# Archiving Data Objects using Web Feeds*

Marilena Oita
INRIA Saclay – Île-de-France
& Télécom ParisTech; CNRS LTCI
Paris, France
marilena.oita@telecom-paristech.fr

Pierre Senellart
Institut Télécom
Télécom ParisTech; CNRS LTCI
Paris, France
pierre.senellart@telecom-paristech.fr

## ABSTRACT

In this paper, we show how Web feeds can be used to archive Web pages that contain temporal data objects, such as blog posts or news items. We use RSS or Atom feeds to extract these Web objects and to detect change in the context of an incremental crawl. We first describe some statistics on Web feeds, by studying the evolution of a collection of feeds for a period of time and observing their temporal aspects. For detecting change on crawled Web pages that have a Web feed associated, we present an algorithm that extracts the information of interest (the data object), with the aim of analyzing changes effectively, without being tricked by possible changes in the surrounding boilerplate. Our algorithm applies a bottom-up strategy on the HTML DOM tree and uses $n$-grams extracted from the title and the description of a feed item to match conceptual leaf nodes in the HTML page. These conceptual nodes will be clustered in function of their lowest block-level common ancestor. The resulting block-level nodes will correspond to semantic zones in the Web page, and by taking the one that is the most semantically dense, the algorithm identifies the node that acts like a wrapper for the article. We extract then the textual content and the references of the article from this node and encapsulate the result in a timely unit. Experiments are done in order to validate our approach, with good results even for pages for which the extraction of the main content is a challenge to other techniques. We finally discuss important applications based on the extraction and change detection of Web objects.

## General Terms

Experiments, Algorithm

## Keywords

Web archiving, data object, Web feed, change detection

*This research was funded by the European Research Council grant Webdam FP7-ICT-226513.


*IWAW 2010* September 2010, Vienna, Austria


## 1. INTRODUCTION

Web archiving [10] is the process of repeatedly collecting the content of parts of the World Wide Web, to ensure its preservation and to allow access to the information even after it disappears from the Web. A Web archiving crawler follows the same basic steps as a search engine crawler does to construct indexes for Web pages. However, an archive crawler does not drop indexes to old versions when new ones are discovered, but instead stores and references versions in time. The final result is a collection of Web pages that can be browsed off-line and, in ideal settings, can as well be temporally and semantically queried.

Performing a coherent crawl of sites that are updating their supply of information very rapidly it is not an easy task. The common choice of crawling in snapshots, i.e., crawling the entire collection at distant (but frequent enough) intervals of time, can be quite expensive in terms of network bandwidth, and in the end, both redundant for some zones of the site, and incomplete for other zones that are more dynamic. Additionally, as running an integral crawl takes time and in the meanwhile the resource might change, it raises issues of temporal incoherence of a given snapshot [18].

What "frequent enough" means is also to be clarified when one decides to perform an incremental crawl instead, which is better adapted to sites that have temporally heterogeneous parts in their structure. An incremental crawler will crawl once a full version of the site and, from that point, supposing the existence of a trigger that informs the crawler which content of a site has been added or updated, it will crawl only the modified content and store it as a delta data structure. The difficulty of performing an incremental crawl is essentially to determine this dynamics (how often new Web pages are added or current ones are modified), and to make this change detection effective, knowing that there are a lot of factors that can wander the focus of the detection process.

Our contribution is mainly related to techniques that can be used to improve this detection process, in the particular case where the Web pages crawled have semantic and temporal information attached to them, in the form of RSS or Atom Web feeds. We also propose a method that can be used to attach more semantics to Web archives, through the extraction of *data objects* (e.g., blog posts, new items) from Web pages.

The traditional way of detecting changes between two successive versions of a Web page is to perform a comparison on content, using a similarity metrics, varying from hashes, content signatures to edit distances for flexibility. Whatever the way the change is detected, the importance of this

change (regarding the types of content of a Web page) is not evaluated. Nevertheless, it is crucial to understand whether changes are relevant to the main content of the Web page from a semantic point of view, since, in some applications, changes that only impact *boilerplate* [7] parts of the page, such as menus, presentation templates, or advertisements, might be safely ignored.

Web feeds have been poorly studied, yet are a rapidly evolving phenomenon. We draw attention to the fact that they can be used as instruments in the analysis of a Web site before and during the crawl. In addition to being a way to advertise content, Web feeds are also used to classify sources of information and their types of content by search engines. In a nutshell, from Web feeds important aspects of a dynamic website can be extracted and exploited in the context of a Web crawl in order to make it more aware of the information it treats.

The nature of a feed is:

**informative:** it synthesizes what and when fresh information is published by a channel;

**descriptive:** it describes what kind of new resources are added, with the title, description, and other possible *tag elements*.

We aim at extracting, with the help of feeds, structured data from Web pages. The basic intuition behind our approach is that an item of a channel matches a data object in a Web page. Therefore, the metadata that we can get about the item in the feed can be used to recognize and extract the data object in the Web page.

The notion of data object has different interpretations in computer science; in order to clarify its signification, in our context a data object is an instance of a resource referenced by a feed, and which has some special properties [8].

Even if the object is often a Web article, it can also be a dictionary entry, a comment, a forum message, a video, a status, and any other kind of resources that are uniquely associated with Web feed items. Direct approaches to identifying the main content of a Web page, such as taking the elements containing the largest amount of information from the HTML code after cleaning, studying the density of the text in specific regions of the page [7], or even identifying the most prominent visual areas of the page [21], are not applicable here, since data objects can very well correspond to small, simple, portions of a page, with one or multiple data objects per page, depending on the context.

We leverage the semantics that is brought by Web feeds to dynamic Web pages. Written in a XML-like form, with standard elements, feeds can be used to capture some important aspects of the information that we want to extract. Moreover, a feed captures the dynamics of a data object, and reveals its nature. We get from the structure of a feed item, using classical information retrieval techniques, a semantic descriptor of the object that will be used as an input for the extraction algorithm. Next, we identify the zone in a Web page that contains the data object and extract it by means of semantic density analysis. Having extracted the content in time and its associated components, complex queries can be run, from the temporal and semantic point of view. Value-added applications can be imagined for users and archivists, services that could exploit the semantics and temporality of the data objects extracted and incrementally crawled.

We present in the next section some related work, and describe in Section 3 the results of our studies on Web feeds done in order to determine the value of these feeds in the Web archiving process. We explain in Section 4 how the semantics revealed by specific elements in a feed can be used to extract data objects. In Section 5, experiments relative to the extraction of data objects are shown. We conclude by outlining the importance of feed timestamps and extracted data objects in the context of an incremental crawl.

## 2. RELATED WORK

Web archiving, seen as a necessity or duty, is acquiring a lot of importance lately due to the volatile nature of the Web, and more specifically due to the value that the lost information could have for future generations. Internet Archive [6] is one of the initiators of the Web archive movement. In spite of the fact that there are many other actors that actively run crawls as part of their heritage preservation mission, the collective effort is not convergent into a unified, global Web archive collection.

Though Web feeds are typically indexed as other kinds of Web documents by archive crawlers, limited effort has been made on exploiting their specificity in the archiving process. ArchivePress, a blog-archiving project [14] has been developing a Wordpress plugin that archives posts using Web feeds. The principal drawback is that only the content that can be delivered by the RSS feed is captured. In effect, an RSS feed can have a full coverage of the article and its media files, but this case is quite rare because a feed is often just a way of advertising content. The essential difference with our approach is that we harness the feed clues with the sole goal of identifying semantic and temporal attributes of posts and then use this feed data to extract the actual objects associated. Additionally, we do not limit ourselves to a blogging platform in particular.

The extraction of data objects is meant to be useful in the task of Web page change detection. In order to detect and store only the modified content of a Web page, the open-source Heritrix archive crawler [16] uses heuristics and regular expressions to filter irrelevant changes, in the attempt to fairly estimate the Web page change rate. A more formal model of prediction is studied by Cho and Garcia-Molina in [2], where the authors study whether changes of a Web page follow a Poisson process, and estimate the change rate using this model. However, accurately identify the change rate is stated by both approaches as a challenge. In contrast with our method, none of these techniques takes into account the semantic of the content that is to be crawled.

Even if we consider the change rate issue solved, the problem of clearly identifying the fresh content published from a version to another remains. Supposing they have acquired two successive versions by using some technique for detecting Web page change, Pehlivan, Ben Saad, and Gançarski [13] focus on detecting the changes that have occurred in the new version, based on the old one. For that purpose, the VIPS [21] algorithm is used to identify semantically related parts of a Web page, which are compared in order to detect structural and content changes. Heuristics on the visual appearance of a Web page are made to group together content that seems to have similar importance in the page. These heuristics do not always work or they are not sufficient, and the algorithm is computationally expensive. We restrict ourselves to the context in which feed information is available and, in this

case, are able to identify the data that has been added or updated. Moreover, we focus our attention on changes that occur in the area that matches the content of the Web feed item, without digression to other zones in the Web page.

We can identify the data object by using the feeds, but still we have to extract it from the HTML code of the associated Web page. Much work has been done on unsupervised extraction of structured data from Web pages; many of these are based on MDR [9], ExAlg [1] or RoadRunner [3]. Basically, these methods try to automatically generate a wrapper by inferring a grammar for the HTML code that contains the content of interest (generally called data records), in such a way that it does not rely on any a priori knowledge about the target pages and their contents. Usually, various pages that have the same template are needed in order to compare them in pairs and discover shared patterns and code regularities. These regularities are discovered either by studying the similarities and dissimilarities between pages [3], or by constructing classes of equivalence [1]. As opposed to the previous works, MDR [9] considers the DOM tree structure of the HTML page, and identifies the data region by finding the node that contains the largest number of children presenting the same patterns according to a similarity measure. Even though we are concentrating our attention on sites that contain Web articles, that are not necessarily consecutive in the way they appear in the Web page, we have a similar problem of extracting structured data (possibly of a more complex and isolated nature than in these approaches) and, as described in Section 4, we use somehow related techniques.

Two other approaches to the problem of extracting the main article from a Web page have been proposed recently by Kholschutter, Fankhauser, and Nejdi [7] and Pasternack and Roth [12]. While [7] uses the text density on the page to identify the data region, [12] uses a subsequence segmentation technique on the text in the page to the same effect. Our algorithm of data object extraction uses, similarly to MDR, some heuristics on the HTML code in order to identify the region where the article might be found, but in contrast with all enumerated approaches, we use additional semantics retrieved from the feed to extract the data objects. This is obviously possible only for pages who are linked to a Web feed. In Section 5, we compare our results to those of the Boilerpipe [7] algorithm, seen as a baseline of what can be done without exploiting Web feed information. To our knowledge, there is no previous work that leverages semantic information about the data objects present in a Web page (information that may come from a Web feed or from any other source) to extract the relevant portion of the Web page, and this portion only, in a general and unsupervised manner.

# 3. WEB FEEDS: STATISTICS AND TRENDS

In order to affirm the value of feeds as analysis tools in the process of Web change detection, we have crawled over a period of a little more than one month, twice a day, a number of 400 Web feeds with all associated Web pages. We first describe how these feeds were selected and then report some statistics of interest of our dataset.

## 3.1 Acquisition

The set of feeds was collected from the Web by passing in large part through a feed search engine called Search4RSS [15]. This search engine returns a number of feeds and associated channel pages for a given keyword. We scraped the way the interface returns the results as records and put all URLs parsed in a file list to be further analyzed.

**Table 1: Dataset feed types**

| Type | Number | Proportion |
|---|---|---|
| Atom | 21 | 6.1% |
| RDF | 30 | 8.8% |
| RSS 0.91 | 1 | 0.2% |
| RSS 2.0 | 288 | 84.7% |
| **Total** | 340 | 100.0% |

The keywords chosen in order to probe the search interface are names of domains: *art*, *biology*, *environment*, *medicine*, *science* and *universe*. We have used Wordnet in order to get hyponyms of these words (for example for art, a hyponym is photography) and constructed a bag of terms representing subdomains.

These bags of terms were used to automatically probe the Search4RSS interface and to construct, for each domain, a list of feed URLs. Semantics of terms permitted us to focus the search for feeds and to identify Web pages that were treating a certain subject. The reflection in the media of domains of interest can be observed through the eyes of feeds captured in this way, that may present some specific patterns and characteristics.

The purpose of this selection was to get an insight into the diversity of feeds, in terms of formats, update patterns, and varied structures of the corresponding Web pages. Additionally, to ensure coverage of common blog platforms as well as the more news-oriented results returned by Search4RSS, a number of blog sites were manually selected from a list of "best blogs" [17]. We obtained thus a list of about 400 sites that were systematically crawled (this number ensures coverage of the variety of Web feeds, while remaining manageable without any involved archiving infrastructure). At the end of the crawling period, we noticed that some of the feeds had not been updated at all, some others had disappeared, and some could not be parsed. Filtering them out, we obtained an archive of 340 *active* feeds and their associated pages.

A Web feed refers to a primary Web page, the *channel*, which is usually either the home page of the site, or a hub from which we can find links to information presented in the form of Web articles. The rest of the feed describes individual feed items corresponding to new or updated articles. For each domain, for each followed site, we have stored the feed and the resources associated, mainly the channel page, and the Web pages that were referred to by each item. In order not to lose new items that could be added in the feed, these crawls were performed twice a day (in addition to the crawls of the same feeds run daily by European Web Archive).

## 3.2 Web feed characteristics

For the feed analysis, we have used Eddie [4], a feed parsing library for Java, based on a SAX-based parser capable of parsing even (some of) the real-world ill-formed XML. Eddie supports the standard RSS, Atom, and RDF formats for feeds. The `FeedData` structure returned by this parser can be exploited to extract all kind of useful information about the channel and the composing items. In particular, for the

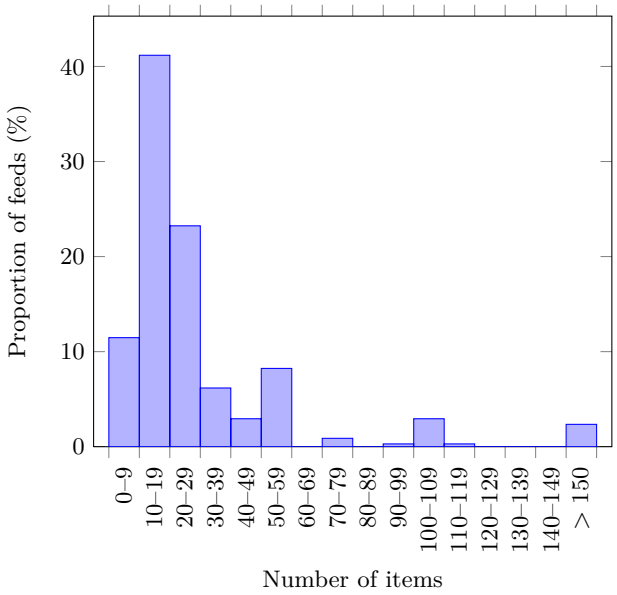


**Figure 1: Number of items per feed in the dataset**

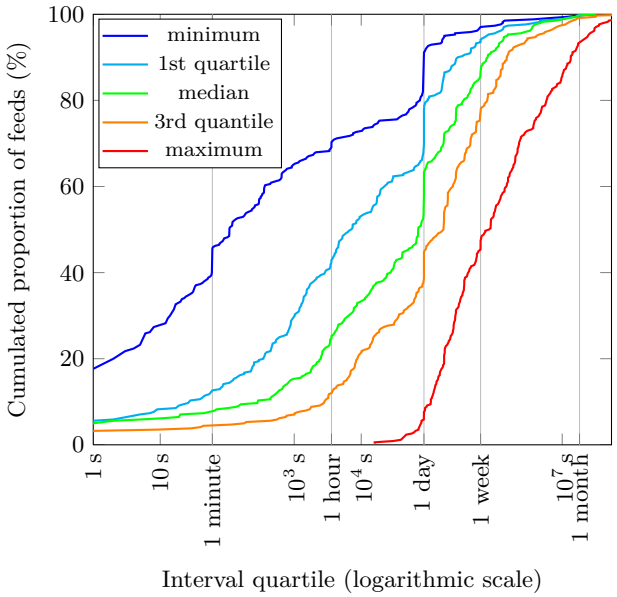


**Figure 2: Cumulated proportion of feeds with a given quartile value of interval between updates**

channel, metadata like language, tagline, description and title; as well for each item, plus author and categories in which the article is classed.

*Feed type.* Let us first look at the types of feed formats that we encountered in our dataset. As shown on Table 1, most feeds use the 2.0 dialect of RSS, while a minority use Atom or RDF. RSS 0.91 was only used once among the 340 feeds, and RSS 1.0 never at all, which might suggest the coming obsolescence of these two feed formats. However, it is quite possible that these numbers are also biased by the use of Search4RSS as our main source for feeds.

*Number of items.* We have looked at the number of items that were presented in a given feed. Indeed, though it is theoretically possible for a feed to refer to all previously published items, it is rarely done so to limit the size of the resulting Web feed. In effect, most feeds are truncated and present only the $k$ most recent items for a given $k$. We show in Figure 1 a histogram of the number of items per feed in the dataset. Roughly 75% of the feeds present information about less than 30 items at a time. The other peaks observed in Figure 1 are explained by the "magic" values of $k = 50$ and $k = 100$. If a feed only contains 10 items (the most frequent number), it means that by crawling it twice daily, we could capture a maximum of 20 new articles per day. As we shall see, there is a minority of feeds with a higher update frequency than that, for which some of the updates were missed in our crawl (basically, these feeds needed to be crawled more often than twice a day).

*Temporal information.* In the RSS specification (and similarly in other feed formats) temporal information can be given through the elements `lastBuildDate`, `ttl`, and `updateFrequency` for the channel, and `pubDate` and `lastModified` for items. From our experiments, we have observed that although pubDate is an optional element, it is present in the vast majority of feeds. This is not the case for the other types of time-related elements mentioned above, though `lastBuildDate` can be somehow inferred as the publication date of the most recent item. This observation is important as it shows that feeds can be used to determine when new data is added to a channel and of help in the task of change detection. By doing the analysis of a feed for a period of time, we can identify patterns in the publishing strategy and automatically adapt the crawl to it.

*Update intervals.* We have collected all publication dates corresponding to the items appearing during the period of experiments; as an item has one publication date, the number of publication dates is equal to that of items. We are interested in the range of update intervals between two publications, as well as indications whether a feed has a regular publishing strategy. In Figure 2, we present the median update interval between two publications of each feed as a cumulative plot (in green, in the middle). Note that the x-axis has logarithmic scale. Figure 2 shows for instance that 20% of the feeds have a median update interval of less than an hour, and that around 10% of the feeds have a median update interval of exactly one day, which corresponds to feeds having regular, automatic, updates, every day. Globally, it is important to note that there is no typical update interval, and that, even disregarding extreme cases, it can range to less than a hour to more than a week. Figure 2 also shows other quantities of the update interval of each feed, which helps to see the diversity of update patterns for a given feed: thus, even though 60% of the feeds have a median update interval of a day or less, less than 10% of them always have at least an update per day, and more than 90% of them have at least had a daily update in the period of observation. There is actually a big gap between the median update interval values and minimum and maximum values, which makes difficult to predict the next update of a given feed.

Table 2: Feed statistics per domain

| Domain | Number of feeds | Average mean update interval | Pooled standard derivation of update interval |
|---|---|---|---|
| Art | 87 | 12 days, 14 hours, 12 min | 82 days, 6 hours, 32 min |
| Biology | 80 | 7 days, 13 min | 8 days, 17 hours, 43 min |
| Blogs | 29 | 15 hours, 35 min | 8 hours, 39 min |
| Environment | 7 | 19 hours, 49 min | 4 days, 15 hours, 18 min |
| Medicine | 8 | 3 days, 19 hours, 16 min | 1 day , 22 hours, 43 min |
| Other | 13 | 4 days, 16 hours, 48 min | 4 days, 19 hours, 46 min |
| Science | 112 | 22 days, 12 hours, 45 min | 14 days, 21 hours, 35 min |
| Universe | 4 | 4 hours, 44 min | 7 hours, 5 min |
| **Total** | 340 | 12 days, 15 hours, 17 min | 37 days, 16 hours, 49 min |

We show in Table 2 some other statistics on update intervals at the level of domains. For each domain, the average mean update interval is given, as is the *pooled standard deviation* of update intervals, which is the statistically-founded way to summarize the deviation of the union of series of numbers. As can be seen, there are huge variations among domains, another indication of the absence of typical update interval. We also note here the very high standard deviations in some domains[1], especially since a given domain can represent Web sites of very different natures (news items, blogs, wiki entries, etc.). In the art domain, for instance, we have observed that there are several sites that publish small articles about paintings or photographs of the order of 100 entries per day. So the notion of item varies from a specialized article, that might contain a lot of text (as it is the case for news), to an article exclusively composed of images or videos. The more structurally homogeneous category of "popular" blogs has a more reasonable deviation of update intervals.

We conclude here the study of the dataset, that reflects in a certain measure the status-quo and diversity of Web feeds, to turn to the discussion to our technique of Web data objects extraction.

# 4. DATA OBJECT EXTRACTION

We describe in this section our algorithm for finding the data object in a given Web page, object that corresponds to a Web feed item.

## 4.1 Gathering semantic information

We first construct a semantic context from feed items, that will serve to extract the corresponding object. Feed items have three compulsory elements: link, title and description.

### *Link.*

The link gives us the URL of the Web page in which the data object resides.

### *Title.*

A feed item is required to have a title, which is usually a short textual description. This title will appear also as the title of the data object. To identify the data object, trying to search for the exact same title in the Web page can work, but the effectiveness of this technique is limited, as it represents a naïve approach, especially when the title is quite short or the actual sequence of words can be found also in other zones of the Web page.

[1]Note that it is not impossible for a standard deviation to be much greater than a mean value, it just means that there are a few values much greater than the mean in the distribution.

### *Description.*

The item's description can be of great help when the title, for various reasons, is not enough. Most of the time, the description does not contain the whole content of the data object, but only the first few lines of the article, encoded in HTML for presentation purposes, with a link at the end ("Read more...") to the original article. In other cases, the description contains only a sentence, which summarizes the article rather than containing the first lines of it. Whatever the precise case, we can extract some reliable knowledge about the data object by exploiting this description.

We start by retrieving all textual content from the title and description; the HTML code of the description is stripped, only text is kept. The result has the form of sequences of words that will be used to build the semantic context of a feed item.

The sequences of words are transformed next into two kinds of semantic entities: concepts and $n$-grams.

To obtain the concepts, we tokenize and stem the words, sort the resulting lexemes according to their frequency (seen as a measure of their importance) and keep only outstanding ones. In a sense, a concept will resemble a tag, being a term that describes a data object. We also retain the terms that correspond to these concepts (or to a part of them, if their number is greater than the necessary for extraction), in the precise way they appear in the title and description. These terms coming from concepts can indeed be used as keywords to look for in a Web page, in order to identify the data object region, but can frequently digress the attention to other zones that are also rich in concepts, like the ones that contain comments or tags for a given news article.

This is the reason why we turn our focus to $n$-grams. A $n$-gram represents in our context a sequence of $n$ terms, taken as they appear, from the title and the description. The choice for $n$ is a compromise between false positives and false negatives in the extraction process and it is further discussed.

## 4.2 Extraction

We present here a bottom-up algorithm that, given a feed item and the associated Web page, finds the wrapper element

of the data object by matching $n$-grams of the semantic context against the textual content of the leaf nodes extracted from the HTML Web page. This algorithm is summarized in Algorithm 1. Our algorithm relies on the assumption that a data object is encoded as a contiguous zone of (partially) formatted leaf nodes that contain either text or references (anchors and images).

We first introduce the notion of conceptual node:

Definition 1. *A conceptual node is a leaf node (a node with no children) that contains in its textual content at least one element (concept or n-gram) from the semantic context.*

We extract all leaf nodes of the page and, for each, establish its semantic density.

Definition 2. *The semantic density of a conceptual node is defined as the number of matched concepts or n-grams, divided by the length of the textual content of the respective node.*

We classify the conceptual nodes according to the closest ancestor that is a block-level element. An effective heuristic is actually to take the closest ancestor that is a `div` element, since we have observed in our experiments that data objects are almost always confined in a div element. After this analysis, we are able to say which are the conceptual nodes that share the same ancestor.

The list of ancestors will give us the semantic zones of the page, modeled at code level by semantic nodes.

Definition 3. *A semantic node is the lowest block-level common ancestor of a set of conceptual nodes.*

In order to clarify which of the semantic nodes represents the wrapper of the article, we calculate the following semantic density measure for each and take the node that has the largest value for it.

Definition 4. *The wrapper node of the data object is the semantic node that contains the largest number of dense conceptual nodes.*

In the previous description, we have allowed the use of either concepts or $n$-grams for finding conceptual nodes and computing semantic density. When matching with terms that correspond to concepts, the number of semantic zones will increase, while doing the matching with $n$-grams will visibly increase the possibility of having just one ancestor as result, the one that represents the wrapper for the article.

This happens because $n$-grams are more significant than concepts, relative to the data objects. Nevertheless, in certain cases, the choice of $n$-grams is too restrictive. This happens quite rarely, when the semantic context is too limited or when it comes from a description that is summarizing the article with different words, rather than representing the first lines of it. Consequently, in order to detect this kind of cases, we introduce the notion of semantic node consistency.

Definition 5. *A node is semantically consistent if its text contains a large proportion of the concepts acquired from the semantic context.*

We say a "large" proportion (in practice, $\geq 0.5$) of concepts because it is not necessary to check for the presence of all

**Input**: a URL of a Web feed $feedUrl$
**Output**: extracted data objects *dobs*

```
feedData = getFeedDataStructure(feedUrl);
items = feedData.getItems();
foreach item in items do
    dob.created = item.getPubDate();
    semanticInput = item.getTitle() +
      cleanHTMLtags(item.getDescription());
    tokens = tokenize(semanticInput);
    terms = rejectStopWords(tokens);
    normalizedTerms = applyStemming(terms);
    tags = setFrequencyAndFilter(normalizedTerms);
    dob.descripTags = tagConcepts;
    3grams =
      getSubsequencesOf3Tokens(semanticInput);
    articlePage = item.getLink();
    cleaned =
      getCleanedHTMLCodeFrom(articlePage);
    leafTagNames =
      cleaned.evaluateXPath("//*[not(*)]/name()");
    foreach leafType in leafTagNames do
        leafNodes =
          cleaned.getElementsByName(leafType);
        foreach leafNode in leafNodes do
            nbOfSemanticMatches =
              leafNode.nbMatches(3grams);
            if nbOfSemanticMatches ≥ 1 then
                conceptualNodes.put(
                  leafNode, nbOfSemanticMatches);
            end
        end
    end
    foreach cnode in conceptualNodes do
        while cnode isNotA block do
            ancestor = cnode.getParent();
        end
        semanticZones.add(ancestor);
    end
    foreach ancestor in semanticZones do
        nbConceptualNodes =
          intersectionSize(
            ancestor.descendant(),conceptualNodes);
        semanticDensitySum[ancestor] =
          ∑_{n=1}^{nbConceptualNodes} cnode.nbOfSemanticMatches / cnode.textualLength;
    end
    wrapperNode = the ancestor with the biggest value
      for semanticDensitySum ;
    if wrapperNode != null (matches at least a 3-gram)
    then
        while wrapperNode
          doesNotContainsEnoughConcepts do
            wrapperNode = wrapperNode.getParent();
            if wrapperNode is the root then
                wrapperNode=null;
                exit the while;
            end
        end
        if wrapperNode != null then
            dob.setFullTextAndReferencesFrom(wrapperNode);
        end
    end
    if dob.object is not set (wrapperNode was null)
    then
        repeat the operation by using this time random
        terms coming from the concepts
    end
    dobs.add(dob);
end
return dobs
```

**Algorithm 1**: Data Object Extraction

of them in order to assert a wrapper node. On the other hand, when the candidate ancestor node does not contain half of the semantic context, we might suspect that it is not the wrapper of the article. If this happens, we conclude that the $n$-grams were not appropriate due to imperfection of the semantic context, and therefore we can diminish the n of $n$-grams and repeat the method of getting the wrapper node. In our experiments, we start with $n = 3$ which gave best results, and, in case of failure, directly tried matching with concepts. In general, data objects (news articles, blog posts) have comments associated. Our technique aims at not including the comments in the resulting data object because:

1. from a conceptual point of view, the information of interest in an article is not the same as comments about it;

2. the crawl of the article should be separated from the crawl of the comments: if each time a comment is added, the article is considered to be changed, a new crawl is needed and the resulting object can be very redundant in comparison with its previous version; instead, the article should have a reference to its comments, tracked separately and synchronized on update;

3. usually, the webmaster takes care to add in the feed the URL that can be used to track comments, but when it is not the case, we can still identify the zone of comments by using heuristics in our algorithm.

# 5. EXPERIMENTS

In order to prove the validity of our approach for extracting data objects, we have fully implemented the system and performed experiments in order to evaluate its precision.

The experiments were done using the feeds collected as responses from the Search4RSS engine, the same dataset mentioned in Section 3. Recall that the dataset was quite diversified, in terms of structure and types of data objects.

For each feed, we have retrieved the channel data structure and applied the method of extraction for all of its item components.

As a first test, we have tried to return the zone of the Web page associated with the item where the item title could be found exactly as it appears. This method was giving poor results, however, for several reasons: the title may not fully match due to encoding characters, or it may appear at several different places in the page. Additionally, given only the location of the title, and its levels of imbrications in blocks of the HTML code, it is not an easy task to identify the limits of the whole data object.

We compare now the performance of our algorithm, i.e., the precision of extracted data objects, to Boilerpipe [7] which constitutes a state-of-the-art method for identifying the main content of a page in the absence of extra semantic information. We stress that, as we use more information than Boilerpipe has access to, obtaining a better precision does not diminish the interest of this approach, which is more general.

We observe that our result is often more precise, because we are not simply considering the text density in the Web page, but rather its semantic coherence according to the item. There are cases where a node may contain a lot of text but may be judged as worthless with respect to our semantic density measure. Moreover, note that when the Web page contains various consecutive articles, our method will make the difference between them and identify the specific article that corresponds to an item. In contrast, Boilerpipe will take the textual content of all the articles or just of the densest one, depending on the case.

**Table 3: Experimental results**

| Method | Correctly extracted | Precision |
|---|---|---|
| Our technique | 1038/1314 | 79.0% |
| Boilerpipe | 821/1314 | 62.5% |

The experiments were performed for 60 randomly selected sites from the dataset in the art domain (the first one), corresponding to a total of 1314 feed items. We manually checked the Web article, the result of our algorithm of extraction and that of Boilerpipe [7]. We only compare the textual result of the extraction because it is the output of the freely usable Boilerpipe implementation[2]. Our method actually extracts the whole content of the identified zone, including links and images.

Numerical results are given in Table 3. We consider an object as correctly extracted when its text is exactly that of the gold standard, obtained by manual annotation of the Web page. Partial matches are disregarded. The overall precision of the algorithm reaches the satisfactory number of around 79%, compared to the 62% precision obtained by Boilerpipe.

Finally, note that when our technique fails, the algorithm actually manages to identify a richer conceptual zone that the one of the data object, which is still relevant to the article, even though not strictly identifying with it.

# 6. DISCUSSION OF APPLICATIONS

We conclude this article by discussing a number of applications that can be integrated in the Web archiving process, and which use the data object extraction approach we propose.

*Perenniality of Web archives.* Time is a particularly important factor of influence also for interpreting crawled content. While data might remain intact, the way we understand it and the rendering of it changes, mostly due to the fact that the language itself, the culture and the technological means of expression evolve. One of the most serious problems encountered in Web archiving is when the format of crawled data becomes obsolete or not generally used. The solutions given by the authors of [5], [20] and [19] are software or hardware emulation, migrating content, or to include a proxy that will incorporate format translating capabilities and will do this dynamically, at user request or when the need is detected. While these works are trying to fight technology evolution, it is possible to take the opposite approach: adapt data to existing technology. In order to do this, one can imagine to encapsulate the relevant information of the crawled Web page, and to store the resulting object independently of the original encoding format. In this manner, the fact that technology evolves will not be anymore something to resist against, instead it will enhance the possibility of rendering the existing data in new ways, adapting the actual content to

[2] `http://code.google.com/p/boilerpipe/`

preferences. The extraction of data objects that we highlight in this paper is a first step towards storage of information freed from the specific way of encoding.

While it can be argued that for Web archiving the original form of a Web page matters, this is more related to the requirement of placing the actual data in its correct context. We note that our goal is not to try to change the current ways of storing the archived content, but for the cases in which it works (dynamic data with feed associated), some interesting overlay applications can be created for the people using the collection or for the archivists themselves. The extraction of data objects in the context of Web archiving is closely related to the semantic analysis of content in time and the possibility of providing value-added services. Indeed, what can very useful is to be able to run complex queries over the content, add interactivity and facilities to target consumers of the information presented as a Web archive.

*Reconstruction of Web pages.* Recall from Figure 2 that Web pages associated with Web feeds can have very lively dynamics, in some cases with update intervals of the order of the minute. In these conditions, it becomes unreasonable to try and capture every successive version of the corresponding channel Web page. However, since the Web feed keeps reference to a number of items (and, hopefully, a rather large number for such a lively channel), it is still possible to regularly crawl and archive it. In the cases where (most of) the content of each data object is stored in the Web feed, one application of the data extraction technique can be the reconstruction the Web page at a given point in time using the crawled Web feed items and references to template elements, inferring thus a version of a Web page that had not actually been crawled in the classical way.

Moreover, using our algorithm (and possibly some heuristics), we can detect not only the DOM node that contains the article, but also other semantic nodes of the page: the ones that contain the comments, categories or tags. These zones can be mined for textual content and references, and stored independently of the template of the page (that consists basically of everything that remains after the extraction). A relation between the resulting components can be recreated or inferred using the semantics, and used to analyze these components. We could as reinvent the way information is rendered by mixing the objects (in mashups) to adapt to user preferences.

Figure 3 depicts the flow of a demonstration scenario pertaining to this reconstruction of Web pages from data objects, as described in [11].

*Crawling and semantic exploitation of archives.* Finally, we want to briefly recall two other applications, already mentioned earlier in this paper. Firstly, it is possible to use the data object extraction technique for change detection. Using a technique of temporal analysis on feeds similar to the one presented in this paper, we can determine the strategy of publishing of the channel that represents the dynamic part of the website. Having this awareness, we can adapt the rate of crawl to the approximated frequency. Moreover, for already crawled articles, we can detect if a new version has appeared in order to crawl it or not. This can be done by applying the algorithm of data object extraction on the crawled article and on the possibly new version of it (the current Web page). By comparing the resulting data objects referring to the same URL, we can see if a change in the text or references has occurred in the intervening period of time. Secondly, a Web archive that contains data objects (maybe in addition to Web pages) can be used by analysts more effectively than just an archive of Web pages. For instance, a linguist could focus on the new terms that appear in newspaper articles, without bothering about the terms that appear in the comments. Generally, the aim is to add exploitable meaning and temporality (at finer-grained levels) to collections of Web archives, that can become more expressive and adapted to user needs.

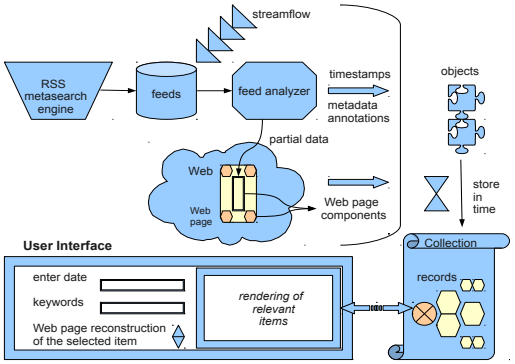


**Figure 3: Application for reconstructing Web pages from data objects**

*Acknowledgments.*

We would like to thank European Web Archive (especially, Julien Masanès, Gabriel Vasile and Radu Pop) for discussions about this topic and their help in obtaining the experimental dataset.

## 7. REFERENCES

[1] A. Arasu and H. Garcia-Molina. Extracting structured data from Web pages. In *Proc. SIGMOD*, San Diego, USA, June 2003.

[2] J. Cho and H. Garcia-Molina. The evolution of the Web and implications for an incremental crawler. In *Proc. VLDB*, Cairo, Egypt, Sept. 2000.

[3] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large Web sites. In *Proc. VLDB*, Roma, Italy, Sept. 2001.

[4] Eddie java feed parser. Website, 2010. `http://www.davidpashley.com/projects/eddie.html`.

[5] J. Hunter and S. Choudhury. Implementing preservation strategies for complex multimedia objects. In *Proc. ECDL*, Trondheim, Norway, Aug. 2003.

[6] Internet Archive. Website, 2010. `http://web.archive.org/collections/web.html`.

[7] C. Kholschutter, P. Fankhauser, and W. Nejdi. Boilerplate detection using shallow text features. In *Proc. WSDM*, New York, USA, Feb. 2010.

[8] G. Knight and M. Pennock. Data without meaning: Establishing the significant properties of digital research. *International Journal of Data Curation*, 4(1), 2009.

[9] B. Liu, R. Grossman, and Y. Zhai. Mining data records in Web pages. In *Proc. KDD*, Washington, USA, Aug. 2003.

[10] J. Masanès, editor. *Web Archiving*. Springer-Verlag, Heidelberg, Allemagne, 2006.

[11] M. Oita and P. Senellart. Archivage du contenu éphémère du Web à l'aide des flux Web. In *Proc. BDA*, Toulouse, France, Oct. 2010. Conference without formal proceedings. (Demonstration).

[12] J. Pasternack and D. Roth. Extracting article text from the web with maximum subsequence segmentation. In *Proc. WWW*, Madrid, Spain, Apr. 2009.

[13] Z. Pehlivan, M. Ben Saad, and S. Gançarski. A novel Web archiving approach based on visual pages analysis. In *Proc. IWAW*, Corfu, Greece, Sept. 2009.

[14] M. Pennock and R. Davis. ArchivePress: A really simple solution to archiving blog content. In *Proc. iPRES*, San Francisco, USA, 2009.

[15] Search4RSS feed search engine. Website, 2010. `http://www.search4rss.com/`.

[16] K. Sigurðsson. Incremental crawling with Heritrix. In *Proc. IWAW*, Vienna, Austria, Sept. 2005.

[17] Blogger's choice awards. Website, 2010. `http://bloggerschoiceawards.com/`.

[18] M. Spaniol, D. Denev, A. Mazeika, and G. Weikum. Catch me if you can. Temporal coherence of Web archives. In *Proc. IWAW*, Aarhus, Denmark, Sept. 2008.

[19] S. Strodl, P. P. Beran, and A. Rauber. Migrating content in warc files. In *Proc. IWAW*, Corfu, Greece, Sept. 2009.

[20] D. S. Swaney, F. McCown, and M. L. Nelson. Dynamic Web file format transformations with grace. In *Proc. IWAW*, Vienna, Austria, Sept. 2005.

[21] S. Yu, D. Cai, J.-R. Wen, and W.-Y. Ma. Improving pseudo-relevance feedback in web information retrieval using web page segmentation. In *Proc. WWW*, Budapest, Hungary, May 2003.