Synergic Data Extraction and Crawling for Large Web Sites

Celine Badr, Paolo Merialdo, Valter Crescenzi
Dipartimento di Ingegneria
Università Roma Tre
Rome - Italy
{*badr, merialdo, crescenz*}*@dia.uniroma3.it*

***Abstract*—Data collected from data-intensive web sites is widely used today in various applications and online services. We present a new methodology for a synergic specification of crawling and wrapping tasks on large data-intensive web sites, allowing the execution of wrappers while the crawler is collecting pages at the different levels of the derived web site structure. It is supported by a working system devoted to non-expert users, built over a semi-automatic inference engine. By tracking and learning from the browsing activity of the non-expert user, the system derives a model that describes the topological structures of the site's navigational paths as well as the inner structures of the HTML pages. This model allows the system to generate and execute crawling and wrapping definitions in an interleaved process. To collect a representative sample set that feeds the inference engine, we propose in this context a solution to an often neglected problem, called the Sampling Problem. An extensive experimental evaluation shows that our system and the underlying methodology can successfully operate on most of the structured sites available on the Web.**

***Keywords*-data extraction; crawler; web wrapper; sampling;**

I. INTRODUCTION

Large data-intensive web sites contain information of interest to search engines, web applications, and various online service providers. These sites often present structural regularities, embedding content into predefined HTML templates using scripts. Regularities are also apparent at the topological level, with similar navigational paths connecting the pages obeying to a common template. In this paper, we extend the work introduced in [1], to address two related issues for capturing useful information from structured large web sites: first, the pages containing the information of interest need to be downloaded; second, the structured content needs to be extracted by web wrappers, i.e., software modules that collect page content and reorganize it in a format more suitable for automatic processing than HTML.

In general, crawlers and wrappers generation have been studied separately in the literature. Numerous tools exist for generating wrappers, with different levels of automation. They usually aim at extracting information from semi-structured data or text, and they use to that effect scripts or rule-based wrappers that rely on the structure or format of the source HTML. In some cases, wrappers are based on ontlogies or NLP techniques. Concerning the level of automation, hand-coded wrappers require a human expert, which becomes a cumbersome task for data extraction on large data-intensive web sites. Fully automatic wrappers have also been implemented [1], [2], but they necessitate considerable data post-processing and may suffer from lower accuracy. In semi-automatic wrapping, the main focus has been on learning approaches that take several positive and negative labeled examples as input.

Various tools also exist to crawl web pages and entire web sites. Popular techniques start with seed URLs and either search on the pages for hyperlinks matching certain patterns, or consider all encountered hyperlinks and then apply selection and revisit techniques for downloading target pages based on content or link structure [3], [4]. For deep Web crawling, some work has been proposed to obtain and index URLs resulting from different form submissions, e.g., [5]. However, the production of crawlers for structured web sites remains a subject with large room for improvement.

When it comes to large web data-intensive sites, it is commonly useful to extract data from a subset of the pages, in general pertaining to vertical domains. For example, in a finance web site, one may be interested in extracting company information such as shares, earnings... In this case, there is no need to download all the site's pages about market news, industry statistics, currencies, etc. Thus we propose a new approach in which the two problems of crawling and wrapping are tackled concurrently, where the user indicates the attributes of interest while making one browsing pass through the site hierarchy. The specifications are created in the same contextual inference run. Moreover, the execution of the wrappers takes place while the crawler is collecting pages at the different levels of the derived web site structure. The mutual benefits are manifested as the produced simple wrappers extract the specific links followed by the crawling programs, and the crawlers are in turn used to obtain sample pages targetted for inferring other wrappers. We have developed a working system that offers these capabilities to non-expert users. It is based on an active-learning inference engine that takes a single initial positive example and a restricted training set of web pages. We also define in this context the *Sampling Problem*, a problem often undervalued in the literature, and we show how it is mitigated by our approach when collecting a representative training set for

 ISBN: 978-1-61208-280-6

the inference process.

Throughout the paper, we consider an example web site that offers financial information on companies. We are interested in extracting from company pages data about stock quotes that is accessible in two different navigation sequences in the site topology: first, the home page displays companies' initials as links. Clicking on a letter leads to a listing of all the companies with this given initial. Each company name in turn links to a page containing the data to extract. Second, by filling and submitting a form on the home page, one reaches index pages grouping the companies by financial sector. Index pages are paginated, and by following the links provided in each paginated result list, the target company pages can be finally reached.

The rest of the paper is organized as follows: Section II presents our web site abstract model, on which we build the interleaved crawling and wrapping definitions; Section III lists and explains the crawling algorithm; Section IV defines the extraction rule classes used for extracting both data and links leading to the pages where these data are located; Section V presents the wrapper and assertion constructs that enhance our synergic crawling and wrapping tasks; in Section VI, we define the sampling problem and present our approach to collect a sample set; Section VII summarizes the results of the extensive experimental activity we conducted; Section VIII discusses related work; and finally, Section IX concludes the paper and presents some possible future developments.

II. An Abstract Model for Describing Large Web Sites

To access data from data-intensive web sites, we aim to reach these data on pages collected by a crawler using wrappers tailored to the user's information needs. We note that large web sites are composed of hundreds of pages that can generally be grouped in few categories, such that pages of the same category share a common HTML template and differ in contents. These sites also exhibit topological regularities in the navigational paths that link the pages and page categories. By capturing these regularities, we describe large web sites with an abstract model of three interrelated levels: the *intensional, extensional,* and *constructional* levels.

A. Intensional level description

Our intensional model defines two main constructs for building schemes: the *Page-Class* construct describes a set of similar pages of the same category, while the *Request-Class* construct models a set of homogeneous *requests* (GET or POST) to navigate from the pages of one Page-Class to those of a consecutive Page-Class. In our model, *Request-Classes are typed*, in the sense that each Request-Class specifies the Page-Class that it leads to.

The above concepts are illustrated in the graph of Figure 1: for our example site, the home page can be modeled by a singleton Page-Class HOME from which depart two Request-Classes, BYINITIAL and BYSECTOR, leading to two Page-Classes, INDEXBYINITIAL and SECTOR. INDEXBYINITIAL models the index pages grouping companies by initials while SECTOR describes index pages grouping companies by financial sector.

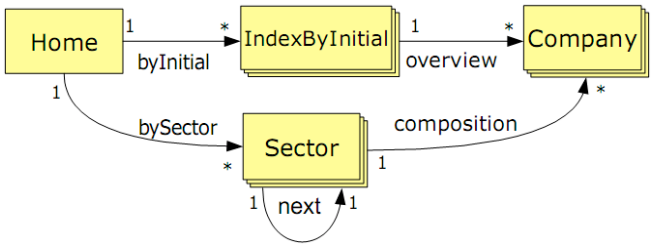

Figure 1. Intensional model example.

Both Page-Classes, by means of Request-Classes OVERVIEW and COMPOSITION respectively, lead to the last Page-Class COMPANY whose pages contain detailed information about companies, one company on each page. In particular, Request-Class NEXT models the link leading to another page instance of the same Page-Class SECTOR.

B. Extensional level description

An extensional listing of a Page-Class is a set of pages called Page-Class *support* that: (i) obey to a common HTML template and hence have similar structure; (ii) are about a common topic (each page represents a different instance of the same conceptual entity); (iii) are reachable by similar navigational paths, that is, by crossing pages that also correspond to predefined Page-Classes. Similarly, the extensional definition of a Request-Class is a set of requests called Request-Class *support* that: (i) lead to page instances of the same Page-Class; (ii) can be generated by clicking on comparable links or by different submissions of the same web form.

The two intensional constructs can be used together to build complex schemes, and to each scheme an extensional counterpart can be associated. That is, a scheme instance can be obtained by arranging the supports of every Page-Class and Request-Class involved into a graph that reflects the logical organization of the modeled site, where nodes represent page instances of a Page-Class in the scheme, while edges represent request instances of a Request-Class. We call an instance *empty* whenever every Page-Class (and hence every Request-Class) has empty support. An extensional graph for our example is partially shown in Figure 2.

C. Constructional level description

The constructional level in our model bridges the intensional and extensional descriptions by providing all the operative details needed to build the schema instances. Constructional elements are in fact the information that the

 ISBN: 978-1-61208-280-6

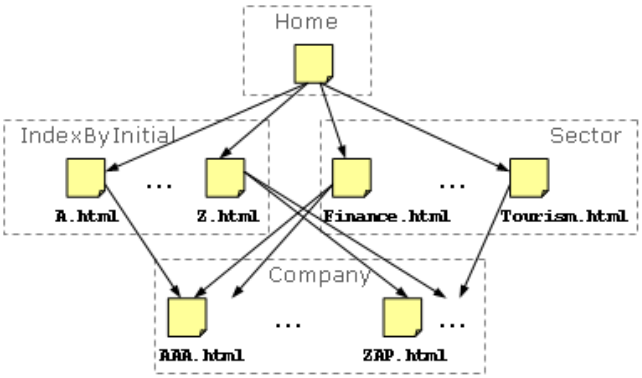

Figure 2. Extensional model example.

system needs to start from the entry page, determine the possible navigational paths to follow, proceed to subsequent pages, and extract the attributes of interest to the user. These elements consist of: (i) for the *entry* Page-Class, the set $E = \{e_1, \ldots, e_n\}$ of addresses of the pages in its support (typically the URL e of the page is sufficient); (ii) for each Request-Class r, a function er^r that builds its support, given a page instance of the Page-Class from which it departs. We call this function an extraction rule since it extracts requests from pages, e.g., the initials hyperlinks on the home page. Two other constructional elements, *assertion* and *wrapper*, are added to our model in order to enhance the Page-Class constructs. They are detailed in Section V.

III. Crawling Algorithm

Having defined our abstract model and how to build its instances, we explain how the crawler operates in this context. We first formalize the definition of navigational path, a main component for our crawling algorithm. On the intensional level, a navigational path $\mathcal{N} = (P_0 \cdot r_0^1 \cdot P_1 \cdot r_1^2 \cdot \ldots \cdot r_{h-1}^h \cdot P_h)$ is a sequence of Page-Classes and Request-Classes such that each Request-Class r_k^{k+1} is outgoing from Page-Class P_k and ingoing to Page-Class P_{k+1}, $k = 1..h-1$. It essentially corresponds to a path in the graph associated with the scheme. For our purposes, the interesting navigational paths start from an *entry* Page-Class (HOME, in our example) and reach a *target* Page-Class containing the relevant data (COMPANY). On the extensional level, the navigational path consists of *navigational trees*, where each tree is rooted at one entry page of the site and its descendants represent the pages visited by following the various requests in the support of the Request-Classes traversed.

A generalization of the user's sample browsing activity derives the navigational path definition as a sequence of constructional Page-Class and Request-Class elements. The crawler then finds all the navigational tree instances of this given path in the web site, in order to eventually download -strictly- all the pertaining pages.

Algorithm 1: Crawling algorithm based on the abstract model for large web sites

Input : A navigational path N = $(P_0 \cdot r_0^1 \cdot P_1 \cdot r_1^2 \cdot \ldots \cdot r_{h-1}^t \cdot P_h)$;
addresses $E = \{e_1, \ldots, e_n\}$ of P_0's pages;
a set of extraction rules $\{er^{r_k^{k+1}} | k = 0, \ldots, h-1\}$

Output: The navigational trees T instances of N.

Let $T = \{t_1, \ldots, t_n\}$ **be** an empty instance of N;
foreach $e_i \in E$ **do**
  add the page with address e_i as root of t_i;
for $k = 0$ **to** $h-1$ **do**
  foreach *page* $p \in \texttt{support}^{t_i}(P_k)$ **do**
    foreach *request* $r \in er^{r_k^{k+1}}(p)$ **do**
      Let d **be** the page obtained by means of r;
      insert d as a child of p in t_i;
      insert d in $\texttt{support}^{t_i}(P_{k+1})$;
return T;

$support^{t_i}(P)$	support of P in the navigation tree t_i
$er^{r_k^{k+1}}(p)$	set of requests associated with the r_k^{k+1} and extracted by applying the extraction rule er on page p

To do so, a crawler operates according to Algorithm 1. It starts with an input navigational path and an empty set of the corresponding navigational tree instances. Then it incrementally builds each instance by first adding a root page from the support of the entry Page-Class P_0, using to that effect the input addresses E. Subsequently, for each page p in the Page-Class under consideration, the algorithm uses $er^r(p)$ to build the support of the outgoing Request-Class r, that is, to extract on p the actual set of requests corresponding to r. These requests are sent to the web server in order to obtain new pages. The latter are added to the support of the Page-Class that constitutes the destination of the processed Request-Class. The crawler continues by iteratively picking each page p from the support of an already populated Page-Class and following its corresponding requests, once extracted. It thus incrementally builds other instances and the algorithm stops when all the requests have been sent.

IV. Specification of Extraction Rules

To perform wrapping and crawling tasks in an interrelated mode, our system infers extraction rules. These rules are used in crawling to locate the requests to be followed, and in wrapping to extract the relevant data from the downloaded target pages.

In Section II, we have already discussed extraction rules for requests. We revisit the previous definition to also

 ISBN: 978-1-61208-280-6

model wrappers for extracting relevant data from a page. An extraction rule er is then more generally defined as a function that locates and returns a set of strings s_i from the HTML code of page p: $er(p) = \{s_1, \ldots, s_k\}$.

In order to infer extraction rules, our algorithm takes as input a few positive examples provided by the user, highlighting the strings to extract (links or attributes). Eventually, only the rules compatible with collected user feedback are kept. Defining the class of inference rules to be generated constitutes then an important design choice. On one hand, a large and expressive class is more likely to include all the useful rules but may require many initial samples and a lot of user interaction before the correct rules are discerned. On the other hand, a limited and concise class of extraction rules requires less samples, and therefore less user interaction, but is also less expected to include correct extraction rules that are valid on all the pages. We address this tradeoff by opting for extraction rules based on XPath expressions and obtained from the union of three simple classes:

ABSOLUTE containing all absolute XPath expressions with positional predicates in each location step, generalized when a superset of several samples is needed

URL-REGEX obtained by filtering the rules in the ABSOLUTE class with simple regular expressions, and used mainly for extracting links where the URL value obeys to a general pattern

LABELED consisting of relative XPath expressions that make use of a fixed node considered part of the template to locate the string to extract

These classes proved to work on more than one hundred real web sites with very good performance results, while maintaining simplicity and requiring very limited user effort to discern a correct rule.

V. WRAPPER AND ASSERTION CONSTRUCTS

As mentioned, wrappers and assertions are constructs used in our constructional model to enhance the system's definitions of crawling and data extraction activities.

Wrappers are tools for extracting data values on web pages. Data wrappers generated by our system consist of rules taken from the LABELED class as they are less sensitive to variations on the page. They require that the node chosen as a reference be present in most of the target pages while being as close as possible to the data node. We associate the wrapper element with the Page-Class construct so that when crawling to build a Page-Class support, if a wrapper definition is encountered, it is executed instantly on the downloaded page.

Assertions are Boolean predicates over web pages. They are formulated as a set of XPath expressions locating valid template nodes, and a page is said to satisfy an assertion if and only if it agrees with all the XPath expressions associated with the template of its respective Page-Class. As we relax extraction rules and allow them to extract a superset of the correct links, the system can detect and discard any *false positive* pages crawled by checking whether their template matches or not all the established assertions. Consequently, assertions replace the need for adding more expressive classes of extraction rules when the crawler's performance is not satisfactory, which means fewer rules produced in response to the examples provided by the user, and fewer subsequent examples required from the user to discern the correct rules [6].

VI. SAMPLE SET SELECTION

The input required for the semi-automatic generation of the crawling programs and annexed wrappers is provided through interaction of a non-expert user with our system through a browser-based interface. For the model generation, the system tracks user navigation and derives an *internal schema* from the user's browsing activity as per the model listed in Section II, with the Page-Classes and Request-Classes of the web site along with the needed extraction rules. As for wrappers inference, the system generates extraction rules for the data selected by the user and evaluates them with the active learning module of the underlying inference engine. Then for any page with uncertainty, be it for the template or for the extracted values, the user is prompted to confirm the results, correct them, or even discard the page.

A. Sampling Problem

In order to produce correct and effective extraction rules, automatic and semi-automatic inference methods require samples with a certain quality of representativeness [6] that inexpert users cannot provide. Therefore the sample pages chosen by the tool to collect user feedback need to be representative of their corresponding Page-Classes. In addition, the navigational paths linking them together need to cover a rather wide variety of the possible paths in the selected informative domain. As a result, selecting sample pages introduces what we define as the *Sampling Problem*. This problem, often neglected, consists in determining which sample pages to choose in order to have "good" positive examples and guide an inexpert user in the inference process. Consider the following two scenarios:

Example 1: the downloaded sample pages all belong to one specific subset; say company pages from the Agriculture sector in our running example. Such pages would all contain the token "Sector: Agriculture".

Example 2: the downloaded sample pages share a particular variation in the template, such as the pages of top-ranked companies in our example containing a table with statistical data. The headers of this table constitute tokens that are not shared by other pages that do not rank first in their respective sector.

In these examples, sample pages may not be representative of all their Page-Class instances, as they contain some

 ISBN: 978-1-61208-280-6

specific tokens that are not present in the remaining target pages. This can in turn affect negatively the crawler, were wrong template assertions derived from these tokens, as well as the data extraction process, were any relative rules to be built around these tokens.

B. Resolution

The manifestations of the sampling problem are: (i) in characterizing the valid pages template, (ii) in collecting good navigational path instances, (iii) and in generating accurate extraction rules for the data of interest. In the first case, the problem is to generate a template characterizing only relevant web pages and discard "false positives". A template consists of a set of valid tokens that are present in most of the sample pages at the same XPath location. The second case relates to the need of covering the domain's navigational paths with minimum bias, and is addressed by tracking and generalizing the user's navigation at each step. The third case can then be resolved having collected a diversified page sample and derived a valid template. We propose the following approach to characterize valid page templates based on a statistical analysis of page contents and a learning algorithm requiring limited user effort:

- at each navigation level, consider all the pages obtained by generalizing selected links or form requests
- from this set of pages, which can be very large, choose a fixed percentage of random sample pages to download
- analyze tokens occurrence and data extraction results on the sample pages to train the classifier
- apply uncertainty sampling techniques to select query pages to propose to the user for feedback
- update tokens set, assertions, and extraction rules from user feedback

With this technique, our system is able to collect and use a representative subset from the site pages to infer performing wrappers and crawlers relevant to the user's needs. At execution time, the constructional details, as described in Section II-C, recorded in XML format, are used to guide the interleaved crawling and wrapping on the large web site.

VII. Experiments

In this section, we summarize our experiments conducted with the system prototype, called CoDEC, implemented as a Mozilla Firefox extension. We used our prototype for generating specifications and executing them on real web sites to analyze the performance on a wide variety of heterogeneous topics, page templates, and navigational paths. We evaluate our experiments by computing the F-measure as

$$F = 2\frac{P * R}{P + R} \quad (1)$$

This value ranging between 0 and 1 reports the harmonic mean of the precision *P* and recall *R*. For crawling, it evaluates the set of downloaded pages as compared to the set of actual target pages. Whereas for wrapping, the F-measure evaluates the values extracted by the generated rules with respect to the correct set of attribute values on the target pages. Larger F-measure values imply better results.

Table I sums up the experiment results of our crawling techniques on 100 different sites belonging to various domains. The simplest class ABSOLUTE was sufficient to extract most of the links leading to target pages. In few cases, where other links leading to non-target pages were located very close to the correct links, URL-REGEX extraction rules improved the precision by discarding the links that do not match an inferred URL pattern. One crawling limitation was the inability to discard target pages with the same template but different semantic entities, such as pages for coaches downloaded with those of players in a soccer web site.

Table II summarizes the results of testing our wrapping techniques on a subset of the previous sites, where we manually chose some attributes to extract. Optional attributes are those that occur only on a fraction of the pages in the same Page-Class. When several attributes on a page are optional, their inconsistent occurrence and location are likely to cause extraction rules to fail, so we observe low recall for these tests. Moreover, valid extraction rules cannot be generated when poor HTML formatting affects user selection of data. All in all, the overall results collected on the different web sites support the effectiveness of our tool.

VIII. Related Work

Data extraction for structured web sources has been widely studied, as shown in the various surveys on wrapper generation [7], [8], [9], and the several works on wrapper specification, inference and evaluation of extraction rules (such as [10]). However, our approach focuses on how to contextually specify, create, and execute simple and interrelated crawling and wrapping algorithms, rather independently from the underlying inference mechanisms of extraction rules. A few works [11], [12], [13] have addressed the automatic discovery of content pages and pages containing links to infer a site's hierarchical organization. These approaches mainly aim at finding paths to all generic content pages within a web site, often with some limitations of specific hypotheses to allow automation. In contrast, we aim at semi-automatically gathering pages from a selected class type of interest to a user, with a minimal human effort. The work by [14] partially inspired our URL-REGEX class, as they use URL patterns to specify crawlers in their GoGetIt! system. However, our experiments show that many web sites cannot be crawled in this restrictive way alone. In addition, they adopt a breadth-first strategy to compare the site's pages DOM trees with a provided sample page, while our system works on a small set of sample pages presented to a user for feedback. Sellers A. et al [15] propose a formalism for data extraction by describing and simulating user interactions on dynamic sites. However, the declarative

 ISBN: 978-1-61208-280-6

Table I
CRAWLING RESULTS SUMMARY

Total # of web sites	Total # of pages crawled	Overall F-measure
100	208769	0.99

Table II
WRAPPING RESULTS SUMMARY

Domain	# of web sites	# of pages	# of attributes	# of optional attributes	Overall F-measure
FINANCE	3	610	5	0	1.00
BASKETBALL	4	2310	7	1	0.99
SOCCER	4	1214	5	0	0.99
MOVIES	3	3046	7	6	0.92

specifications are defined by an expert programmer and not derived from an actual user's navigation. Finally, [16] implement a web scale system for data extraction that generates extraction rules on structured *detail* pages in a web site. They apply extensive calculations for clustering pages according to template similarity and rely on several user inputs for annotation and rule learning. Their work is different in scope from ours since we work on synergic crawling and wrapping that can cover various Page-Classes in a web site while deriving information from the user's browsing activity.

IX. CONCLUSION AND FUTURE WORK

We presented in this article a new semi-automatic methodology for synergic crawling and wrapping in the scope of information retrieval. Our contributions can be summarized by: (i) a formalism to specify interleaved crawling programs and wrappers concurrently over structured web sites; (ii) the introduction of the *Sampling Problem*, which illustrates how randomly chosen samples can be biased and negatively impact the inference tasks; (iii) an approach to mitigate the effects of the sampling problem by requiring minimal effort from an inexpert user; (iv) and an experimental evaluation to validate our proposed techniques. Our experiments revealed encouraging results, and can be further improved with the potential inclusion of semantic assertions and a mechanism to deal with any optional attributes with changing location on the page. The quantification of user effort and its variation with the learning implementation parameters is still the subject of an ongoing examination.

REFERENCES

[1] C. Bertoli, V. Crescenzi, and P. Merialdo, "Crawling programs for wrapper-based applications," in *IRI*, 2008, pp. 160–165.

[2] Y. Zhai and B. Liu, "Structured data extraction from the web based on partial tree alignment," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 18, no. 12, pp. 1614–1628, 2006.

[3] S. Chakrabarti, M. Van den Berg, and B. Dom, "Focused crawling: a new approach to topic-specific web resource discovery," *Computer Networks*, vol. 31, no. 11, pp. 1623–1640, 1999.

[4] M. Jamali, H. Sayyadi, B. Hariri, and H. Abolhassani, "A method for focused crawling using combination of link structure and content similarity," in *WI 2006*. IEEE, 2006, pp. 753–756.

[5] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy, "Google's deep web crawl," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1241–1252, Aug. 2008.

[6] V. Crescenzi and P. Merialdo, "Wrapper inference for ambiguous web pages," *Applied Artificial Intelligence*, vol. 22, no. 1&2, pp. 21–52, 2008.

[7] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira, "A brief survey of web data extraction tools," *SIGMOD Record*, vol. 31, no. 2, pp. 84–93, 2002.

[8] S. Flesca, G. Manco, E. Masciari, E. Rende, and A. Tagarelli, "Web wrapper induction: a brief survey," *AI Commun.*, vol. 17, no. 2, pp. 57–61, 2004.

[9] C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan, "A survey of web information extraction systems," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 10, pp. 1411–1428, 2006.

[10] J. Carme, M. Ceresna, and M. Goebel, "Web wrapper specification using compound filter learning," in *IADIS*, 2006.

[11] V. Crescenzi, P. Merialdo, and P. Missier, "Clustering web pages based on their structure," *Data Knowl. Eng.*, vol. 54, no. 3, pp. 279–299, 2005.

[12] H.-Y. Kao, S.-H. Lin, J.-M. Ho, and M.-S. Chen, "Mining web informative structures and contents based on entropy analysis," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 1, pp. 41–55, 2004.

[13] Z. Liu, W. K. Ng, and E.-P. Lim, "An automated algorithm for extracting website skeleton," in *DASFAA*, 2004, pp. 799–811.

[14] M. L. A. Vidal, A. S. da Silva, E. S. de Moura, and J. M. B. Cavalcanti, "Gogetit!: a tool for generating structure-driven web crawlers," in *WWW*, 2006, pp. 1011–1012.

[15] A. J. Sellers, T. Furche, G. Gottlob, G. Grasso, and C. Schallhart, "Taking the oxpath down the deep web," in *EDBT*, 2011, pp. 542–545.

[16] P. Gulhane, A. Madaan, R. Mehta, J. Ramamirtham, R. Rastogi, S. Satpal, S. Sengamedu, A. Tengli, and C. Tiwari, "Web-scale information extraction with vertex," in *ICDE 2011*. IEEE, 2011, pp. 1209–1220.

 ISBN: 978-1-61208-280-6