# INTERNALS OF AN AGGREGATED WEB NEWS FEED

*Mitja Trampuš, Blaž Novak*
Artificial Intelligence Laboratory
Jozef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
e-mail: {mitja.trampus, blaz.novak}@ijs.si

**ABSTRACT**

**We present IJS NewsFeed, a pipeline for acquiring a clean, continuous, real-time aggregated stream of publically available news articles from web sites across the world.**
**The articles are stripped of the web page chrome and semantically enriched to include e.g. a list of entities appearing in each article. The results are cached and distributed in an efficient manner.**

## 1 INTRODUCTION

The news aggregator is a piece of software which provides a real-time aggregated stream of textual news items provided by RSS-enabled news providers across the world. The pipeline performs the following main steps:

1) Periodically crawls a list of RSS feeds and a subset of Google News and obtains links to news articles
2) Downloads the articles, taking care not to overload any of the hosting servers
3) Parses each article to obtain
    a. Potential new RSS sources, to be used in step (1)
    b. Cleartext version of the article body
4) Process articles with Enrycher (see Section 3.2)
5) Expose two streams of news articles (cleartext and Enrycher-processed) to end users.

## 2 SYSTEM ARCHITECTURE

Figure 1 gives a schematic overview of the architecture. The first part of the aggregator is based around a PostgreSQL database running on a Linux server. The database contains a list of RSS feeds which are periodically downloaded by the RSS monitoring component. RSS feeds contain a list of news article URLs and some associated metadata, such as tags, publication date, etc. Articles that are not already present in the database are added to a list of article URLs, and marked for download. Tags and publication date are also stored alongside, if found in the RSS.

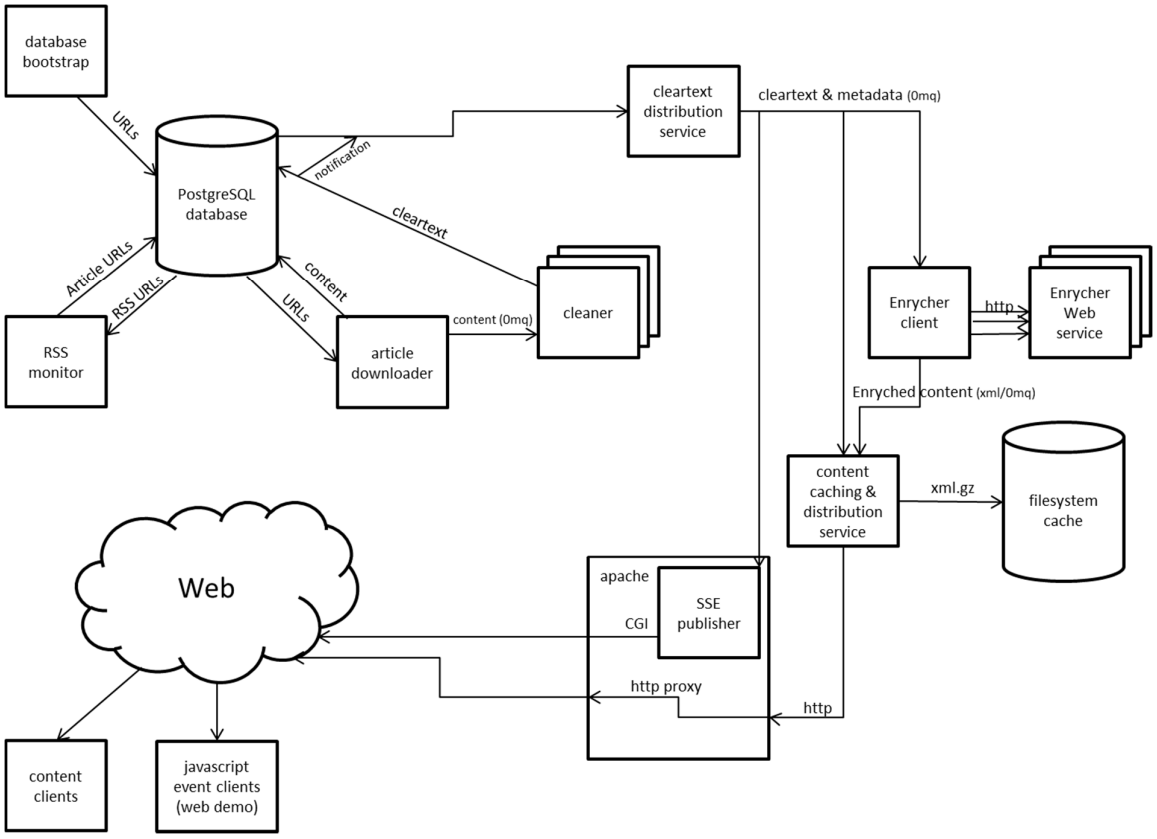


Figure 1: *The system architecture of IJS NewsFeed.*

A separate component periodically retrieves the list of new articles and fetches them from the web. The complete HTML is stored in the database, and simultaneously sent to a set of cleaning processes over a 0mq message queue.

The cleaning process converts the HTML into UTF-8 encoding, determines which part of the HTML contains the useful text, and discards the remainder and all of the tags. Finally, a classifier is used to determine the primary language.

The cleaned version of the text is stored back in the database, and sent over a message queue to consumers.

Documents in English language are sent to the Enrycher web service, where named entities are extracted and resolved, and the entire document is categorized into a DMOZ topic hierarchy.

Both the cleartext and the enriched versions of documents are fed to a filesystem cache, which stores a sequence of compressed xml files, each containing a series of documents in the order they have arrived through the processing pipeline. The caching service exposes an HTTP interface to the world through an Apache transparent proxy, serving those compressed xml files on user request.

The Apache server also hosts a CGI process capable of generating HTML5 server-side events, which contains the article metadata and cleartext as payload. These events can be consumed using Javascripts EventSource object in a web browser.

## 3 DATA PREPROCESSING

Data preprocessing is an important part of the pipeline, both in terms of the added value provides and in terms of challenges posed by the data volume. The articles themselves are certainly useful, but almost any automated task dealing with them first needs to transform the raw HTML into a form more suitable for further processing. We therefore perform the preprocessing ourselves; this is much like the practice followed by professional data aggregation services like Spinn3r or Gnip.

In terms of data volume, preprocessing is the most interesting stage and the one at which the most tradeoff can be made. The present data download rate of about one article per second is nothing extreme, especially if we consider scaling to multiple processing nodes; however, it is nontrivial in that adding complex preprocessing steps (e.g. full syntactic parsing of text) or drastically increasing data load (e.g. including a 10% sample of the Twitter feed) would turn preprocessing into a bottleneck and require us to scale the architecture.

### 3.1 Extracting article body from web pages

Extracting meaningful content from the HTML is the most obviously needed preprocessing step. As this is a pervasive problem, a lot has been published on the topic; see e.g. Pasternack (2009), Arias (2009), and Kohlschütter (2010).

We initially implemented the algorithm by Pasternack because of its simplicity and reported state-of-the-art performance. The algorithm scores each token (a word or a tag) in the document based on how probable it is to comprise the final result (the scores are trained); then it extracts the maximum token subsequence.

**Datasets**

We tested the initial algorithm on three manually developed datasets. Each of the three consists of 50 articles, each from a different web site.

- **english** – English articles only.
- **alphabet** – Non-English articles using an alphabet, i.e. one glyph per sound. This includes e.g. Arabic.
- **syllabary** – Non-English articles using a syllabary, i.e. one glyph per syllable. This boils down to Asian languages. They lack word boundaries and have generally shorter articles in terms of glyphs. Also, the design of Asian pages tends to be slightly different.

Some of the input pages (about 5%), realistically, also do not include meaningful content. This is different from other data sets but very relevant to our scenario. Examples are paywall pages and pages with a picture bearing a single-sentence caption.

The fact that each of the 150 articles comes from a different site is crucial – most of the papers on this topic evaluate on a dataset from a small number of sites, which leads to overfitting and poor performance in the general case. This was also the case with Pasternack's algorithm. As the performance was unsatisfactory, we developed three new algorithms.

**Algorithms**

- **WWW** – an improved version of Pasternack (2009), it extracts *two* most promising contiguous chunks of text from the article to account for the fact that the first paragraph is often placed separately from the main article body.
- **WWW++** – a combination of WWW and heuristic pre- and post-processing to account for the most obvious errors of WWW. For instance, preprocessing tries to remove user comments.
- **DOM** – a completely heuristics-based approach proposed here which requires the DOM tree to be computed. With the fast libxml package, this is not a limiting factor. The core of the heuristic is to take the first large enough DOM element that contains enough promising <p> elements. Failing that, take the first <td> or  element which contains enough promising text. The heuristics for the definition of "promising" rely on metrics found in other papers as well; most importantly, the amount of markup within a node. Importantly, none of the heuristics are site-specific.

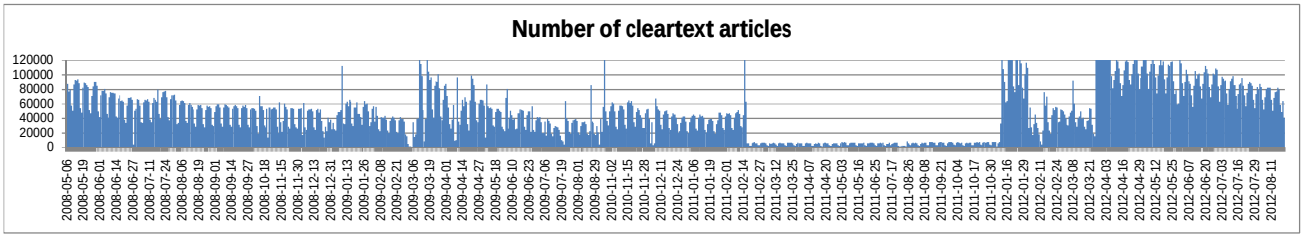


Figure 2: *The daily number of downloaded articles. A weekly pattern is nicely observable. Through most of 2011, only Google News was used as an article source, hence the significantly lower volume in that period.*

In all three algorithms, all pages are first normalized to the UTF-8 character set using the BeautifulSoup package (which in turn uses a combination of http headers, meta tags and the chardet tool).

**Evaluation**

We evaluated two of the three pairs of algorithms by comparing per-article performance. We did compare WWW and DOM; based on informal inspection of outputs, DOM would be certain to perform better.

| Algo / Dataset | WWW vs WWW++ number of articles where one of the algorithms performs better | | | WWW++ vs DOM number of articles where one of the algorithms performs better | | |
|---|---|---|---|---|---|---|
| | **WWW** | tie | **WWW++** | **WWW++** | tie | **DOM** |
| **English** | 2 | 43 | 4 | 7 | 34 | 8 |
| **alphabet** | 4 | 37 | 8 | 6 | 36 | 7 |
| **syllabary** | 0 | 44 | 6 | 2 | 12 | 32 |

*Table 1. Performance comparison of webpage chrome removal algorithms*

The differences between the algorithms are statistically significant with a 5% confidence interval only on the syllabary dataset; it is however clear from the data that overall, WWW++ performs better than WWW and DOM performs better still. DOM is therefore our algorithm of choice.

For DOM, we additionally performed an analysis of errors on all three datasets. As the performance did not vary much across datasets, we present the aggregated results. For each article, we manually graded the algorithm output as one of the following:

- **Perfect [66.3%]** – The output deviates from the golden standard by less than one sentence or not at all: a missing section title or a superfluous link are the biggest errors allowed. This also includes cases where the input contains no meaningful content and the algorithm correctly returns an empty string.
- **Good [22.1%]** – The output contains a subset or a superset of the golden standard. In vast majority of the cases, this means a single missing paragraph (usually the first one which is often styled and positioned on the page separately) or a single extraneous one (short author bio or an invitation to comment on the article). A typical serious but much rarer error is the inclusion of visitors' comments in the output.
- **Garbage [5.8%]** – The output contains mostly or exclusively text that is not in the golden standard. These are almost always articles with a very short body and a long copyright disclaimer that gets picked up instead.
- **Missed [5.8%]** – Although the article contains meaningful content, the output is an empty string, i.e. the algorithm fails to find any content.

If we combine "Perfect" and "Good" (where the outcome is most often only a sentence away from the perfect match) into a "Positive" score, both precision and recall for DOM are 94%. This (article-based) metric is arguably comparable with the word- or character-based metrics employed in some other papers on state of the art methods (Kohlschütter 2010); those also report precision and accuracy of at most 95%.

### 3.2 Extracting semantic information from clear text

For most of semantic processing, we rely on Enrycher (Štajner 2009) running as a service. In order to increase error resiliency, improve the utilization of the service and avoid undue delays in the preprocessing pipeline, we access the service in a multithreaded fashion. For performance evaluation and other information, please refer to the paper by Štajner.

Enrycher annotates each article with named entities appearing in the text (resolved to Wikipedia when possible), discerns its sentiment and categorizes the document into the general-purpose DMOZ category hierarchy.

We also annotate articles with a language; detection is provided by a combination of Google's open-source Compact Language Detector library for mainstream languages and a separate Bayesian classifier. The latter is trained on character trigram frequency distributions in a large public corpus of over a hundred languages. We use CLD first; for the rare cases where the article's language is not supported by CLD, we fall back to the Bayesian classifier. The error introduced by automatic detection is below 1% (McCandless, 2011).

## 4 DATA PROPERTIES

In no particular order, we list some statistics of the data provided by the news aggregator.

### 4.1 Sources

The crawler actively monitors about 75000 feeds from 1900 sites. The list of sources is constantly being changed – stale sources get removed automatically, new sources get added from crawled articles. In addition, we occasionally manually

prune the list of sources using simple heuristics as not all of them are active, relevant or of sufficient quality. The feed crawler has inspected about 350000 RSS feeds in its lifetime. The list was bootstrapped from publically available RSS compilations.

Besides the RSS feeds, we use Google News (news.google.com) as another source of articles. We periodically crawl the US English edition and a few other language editions, randomly chosen at each crawl. As news articles are later parsed for links to RSS feeds, this helps diversify our list of feeds while keeping the quality high.

We also support additional news sources with custom crawling methods. The sources are not limited to any particular geography or language.

### 4.2 Language distribution

We cover 37 languages at an average daily volume of 100 articles or more. English is the most frequent with an estimated 54% of articles. German, Spanish and French are represented by 3 to 10 percent of the articles. Other languages comprising at least 1% of the corpus are Chinese, Slovenian, Portugese, Korean, Italian and Arabic.

### 4.3 Data volume

The crawler currently downloads 50000 to 100000 articles per day which amounts to roughly one article per second. The current archive contains about 40 million articles and begins in May 2008. See Figure 2.

The median and average article body lengths are 1750 and 2400 bytes, respectively.

### 4.4 Responsiveness

We poll the RSS feeds at varying time intervals from 5 minutes to 12 hours depending on the feed's past activity. Google News is crawled every two hours. All crawling is currently performed from a single machine; precautions are taken not to overload any news source with overly frequent requests.

Based on articles with known time of publication, we estimate 70% of articles are fully processed by our pipeline within 3 hours of being published, and 90% are processed within 12 hours.

## 5 DATA DISSEMINATION

Upon completing the preprocessing pipeline, contiguous groups of articles are batched and each batch is stored as a gzipped file on a separate distribution server. Files get created when the corresponding batch is large enough (to avoid huge files) or contains old enough articles. End users poll the distribution server for changes using HTTP.This introduces some additional latency, but is very robust, scalable, simple to maintain and universally accessible.

Independent of this server-side, filesystem-based cache, a complete copy of the data is still kept in the traditional structured database (see Section 2). This is the only copy guaranteed to be consistent and contain all the data; from it, the XML files can be regenerated at any time. This is particularly useful in case of XML format changes and/or improvements to the preprocessing pipeline.

## 6 CONCLUSION

We presented a news crawling and processing engine that is scalable, responsive and achieves state of the art performance in most of the processing stages.

The data provided by the pipeline is being successfully used in several multilateral projects with expected applications in cross-lingual text mining, opinion mining and recommender systems.


*Figure 3: A real-time preview of the stream demonstrating some of the semantic annotations. See http://newsfeed.ijs.si/visual_demo/*


### Acknoledgements

This work was supported by the Slovenian Research Agency and the ICT Programme of the EC under RENDER (ICT-257790-STREP), XLike (ICT-STREP-288342), PlanetData (ICT-NoE-257641), MetaNet (ICT-249119-NoE).


### References


[1] Arias, J., Deschacht, K., & Moens, M. (2009). *Language independent content extraction from web pages.* Proceedings of the 9th Dutch-Belgian information retrieval workshop.

[2] Kohlschütter, C., Fankhauser, P., & Nejdl, W. (2010). *Boilerplate detection using shallow text features.* Proceedings of WSDM 2010.

[3] McCandless, M. (2011). *Accuracy and performance of Google's Compact Language Detector*. Retrieved from http://blog.mikemccandless.com/2011/10/accuracy-and-performance-of-googles.html

[4] Pasternack, J., & Roth, D. (2009). *Extracting article text from the web with maximum subsequence segmentation.* Proceedings of the 18th WWW conference

[5] Štajner, T., Rusu, D., Dali, L., Fortuna, B., Mladenić D., Grobelnik, M. (2010). *A service oriented framework for natural language text enrichment. Informatica (Ljublj.), 2010, 34:3, pp. 307-313.*