Multi-pattern wrappers for relation extraction from the Web

Benjamin Habegger[1] and **Mohamed Quafafou**[1]

Abstract. Numerous sources of data are available on the web, for instance, product catalogs, multiple directories, conference and event sites, etc. The extraction of information from the content of these sources is a challenging problem and a hard task since they are heterogeneous and dynamic. This paper presents a new method for extracting wrappers and relations from the web using both page encoding and context generalization. Its starting point is a training set of instances of the relation the user wishes to extract. Multiple patterns are then extracted considering the occurrences of the input instances in the data source. The generalization of these patterns allows us to identify new instances of the relation in the same data source. The main features of this method are its simplicity, genericity and robustness faced to the diversity of sources. Its efficiency is shown by the experimental results on different sources, i.e., search engines, shopping, product catalogs, paper listings, etc.

1 INTRODUCTION

With the growth of the web, many data sources have been made accessible through the web. These go from search engines, online product catalogs, advertisements, to bibliographic listings. The potential use of such information sources has lead to research on how to encapsulate such sources to make them available to computer programs and lead to the introduction of programs called wrappers. These programs allow to query and retrieve the results in a manner computer programs can use them, ie. with well defined queries and structured results. Faced to the constant evolution of the web theses wrappers need to be maintained and renewed often. To face this problem it became interesting to build wrappers automatically or semi-automatically.

In this paper we present a new approach to wrapper construction. Our method represents a wrapper as a set of patterns. We consider that the user wants to extract a relation from a given document or set of documents. Furthermore while other example-based approaches usually use tagged pages as examples, we only require the user to give a set of example instances of the relation he/she wants to extract. The contexts of these examples are then extracted and generalized in order to build the set of patterns allowing to extract the other instances of the relation. Wrappers are easily generated, by only giving a small set of example instances, and can be built for many different types of web sites.

This paper is organized as follows. Section 2 presents the basic concepts used and the work done by others in the domain of wrapper generation for the web. Section 3 gives a formalization of the problem and introduces the notations we use. In section 4 we describe our approach to the wrapper extraction problem and the system we have built. An evaluation of the system is given in section 5. Finally we conclude and present future work in section 6.

[1] IRIN : Institut de Recherche en Informatique de Nantes, University of Nantes, France. {habegger,quafafou}@ irin.univ-nantes.fr

2 RELATED WORK

Many researchers have tackled problems related to wrappers or more generally information extraction from the web. These go from toolkits to aid in building wrappers manually and wrapper induction to the extraction of relational data from large collections of web documents or extraction of the symbolic knowledge.

Most wrapper construction methods are automatic or semi-automatic and based on inductive learning [10, 8, 12]. Structure discovery or extraction can also be used to resolve the wrapper generation problem [1, 3]. In this case, PAT-trees can be used to find maximal prefixes of the input HTML string [3]. When considering the problem of relation extraction or mining, an approach is to consider that their exists a duality between patterns and relations [2, 15]. Our point of view focuses on these three main directions to wrapper construction.

Wrapper induction Wrapper induction was first introduced by N. Kushmerick [10]. The input of his method is a set of fully tagged documents from a given web data source and the output is a wrapper for that source. The general algorithm is to find a delimiter, extract until another delimiter, find the next delimiter, etc. Different classes of wrapper are presented and an evaluation on a per correctly wrapped page basis are proposed.

Due to the general wrapper algorithm basis, this method does not always lead to the generation of a wrapper. For example $Av_1BxCyCv_2D$ is the general form of the text from which we want to extract the instances (v_1, v_2) of a relation $R(A_1, A_2)$ and where A, B, C are constant text while x, y a arbitrary text. The method will be incapable of constructing a valid left delimiter for the second attribute of the relation (ie. all possible candidates are suffixes of C and all suffixes of C are not proper suffixes of $xCyC$ [9] for details validity of delimiters). Moreover, even though corroboration is introduced, it remains a tedious task to tag by hand all the instances contained in every example document. Furthermore the proposed evaluation gives poor results to wrappers favoring robustness to perfectness.

Relation discovery This problem, to our knowledge, first appeared in [2]. The method presented is used to extract relations from a very large set of web pages. It relies on the hypothesis that there exists a duality between a relation and the patterns allowing to extract the occurrences of its instances. The example problem in [2] is

to find instances of the (author,book) relation by giving a small set of examples. With 5 example instances 199 occurrences were found in a repository of about 5 million web pages. The generalization of the contexts in which the occurences were found generated 2 patterns. Once applied these patterns, 4047 total instances were found.

While sufficient when used on a very large number of documents, the context generalization used can not be applied to wrapper construction since it is very limited. Given two contexts, only the head and tail parts are generalized, keeping the middle over-specific and therefore making the method very limited to extract wrappers.

Structure extraction The method proposed by [3] to resolve this problem is done by searching for a structure, or logical organization of the data using an automatic and unsupervised method. After having encoded the HTML input document by abstracting content text and coding tags the algorithm presented uses PAT-trees and a maximal prefix search method to build patterns. The extracted structures allow the construction of a pattern capable of extracting the occurrences at similar logical positions in the document. A document can have multiple structures and therefore heuristics to reduce the number of patterns are applied.

A problems with this approach are that the encoding used does not allow to consider structure found in content text. Typically, in the example of figure 1, this method would not be able to separate the prices in different currencies. Furthermore though the structure discovery process is unsupervised, the user still has to be consulted afterwards, leaving him the choice between patterns which may not have much sense to him/her (for example `<DT>TEXT</DT><DD>TEXT<BR>TEXT</BR></DD><BR>`).

Other approaches and problems Many other wrapper induction methods also exist such as [8] based on finite state automata, and [12] based on embedded catalog trees. Researchers have also tackled problems related to wrappers, such as their description using XML [11, 14], building knowledge-based wrappers [6, 14], or extracting symbolic knowledge from the web [4].

The work presented in this paper handles the task of extracting a wrapper as is done in [9] but by using patterns built by generalizing contexts. In that, it overlaps with the method proposed by [2], but the generalization we use enables us to have patterns covering more occurrences on a single page, and therefore is more appropriate to wrapper construction. It also uses encoding techniques inspired by [3] but without encoding text to allow existing non-html structure to be used.

3 PROBLEM FORMALIZATION

From a database point of view a wrapper is used to encapsulate a data source to make it available to applications [13]. This means having a shared query language and result format for all the data sources. In the context of the web the term wrapper can have the same meaning [7] but is also often used to designate the procedure of extracting and returning formatted data ignoring the querying problem [10, 5]. In this paper the term wrapper will refer to such a procedure. A *wrapper* for the web can therefore be seen as a function allowing to *extract structured data contained in a set of a partly but similarly structured web pages*.

The problem we resolve in this paper is to build a wrapper for a data source which can extract a relation it contains. The description of the relation to extract is given in the form of a set of example instances. To resolve this problems some terminology needs to be introduced.

- An *instance* refers to the tuple of values of a relation.
- An *occurrence* of such an instance refers to its appearance within its context in a web page. Formally the occurrence o of an instance t can be represented by the positions of the values of t.
- The *context* of an occurrence is the text surrounding its values. Given a document d, and an instance $t = (v_1, \ldots, v_n) \in R$ of the target relation R the context of an occurrence of t in d is an $n+1$ tuple $c = (c_1, \ldots, c_{n+1})$, where is c_1 is all the text before the occurrence, c_{n+1} all the text after the occurrence, and the $c_k, k \in [2, n]$ are the text between the values v_{k-1} and v_k.
- A *pattern* is a general description of the contexts allowing to extract pieces of text which correspond to occurrences of instances we want to extract. It is made up of a sequence of parts which represent text to be match in the order of the sequence. Each part of a pattern is itself divided into tokens. A token can either be a the code of a tag (see section 4), a string of text which should match exactly in the document or a joker which matches any substring of content text (ie. non-tag text).

Formally a wrapper $\mathcal{W}$ can be considered as a function having as its input a document d belonging to or generated by a source $\mathcal{S}$ and as output the subset of instances I of a relation R which appears in d. For short, we will consider that the source is simply a set of documents D sharing the same structure, and that our relation is the set of instances I_D appearing in the set of documents D to be extracted. A document can be seen as a string of text constructed over an alphabet Σ. The problem we want to solve is then to build a program capable of extracting from D all the instances of I_D given a subset E of I_D.

The type of wrapper we want to build are multi-pattern wrappers. A multi-pattern wrapper is a wrapper which can be represented as a set of patterns P each allowing to extract subsets of I_D from the documents in D according to a set of instances of R. Each pattern of P is said to cover a subset of instances of I_D. A multi-pattern wrapper is complete when the generated instances do cover all the instances of R. It is said consistent when no more than the instances of R are covered. Respectively recall and precision allow to measure the completeness and consistency of a multi-pattern wrapper.

4 WRAPPER DISCOVERY METHOD

The problem we consider can be seen as the wrapper construction problem, but the approach we take is slightly different from the one described in section 2. As in the general problem we consider a static or dynamic web data source whose pages contain structured data. *However we do not consider that the wrapper should extract all the structured data, but only the instances of a given relation R.* This relation is expressed by giving examples of its instances. The objective is then to build a wrapper capable of extracting the given relation R from the pages generated by the source $\mathcal{S}$. Since we want to extract a relation we consider that the data has a tabular form and that the values are in the same order as the examples. These two restrictions are not real limitations since we can extract multiple relations. Complex data structures can be obtained by the combination of relations and each ordering can be represented by a different relation.

The html source given in figure 1 gives an example document containing a relation "Prices(Food,PriceFF,PriceEUR)" given in table 4.

Our method is based on encoding and generalization. Encoding is used to simplify the input document to make the inherent structure more apparent by cleaning up unnecessary pieces such as white

```
<html>
  <body>
    <h1>Coffee Machine Prices</h1>
    <table>
      <tr><td>Coffee</td>
          <td>0.40 EUR (2.96 F)</td></tr>
      <tr><td>Soda</td>
          <td>0.75 EUR (4.92 F)</td></tr>
      <tr><td colspan="2"><hr></td></tr>
      <tr><td><a href="cakes.html">Cake</a></td>
          <td>0.50 EUR (3.28 F)</td></tr>
    </table>
  </body>
</html>
```

Figure 1. An example web document source

Food	Price in EUR	Price in FF
Coffee	0.40	2.96
Soda	0.75	4.92
Cake	0.50	3.28

Table 1. Example relation to be extracted

spaces, indenting, carriage-returns, low-level tags, and by removing ambiguities caused by characters appearing both in content and tag text. Pattern generalization is used on the encoded input to construct a wrapper capable of extracting a relation based on the contexts of given example instances. The general wrapper extraction algorithm is presented below, and the details of the encoding method and pattern generalization process are detailed afterwards.

4.1 Wrapper extraction algorithm

Algorithm 1 Wrapper extraction algorithm

Input: An HTML document D and an example set E of instances of a relation R
Output: A multi-pattern wrapper for R
$enc \leftarrow encode(D)$
for all $e \in E$ **do**
  $C \leftarrow C \cup extract\_contexts(E, enc)$
end for
$C_f \leftarrow \emptyset$; $C' \leftarrow \emptyset$
while $C \neq \emptyset$ **do**
  $c \leftarrow pop\_any(C)$; $c_g \leftarrow$ '-'
  while $C \neq \emptyset$ **do**
    $c' \leftarrow pop\_any(C)$; $c^g \leftarrow gerneralize(c, c')$
    if $c^g \neq$ '-' **then**
      $C' \leftarrow C' \cup C \cup \{c^g\}$; $C \leftarrow \emptyset$
    else
      $C' \leftarrow C' \cup c'$
    end if
  end while
  if $c^g \neq$ '-' **then**
    $C_f \leftarrow C_f \cup \{c\}$
  end if
  $C \leftarrow C'$; $C' \leftarrow \emptyset$;
end while
return C_f

The extraction process has as its input a document d containing a set I of instances of a relation R. Furthermore, an example subset of the set of these instances E is also given. Its output is a multi-pattern wrapper for the set of instances I. To generate this wrapper the input document d is first encoded. The resulting encoded string is then searched to extract the contexts of the occurrences of the example instances of E. Once the different contexts have been extracted, the contextes are generalized by pairs. When the generalization of a pair succeeds, the pair is replaced by the new pattern which is kept for furthur generalization with other contexts. This general wrapper construction process is described in algorithm 1. This algorithm supposed that the generalization function $generalize$ is order independant. Such a function will is given in subsection 4.3. The function pop_any takes chooses and removes and element from its input set. No hypothesis is made on the manner to choose such an element.

4.2 Encoding scheme

The first step of our approach is to encode the input HTML document. Encoding is used to disambiguate the characters, consider tags as only unique tokens where textual parts need to be considered as independent characters, and also ease the context extraction and generalization. The result of encoding is a string of tokens where each token is either a tag or a string of content text. In figure 2 is given the coded form of the page source given in figure 1. The bold number correspond to the encoded tags, for example, **1**,**8** and **10** are, respectively, the `<html>`, `</td>`, and `<td colspan="2">` tags in the original document. A simple heuristic on tags can be used. This is

1-2-3 Coffee Machine Prices**4-5-6-7**Coffee**8-7**0.40 EUR (2.96 F)**8-9-6-7**Soda**8-7**0.75 EUR (4.92 F)**8-9-6-10-11-8-9-6-7**Cake**8-7**0.50 EUR (3.28 F)**8-9-12-13-14**

Figure 2. Example encoded page

done by considering different levels of tags. It is specially useful in eliminating text level tags which usually do not contain structural information and can be a barrier in the discovery of patterns. The different levels of tags can be separated as : (1) document or high level tags : *html*, *body*, *head*, etc. (2) paragraph or middle level tags : *br*, *p*, *div*, etc. (3) text or low level tags : *b*, *i*, *em*, *a*, etc.

During the encoding process low-level tags are removed. They are rarely necessary in the pattern construction process, while often are the source of exceptions. For example, in figure 1 the anchor tag *a* allows the user to see the list of cakes, but also makes the context of "Cake" different from that of "Coffee" or "Soda" even though in the relation we want to extract they are values of the same attribute.

The other types of tags are encoded according to their full text string and not only according to their type. This is due to the fact that a same tag with different present attributes and attribute values are structural clues. In the example of figure 1, the *td* tags are the common context of the values the attributes of the relation to be extracted except in the case where it has an attribute *colspan="2"* in which case it is just used for presentation matters (ie. to add a separating horizontal rule between drinks and food).

4.3 Pattern generalization

The generalization of a pattern is done by separately generalizing each part of the pattern. Each part can itself be decomposed into a list

Algorithm 2 Pattern generalization algorithm

Input : Two n-part patterns $T = (T_1, \ldots, T_n)$ and $T' = (T'_1, \ldots, T'_n)$
Output : A generalized pattern $T^g = (T^g_1, \ldots, T^g_n)$
for all $k \in [1, \ldots, n]$ **do**
  Let $s_1 \cdot \ldots \cdot s_l = T_k$
  Let $s'_1 \cdot \ldots \cdot s'_l = T'_k$
  $T^g_k = G(T_k, T'_k)$
  for all $i \in [1, \ldots, l]$ **do**
    if $s_i = s'_i$ **then**
      $s^g_i \leftarrow s_i$
    else if $tag(s_i)$ or $tag(s'_i)$ **then**
      $s^g_i \leftarrow$ '-'
    else
      if $i = 1$ **then**
        $cs \leftarrow commun\_suffix(s_i, s'_i)$
        $s^g_i \leftarrow length(cs) = 0 :$ '-' ? '*' $\cdot\ cs$
      else if $i = l$ **then**
        $cp \leftarrow commun\_prefix(s_i, s'_i)$
        $s^g_i \leftarrow length(cp) = 0 :$ '-' ? $cp \cdot$ '*'
      else
        $s^g_i \leftarrow$ '*'
      end if
    end if
  end for
end for
$T^g_1 \leftarrow keep\_nodash\_suffix(T^g_1)$
$T^g_n \leftarrow keep\_nodash\_prefix(T^g_n)$
if $has\_dashes(T^g)$ **then**
  $T^g \leftarrow$ '-'
end if

of tokens which are each either a tag or a string of text which can not be subdivided. Two parts are generalized when all of their tag *tokens* at the same position *match exactly*. This means that we keep tokens as is and they can only be removed either in the cleaning process as low-level tags or in the head and tail parts of the pattern as described below. The text parts, either match and are kept as is to make up the corresponding generalized token or abstracted into a new virtual tag "text". An exception to this rule is made when considering first and last text tokens which we do not want to be generalized (the border limit of the values would be lost).In the case of a first (resp. last) text token we only allow a generalization when we can find a common prefix (resp. suffix). As they are a bit different, the head and tail parts are handled separately ($i = 1$ or $i = l$ in algorithm 2). For the head (resp. tail) parts we only need to have a common generalized suffix (resp. prefix). Therefore we only need last (resp. first) part to be generalized. The preceding (resp. following) parts are kept when they match (generalized text is not kept). The pattern generalization method we used is given in algorithm 2.

It is important to note that not only the head and tail parts of a pattern play a role in identifying an occurrence. The easiest way to see this is that if the head and tail *were* sufficient to delimit an occurrence, it wouldn't be possible to find an occurrence of the head and tail delimiters in the middle parts. This is why searching for a general pattern can not be divided into the search of local delimiters. (The wrapper classes proposed by Kushmerick did not consider such general properties see section 2). Another aspect of generalization, is the fact that patterns can consist of tags, but may also contain free textual parts. This can be asily seen in the example given in figure 1. By considering only tags it is impossible to separate prices in French Francs and in Euros. But by letting the textual parts "EUR (" and "F)" be delimiters this is made possible. This is why text cannot be totally encoded when looking for the structure of the document since it can also contain such structural information (In IEPAD these types of delimiters can not be detected since they were abstracted into a single text token). Our pattern generalization scheme takes all these aspects into account.

Context of $('Coffee', '0.40', '2.96')$
1-2-3 Coffee Machine Prices**4-5-6-7**
8-7
EUR (
F)**8-9-6-7**Soda**8-7**0.75 EUR (4.92 F)**...**

Context of $('Soda', '0.75', '4.92')$
...70.40 EUR (2.29 F)**8-9-6-7**
8-7
EUR (
F)**8-9-6-10-11-8-9-6**Cake**8-7**0.50 EUR (3.28 F)**...**

Generalized pattern
6-7
8-7
EUR (
F)**8-9-6**

Figure 3. Pattern generalization (each line separated the parts of the pattern)

This is done by generalizing separately each part of the pattern. To do the generalization of the two patterns, only one simultaneous pass through the different parts of the pattern is needed. This means that in the worst case (when a generalization exists) the generalization is done in linear time.

5 EVALUATION AND EXPERIMENTAL RESULTS

We have evaluated our method on three different types of data sources : search-engines, product catalogs and bibliography listings. The separate results for each of these parts are presented first and we then discuss them from a general point of view. Since the basis of our valuation is to count the number of instances retreived and not the number of pages wrapped correctly, it is difficult to compare our results to those of other methods. However the following results show that while it is simple, it is robust and efficient. Our method has a global pattern for the whole instances and thus does not consider that the values have to be delimited independantly. Therefore we can from a greater range of web sites than delimiter-based methods.

Search Engines Our method works quite well on the output of search engines as it is shown by table below. With only few examples the built wrapper can extract most of the results. For all the tested sites, except AltaVista, all extract instances are correct ones. In the case of AltaVista, all the correct instances are effectively extracted, but the generated wrapper has a pattern which is a too general. This lead to the extraction of wrong instances. However, the wrong instances generated are the same as the correct one except that there extra text is found at the end of the url attribute.

Shopping and product catalogs The results for the shopping sources are also very encouraging. All the built wrappers correctly extract the full page with a couple of instances as can be seen the table below. The missing instance in the "Alapage" result reveal recur-

Source	*Ex.*	*Inst.*	*Retr.*	*Recall*	*Accurracy*
AltaVista	4	50	67	1.00	0.74
Excite	4	10	9	0.90	1.00
Galaxy	2	9	9	1.00	1.00
Infoseek	3	15	15	1.00	1.00
Metacrawler	4	30	28	0.93	1.00
Savvysearch	5	12	11	0.92	1.00
Webcrawler	4	20	19	0.95	1.00
Google	3	50	45	0.90	1.00

Table 2. *Search Engine* extraction results (*Ex.*= number of examples, *Inst.*= number of instances in the document, *Retr.*= number of extracted instances.)

ring problem, which is that the last (resp. first) occurrences of a page are often missed. This due to the fact that hey do not share the same following (resp. preceding) context and that the generalized patterns for the other occurrences are to specific to handle that difference. As shown it can easily be corrected by given as examples the first and last occurrences of a page.

Source	*Ex.*	*Inst.*	*Retr.*	*Recall*	*Accurracy*
Alapage	3	12	13	0.92	1.00
Conforama	3	6	6	1.00	1.00
Amazon	7	25	19	0.76	1.00
Darty.fr	4	10	10	1.00	1.00
Fnac	5	13	13	1.00	1.00

Table 3. *Shopping* extraction results (*Ex.*= number of examples, *Inst.*= number of instances in the document, *Retr.*= number of extracted instances.)

In the case of Amazon many examples are need to build a wrapper which can correctly extract all the occurrences. This is because there are many different data presentations on the same web page. In the Amazon example we always have the Amazon price but sometimes is also added the list price, or special rebates with/without a summary price. Also, the relation we chose to extract did not include shipping time, creating more varieties of formats which have to be generalized, and therefore need to appear in the examples. In the worst case, when all the examples need to be given, the wrapper generated can still work on other generated pages.

Bibliography and article listings After having tested our approach on search-engines and catalogs we also wanted to try it to extract bibliographic data from sites such as http://dblp.uni-trier.de/. The results are presented in the table below. The data contained on such sites is much less structured than for the previous examples. Furthermore the poor structure still available is not contained in HTML tags but is rather in punctuation symbols. Also two different relations are present, article in proceedings entries, and book entries. It is the presence of these two types which generated the extra occurrences extracted in the DBLP example. The relation we wanted to extract was the following :

$$article(authors, title, book, conference, year, pages)$$

Unfortunately, the presence of book description on the pages lead to the extraction of erroneous instances.

6 CONCLUSION AND FUTURE WORK

This paper presented a new approach to wrapper and relation extraction form web data sources. The method proposed is based en

Source	*Ex.*	*Inst.*	*Retr.*	*Recall*	*Accurracy*
DBLP	3	21	27	1.00	0.78
liinwww	3	40	31	0.78	1.00

Table 4. *Other* extraction results (*Ex.*= number of examples, *Inst.*= number of instances in the document, *Retr.*= number of extracted instances.)

document encoding an pattern generalization. The relation to extract is defined by the user of the wrapper generator by given example instances of the relation. The proposed method can wrap up many different types of site, is robust to irregularities in format and makes use of structure contained in content text.

Many research direction can follow this work, such as trying to automatically generate examples, use already available instances instead of the examples or integrate the use of generic examples. Also more research has to be done to see how our method works on sources composed of similarly structured documents but containing a unique instance.

REFERENCES

[1] Brad Adelberg, 'NoDoSE–a tool for semi-automatically extracting structured and semistructured data from text documents', pp. 283–294, (1998).

[2] Sergey Brin, 'Extracting patterns and relations from the world wide web', in *Lecture Notes in Computer Science*, eds., Paolo Atzeni, Alberto O. Mendelzon, and Giansalvatore Mecca, volume 1590, pp. 172–183, Valencia, Spain, (March 1998). Springer. ISBN 3-540-65890-4.

[3] Chia-Hui Chang and Shao-CHen Lui, 'Iepad : Information extraction based on pattern discovery', in *Proceedings of the ACM WWW10 Conference*, pp. 681 – 688. ACM Press New York, NY, USA, (2001). ISBN 1-58113-348-0.

[4] Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew K. McCallum, Tom M. Mitchell, Kamal Nigam, and Seán Slattery, 'Learning to extract symbolic knowledge from the World Wide Web', in *Proceedings of AAAI-98, 15th Conference of the American Association for Artificial Intelligence*, pp. 509–516, Madison, US, (1998). AAAI Press, Menlo Park, US.

[5] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo, 'Roadrunner: Towards automatic data extraction from large web sites', in *The VLDB Journal*, pp. 109–118, (2001).

[6] X. Gao and L. Sterling. Semi-structured data extraction from heterogenerous sources, 1999.

[7] Jean-Robert Gruser, Louiqa Raschid, M. E. Vidal, and Laura Bright, 'Wrapper generation for web accessible data sources', in *Conference on Cooperative Information Systems*, pp. 14–23, (1998).

[8] Chun-Nan Hsu and Ming-Tzung Dung, 'Generating finite-state transducers for semi-structured data extraction from the web', *Information Systems*, **23**(8), 521–538, (1998).

[9] Nicolas Kushmeric, *Wrapper Induction for Information Extraction*, Ph.D. dissertation, University of Washington, 1997.

[10] Nicholas Kushmerick, 'Wrapper induction: Efficiency and expressiveness', *Artificial Intelligence*, **118**(1-2), 15–68, (2000).

[11] Ling Liu, Calton Pu, and Wei Han, 'XWRAP: An XML-enabled wrapper construction system for web information sources', in *ICDE*, pp. 611–621, (2000).

[12] I. Muslea, S. Minton, and C. Knoblock, 'Stalker: Learning extraction rules for semistructured, web-based information sources', in *In Proceedings of AAAI-98 Workshop on AI and Information Integration*. AAAI Press, (1998).

[13] M. Roth and P. Schwarz, 'Don't scrap it, wrap it! a wrapper architecture for legacy data sources', in *Proc. of the 23 VLDB Conference*, (1997).

[14] Heekyoung Seo, Jaeyoung Yang, and Joongmin Choi, 'Knowledge-based wrapper generation by using xml', in *IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, (2001).

[15] Neel Sundaresan and Jeonghee Yi, 'Mining the web for relations', in *Proceedings of the WWW9 Conference*, pp. 699–711. Elsevier Science, (2000).