

Information Extraction for Semi-structured Email Corpora

Hendrik Adam¹ and Philipp Schaer²[0000–0002–8817–4632]

¹ Science Media Center, Cologne, Germany

`firstname.lastname@sciencemediacenter.de`

² TH Köln - University of Applied Sciences, Cologne, Germany

`firstname.lastname@th-koeln.de`

Abstract. Information extraction is a requirement for enhanced IR techniques. To surpass rigid extraction rules in wrappers based on XPath or CSS selectors, we present a new extraction method extending the FleXPath method that was used for structured XML retrieval in INEX. We expand this method to work with semi-structured HTML and present a case study and a short evaluation based on a corpus of emails from scientific publishers.

Keywords: Information extraction · Semi-structured documents · Emails

1 Introduction

Information extraction is a requirement and necessity for enhanced information retrieval techniques like entity-based search, semantic search, or other approaches that make use of properly structured and annotated information [3]. Searching for or within structured information like XML documents (INEX) allows to make use of semantic annotations directly, but usually, source documents are not semantically structured at all and only allow full-text search. In this work, we use a newsletter corpus that is not formally structured but semi-structured with some reoccurring parts of the mails (like title, dates, or authors). We want to extract this information to allow rich IR techniques like filtering, browsing, or ranking based on these semantic annotations.

Usually information extraction [6] relies on structural layouts and syntactical patterns [5] within the source documents. *Wrappers* are used to make use of these structures. Wrappers use pattern matching procedures and heavily rely on pre-defined or learned extraction rules [9]. A more general approach for wrapper construction is XPath [8] or derivation like OXPath [10]. Both systems are used to address and locate parts and nodes in XML / HTML documents and to extract their content. Especially for HTML documents CSS selectors are an alternative to XPath [4].

Copyright ©2019 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

A common issue in real-life web information extraction is the fact that XPath or CSS-based extraction rules are not flexible and are prone to even slight changes in the source documents' structures. Due to this, the process of adjusting an existing wrapper to new requirements is costly. Amer-Yahia et al. introduced a technique called *FleXPath* [1] to surpass this issue. They developed a mixture of structured XPath and full-text XQuery-based search techniques to extract information from structured XML documents. Thus FlexPath allows using both: database style querying and full-text search. The results from both query approaches are scored and ranked on structural and full-text search features. This approach was successfully evaluated in INEX [2].

We argue that automated web information extraction could benefit from this idea and that the FlexPath approach could be expanded for semi-structured HTML documents. Our main goal is to yield better coverage for structured information extraction. Due to the nature of XPath or CSS-based selectors, any structural changes to the source of extraction will affect the quality of the returned data. This paper aims to create and to test an information extraction concept for semi-structured documents.

We will demonstrate the feasibility of our approach by implementing our algorithm in a demonstrator called FlexIpy and test it against a sample document collection containing different types of semi-structured HTML data. We will test whether our approaches will result in a more robust and more complete outcome in contrast to "normal" XPath/CSS-based extraction frameworks.

2 FleXipy - Information Extraction for HTML

In this section we will present *FleXipy*, an extraction mechanism that incorporates both *path-based* and *text-based* features. FlexIpy always starts with a strict path-based extraction rule that was configured by the wrapper generator. Only if this strict extraction rule does not verify and returns an empty node, the additional extraction features are activated. First, we generate a list of possible node candidates by looking in the neighborhood of the original configured path. In a second step, we use different text-based features to rank the different node candidates to find the best-matching node and path to correct the wrong extraction rule.

2.1 Main Components of FlexIpy

The core of FlexIpy is divided into three major components: (1) Modules for path-based tree interactions that use rigid patterns like XPath, modules for textual, (2) content-based interactions that use full-text search patterns and finally (3) a module to rank the node candidates.

Path-based Modules There are two different methods for finding node candidates with FlexIpy. One requires a full XPath to nodes where the desired text

should be found. The other one requires an XPath to select a structural description to look for anywhere in the DOM tree. When giving a full XPath, we will use it as a template. This method is useful in cases where the desired text *should be* in this node but moved to a nearby sibling due to minimal structural changes. Using a limited breadth-first search on the DOM tree and a following depth-limited search, we will check any siblings for existing text and their full XPath will be added to the list of possible candidates. In Flexipy, this is called *template search*.

The second method requires a structural description of a subtree to search for in the DOM tree, e.g `//tbody/tr/td/h1`. Any found expression containing text will be considered a candidate for further analysis. In cases where no candidate can be found, it is recommended and supported to reduce the size of the subtree to enlarge the search space. This is called *subtree search*.

The two path-based searches return a list of candidates. We calculate the distances between the candidates in the DOM tree and the original configured XPath expression. This distance will be normalized for the template search. For subtree search, no distance can be calculated. Therefore the distance is set to $d = 0$. All other distances contribute to the path score like follows: $p = 1 - d/d_{max}$. A smaller distance will lead to a higher path score of $0 \leq p \leq 1$.

Text-based Modules Text- and content-based modules are components that test the content of all candidates found by the path-based modules. These modules represent a verification process. Depending on the configured rules, the interaction can be expressed by using a Radcliff/Obershelp fuzzy string matching algorithm or by using a combination of rules to verify the structural features of the extracted text. These rules can match text features like length and formatting, check for prefixes or use text similarity scores for text which may only change slightly for multiple sources. Every configured rule symbolizes a verification task for the extracted text and should “penalize” candidates for not passing a pattern rule. The text score is $t = 1 - f/n$, where f is the number of failed verifications and n is the number of configured rules. The result of this process is a score of $0 \leq t \leq 1$ for all text-based features.

Candidate Ranking The two module types return two scores: p for for all path-based features and t for all textual features for all node candidates. Using a simple linear combination of both normalized scores p and t will then lead to a final ranking R (Eq. 1):

$$R = p \times (1 - \alpha) + t \times \alpha \tag{1}$$

where $0 \leq \alpha \leq 1$ is usually set to 0.7. The value for α is a purely heuristically determined best-practice value that works for the test data sets in our case study. (see section 3).


2.2 Example Extraction with FleXipy

To further clarify how the FleXipy algorithm works, we will showcase a step by step extraction in comparison to a conventional XPath based extraction approach.

#1: EMBARGOED PRESS RELEASE
STRICTLY UNDER EMBARGO UNTIL 12:00PM NOON ET (US) ON THURSDAY, APRIL 12, 2018
 Region(s) of Interest: United Kingdom
 Institution(s): University of Oxford

Sweet Potato History Casts Doubt on Early Contact between Polynesia and the Americas

Evidence reported in the journal *Current Biology* on April 12 shows that sweet potatoes arose before there were any humans around to eat them. The findings also suggest that the sweet potato crossed the ocean from America to Polynesia without any help from people. The discovery raises doubts about the existence of pre-Columbian contacts between Polynesia and the American continent.



```

1  ...
2  <div id="email-wrapper">
3    <div>
4      <p style="margin:0pt">...</p>
5      <p style="margin:0pt">...</p>
6      <p style="margin:0pt">...</p>
7      <p style="margin:0pt">...</p>
8      <p style="margin:0pt">...</p>
9      <p style="margin:0pt">...</p>
10     <p style="margin:0pt">...</p>
11     <p style="margin:0pt; text-align:center"><span style="...">Sweet
        Potato History Casts Doubt on Early Contact between
        Polynesia and the Americas</span>
12     <br>
13     <span style="...">...</span>
14     </p>
15     <p style="margin:0pt">...</p>
16     ...
17   </div>
18 </div>
19 ...
  
```

Fig. 1. Top: Section from an embargo email from the journal *Cell*. Bottom: Corresponding HTML code for this email.

Figure 1 shows the head of an article in one of the typical embargo emails handled in PRIOR (For explanation of PRIOR see section 3). For this example, we would like to extract the title (“Sweet Potato History Casts ...”). The starting point for the extraction for both methods is manually finding the XPath pointing to the headline.

Using the HTML snippet from Figure 1, the XPath expression needed is `//*[@id="email-wrapper"]/div/p[7]/span[1]`. We can see from the expression, that there are two nodes with siblings of the same kind (`p` and `span`). Due to the heterogeneous nature of emails, this could cause problems since the headline node could still be *moving* in the DOM tree of the same provider. A typical XPath based wrapper will then extract no or unwanted data.

When using FleXipy, we can make use of the text-based modules for identifying unwanted or no data. Looking at our example, it is clear that the headlines,

in this case, are formatted using CSS, and we also assume that headlines have a minimum length of 30 characters. Therefore, we can make use of functions provided by the FleXipy framework addressing these characteristics. These directives will be added to the FleXipy configuration file.

During a simulated extraction process, we got an email where the title can be found not in `//div/p[7]/span[1]` but in `//div/p[8]/span[1]`. Using a normal XPath-based wrapper, the extraction will fail since the path has changed. The FleXipy framework will, first of all, check the given XPath expression against the configured directives. If they match, FleXipy is done, if not FleXipy will use its path-based modules to look in the surroundings of the configured XPath for nodes that contain any form of text assuming that the wanted text only *moved* slightly. In this case, the mentioned template search will look for siblings for `p` and `span` nodes, ultimately also cover the wanted text. When reaching the configured limits, all found nodes containing text will then be checked against the text-based directives. Combining the distance and probability that a found text is the wanted text, the framework will then calculate a ranking of candidates and provide it for further processing.

3 A Case Study on FleXipy for Emails

To test the feasibility of the FleXipy approach, we will present the results of a case study with semi-structured email texts. The source of our test collection is a set of extracted email bodies in HTML from the Google Digital News Initiative-funded PRepublicatIOn Radar (PRIOR) project³. The background of PRIOR is to enable science journalists to keep up with the latest scientific research in relevant domains of knowledge. Scientific publishers offer science journalists exclusive access to prepublications of upcoming research articles under a very strict embargo. These prepublications are distributed exclusively via email and are very heterogeneous by nature as each publisher has its own format. The PRIOR project uses a fully automated extraction pipeline written in Python that utilizes the web crawling framework *scrapy* and XPath-based wrappers for information extraction. The daily work with these emails shows that these wrappers are highly vulnerable against deviations in the source data, which is the case for most of the emails. Most of the time, the emails are highly unstructured or do not share a common set of formatting rules, making them nearly impossible to process by standard wrappers. These issues make the unstructured and heterogeneous email bodies from the PRIOR project a perfect test bed for evaluating the FleXipy approach.

3.1 Evaluation Setup

We built a small test collection out of 75 PRIOR emails from the four scientific journals Science, Jama, Lancet, and Cell. Each email may contain many

³ <https://ir.web.th-koeln.de/projects/prior/>

embargo announcements that consist of different metadata fields. We manually identified and annotated 202 of these metadata fields (also called slots) within the source emails. The fields were of the four following types: Embargo dates, contact details, titles of the embargoed articles, and short abstracts. For each field, we manually defined the correct XPath to extract the information to test the different system configurations.

Two different configurations were part of our evaluation: A simple XPath-based extraction without any modifications (called baseline) and a full features FleXipy configuration (called Full-FleXipy). The baseline represents the standard extraction process done by frameworks like Scrapy. From each journal, we randomly took one email as training data to manually extract the XPath expression and used it to create a simple wrapper for the information extraction. The same XPath expression was later configured in FleXipy. The training emails were removed from the evaluation corpus. These extracted XPath expressions were then used as a rigid set of extraction rules for the remaining emails. Since emails are heterogeneous by nature, the results of this simulated extraction process are expected to be insufficient. After generating our baseline, we created additional test-based rule sets (like prefix patterns) for the training data and activated the path-based modules.

3.2 Evaluation Metrics

We compared the results of FleXipy against manually annotated data for all slots. The comparison relies on two different measurements: F_1 and the *slot error rate* [7] (SER). When evaluating, we have four different kinds of results for a slot: (1) We correctly extracted a slot (*hit*), (2) we have slots which contain incorrect data (*substitution*), (3) we have slots without any data (*deletion*) and (4) slots which were configured and extracted but cannot be found in the source data and should not have been extracted (*insertion*).

Returned slots are either a hit, a substitution or an insertion, but only slots which count as a hit are relevant in terms of precision and recall and therefore F_1 . We decided to use the slot error rate as a additional measurement as it introduces a performance measurement for information extraction which gives adequate weight to the different error types. It is defined as follows (Eq. 2):

$$SER = \frac{(substitutions + deletions + insertions)}{(hits + substitutions + deletions)} \quad (2)$$

where a lower SER value indicates a better extraction performance.

3.3 Findings

We evaluated four different entry types (embargo dates, contact information, titles, and abstracts) for four different scientific journals (Science, Jama, Lancet, and Cell). We report SER and F_1 -scores (see Table 1). The statistical significance of differences in the average performance is determined using a two-sided

		Embargo		Contact		Title		Abstract	
		SER	F_1	SER	F_1	SER	F_1	SER	F_1
Baseline	Science	0.38	0.69	0.69	0.40	0.50	0.53	0.63	0.38
	Jama	0.00	1.00	nan	0.00	0.00	1.00	0.12	0.88
	Lancet	0.73	0.29	1.50	0.00	0.93	0.13	0.80	0.21
	Cell	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.00
	avg	0.53	0.49	1.06	0.10	0.61	0.41	0.64	0.37
Full-FleXipy	Science	0.44	0.72	0.00	1.00	0.19	0.84	0.38	0.63
	Jama	0.00	1.00	nan	0.00	0.00	1.00	0.12	0.88
	Lancet	0.00	1.00	0.50	0.67	0.33	0.67	0.80	0.21
	Cell	0.00	1.00	0.00	1.00	0.00	1.00	1.00	0.00
	avg	0.11	0.93	0.17*	0.67*	0.13	0.88	0.57	0.43

Table 1. Results of the information extraction for four different field types and four different journals comparing the baseline approach and the full FleXipy implementation. We report the slot error rate (SER) and F_1 -scores. Improvements are marked green and losses are marked red. Significance was tested using a two-sided t-test. Field types not existing in a journal are represented by nan values.

Student’s t-test. Significant improvements over the baselines ($p < 0.05$) are indicated with an asterisk.

We see clear improvements in the average extraction performance in all four extracted entry types on both F_1 and SER. Especially the contact information shows a significant improvement from an average SER of 1.06 to 0.17 and an improvement of F_1 from 0.1 to 0.67. The other improvements are still evident but not statistically significant. A small increase can also be seen for the most heterogeneous slot defined increasing the F_1 average for abstracts from 0.37 to 0.43. Only one single slot entry type for one single journal had a loss compared to the baseline: the embargo dates from the Science journal. Here the SER value raised from 0.38 to 0.44 while the F_1 still increased from 0.69 to 0.72.

We can also see that the information saved in the embargo slot was extracted perfectly for three of the four providers. For Cell, FleXipy was capable of finding all missing titles and contacts and fix them accordingly. For Jama, there is no change in the result at all, since the baseline result was already near perfect from the beginning.

4 Discussion and Outlook

We presented and evaluated an information extraction method to complement simple XPath or CSS-based wrappers. We implemented our approach that incorporates path-based and text-based extraction rules and a candidate ranking module in a demonstrator called FleXipy. The evaluation of email newsletters from different scientific publishers and journals showed clear improvements compared to a simple XPath baseline. The improvements are possible as FleXipy can

search for alternative candidates when the original XPath-selector would only return an empty node (deletions). Additionally, FlexIpy could be used to correct the two other extraction errors types substitutions and insertions as the text-based modules would allow checking on additional features (like text patterns) rather than only on path features.

Although the evaluation showed a clear improvement and returned nearly perfect results for some entry types, we have to emphasize the preliminary character of this evaluation. It is only a case study with 202 manually annotated slots resulting in a rather small test collection. To get a more reliable result, the size of the collection should be increased. Also, the configured rules used to verify the data can be improved by increasing the size of the training data.

An interesting result is the opposing performance of SER and F_1 for the embargo entries in Science. This is the case when the extraction results shift from cases of substitutions to cases of deletions. This leads to a decreasing number of returned documents and therefore influences the precision and recall. The increase in F_1 does not clearly describe an overall increase in the extraction result since there could still be the case of less correctly extracted slots. These contradicting results for the embargo dates in Science are a compelling case that demonstrates the justification for using SER to complement the precision/recall-fixed perspective of F_1 . While we increase the number of relevant slots in the result, we introduce more and different error types.

The overall results of this case study can be promising for further development of this approach to improve web information extraction. FlexIpy is not a complete extraction framework (like e.g., Scrapy) but can be part of such frameworks to complement XPath or CSS-based wrappers. For productive use, this framework should be extended with more possibilities to write extraction rules for the source data.

Acknowledgments

This work was part of the PRIOR project that was funded by the Google Digital News Initiative⁴.

References

1. Amer-Yahia, S., Lakshmanan, L.V.S., Pandit, S.: Flexpath: Flexible structure and full-text querying for xml. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data. pp. 83–94. SIGMOD '04, ACM, New York, NY, USA (2004). <https://doi.org/10.1145/1007568.1007581>, <http://doi.acm.org/10.1145/1007568.1007581>
2. Amer-Yahia, S., Lalmas, M.: XML search: languages, INEX and scoring. ACM SIGMOD Record **35**(4), 16–23 (Dec 2006). <https://doi.org/10.1145/1228268.1228271>, <http://portal.acm.org/citation.cfm?doid=1228268.1228271>

⁴ <https://newsinitiative.withgoogle.com/dnifund/dni-projects/prior-prepublication-radar-round-4/>

3. Balog, K.: Meet the Data. In: Balog, K. (ed.) Entity-Oriented Search, pp. 25–53. The Information Retrieval Series, Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-93935-3_2, https://doi.org/10.1007/978-3-319-93935-3_2
4. Chamberlain, S., Ram, K., Grolemond, G.: Extracting data from the web apis and beyond. In: The R User Conference 2016. Stanford University, Stanford, California (2016), <https://github.com/ropensci-training/user2016-tutorial/blob/master/slides.pdf>
5. Chang, C.H., Kayed, M., Girgis, M.R., Shaalan, K.F.: A survey of web information extraction systems. *IEEE transactions on knowledge and data engineering* **18**(10), 1411–1428 (2006)
6. Jurafsky, D., Martin, J.H.: *Speech and language processing*, vol. 3. Pearson London (2014)
7. Makhoul, J., Kubala, F., Schwartz, R., Weischedel, R., et al.: Performance measures for information extraction. In: *Proceedings of DARPA broadcast news workshop*. pp. 249–252. Herndon, VA (1999)
8. Melton, J., Buxton, S.: *Querying XML: XQuery, XPath, and SQL/XML in context*. The Morgan Kaufmann Series in Data Management Systems, Elsevier Science (2011), <https://books.google.de/books?id=EuYRXgDqVp0C>
9. Michels, C., Fayzrakhmanov, R.R., Ley, M., Sallinger, E., Schenkel, R.: Oxpath-based data acquisition for dblp. In: *2017 ACM/IEEE Joint Conference on Digital Libraries, JCDL 2017, Toronto, ON, Canada, June 19-23, 2017*. pp. 319–320. IEEE Computer Society (2017). <https://doi.org/10.1109/JCDL.2017.7991609>, <https://doi.org/10.1109/JCDL.2017.7991609>
10. Neumann, M., Steinberg, J., Schaer, P.: Web-Scraping for Non-Programmers: Introducing OXPath for Digital Library Metadata Harvesting. *Code4Lib Journal* **2017**(38) (Oct 2017), <https://journal.code4lib.org/articles/13007>