permalinks:* parse the feed and identify permalinks and other data elements. The feed may only contain most recent posts; however, the available number of posts should be sufficient for generating the wrapper.
3. *Generate Wrapper:* locate the permalinks and automatically generate a wrapper as discussed in section 6.
4. *Locate Post Listing*: identify the list of posts displayed on the accessed blog page. Information from the earlier performed generation of the wrapper should be used to locate permalinks. Alternatively, patterns of repeated groups should be used to locate permalinks.
5. *Get All Permalinks*: locate the permalinks from the given listing.
6. *Navigate to Next Post Listing*: navigate the page which contains the next (usually older) list of blog posts. Mechanisms for navigations will be necessary due to the possibility of incomplete data provided via web feeds.
7. *Capture Blog Entries:* captures the blog posts for identified permalinks according to the generated wrapper.

Extraction of Blog Pages: The process for identifying the web pages associated with the blog should differ from the one discussed above. The reason for this is that blog pages are not usually distributed via web feeds. Hence, the process needs to be adjusted to accommodate the data extraction from pages. The task of data extraction can be divided into two sub-tasks: locating URLs, distinguishing them from posts and capturing their content.

As mentioned above, blog pages are not distributed via web feeds. Unlike posts they are not usually included in ping notifications received from a ping server. Hence, the sub-task of locating a page URL requires some attention. One of the approaches relevant for identifying the non-post pages can be performed by employing domain crawling. Domain can capture all the pages linked or exhibited on the blog. One of the limitations of this approach is the required scheduling for periodic domain crawls aimed to capturing newly added page entries. Another limitation is the need to crawl the entire blog which is an expensive process that confined by crawling policies.

Alternatively, location of blog pages can be conducted by analysing the blog structure. By looking into the structure of blogs it will be possible to locate sections of the HTML that are more likely to contain blog pages. References to HTML5 <nav> tag or similar CSS naming conventions can provide clues on locating the blog pages for extraction. Since, many blogs list additional pages such as

'About' or 'Contact' as part of the navigation section and marked with <nav> tag, basic search for the necessary HTML tag can provide good results in identifying a large number of non-post pages associated with the blog. Hence, searching for blog pages within the header or a footer of the HTML page can spare the expense of domain crawling.

The sub-task for distinguishing web pages can exploit the heuristics of blog URLs. The patterns of permalinks and URLs blog pages differ unless the generated URLs are specifically customised. In the case of WordPress, for example, the system can generate permalinks that contain the date and identifiers related to the post (i.e. title). The page URLs within the same system can include a category name or include a specific hierarchy. The heuristics of the URLs has been successfully used in the crawling domain. For instance, Zheng [108] describes the method for exploiting URL heuristics for focused crawling, which can be adopted for task of distinguishing pages from posts.

The sub-task of capturing and extracting the content form the pages and the possible differences between the two types of resources is discussed in section 7.1.2.

7.1.2 Capture/Update Blog Entry or Comment

Capturing the entries or comments requires the presence of a generated wrapper (the process of generating and applying a wrapper has been discussed in section 6). If earlier generated wrapper is not available for the blog, it can be assumed that the blog is new to the repository.

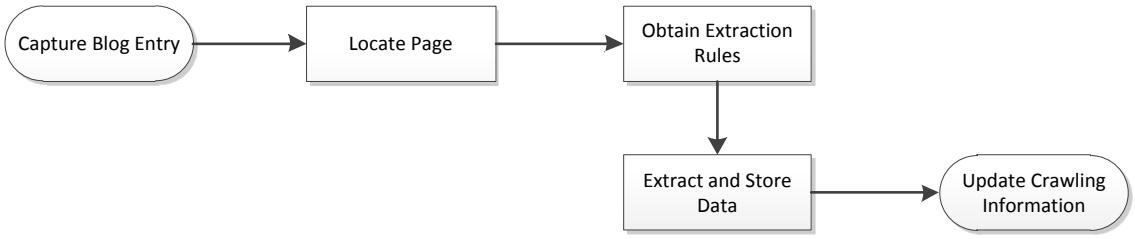

Figure 27: Data extraction from a blog entry.

Capturing the new blog entry or its update is essentially the same from the perspective of data extraction. This process will need to rely on the same extraction rules from the earlier generated wrapper. However, prior to storing a newly extracted post, it will be necessary to check whether the post has not been ingested into the repository earlier. If the post is being processed as an updated post as a result of a registered ping, then comparison with the stored records is needed to ensure that the record has indeed been updated and not, for instance, re-pinged without any change. We attempt to integrate these diverse operations into a single workflow after highlighting the prerequisites that may optimise the above mentioned processes.

7.2 Suggestions for Storage and Processing

As discussed earlier in section 6, the data extraction process may be computationally expensive. The identification of web feeds, repeated blocks, navigational elements, and generation of wrappers can require numerous traversals of the parsed documents and extensive training. All these operations may considerably extend the timing necessary for performing the data extraction. Hence, it is important to introduce mechanisms that could optimise this process.

More specifically, we suggest the following.

1. Store the URL to appropriate web feeds after they are located. Future use of the feeds can be necessary for re-generating the wrapper when the template of the blog has been updated.

2. Store the wrapper (as data extraction rules) for each of the blog, to avoid the need to generate rules of extraction for every post.
3. Associate data extraction rules with blog entries within the repository to optimise the time of retrieving earlier stored versions of entries.

All of those supporting measures can be integrated into the database module of System Manager component of the BlogForever Crawler.

7.3 Wrapper-Based Data Extraction Workflow

The following data extraction workflow attempts to integrate the discussed operations and suggestions (see sections 7.1and 7.2). It represents the sequential process of data extraction and can be used to inform the design of the data extraction system of the BlogForever platform. Figure 28 introduces the workflow in a diagrammatic form. It describes the process of capturing and extracting data from a given URL.

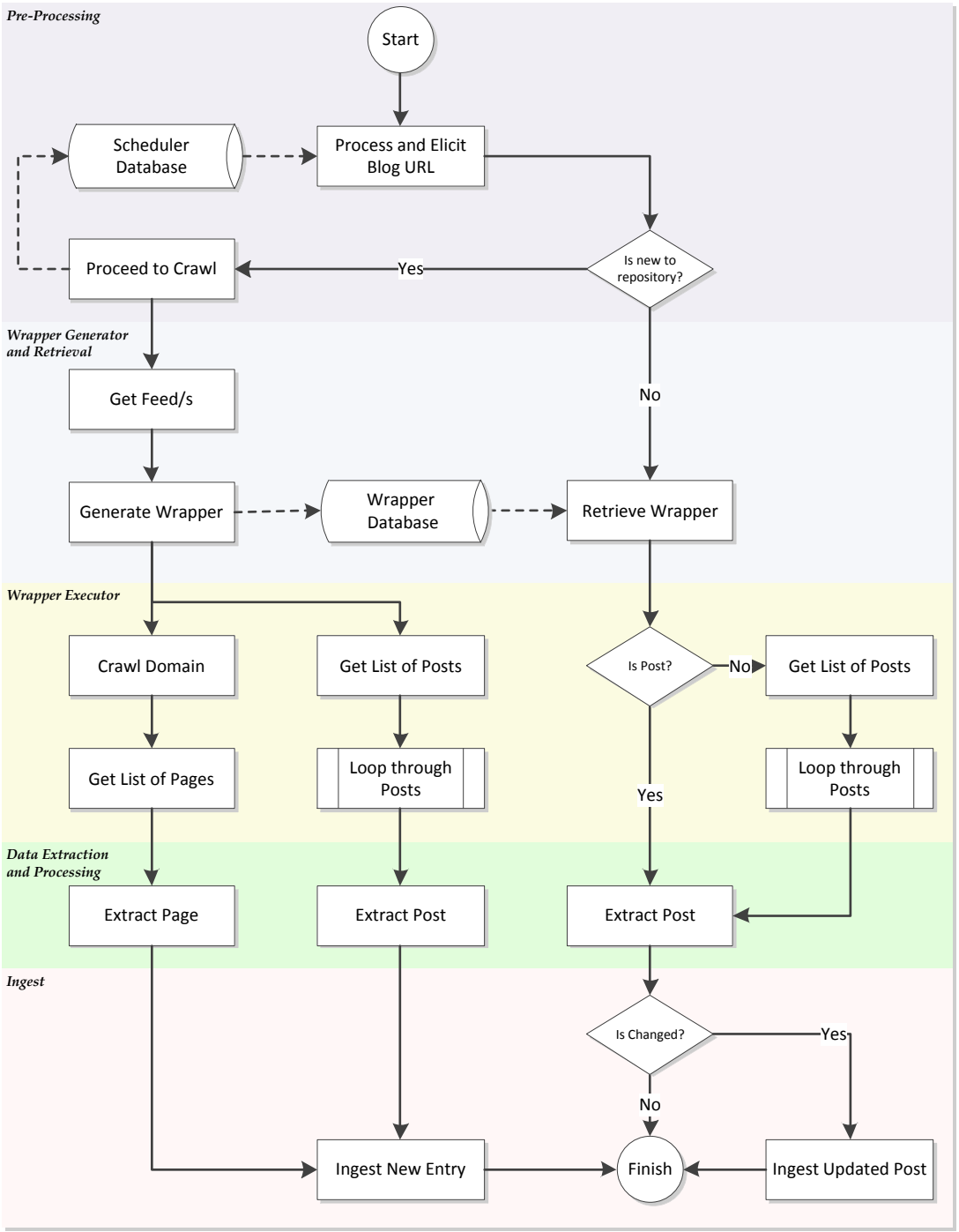

Figure 28: Wrapper-based data extraction workflow (*Note: Dashed arcs indicate data flow***)**

Some of the processes described in the flow diagram are represented at a higher and more abstract level, for instance, the process 'Generate Wrapper'. This is due to having more detailed description of this process presented in section 6.

The workflow, as presented above, branches depending on a number of conditional checks (depicted as diamonds). The first check looks for an available wrapper. The other branches determine the flows that enable both capturing and updating posts. The loops within the diagram illustrate the processes where more than one entry is subject to extraction.

The process, described in the flow diagram can be discussed in the context of the BlogForever spider design. Each of the described processes can be associated with the components of the data extraction. Depending on the design of the Spider component, the following broadly defined components (see [4]) can be involved in performing the task of data extraction: Worker (i.e. including Fetcher and Parser) and Scheduler. Where crawling policies are to be taken into account, the Scheduler can play a greater part for conducting the data extraction.

The process described in the diagram above as Generate Wrapper employs the use of web feeds. Due to similarities between the blog posts and blog pages the application of the wrappers generated on the basis of the web feeds may be sufficient. The performance of the wrappers generated using blog feeds remains to be evaluated when applied to page entries. Both posts and pages considered to be blog entries. There are a number of conceptual differences between the posts and the pages. For instance pages retain their static position within the blog, while post entries, when published in blogs that retain a reversed chronological order, are automatically pushed down with the publication of new posts. Yet, despite their differences, they share a range of structural similarities. They often follow same presentation rules defined by associated CSS. They share functional elements such as possibility of allowing commenting or sharing. Hence, there are indications for possible reuse of the wrappers. However, an empirical evaluation is necessary for providing a definitive answer. Hence, alternative solutions for automatically identifying the components of the page may also be necessary to consider. A possible alternative is to employ a solution similar to Boilperpipe[123]. The granularity of this approach may not be comparable to the extraction of posts. However, the use of Boilerpipe can be computationally less expensive. Otherwise, if greater level of granularity is required when capturing pages, alternative approaches that include unsupervised machine learning based on the exhibited pages of the particular blog might be necessary. The common challenge in employing such a method is defined by the complexity of identifying, aligning and labelling page components [45].

The evaluation of the automatic wrapper generation based on web feeds (see Section 6) demonstrates sufficient capabilities of performing data extraction without user intervention. However, some control mechanisms can further improve and optimise the process of data extraction. These mechanisms are justified by the possible (and at times frequent) changes within blogs that may hinder successful data extraction by using the earlier generated wrappers. Additionally, they can be used to enhance the process of data extraction by using human input where automatic methods fail. Hence, the control mechanisms can include a: [a] notification mechanism for detecting failure in data extraction; [b] attempt to automatic correction mechanism (e.g. re-generation of a wrapper); and [c] mechanisms for user intervention and manual correction.

Similarly to the general data extraction system architecture as described by Baumgartner [14] we introduce an architecture of a data extraction component of the spider. It encompasses the use of web feeds for automatic wrapper generation and extraction as well as a control unit and a user interface to enable automatic or manual correction (see Figure 29). The diagram is colour coded to propose a possible mapping of the data extraction architecture to the main components of the BlogForever Spider [4].

[123] http://code.google.com/p/boilerpipe/

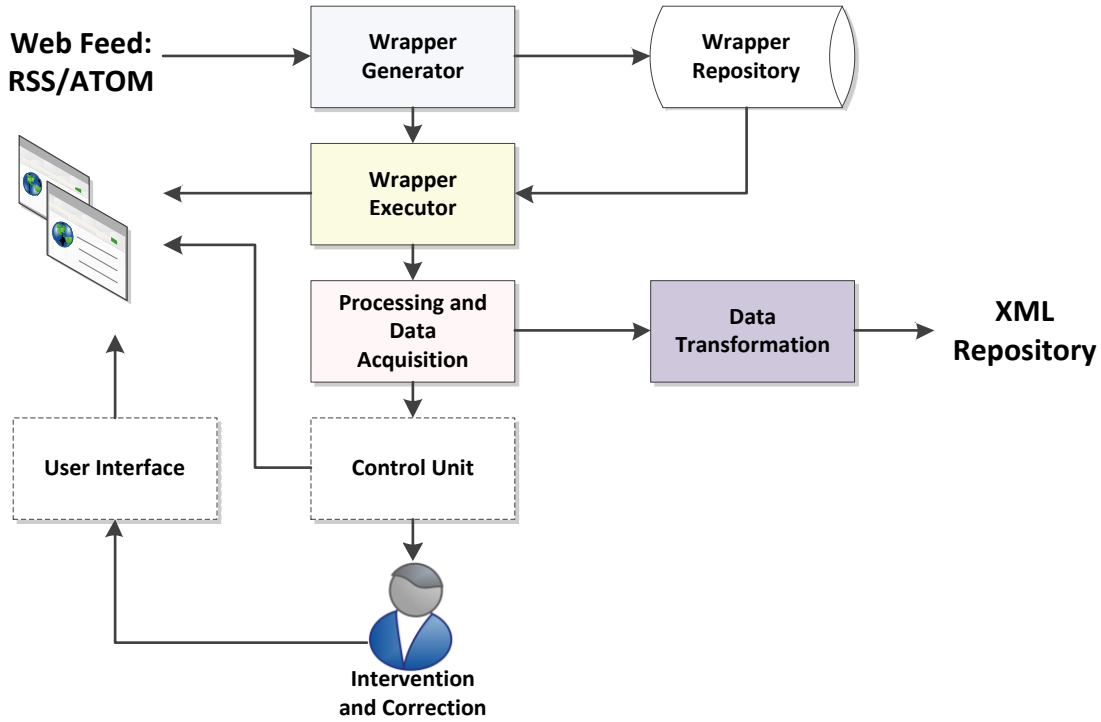

Figure 29: Data extraction component architecture

The Control Unit, as depicted above can include a notification mechanism triggered when data extraction task on a specific field of a blog fails. The notification can subsequently trigger the process for regenerating the wrapper (if necessary) or seek human input for resolving the failure. The User Interface can include a browser to render the blog and enable users to interact with the content. When the notifications of failed extraction are triggered the users can intervene (using the visual interface) for correcting extraction rules where necessary. The visual interface can be equipped with other features to support user interaction with the blog and the generation of a wrapper. The design and development of the control mechanisms can utilise available tools such as OXPath [100] and/or programmatic web browser libraries like HTMLUnit[124]. Ready-made solutions, however, as discussed in section 6, remain limited in their application within the blog domain. Hence, they should be customised or adapted to the needs of the BlogForever project.

In summary, this section consolidates the review of the related work and the results of the evaluation to propose a workflow of data extraction from blogs. It described possible actions that enabled discussing the process of data extraction from a more abstract level that can be easily associated with the use of the BlogForever platform.

7.4 Conclusions

This section consolidated the earlier discussions (Sections 3-6) and proposed directions for developing the BlogForever platform. Unlike the previous sections, it zoomed out to discuss the methodology at a higher level. This section provided examples directly related to data extraction from blogs and discussed these in the context of BlogForever. Unlike the former parts of this report this section focused exclusively on data extraction from blogs only. Namely, it summarised the processes for extracting and updating posts and pages from blogs. Additionally, this section made suggestions for data storage (section 7.2) and a workflow of the wrapper-based extraction flow (section 7.3). This higher level perspective, compliments the earlier reviewed methodologies and

[124] http://htmlunit.sourceforge.net/

approaches, but also, it demonstrates the relationships of the methodologies with the larger components of the BlogForever systems such as the spider and the repository system.

8 Post-Processing: Named Entity Recognition

In this report, we have already defined and discussed extensively the notion of data extraction from HTML or XML-based resources in order to extract data objects. Another important aspect of weblog data extraction methodologies is the post-processing of aggregated content. The fields of text analysis and computational linguistics have been developing along with the growing availability of Web data. Processing the aggregated content can be used to infer new knowledge from free-form text elements. Understanding the available opportunities for conducting post-processing can be informative for making decisions on further improvement of the BlogForever system. This section, in particular, discusses the concept of Named Entity Recognition and presents some prototype applications that implement this functionality. Last but not least, outcomes are discussed and suggestions are formulated for their implementation in the context of the BlogForever platform development.

8.1 Named Entity Recognition

Named Entity Recognition is the process of locating and classifying atomic elements in text into predefined categories such as names, persons, organizations, locations and dates. Extracting named entities can result not only to improving keyword search but also to empowering semantic search, faceted search, content clustering and document repurposing in general. Most research on NER systems has been structured as taking an unannotated block of text such as this one:

> *Jim bought 300 shares of Acme Corp. in 2006.*

And producing an annotated block of text, such as this one:

> *<ENAMEX TYPE="PERSON">Jim</ENAMEX>bought<NUMEX TYPE="QUANTITY">300</NUMEX>shares of<ENAMEX TYPE="ORGANIZATION">Acme Corp.</ENAMEX> in <TIMEX TYPE="DATE">2006</TIMEX>.*

NER systems have been created that use linguistic grammar-based techniques as well as statistical models. Hand-crafted grammar-based systems typically obtain better precision, but at the cost of lower recall and months of work by experienced computational linguists. Statistical NER systems typically require a large amount of manually annotated training data.

It is often stated that over 90% of the useful information in any organization is stored in unstructured text documents – not in structured relational databases [109]. We argue that this is also the case with blogs as most textual content is free text and the use of semantic markup is very low, a fact that is also confirmed by the BlogForever Survey [2, Section 5.2.4, p. 62]. We believe that Named Entity Recognition in the context of blogs would be an interesting and useful case. Towards this end, we have developed two alternative prototypes that confirm our claim.

8.2 Software Prototypes

In order to explore NER functionality in the context of BlogForever, two different software solutions were implemented using established named entity extraction software and services. The main idea behind these applications was to extract entities from common blog content and store them in a database in order to utilize them later on. Possible utilisation would be the association of entities with blog post and the establishment of links between posts based on the fact that they share common entities.

8.2.1 Stanford Named Entity Recognizer (NER)

Stanford NER (also known as CRFClassifier) is a Java implementation of a Named Entity Recognizer. Named Entity Recognition (NER) labels sequences of words in a text which are the names of things, such as person and company names, or gene and protein names. The software provides a general (arbitrary order) implementation of linear chain Conditional Random Field (CRF) sequence models, coupled with well-engineered feature extractors for Named Entity Recognition [110, 111]. Included in the core software distribution are different models, each one trained with different data sets. The broadest model supports 7 different classes that are shown in Table 22.

Table 22: Stanford NER supported named entities

Entity	***Example***
Time	15:33
Location	Washington
Organization	Aristotle University
Person	Barrack Obama
Money	53$
Percent	78%
Date	Monday 15th April 2010

For the purposes of evaluation, a Python application was developed in order to analyse blog content using Stanford NER and extract Person, Organization and Location entities. In order to evaluate the prototype we used the same dataset of 2.547 blog posts used in data extraction evaluation (Section 6.4) of this report. The prototype analyzed the dataset and stored results in a MySQL database. The database schema is shown at Figure 30.

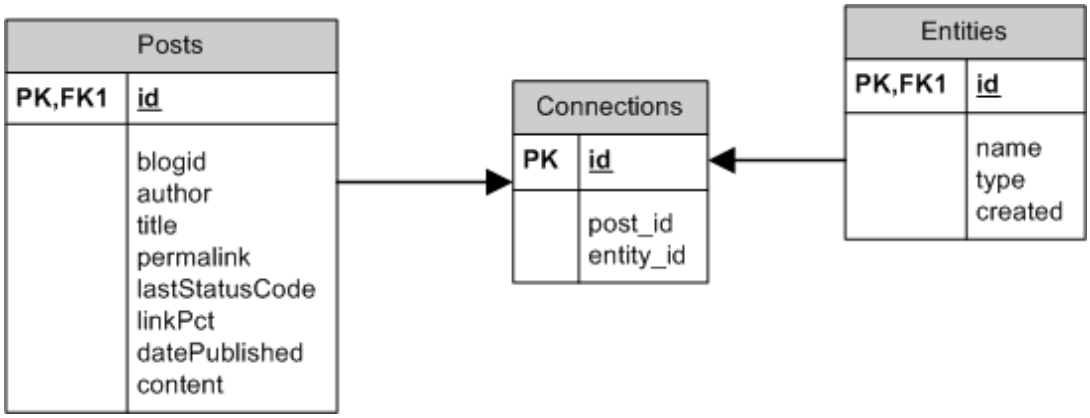

Figure 30, Prototype database schema

Results were evaluated using SQL. It is obvious that the entity extraction process is meaningful and important for the enrichment of the data. Many entities were identified (Table 23 and Table 24) and connected with blog posts.

Table 23: Entity occurrences

Entity Types	#
LOCATION	665

ORGANIZATION	1010
PERSON	2731
Total	**4401**

Table 24: Top 10 entities for each different type

Top Organizations	*#*	***Top Persons***	*#*	***Top Locations***	*#*
StumbleUpon	369	Obama	289	Reddit	290
Digg	314	Baloch	57	Pakistan	167
United	113	Jesus	56	America	148
DC	58	John	48	Seattle	105
Google	58	Brossman	45	London	102
Congress	47	Michael	43	Japan	98
House	37	Bush	35	U.S.	89
Senate	32	Martin	34	Balochistan	88
Microsoft	30	Moore	34	Wordpress.com	82
ACP	27	Sarah	32	China	82

Evaluating Stanford NER precision with sample blog content from our dataset, the following named entities were identified:

> Montlake has a neighborhood news site -- Meet the Montlake Over the years, CHS has told a story or two from Montlake. We're connected to our neighbors -- in many interesting ways. For those of you who have followed our effort to gather news and information around Capitol Hill know, we take the job (mostly) seriously and with a sense of pride powered by a competitive nature to be the best. That doesn't mean there isn't room for others to tell the stories, too -- especially when they are dedicated and do a good job of it. So welcome Montlaker to the mix of telling the stories of Seattle. New neighborhood news sites are fragile things. Many have come and gone. Here's wishing good luck to http://montlaker.com. We look forward to working with you -- and, occasionally, kicking your ass on a story. You can also follow along via Twitter with @montlaker. Here's what the person behind Montlaker told CHS about their hopes for the site: One of the major reasons I wanted to start this project is to give the neighborhood a voice in the new world of hyper local media. The Sound Transit tunnel noise debacle made clear to me that? Montlake is not so well represented online and often ends up tagged as NIBMY-land. The label is sometimes justified and sometime not, and a neighborhood blog at least is a chance to make the case?
>
> Named Entities: LOCATION ORGANIZATION PERSON MISC

Last but not least, it must be noted that one of the advantages of Stanford NER is its open source nature. Anyone is free to use it or modify it according to one's needs. For instance, University of Heidelberg, Germany has developed German Named Entity Recognition Models http://www.nlpado.de/~sebastian/software/ner_german.shtml.

8.2.2 OpenCalais

OpenCalais is a free web service provided by company Thomson Reuters[125] since 2008. OpenCalais web service reads unstructured text and return RDF documents identifying specific entities within the text. The full list of support entities is listed in Table 23. OpenCalais uses natural language processing technologies and supports English, French and Spanish. Unfortunately, due to the closed nature, no information is available regarding the algorithms and technology used.

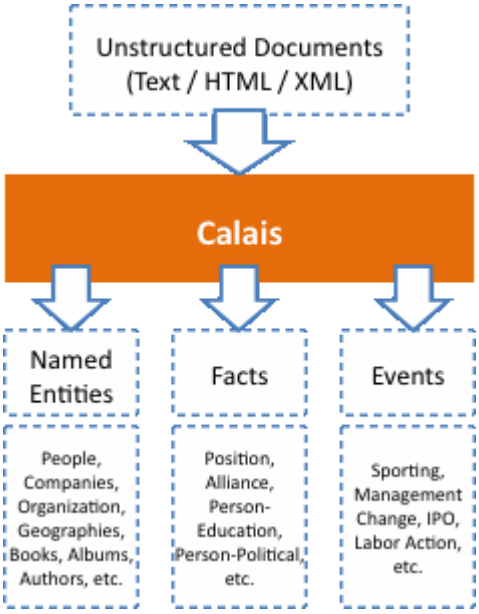

Figure 31: OpenCalais overview

Table 25: OpenCalais supported named entities

Entity	***Example***
City	Sacramento
Company	Dow Jones & Company, Inc
Country	Greece
EntertainmentAwardEvent	The Oscars
Holiday	Memorial day
MarketIndex	S&P
MedicalCondition	Pancreatic cancer
MedicalTreatment	acupuncture
Movie	Seven
MusicAlbum	Wish you were here
MusicGroup	Pink Floyd
OperatingSystem	Debian
ProgrammingLanguage	Python
ProvinceOrState	Saskatchewan, Canada

[125] http://thomsonreuters.com/

PublishedMedium	The Boston Globe
RadioStation	WBUR
SportsEvent	The FIFA World Cup
SportsGame	Basketball
SportsLeague	Scottish Football Alliance
TVShow	The X-Files
TVStation	WGBH

In order to evaluate OpenCalais, we implemented a Python application that reads blog content, detects entities and stores them in a MySQL database. To evaluate the prototype, we used the same dataset of 2.547 blog posts used in data extraction evaluation (Section 6.4) of this report. . The prototype analyzed the dataset and stored results in a MySQL database. The database schema is shown at Figure 30.

Table 26: Entity occurrences

Entity type	**#**
Person	968
Position	351
Organization	221
IndustryTerm	220
Company	149
Facility	120
City	92
Movie	65
URL	65
Country	56
ProvinceOrState	46
MedicalCondition	44
Technology	42
SportsEvent	38
NaturalFeature	32
Product	28
PublishedMedium	21
EntertainmentAwardEvent	17
Holiday	13
MusicGroup	13
Region	12
PhoneNumber	10

MusicAlbum	8
Continent	7
EmailAddress	7
Currency	7
OperatingSystem	6
PoliticalEvent	6
SportsLeague	5
SportsGame	5
MedicalTreatment	5
RadioStation	3
TVShow	3
TVStation	2
FaxNumber	1
Total	**2.688**

Result data were evaluated using SQL. Again, entity extraction results were plentiful, 2.688 entities were identified, spanning across 34 different entity types.

Table 27: Top 10 entities occurrences

Entity	***Type***	#
Twitter	Company	59
Facebook	Company	50
USD	Currency	48
United States	Country	27
Wordpress.com	URL	17
America	Continent	17
Afghanistan	Country	16
pence	Currency	15
president	Position	14
United Kingdom	Country	14

Evaluating OpenCalais precision with the same sample blog content as in the previous subsection, the following named entities were identified:

> Montlake has a neighborhood news site -- Meet the Montlake Over the years, CHS has told a story or two from Montlake. We're connected to our neighbors -- in many interesting ways. For those of you who have followed our effort to gather news and information around Capitol Hill know, we take the job (mostly) seriously and with a sense of pride powered by a competitive nature to be the best. That doesn't mean there isn't room for others to tell the stories, too -- especially when they are dedicated and do a good job of it. So welcome Montlaker to the mix of telling the stories of Seattle. New neighborhood news sites are

> fragile things. Many have come and gone. Here's wishing good luck to http://montlaker.com. We look forward to working with you -- and, occasionally, kicking your ass on a story. You can also follow along via Twitter with @montlaker. Here's what the person behind Montlaker told CHS about their hopes for the site: One of the major reasons I wanted to start this project is to give the neighborhood a voice in the new world of hyper local media. The Sound Transit tunnel noise debacle made clear to me that? Montlake is not so well represented online and often ends up tagged as NIBMY-land. The label is sometimes justified and sometime not, and a neighborhood blog at least is a chance to make the case?
>
> Named Entities: URL, IndustryTerm, Person, Company, City

Last but not least, OpenCalais has some limitations such as limited language support (only English, French and Spanish) and a quota of 50.000 transactions per day with a frequency of maximum 4 transactions per second.

8.2.2.1 DBPedia Spotlight

OpenCalais is one of many entity extraction web services on the web such as Veeb[126] and Alchemy API[127]. The problem with most cases is that these services are provided by commercial companies and are not free. Nevertheless there is a new high quality web service by DBpedia[128] aiming to fill this gap. DBpedia Spotlight [129] is a tool for automatically annotating mentions of DBPedia resources in text, providing a solution for linking unstructured information sources to the Linked Open Data cloud through DBpedia. DBpedia Spotlight performs named entity extraction, including entity detection and Name Resolution. It can also be used for building your solution for Named Entity Recognition, Recognition amongst other information extraction tasks.

Text annotation has the potential of enhancing a wide range of applications, including search, faceted browsing and navigation. By connecting text documents with DBpedia, Spotlight enables a range of interesting use cases. For instance, the ontology can be used as background knowledge to display complementary information on web pages or to enhance information retrieval tasks. Moreover, faceted browsing over documents and customization of web feeds based on semantics become feasible. Finally, by following links from DBpedia into other data sources, the Linked Open Data cloud is pulled closer to the Web of Documents.

The following example showcases basic DBPedia spotlight functionality. The user accesses the web service URL with the following parameters:

- **text**= "President Obama called Wednesday on Congress to extend a tax break for students included in last year's economic stimulus package, arguing that the policy provides more generous assistance."
- **confidence** = 0.2; **support**=20
- whitelist all types.

http://spotlight.dbpedia.org/rest/annotate?text=President%20Obama%20called%20Wednesday%20on%20Congress%20to%20extend%20a%20tax%20break%20for%20students%20included%20in%20last%20year%27s%20economic%20stimulus%20package,%20arguing%20that%20the%20policy%20provides%20more%20generous%20assistance.&confidence=0.2&support=20

The output is an HTML page were named entities were identified and linked to DBPedia URLs.

[126] http://www.veeeb.com/
[127] http://www.alchemyapi.com/
[128] http://dbpedia.org
[129] http://dbpedia.org/spotlight

> President Obama called Wednesday on Congress to extend a tax break for students included in last year's economic stimulus package, arguing that the policy provides more generous assistance.

Evaluating OpenCalais precision with the same sample blog content as in the previous two subsections, the following named entities were identified:

> Montlake has a neighborhood news site -- Meet the Montlake Over the years, CHS has told a story or two from Montlake. We're connected to our neighbors -- in many interesting ways. For those of you who have followed our effort to gather news and information around Capitol Hill know, we take the job (mostly) seriously and with a sense of pride powered by a competitive nature to be the best. That doesn't mean there isn't room for others to tell the stories, too -- especially when they are dedicated and do a good job of it. So welcome Montlaker to the mix of telling the stories of Seattle. New neighborhood news sites are fragile things. Many have come and gone. Here's wishing good luck to http://montlaker.com. We look forward to working with you -- and, occasionally, kicking your ass on a story. You can also follow along via Twitter with @montlaker. Here's what the person behind Montlaker told CHS about their hopes for the site: One of the major reasons I wanted to start this project is to give the neighborhood a voice in the new world of hyper local media. The Sound Transit tunnel noise debacle made clear to me that? Montlake is not so well represented online and often ends up tagged as NIBMY-land. The label is sometimes justified and sometime not, and a neighborhood blog at least is a chance to make the case?

The significant difference between DBPedia Spotlight and the other named entity retrieval solutions presented in this report is that DBPedia Spotlight does not just classify named entities into categories, it also connects identified items with DBPedia entities, providing richer information, metadata and associations with other entities.

8.3 Conclusions and Considerations Regarding the BlogForever Platform

The research conducted regarding named entity recognition highlights the fact that it is feasible to identify named entities in blog text content using a variety of tools. This is very important because although there is an abundance of content on the blogosphere, the vast majority is not annotated. Therefore, the application of named entity detection would increase content accessibility and suggest new relationships between blog content. Our evaluation shows that there are mature commercial and open source solutions ready to be used towards this goal with minimal effort. The question is how to take advantage of this technology in order to maximize benefits. Currently, we have identified the following cases where named entity extraction would be useful in the context of BlogForever:

1. **Automatic tagging**. Aggregated blog content may not have correct metadata (if any) for a variety of reasons. For instance, the blog author did not provide tags at all or the blog spider did not aggregate tags and metadata correctly. Using NER technology we will be able to generate automatically this information from unstructured blog text with high levels of accuracy and performance.
2. **Enable semantic search.** Semantic Search systems consider various points including context of search, location and intent, variations of words, synonyms, generalized and specialized queries, concept matching and natural language queries to provide relevant search results. Using NER we will have fulfilled some of the requirements to enable semantic search.
3. **Link content.** Blog content (posts, comments or other entities) which does not have any connections to another can be assumed as interlinked as long as it shares the same named entity. This way, new connections will be created automatically, enriching the repository and providing foundations for further knowledge discovery.

9 Discussion and Conclusions

This section aims to summarise the findings described in this report and discuss future work related to the task of data extraction within the context of the BlogForever project. The review conducted as part of this report highlighted the complexity of this area and diversity of possible approaches for data extraction. The continuous innovation and rapid growth of the Web makes the area of data extraction highly dynamic. The recent developments towards the semantic web and linked data are most promising in their potential for changing the traditional approaches to data extraction. The increasing adoption of semantic annotation on the Web is opening a wide range of possibilities for accessing, extracting and processing machine-readable data. However, the amount of annotated data remains disproportionately small. The data extraction approaches considered for the BlogForever project should extend beyond the most novel approaches and integrate approaches that enable data extraction from the resources that use more traditional technologies.

As discussed in the early sections of this report, the task of data extraction is challenging. It often requires adoption of machine learning approaches and is computationally hard and expensive. Furthermore, at times, data extraction may require human intervention and lacks scalability. We have provided the necessary background that grounded this report into the established body of related work. We progressed by discussing the possibilities of employing some of the most relevant approaches taken for web data extraction and commented on the use of tools developed alongside these approaches. Prior to proposing the methodology, the report discussed the differences and similarities of blogs from other web resources. It referred to the structure of blogs as discussed in the earlier deliverable [3, Section 9, p. 45], which describes the developed blog data model (see Section 2). The report positioned the inquiry within the established areas of research and practice and defined the boundaries by defining the tasks of data extraction, the anticipated pre-processing and potential post-processing (see Section 3). The report discussed the fundamental techniques and tools associated with data extraction related to parsing and crawling (see Section 4).

In addition to the theoretical inquiry, the report included two case studies. The first case study reported the experience of working with dynamically generated website powered by MediaWiki and discussed possible solutions for avoiding similar challenges when working with blogs (see Section 5).

The second case study reported the benefits and disadvantages of using web feeds for data extraction from blogs (see Section 6). Building on the covered work the report proceeded to propose and test a relevant methodology for data extraction from blogs. This part of the report proposed the use of web feeds to enable unsupervised data extraction from blogs. It followed a mixed strategy of exploiting both feeds and web content. The report outlined the developed prototype and evaluated the results of the proposed approach. Finally, the section continues with a discussion that addresses data extraction from semantically annotated web resources such as microformats and microdata and its consideration for the BlogForever platform.

Section 7 of the report provided a high level overview of the workflow of the BlogForever data extraction process that incorporates the proposed methodology. To achieve this, a set of processes for capturing and maintaining blog entries were outlined and described in detail. The processes were interlinked in order to describe a step-by-step summary of a proposed workflow for data extraction. Furthermore, various architectural components related to both, the spider and the repository, were presented and discussed in the context of BlogForever.

Last, but not least, the report discusses the post-processing of the extracted and aggregated data (see Section 8). The rationale of post-processing the collected data lies in the possibilities of inferring knowledge from the free-form text collected from blogs. The report introduced the notion of Named Entity Recognition (NER) and discussed the prototyped software that enabled commenting on the potential of NER within the BlogForever context. As a result, this report provided a summary of

fundamental technologies and informed directions for implementing the task of data extraction from blogs as part of the BlogForever platform.

The primary direction for conducting future work in relation data extraction remains tightly aligned to monitoring the development of new approaches and tools for automatic data extraction. This is particularly important due to rapid achievements in publishing and reuse of Linked Data and the exponential update of semantic technologies such as Microdata and RDFa. It is also necessary to follow other data extraction initiatives to further refine and improve the performance of data extraction mechanisms proposed for adoption for the BlogForever project. Initiatives developed as part of the DIADEM project, Webdam or GATE remain to be closely followed. The adoption of rapidly emerging semantic web tools and technologies, such as Any23, are also to be evaluated for their adoption within the Blogosphere and the BlogForever project in particular. Finally, the short-term plan is to work into further develop and improve the prototype. While the evaluation of the proposed extraction methodology shows that it is indeed innovative, applicable and of potential interest to the research community, some limitations were identified. In the upcoming tasks, the various technical and theoretical issues will be examined and addressed more thoroughly.

10 References

[1] H. Kalb, N. Kasioumis, J. García Llopis, S. Postaci, and S. Arango-Docio, "BlogForever: D4.1 User Requirements and Platform Specifications Report", Technische Universität Berlin2011.

[2] S. Arango-Docio, P. Sleeman, and H. Kalb, "BlogForever: D2.1 Survey Implementation Report", 2011, Available: http://blogforever.eu/deliverables/.

[3] K. Stepanyan, M. Joy, A. Cristea, Y. Kim, E. Pinsent, and S. Kopidaki, "D2.2 Report: BlogForever Data Model", 2011, Available: http://blogforever.eu/wp-content/uploads/2011/11/BlogForever_D2.2WeblogDataModel.pdf.

[4] M. Rynning, V. Banos, K. Stepanyan, M. Joy, and M. Gulliksen, "BlogForever: D2.4 Weblog spider prototype and associated methodology", 2011, Available: http://blogforever.eu/wp-content/uploads/2011/11/BlogForever_D2_4_WeblogSpiderPrototypeAndAssociatedMethodology.pdf.

[5] Y. Kim and S. Ross, "BlogForever: D2.5 Weblog spam filtering report and associated methodology", 2012.

[6] B. A. Nardi, D. J. Schiano, M. Gumbrecht, and L. Swartz, "Why we blog," *Communications of the ACM,* vol. 47, pp. 41-46, 2004.

[7] P. Pluempavarn and N. Panteli, "Building social identity through blogging", in *Exploring virtuality within and beyond organizations: social, global, and local dimensions*, N. Panteli and M. Chiasson, Eds., ed New York, USA: Palgrave Macmillan, 2008, pp. 195-212.

[8] M. Garden, "Defining blog: A fool's errand or a necessary undertaking," *Journalism,* pp. 1-17, 20 September 2011 2011.

[9] J. Saddington. (2011). *The Ultimate List of Blogging Platforms, Blog Software, Free and Paid (100+!)*. Available: http://tentblogger.com/blogging-platforms/, Accessed: 25.04.2012

[10] W. Shields. (2011). *Why does every man and his dog want to code a blogging engine?* Available: http://stackoverflow.com/questions/471940/why-does-every-man-and-his-dog-want-to-code-a-blogging-engine, Accessed: 25.04.2012

[11] K. Patowary. (2009). *Blogging without a database: 7 database-less Content Management Systems (CMS)*. Available: http://www.instantfundas.com/2009/09/blogging-without-database-7-database.html, Accessed: 25.04.2012

[12] M. D. Dikaiakos, A. Stassopoulou, and L. Papageorgiou, "An investigation of web crawler behavior: characterization and metrics," *Computer Communications,* vol. 28, pp. 880-897, 2005.

[13] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical machine learning tools and techniques*, Second Edition ed.: Morgan Kaufmann, 2005.

[14] R. Baumgartner, W. Gatterbauer, and G. Gottlob, "Web data extraction system," *Encyclopedia of Database Systems,* pp. 3465-3471, 2009.

[15] H. Chen and M. Chau, "Web mining: Machine learning for Web applications," *Annual review of information science and technology,* vol. 38, pp. 289-330, 2004.

[16] B. Liu, *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*: Springer London, Limited, 2011.

[17] S. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *FRONTIERS IN ARTIFICIAL INTELLIGENCE AND APPLICATIONS,* vol. 160, p. 3, 2007.

[18] J. Han, M. Kamber, and J. Pei, *Data mining: concepts and techniques*: Morgan Kaufmann, 2011.

[19] M. Ester, H. P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise", 1996, pp. 226-231.

[20] W. Wang, J. Yang, and R. Muntz, "STING: A statistical information grid approach to spatial data mining", in *International Conference on Very Large Data Bases (VLDB'97)*, Athens, Greece, 1997, pp. 186–195.
[21] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE,* vol. 78, pp. 1464-1480, 1995.
[22] X. Qi and B. D. Davison, "Web page classification: Features and algorithms," *ACM Comput. Surv.,* vol. 41, pp. 1-31, 2009.
[23] M. Álvarez, A. Pan, J. Raposo, F. Bellas, and F. Cacheda, "Using clustering and edit distance techniques for automatic web data extraction", 2007, pp. 212-224.
[24] C. H. Chang, C. N. Hsu, and S. C. Lui, "Automatic information extraction from semi-structured Web pages by pattern discovery," *Decision Support Systems,* vol. 35, pp. 129-147, 2003.
[25] E. Ferrara, G. Fiumara, and R. Baumgartner, "Web Data Extraction, Applications and Techniques: A Survey", Tech. Report2010.
[26] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. Da Silva, and J. S. Teixeira, "A brief survey of web data extraction tools," *ACM Sigmod Record,* vol. 31, pp. 84-93, 2002.
[27] N. Kushmerick, "Wrapper induction: Efficiency and expressiveness," *Artificial Intelligence,* vol. 118, pp. 15-68, 2000.
[28] I. Muslea, S. Minton, and C. A. Knoblock, "Hierarchical wrapper induction for semistructured information sources," *Autonomous Agents and Multi-Agent Systems,* vol. 4, pp. 93-114, 2001.
[29] C. N. Hsu and M. T. Dung, "Generating finite-state transducers for semi-structured data extraction from the web," *Information Systems,* vol. 23, pp. 521-538, 1998.
[30] D. Ciravegna, "Adaptive information extraction from text by rule induction and generalisation," 2001.
[31] J. Turmo and H. Rodriguez, "Learning rules for information extraction," *Natural Language Engineering,* vol. 8, pp. 167-191, 2002.
[32] J. Turmo, A. Ageno, and N. Català, "Adaptive information extraction," *ACM Computing Surveys (CSUR),* vol. 38, p. 4, 2006.
[33] S. Sarawagi, "Information extraction," *Foundations and trends in databases,* vol. 1, pp. 261-377, 2008.
[34] A. J. Sellers, T. Furche, G. Gottlob, G. Grasso, and C. Schallhart, "OXPath: little language, little memory, great value", 2011, pp. 261-264.
[35] A. Hogue and D. Karger, "Thresher: automating the unwrapping of semantic content from the World Wide Web", 2005, pp. 86-95.
[36] K. Lerman and C. A. Knoblock, "Wrapper maintenance," *Encyclopedia of Database Systems,* pp. 3565-3569, 2008.
[37] N. Dalvi, P. Bohannon, and F. Sha, "Robust web extraction: an approach based on a probabilistic tree-edit model", 2009, pp. 335-348.
[38] B. Chidlovskii, B. Roustant, and M. Brette, "Documentum eci self-repairing wrappers: Performance analysis", 2006, pp. 708-717.
[39] K. Lerman, S. Minton, and C. A. Knoblock, "Wrapper maintenance: A machine learning approach," *J. Artif. Intell. Res. (JAIR),* vol. 18, pp. 149-181, 2003.
[40] E. Ferrara and R. Baumgartner, "Automatic wrapper adaptation by tree edit distance matching," *Combinations of Intelligent Methods and Applications,* pp. 41-54, 2011.
[41] R. McCann, B. AlShebli, Q. Le, H. Nguyen, L. Vu, and A. H. Doan, "Mapping maintenance for data integration systems", 2005, pp. 1018-1029.
[42] Y. Zhai and B. Liu, "Extracting web data using instance-based learning," *Web Information Systems Engineering–WISE 2005,* pp. 318-331, 2005.
[43] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu, "Fully automatic wrapper generation for search engines", 2005, pp. 66-75.
[44] M. Bronzi, V. Crescenzi, P. Merialdo, and P. Papotti, "Wrapper Generation for Overlapping Web Sources", 2011, pp. 32-35.
[45] G. Miao, J. Tatemura, W. P. Hsiung, A. Sawires, and L. E. Moser, "Extracting data records from the web using tag path clustering", 2009, pp. 981-990.

[46] Y. Liu, K. Bai, P. Mitra, and C. L. Giles, "Tableseer: automatic table metadata extraction and searching in digital libraries", 2007, pp. 91-100.
[47] L. Ramaswamy, L. Liu, and F. Douglis, "Automatic fragment detection in dynamic web pages and its impact on caching," *Knowledge and Data Engineering, IEEE Transactions on,* vol. 17, pp. 859-874, 2005.
[48] F. Fumarola, T. Weninger, R. Barber, D. Malerba, and J. Han, "Extracting general lists from web documents: A hybrid approach," *Modern Approaches in Applied Intelligence,* pp. 285-294, 2011.
[49] C. X. Lin, B. Zhao, T. Weninger, J. Han, and B. Liu, "Entity relation discovery from web tables and links", in *World Wide Web 2010*, Raleigh, North Carolina, USA, 2010, pp. 1145-1146.
[50] J. L. Hong, E. G. Siew, and S. Egerton, "Information extraction for search engines using fast heuristic techniques," *Data & Knowledge Engineering,* vol. 69, pp. 169-196, 2010.
[51] V. Crescenzi and G. Mecca, "Automatic information extraction from large websites," *Journal of the ACM (JACM),* vol. 51, pp. 731-779, 2004.
[52] V. Crescenzi, G. Mecca, and P. Merialdo, "Roadrunner: Towards automatic data extraction from large web sites", 2001, pp. 109-118.
[53] L. Liu, C. Pu, and W. Han, "An XML-enabled data extraction toolkit for web sources," *Information Systems,* vol. 26, pp. 563-583, 2001.
[54] A. Sahuguet and F. Azavant, "Building light-weight wrappers for legacy web data-sources using W4F", 1999, pp. 738-741.
[55] X. A. Boronat, "A comparison of HTML-aware tools for Web Data extraction," *lipsinformatikunileipzigde,* 2008.
[56] C. Olston and M. Najork, *Web Crawling*: Now Publishers Inc., 2010.
[57] T. Nanno, T. Fujiki, Y. Suzuki, and M. Okumura, "Automatically collecting, monitoring, and mining japanese weblogs", 2004, pp. 320-321.
[58] P. Kolari, T. Finin, and A. Joshi, "SVMs for the blogosphere: Blog identification and splog detection", 2006, p. 1.
[59] M. Ceglowski. (2003). *Identify blogging tools based on URL and content*. Available: http://search.cpan.org/~mceglows/WWW-Blog-Identify-0.06/Identify.pm, Accessed: 10 April 2011
[60] E. Elgersma and M. De Rijke, "Learning to recognize blogs: A preliminary exploration," *NEW TEXT Wikis and blogs and other dynamic text sources,* p. 24, 2006.
[61] D. C. Fetterly and S. S. T. Chien, "Identifying a web page as belonging to a blog", ed: Google Patents, 2009.
[62] D. Winer. (2011). *RSS auto-discovery*. Available: http://scripting.com/stories/2011/12/17/rssAutodiscovery.html, Accessed: 13.04.2012
[63] S. Nowson, "The Language of Weblogs: A study of genre and individual differences," 2006.
[64] H. Qu, A. Pietra, and S. Poon, "Automated blog classification: Challenges and pitfalls", 2006, pp. 184-185.
[65] B. Hammersley, *Developing feeds with RSS and Atom*: O'Reilly, 2005.
[66] R. Davis, "Archiving scientific blogs with ArchivePress", 2009, p. 2010.
[67] R. Baumgartner, S. Flesca, and G. Gottlob, "Visual web information extraction with Lixto", 2001, pp. 119-128.
[68] M. Pennock and R. Davis, "ArchivePress: A really simple solution to archiving blog content", in *iPress 2009*, San Francisco, California, USA, 2009.
[69] N. McIntosh. (2003). *Google buys Blogger web service*. Available: http://www.guardian.co.uk/business/2003/feb/18/digitalmedia.citynews, Accessed: 15.05.2012
[70] C. Rusbridge. (2009). *Preservation for scholarly blogs*. Available: http://www.gavinbaker.com/2009/03/30/preservation-for-scholarly-blogs/, Accessed: 01.05.2012
[71] L. Liu, C. Pu, and W. Han, "XWRAP: An extensible wrapper construction system for internet information", in *16th International Conference on Data Engineering*, San Diego, California, USA, 2000.

[72] P. Geibel, O. Pustylnikov, A. Mehler, H. Gust, and K. U. Kühnberger, "Classification of documents based on the structure of their dom trees", 2008, pp. 779-788.
[73] K. Darlington, *The essence of expert systems*: Prentice Hall, 2000.
[74] J. R. Quinlan, "Induction of decision trees," *Machine learning,* vol. 1, pp. 81-106, 1986.
[75] J. R. Quinlan, "Discovering rules by induction from large collections of examples," *Expert systems in the micro electronic age,* vol. 174, 1979.
[76] J. Quinlan, "Improved Use of Continuous Attributes in C4. 5," *Journal of Artificial Intelligence Research,* vol. 4, pp. 77-90, 1996.
[77] S. Ihara, *Information theory for continuous systems* vol. 2: World Scientific Pub Co Inc, 1993.
[78] K. Giles, K. M. Bryson, and Q. Weng, "Comparison of two families of entropy-based classification measures with and without feature selection", 2001, p. 10 pp.
[79] G. Navarro, "A guided tour to approximate string matching," *ACM Computing Surveys (CSUR),* vol. 33, pp. 31-88, 2001.
[80] W. E. Winkler, "Overview of record linkage and current research directions", 2006.
[81] L. Yujian and L. Bo, "A normalized Levenshtein distance metric," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 29, pp. 1091-1095, 2007.
[82] W. E. Winkler, "String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage," 1990.
[83] M. A. Jaro, "Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida," *Journal of the American Statistical Association,* pp. 414-420, 1989.
[84] M. Oita and P. Senellart, "Archiving data objects using web feeds," 2010.
[85] E. Cohen and B. Krishnamurthy, "A short walk in the Blogistan," *Computer Networks,* vol. 50, pp. 615-630, 2006.
[86] E. Z. F. Liu, R. C. Shih, and Y. L. Tsai, "Hyperlink network analysis of the educational blog," *British Journal of Educational Technology,* vol. 42, pp. E25-E29, 2011.
[87] H. Mühleisen and C. Bizer, "Web Data Commons–Extracting Structured Data from Two Large Web Corpora", presented at the World Wide Web, WWW'2012, Lyon, France, 2012.
[88] P. Mika. (2011). *Microformats and RDFa deployment across the Web*. Available: http://tripletalk.wordpress.com/2011/01/25/rdfa-deployment-across-the-web/, Accessed: 15.05.2012
[89] C. Bizer. (2012). *Topology of the Web of Data*. Available: http://www.wiwiss.fu-berlin.de/en/institute/pwo/bizer/research/publications/Bizer-TopologyWoD-LWDM2012-BEWEB2012.pdf, Accessed:
[90] I. Hickson. (2012). *HTML Microdata*. Available: http://www.w3.org/TR/html5/spec.html, Accessed: 15.05.2012
[91] B. Adida and M. Birbeck. (2008). *RDFa Primer*. Available: http://www.w3.org/TR/xhtml-rdfa-primer/, Accessed: 15.05.2012
[92] H. L. Kim, A. Passant, J. G. Breslin, S. Scerri, and S. Decker, "Review and alignment of tag ontologies for semantically-linked data in collaborative tagging spaces", 2008, pp. 315-322.
[93] W3C. (2012). *HTML5: A vocabulary and associated APIs for HTML and XHTML*. Available: http://dev.w3.org/html5/spec/index.html, Accessed: 08 May 2012
[94] V. Banos, K. Stepanyan, M. Joy, A. I. Cristea, and Y. Manolopoulos, "Technological foundations of the current Blogosphere", in *International Conference on Web Intelligence, Mining and Semantics - WIMS'12*, June 13-15, 2012 Craiova, Romania, 2012.
[95] N. Guarino, D. Oberle, and S. Staab, "What is an Ontology?," *Handbook on ontologies,* pp. 1-17, 2009.
[96] D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, D. W. Lonsdale, Y. K. Ng, and R. D. Smith, "Conceptual-model-based data extraction from multiple-record Web pages," *Data & Knowledge Engineering,* vol. 31, pp. 227-251, 1999.
[97] P. Mika, "Ontologies are us: A unified model of social networks and semantics," *Web Semantics: Science, Services and Agents on the World Wide Web,* vol. 5, pp. 5-15, 2007.
[98] M. Tanaka and T. Ishida, "Ontology extraction from tables on the Web", 2006, p. 7 pp.

[99] C. Bizer. (2012). *Panel Discussion. Microdata, RDFa, Web APIs, Linked Data: Competing or Complementary. (Ivan Herman, Peter Mika, Tim Berners-Lee, Yves Raimond)*. Available: http://events.linkeddata.org/ldow2012/slides/Bizer-LDOW2012-Panel-Background-Statistics.pdf, Accessed: 01.05.2012

[100] T. Furche, G. Gottlob, X. Guo, C. Schallhart, A. Sellers, and C. Wang, "How the minotaur turned into ariadne: ontologies in web data extraction," *Web Engineering,* pp. 13-27, 2011.

[101] T. Furche, G. Gottlob, G. Grasso, Ö. Gunes, X. Guo, A. Kravchenko, G. Orsi, C. Schallhart, A. Sellers, and C. Wang, "DIADEM: Domain-centric, Intelligent, Automated Data Extraction Methodology", in *WWW'12: World Wide Web 2012*, Lyon, France, 2012.

[102] W. Su, J. Wang, and F. H. Lochovsky, "ODE: Ontology-assisted data extraction," *ACM Transactions on Database Systems (TODS),* vol. 34, p. 12, 2009.

[103] H. Roitman and A. Gal, "Ontobuilder: Fully automatic extraction and consolidation of ontologies from web sources using sequence semantics," *Current Trends in Database Technology–EDBT 2006,* pp. 573-576, 2006.

[104] Y. A. Tijerino, D. W. Embley, D. W. Lonsdale, Y. Ding, and G. Nagy, "Towards ontology generation from tables," *World Wide Web,* vol. 8, pp. 261-285, 2005.

[105] M. Rowe, "Mapping between digital identity ontologies through sism", 2009.

[106] K. Sigurðsson, "Incremental crawling with Heritrix," *Proc. IWAW,* 2005.

[107] C. Kohlschütter, P. Fankhauser, and W. Nejdl, "Boilerplate detection using shallow text features", 2010, pp. 441-450.

[108] X. Zheng, T. Zhou, Z. Yu, and D. Chen, "URL Rule based focused crawler", 2008, pp. 147-154.

[109] W. McKnight, "Building business intelligence: text data mining in business intelligence," *DM Review,* pp. 21-22, 2005.

[110] C. Sutton and A. McCallum, *An introduction to conditional random fields for relational learning*: Introduction to statistical relational learning. MIT Press, 2006.

[111] J. R. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling", 2005, pp. 363-370.

11 Appendices

11.1 Appendix A: Use of Regular Expressions in the Web Curator Tool

In the Target's profile override section in Web Curator Tool, we have the option of specifying exclusions and inclusions. These work in sequence:

1. First exclusions can be used to exclude URLs from the harvest by specifying a set of perl-compatible regular expressions (one per line).
2. Second, inclusions can be applied that reverse any exclusions, again using perl-compatible regular expressions.

For example, to exclude URLs that contain postID= we would use the exclusion override:

.*postID=.*

The simple explanation of how this works is that a dot (i.e. period or full-stop) , means "any character" and an asterisk (or "star") means "repeated zero or more times". We call the sequence .* a "dot-star" at it means "any letter repeated zero or more times" or "matches anything".

It's likely that the filter specifications are Regular Expressions (Using .* to mean "a string of 0 or more of any characters" is a giveaway). That means that a plain dot . does not mean a dot, it means "any character". So, in theory, the regex ".gov.uk" doesn't only match the exact string ".gov.uk", but also "Agovluke", and "$gov%uke", and so on. If you want to precisely match a dot, we have to escape it with a backslash, so your regex would be .*\.gov\.uk.*
-- or more correctly still: https?://[^/]+\.gov\.uk/.* which means "http", maybe "s", "://", one or more characters that are not slashes, the string ".gov.uk", a slash, and then 0 or more characters. That ensures it only matches at the end of the hostname part, not elsewhere and not in the path (eg http://www.gov.uk.com, or http://www.foo.com/by_domain/gov.uk/)

We also need to be aware of the characters that have special meaning in regexes. For example, ".*?" does not mean "any number of characters followed by a ?", it means "maybe any number of characters". Again, the question mark needs to be escaped..."*\?"

11.2 Appendix B: Matching between strings

Table 28: The table with the string values used to perform an assessment of the different string matching measurements.

ID	String
1	Warwick is located on the outskirts of Coventry, 5.5 km (3.4 mi) southwest of the city centre (and not in the town of Warwick as its name suggests). The university's main site comprises three contiguous campuses, all within walking distance of each other. The university also owns a site in Wellesbourne, acquired in 2004 when it merged with Horticulture Research International.
2	Warwick is located on the outskirts of Coventry, 5.5 km (3.4 mi) southwest of the city centre (and not in the town of Warwick as its name suggests). The university's main site comprises three contiguous campuses, all within walking distance of each other. The university also owns a site in Wellesbourne, acquired in 2004 when it merged with Horticulture Research
3	June 20, 2011
4	The Expected End of Moore's Law is Good News for Computer Science
5	It seems to be bon-ton these days to talk about the end of Moore's law. While originally stated for the number of transistors stored on a given area of silicon, it has been quickly extended to CPU clock speed and performance as well. In general, for a long time, the number of transistors in a CPU and its clock speed used to double every 1.5 to 2 years. All this was going on until around 5 years ago. Due to laws of physics, the heat and energy grow roughly cubically in the clock speed and so increasing the clock speed above 3 GHz seemed impractical. As a result, rather than increasing the clock speed, chip manufacturers opted for increasing the number of cores in a CPU, thereby increasing the theoretical performance by adding parallelism rather than by increasing the clock speed.
6	It seems to be bon-ton these days to talk about the end of Moore's law. While originally stated for the number of transistors stored on a given area of silicon, it has been quickly extended to CPU clock speed and performance as well. In general, for a long time, the number of transistors in a CPU and its clock speed used to double every 1.5 to 2 years. All this was going on until around 5 years ago. Due to laws of physics, the heat and energy grow roughly cubically in the clock speed and so increasing the clock speed above 3 GHz seemed impractical. As a result, rather than increasing the clock speed, chip manufacturers opted for increasing the number of cores in a CPU, thereby increasing the theoretical performance by adding parallelism rather than by increasing the clock speed. Yet, even this trend may soon be put to a serious challenge, as the distances between transistors quickly become smaller than the minimal required by CMOS technology. All this is great news for computer science in my opinion. For a long time, people got used to being lazy. If computers become twice as fast every 1.5 to 2 years, there is no point in investing much efforts in writing efficient code. If something does not run fast enough, simply wait for the next generation of Intel x86 and everything will be resolved. In particular, CPUs became fast enough that traditional programming languages and efficient data structures and algorithms were being abandoned in favor of high level scripting languages whose most sophisticated data structure is an associative array. Suddenly, every Joe-hoe could become a programmer developing sophisticated Web applications with no effort – no need for hard earned computer science degrees anymore. All these could change back with the end of Moore's law. As CPUs become parallel, programmers need to learn how to write parallel code and deal with all the intricacies of concurrent execution. They need to understand how the system executes their code, dealing with memory consistency issues, avoiding synchronization in order to facilitate parallelism etc. There is suddenly great demand for innovation in compiler technology for automatically parallelizing sequential programs.

	Programming models are suddenly an important topic again. Data structure libraries need to be parallelized in an efficient and scalable manner. Operating systems must be redesigned and re-architected to make an effective use of the many cores that are put at their dispense. Moreover, as the number of transistors might be reaching a limit, this means that even the number of cores on a CPU will likely be limited. Also, given Ahmdel's law, the maximal benefit from parallelism is in any case quite limited. Hence, writing efficient code will suddenly become important again. For this, strong background in computer science is a must!
7	Technology and Society
8	Nice post. Can quantum computer change this again

Table 29: The values of the different string matching measurements of strings of Table 28.

StringA	StringB	JaroWinklerDistance [82]	Levenshtein	Normalised Levenshtein [81]	d_{sum}	d_{max}	d_{min}
1	2	0,012	14	0,037	0,019	0,037	0,038
1	3	0,619	370	0,972	0,946	0,979	28,462
1	4	0,488	332	0,857	0,749	0,878	5,108
1	5	0,355	595	0,675	0,510	0,754	1,574
1	6	0,459	2538	0,881	0,787	0,891	6,714
1	7	0,473	360	0,947	0,900	0,952	16,364
1	8	0,454	336	0,881	0,787	0,889	6,857
2	3	0,618	356	0,971	0,944	0,978	27,385
2	4	0,486	318	0,851	0,741	0,874	4,892
2	5	0,355	600	0,685	0,520	0,760	1,648
2	6	0,460	2545	0,884	0,793	0,894	6,992
2	7	0,472	346	0,945	0,896	0,951	15,727
2	8	0,453	323	0,878	0,782	0,887	6,592
3	4	0,627	61	0,878	0,782	0,938	4,692
3	5	0,583	782	0,987	0,975	0,991	60,154
3	6	0,509	2839	0,996	0,993	0,997	218,385
3	7	0,587	19	0,704	0,543	0,864	1,462
3	8	0,638	44	0,830	0,710	0,898	3,385
4	5	0,500	736	0,926	0,862	0,933	11,323
4	6	0,508	2789	0,978	0,958	0,980	42,908
4	7	0,496	51	0,739	0,586	0,785	2,318
4	8	0,487	53	0,635	0,465	0,815	1,082
5	6	0,241	2058	0,723	0,566	0,723	2,608
5	7	0,505	769	0,973	0,948	0,975	34,955
5	8	0,480	744	0,941	0,888	0,943	15,184
6	**7**	0,50	2826	0,99	0,99	0,99	128,45
6	**8**	0,49	2799	0,98	0,97	0,98	57,12

7	**8**	0,51	39	0,71	0,55	0,80	1,77

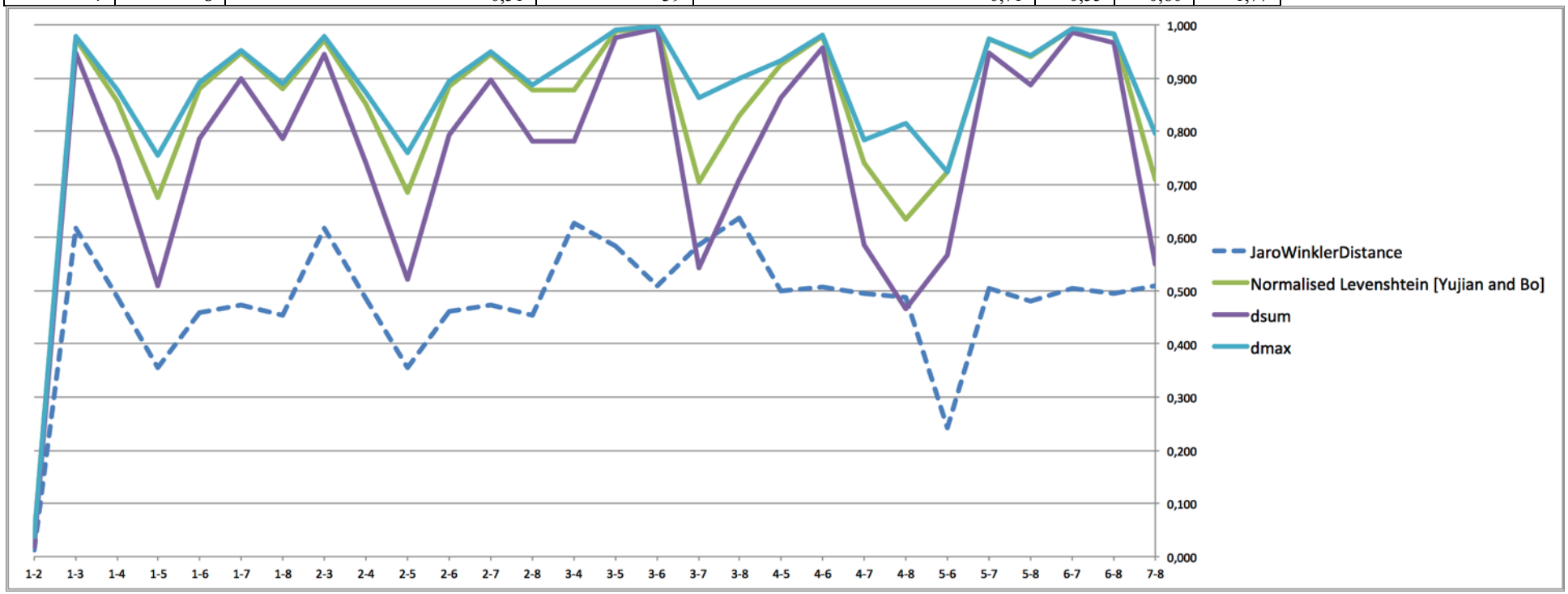

Figure 32: The graph of Table 29.

11.3 Appendix C: Software prototype

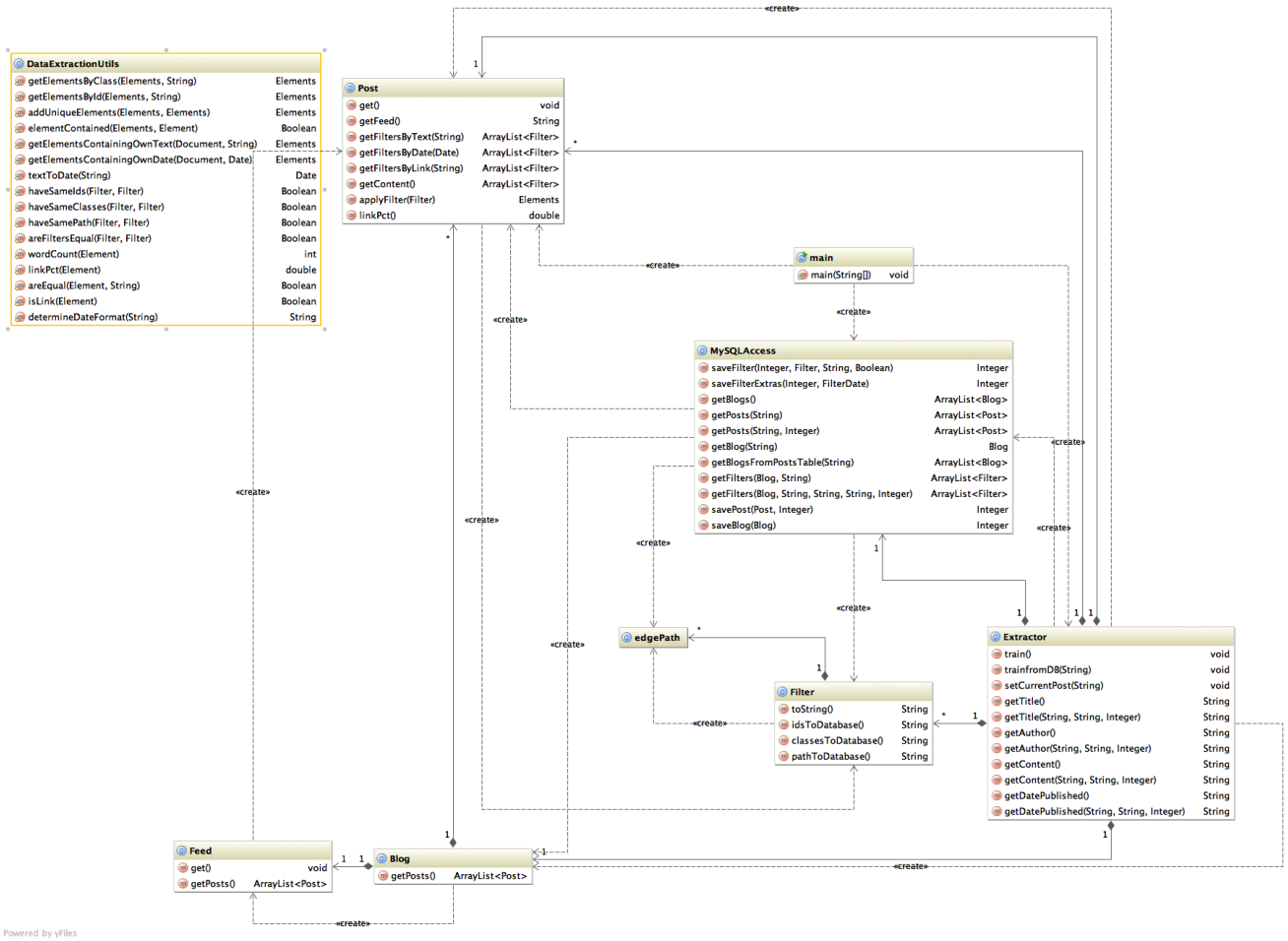

Figure 33: Extended UML Class diagram of the software prototype