Providing Robust Access to Data in Web Pages

Jerome Robinson

Department of Computer Science, University of Essex, Colchester, Essex, U.K.
robij@essex.ac.uk

Abstract.

Much useful e-commerce information is available on web pages, especially those created by queries to web servers. The problem for programs to use that information is how to 'screen-scrape' the data off the web page into machine-usable data structures. Wrappers for web data sources use knowledge of the page layout in order to extract data accurately. So they fail if page format changes. This paper describes a fast method for wrapper production and also a method to automatically detect page format change, before it causes data access to fail. The method works for pages that contain *collections* of items, such as lists, tables and hierarchical structures. It uses a representation of html documents, which makes *repetitive features* apparent. This provides fully automatic wrapper production for a class of web pages, and rapid interactive production for others.

1. Introduction

The task for an extractor program is to identify each data item embedded somehow in the html code. To achieve this it may rely on *landmarks*, which are distinctive features in the html code, to identify each data item. The page description in terms of landmarks is produced by analysing sample pages from the web site. The analyser is a program or person or both. We are interested in *collections* of objects in web pages. A collection is a set of items with similar appearance on the web page. The result set for a query is an example of a collection.

Searching for *repetitive features* in html code is one way to find collections. Previous researchers have sought repetition of substrings, or single tags which recur at approximately equal character-count intervals, or recurrent sub-trees in the html parse tree. Our method, using the recurrence pattern of 'tagSets' (explained below), is a more robust approach; not affected by irrelevant variations in the repetition pattern.

The model used to represent the web page is important. It can have a major impact on the effectiveness of page structure analysis. We use a new page modelling technique, creating a data structure called a 'tpGrid' to represent the html document. This structure is much more amenable to analysis than the raw html code. The following section explains how to create a tpGrid from an html document. That section is followed by an example using a real web page. Then in Section 4 a web page with hierarchically structured data is analysed and discussed.

2. Constructing a tpGrid

This section explains how to create a tagSet Progression Grid (tpGrid). An example from a real web page then follows this preliminary explanation. A tpGrid is an abstraction that reveals repetitive patterns of tags in an html document. It is derived from an html document by a sequence of steps, which are now explained.

The following html fragment represents the start of an imaginary html document:

```
<HTML><HEAD><TITLE>DEXA Conference 2004</TITLE>
  <META http-equiv=Content-Type content="text/html; charset=windows-
1252"><LINK
  id=css_menu href="ecweb04FrontPage_files/menu1.css" rel=stylesheet>
  </HEAD><BODY lang=DE style="BACKGROUND-COLOR: #d6dcdd" vLink=purple
link=blue>
  <P> Call for papers </P>
  <TABLE width="100%" summary="" border=0><TBODY><TR>
  <TD vAlign=top align=middle width=210>
  <IMG height=85 src="ecweb04FrontPage_files/dexa2004.gif" width=204>
  </TD><TD>International Conference on Data Warehousing and Knowledge
Discovery
  </TD></TR><TR><TD> etc, etc.
```

It can be seen that this is made up of sequences of tags between text strings. ('Text string' meaning any sequence of characters not inside angle brackets). Text strings are the main items that are visible in a web page on screen, as well as any pictures produced by <IMG> tags. If we call the text strings 'textStrings' and each sequence of contiguous tags a 'tagString' then the html document is a sequence: tagString - textString - tagString - textString - tagString .. etc. This structure applies to ANY html document. It can also be seen as a sequence: [tagString – textString] - [tagString – textString] - [tagString – textString] .. etc., and this pairing of each textString with its *preceding tagString* is a useful basis for automatic page structure analysis. If we number each pair in the sequence, then a web page or its html document *can be represented as a numbered sequence of* [*tagString – textString*] *pairs.*

After creating this numbered sequence of pairs, which we will call the *pair sequence*, the next stage in deriving the tpGrid is to summarize the tagString in **each** of the [tagString – textString] pairs, by ignoring tag attributes (which follow some tag names, inside the angle brackets). *For example,* the tag `<META http-equiv=Content-Type content="text/html; charset=windows-1252">` in the html code above becomes `<META>` when we ignore tag attributes. Each tag is reduced to its tag *name*. A **tagString** is converted to a **tagSet** by identifying the tag names in it and counting how many times each name occurs in the tagString. *For example,* the tagString:

```
  </P>
  <TABLE width="100%" summary="" border=0><TBODY><TR>
  <TD vAlign=top align=middle width=210>
  <IMG height=85 src="ecweb04FrontPage_files/dexa2004.gif" width=204>
</TD><TD>
```

found in the example above, becomes the sequence of tag names:

```
</P><TABLE><TBODY><TR><TD><IMG></TD><TD>
```

which becomes the tagSet:

```
</P - 1><TABLE - 1><TBODY - 1><TR - 1><TD - 2><IMG - 1></TD - 1>
```

The purpose of this summarization is to eliminate irrelevant over-specification, which can obscure the repetition patterns in the web page. The numbered [tag**String** – textString] sequence of now becomes a numbered sequence of [tag**Set** – textString] pairs, when each tag***String*** is represented by its corresponding tag***Set***.

Certain tagSets occur more than once in the web page, and therefore more than once in the numbered sequence of [tagSet – textString] pairs (i.e. in the *pair sequence*). The next stage in tpGrid production is to identify the all the *distinct* tagSets in the pair sequence. We identify each distinct tagSet by the position of its first occurrence in the pair sequence. This gives each distinct tagSet a unique name, such as T5, T27, etc.

The next step is to identify the *itemSet* associated with each distinct tagSet. An itemSet is a set of rows in the pair sequence such that all pairs contain the same tagSet. We represent the items in an itemSet by their numbers in the pair sequence. *The importance* of an itemSet with more than one element is that it represents a set of text items having the same appearance on the web page. They may be all the items that occupy the second field in each result record, for example.

The final stage of tpGrid production is to create a data structure with *a row for each distinct tagSet*, and each row labelled at the start with the name of the distinct tagSet, such as T5, T27, etc. Each row in the tpGrid displays the itemSet associated with the row's tagSet. So each row shows a sequence of integers which are the numbers of [tagSet – textString] pairs in the pair sequence. If the row for distinct tagSet T5 contains the two numbers 5 and 7, for example, then pairs 5 and 7 in the pair sequence both contain tagSet T5.

An example is now presented, in the following section, to demonstrate this tpgrid derivation process applied to a real web page .. and what it achieves.

3. An example query results web page analysis

As an example of tpGrid production, Fig1 shows the web page produced by a web site www.planepictures.net in response to the query 'de havilland comet', which is a type of aeroplane. Our task during page analysis is to automatically discover the result records in this web page. Then it will be necessary to automatically identify a suitable landmark beside each data item in the html code, so that an extractor program can use those landmarks later, to identify the data items in other result pages from this site.

The html document contained 88 textStrings, which were numbered S0 to S87. The first 33 of these textStrings are shown beside Fig1, and each string can be seen on the web page. The tagString preceding each of the 88 textStrings was converted to a tagSet, so that a *pair sequence* of 88 [tagSet-textString] pairs was produced. Then in these 88 pairs only 20 *distinct tagSets* were found, because the same tagSet was present in many of the pairs. Each distinct tagSet is named by the number of the pair in which the tagSet first occurs in the pair sequence, T2, T10, etc.

Fig 2 shows the itemSet associated with each of the 20 distinct tagSets. It shows, for example, that the pairs numbered 75, 78, 85 all contained the same tagSet, which is named T75 because it occurred first in pair number 75. We can see that tagSet T79 was found in four [tagSet – textString] pairs, and T4 was found in ten pairs.

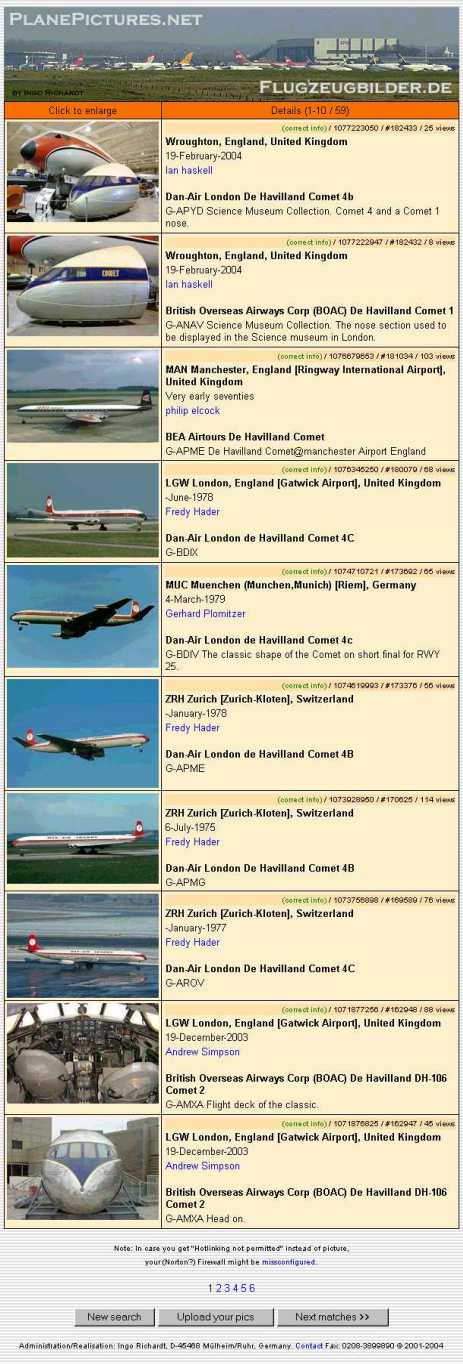


Fig. 1. Query Results web page from www.planepictures.net

The first 33 textStrings from the html document whose web page is shown in Fig1 are as follows:

S0: Planepictures search " de havilland comet "
S1: Click to enlarge
S2: Details (1-10 / 59)
S3: (correct info)
S4: / 1077223050 / #182433 / 25 views
S5: Wroughton, England, United Kingdom
S6: 19-February-2004
S7: Ian haskell
S8: Dan-Air London De Havilland Comet 4b
S9: G-APYD Science Museum Collection. Comet 4 and a Comet 1 nose.
S10: (correct info)
S11: / 1077222947 / #182432 / 8 views
S12: Wroughton, England, United Kingdom
S13: 19-February-2004
S14: Ian haskell
S15: British Overseas Airways Corp (BOAC) De Havilland Comet 1
S16: G-ANAV Science Museum Collection. The nose section used to be displayed in the Science museum in London.
S17: (correct info)
S18: / 1076679653 / #181034 / 103 views
S19: MAN Manchester, England [Ringway International Airport], United Kingdom
S20: Very early seventies
S21: philip elcock
S22: BEA Airtours De Havilland Comet
S23: G-APME De Havilland Comet@manchester Airport England
S24: (correct info)
S25: / 1076345250 / #180079 / 68 views
S26: LGW London, England [Gatwick Airport], United Kingdom
S27: -June-1978
S28: Fredy Hader
S29:Dan-Air London de Havilland Comet 4C
S30: G-BDIX
S31: (correct info)
S32: / 1074710721 / #173692 / 65 views

Some of the words visible on a web page are parts of bitmap images, so will not be found in the list of textStrings for the page.

```
 T0: 0
 T1: 1
 T2: 2
 T3: 3
 T4: 4,11,18,25,32,39,46,53,60,67
 T5: 5,12,19,26,33,40,47,54,61,68
 T6: 6,9,13,16,20,23,27,30,34,37,41,44,48,51,55,58,62,65,69,72
 T7: 7,14,21,28,35,42,49,56,63,70
 T8: 8,15,22,29,36,43,50,57,64,71
T10:10,17,24,31,38,45,52,59,66
T73: 73
T74: 74
T75: 75,78,85
T76: 76,86
T77: 77
T79: 79,80,81,82
T83: 83
T84: 84
T87: 87
```

Fig. 2. The tpGrid for Fig1's web page

The reason for creating a tagSet is to provide a page representation that is more amenable than html code to analysis when searching for repetitive items. We see at once a block of long rows in the tpGrid (Fig2). This indicates collections of similar items in the web page, such as the set of result records we want to find. But in order to see exactly what it signifies, *trail following* is used, as now explained.

The sequence of page items is shown in the tpGrid by the item numbers, 0 to 87 for the 88 [tagSet – textString] pairs. If we join consecutively numbered items with a line then the result is the item sequence *Trail*, on which tpGrid analysis is based. By following the Trail through the tpGrid we can see the pattern of tagSet use in the web page. Items 0 to 3 each have their own distinct tagSet which occurs only once in the web page. Recurrent patterns are associated with long rows in the tpGrid. The tagSets used for items 4,5,6,7,8,9,10 are T4,T5,T6,T7,T8,T6,T10 respectively. This same sequence of tagSets is used for items 11 to 17, and again for items18 to 24, and again for items 25 to 31, etc. The recurrence pattern becomes easier to see in the tpGrid if we put the recurring sequences of items in *vertical columns*, as shown in Fig3.

Now we see a block of long rows that are all the same length (10 items) except the last row (9 items). This block *shape* is characteristic of a set of result records. There are seven rows because each record has seven fields; and there are ten columns for the ten result records shown on the web page on screen.

Looking at the list of textStrings beside Fig1, we can see that items 3, 10, 17, 24, 31 are the first item in each 7-field record. In the tpGrid these items have their characteristic positions as the single item (item 3) before the cluster of long rows, and the last row of the cluster. Item 66 is the first field of the last 7-field record. The automatic analyser does not need to look at the web page.

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7
T3 then T10	T4	T5	T6	T7	T8	T6

Table 1. The tagSet associated with each field

```
 T0:  0
 T1:  1
 T2:  2
 T3:  3
 T4:  4,11,18,25,32,39,46,53,60,67
 T5:  5,12,19,26,33,40,47,54,61,68
 T6:  6,13,20,27,34,41,48,55,62,69
 T7:  7,14,21,28,35,42,49,56,63,70
 T8:  8,15,22,29,36,43,50,57,64,71
 T6:  9,16,23,30,37,44,51,58,65,72
T10: 10,17,24,31,38,45,52,59,66
T73: 73
T74: 74
T75: 75,78,85
T76: 76,86
T77: 77
T79: 79,80,81,82
T83: 83
T84: 84
T87: 87
```

Fig. 3. Rearranged tpGrid for Fig1's web page

The significance of all this is that by constructing the tpGrid we have discovered the set of ten 7-field records and provided all the information an extractor program needs to extract the data into correct field positions.

TagSet T3 occurs once in each results page, just before the first result record and T73 occurs once, just after the last record.

In the results section, each field is identified by the tagSet preceding it in the html code. T4 to T10 are, in effect, labels (in the html code of a results page) for the data item belonging in each field of each result record.

This is a useful achievement.

Landmark-based extractors identify data items by features in the html document. For result pages from this web site an extractor would skipTo(T3), meaning ignore everything in the html code of a results page until tagSet T3 is found. This marks the start of the result set and the first field item immediately follows tagSet T3. After extracting the first field item, each of the other six fields follows the tagSet specified in Table 1. After the first *record* has been extracted, tagSet T10 precedes the first field of the next record. The process of extracting the tagSet-labelled data items continues until tagSet T73 in found, which marks the end of the results.

3.1 Hyperlinked data items

The introductory explanation above considered field items to be textStrings. However, this is just a mechanism to establish the position and shape of records. Once these textString records are found it is easy to identify any pictures and URLs in the tagString preceding each textString data item. These become additional fields that precede the corresponding textString field, because they come from the tagString part of a [tagString – textString] pair. For example, tagString 10 is:

```
</TD></TR></TBODY></TABLE></TD></TR><TR vAlign=top><TD align=middle
width="25%" bgColor=#fef7cd>
<A href="http://www.planepictures.net/netshow.cgi?182432"
target=_blank><IMG height=133 alt=182432 src="files/1077222947_TN.jpg"
width=200 border=0></A></TD><TD width="75%" bgColor=#fef7cd><TABLE
cellSpacing=1 cellPadding=1 width="100%" border=0><TBODY><TR><TD
align=right bgColor=#ffe7ad><FONT size=1>
<A href="http://www.planepictures.net/c.cgi?182432" target=_new><FONT
color=green>
```

It contains two <A> tag URLs and one <IMG> URL. These are in the tagString that precedes each first field textString, so the URL strings can be added as three extra

fields. TagString3, which precedes the first *record*, contains the same three links .. as expected. In the tagStrings preceding each of the other six fields only TagString7 contains a hyperlink, so an extra field to contain that URL is created before Field5.

4. Nested Iterations

Data is sometimes displayed on a web page as a tree structure like the Contents list in a book, rather than a simple sequence of records. Fig 5 shows an example, which is the results page for a query to www.nga.gov.
Here is the tpGrid for the web page shown in Fig 5:

```
T0: 0
T1: 1
T2: 2
T3: 3
T4: 4
T5: 5,47
T6: 6
T7: 7,          16,             27,       34
T8: 8,10,12,14,17,19,21,23,25,28,30,32,35
T9: 9,11,13,    18,20,22,24,    29,31
T15:         15,             26,       33
T36: 36
T37: 37,39,41,43,45,48,50,52,54,56,58,60,62
T38: 38,40,42,44,46,49,51,53,55,57,59,61,63
T64: 64
T65: 65
```

Items 7 to 15 show the characteristic shape in the tpGrid for a set of records. (One item on its own on the row before the block of long rows, and the last long row one item shorter that the others). So do items 16 to 25 and 27 to 32. To see what this signifies, textStrings 7 to 14 can be found beside Fig 5 and identified in the web page. They are the list of 2-field result records below the heading 'Painting'. Items 16 to 25 are the list below 'Drawing'. Items 27 to 32 are the three 2-field records below 'Print', and items 34,35 are the single 2-field record below the final heading, 'Volume'. The single-item in the first row and the items in the slightly short last row in each block of items in the tpGrid correspond to first-field items in the data records.

Rows T6 and T15 in the tpGrid enclose the series of tables just discussed. The single item in row T6 and the set of items in row T15 are first and last rows of another block in the tpGrid. Items 6, 15, 26 and 33 (in rows T6 and T15) are the category headings, Painting, Drawing, Print and Volume, which are visible as headings on the web page in Fig 5. This *nested record structure* in a tpgrid is characteristic of hierarchical data in web pages. Other examples can be seen on our web site [http://www.page-info.info].

The first field in the list of 2-field artObjects is the title, which is clearly a hyperlink in Fig 5. Therefore the tagString preceding each of these textStrings must contain an <A> tag URL which the page analyser will automatically find and add as a third field to the template for records to be extracted.

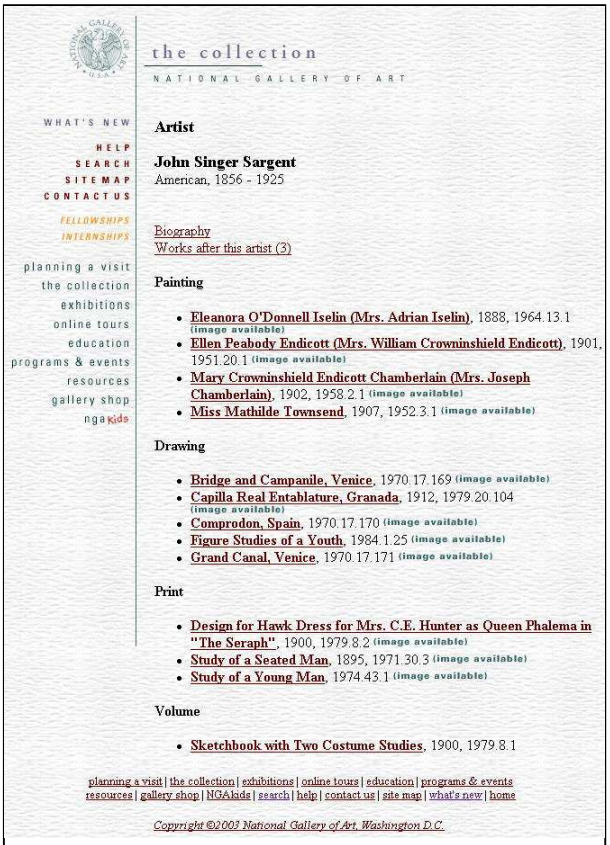
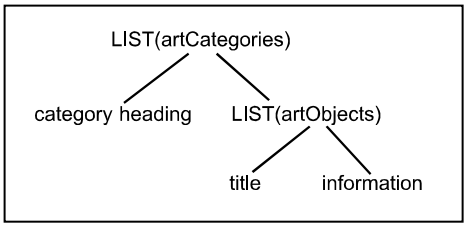


Fig. 5. Query Results web page from www.nga.gov

TextStrings in Fig5's html:

S0: John Singer Sargent
S1: Artist
S2: John Singer Sargent
S3: American, 1856 - 1925
S4: Biography
S5: Works after this artist (3)
S6: Painting
S7: Eleanora O ' Donnell Iselin (Mrs. Adrian Iselin)
S8: , 1888, 1964.13.1
S9: Ellen Peabody Endicott (Mrs. William Crowninshield Endicott)
S10: , 1901, 1951.20.1
S11: Mary Crowninshield Endicott Chamberlain (Mrs. Joseph Chamberlain)
S12: , 1902, 1958.2.1
S13: Miss Mathilde Townsend
S14: , 1907, 1952.3.1
S15: Drawing
S16: Bridge and Campanile, Venice
S17: , 1970.17.169
S18: Capilla Real Entablature, Granada
S19: , 1912, 1979.20.104
S20: Comprodon, Spain
S21: , 1970.17.170
S22: Figure Studies of a Youth
S23: , 1984.1.25
… *etc etc* …
S64: Copyright ©2003 National Gallery of Art,Washin …

The repetitive data in the web page is a *nested iteration*. It is a list of 2-field data items, where the second field is itself a list of 2-field data items. As shown in the diagram on the right.



5. Detecting changes to web page structure

This task, known as Wrapper Verification, is probably as important as wrapper creation because correct data extraction is needed, and if web page layout changes then the wrapper may no longer work properly. Previous research [16,17] on automatic checking to ensure a wrapper still works, has focussed on the data rather than the page. If the wrong data set is extracted then the wrapper is not working properly and this may be because the page format has changed. This is an indirect way to check whether a web page has changed its appearance. It examines the effect rather than the cause. In contrast with this, a tpGrid is a fingerprint for a web page. It shows key features and associates them with tagSets. So if the current tpGrid for a web site differs from the one used to create the wrapper then page appearance has changed. Changes that affect the data area of the page can be distinguished from those that affect other areas and do not affect the tagSets used by the wrapper.

Conclusions

The importance of automatic wrapper production is well established. It is a research area that has been active for many years because of the enormous benefits for applications is the problem can be solved. The problems to be solved are to find the result records in the web page and identify their record structure, to solve the problem of variable length records and records whose fields may occur in a different position in each record, and dealing with hierarchical data. Variable record length is caused by missing fields or by arbitrary repetition of some nested group of fields (as discussed in section 4). In the tpGrid, any missing fields appear as *holes* in the block of result items (Fig 3). But each row (field) is identified by its row's tagSet so there is no difficulty in recognizing which field is missing. Similarly, during extraction of data, the extractor can allocate data items to their correct fields in the record template even though some fields are missing, because of the tagSet label identifying each data item and its field.

The method of page analysis, and data extraction, described in this paper uses a new data structure to represent web pages for analysis. This allows fully-automatic wrapping for web pages with a number of different types of appearance. Samples can be seen at our web site [1]. The extent of this class of automatically wrappable web sites is not yet established. There are many techniques that have not yet been utilized. The tpGrid is a central page analysis technique, to which various other knowledge sources for page structure, including previous methods of web page analysis, can contribute.

References

[1] A substantial bibliography of relevant references can be found via the author's web site at http://www.page-info.info which also contains examples, demo, etc.

[2] R. Kosala, H. Blockeel, *Web Mining Research: A Survey*, ACM SIGKDD Explorations (2)1, 2000.

[3] L. Eikvil, *Information Extraction from World Wide Web A Survey* (1999). Norwegian Computing Center report, available via web.
[4] Kushmerick, N. & Thomas, B. *Adaptive information extraction: Core technologies for information agents*. In Intelligent Information Agents R&D in Europe: An AgentLink perspective, Springer, 2003. LNCS 2586.
[5] Stephen W. Liddle, K. A. Hewett, and D. W. Embley, *An Integrated Ontology Development Environment for Data Extraction*, submitted, April 2003.
[6] S.W. Liddle, D.W. Embley, D.T. Scott, and S.H. Yau, *Extracting Data Behind Web Forms*, Proc. of the Workshop on Conceptual Modeling Approaches for e-Business, 2002.
[7] I. Muslea, S. Minton, C. Knoblock, *Hierarchical Wrapper Induction for Semistructured Information Sources*, J. Autonomous Agents and Multi-Agent Systems, (4) pp 93-114, 2001.
[8] W.W. Cohen, W. Fan, *Learning Page-Independent Heuristics for Extracting Data from Web Pages*, Proc 8th International World Wide Web Conference, 1999.
[9] Stephen Soderland, *Learning Information Extraction Rules for Semi-structured and Free Text*, Machine Learning 34(1-3) pp 233-272, 1999.
[10] D. Buttler and L. Liu and C. Pu, *A Fully Automated Object Extraction System for the World Wide Web*. Proc. Intl. Conf. on Distributed Computing Systems, 2001. pp 361 - 371.
[11] V. Crescenzi, G. Mecca and P. Merialdo. *Roadrunner: Towards automatic data extraction from large web sites*, Proc 27th Very Large Databases Conference, VLDB'01, pages 109-118, 2001.
[12] David W. Embley, Y. S. Jiang, Yiu-Kai Ng, *Record-Boundary Discovery in Web Documents*, Proc. ACM SIGMOD Conference 1999, pages 467-478.
[13] W. Cohen, M. Hurst, L. Jensen, *A Flexible Learning System for Wrapping Tables and Lists in HTML Documents*, in WWW-2002. http://www2002.org/CDROM/refereed/355/
[14] M. Neiling, M. Schaal, M. Schumann, *WrapIt: Automated Integration of Web Databases with Extensional Overlaps*. 2nd Intl.Workshop of Web and Databases, WebDB 2002, pp 184-198. (LNCS 2593)
[15] J. Wang and F.H. Lochovsky, *Data Extraction and Label Assignment for Web Databases*, Proc. 12th International World Wide Web Conference, WWW03.
[16] Kushmerick, N. *Wrapper Verification*. World Wide Web J. 3(2):79-94 (special issue on Web Data Management). 2000.
[17] Kristina Lerman, Steven Minton and Craig Knoblock, *Wrapper Maintenance: A Machine Learning Approach*, Journal of Artificial Intelligence Research 18:149-181, 2003.
[18] A. Hemnani and S. Bressan, *Information Extraction - Tree Alignment Approach to Pattern Discovery in Web Documents*, Proc Database and Expert Systems Applications, 13th International Conference, DEXA 2002, (LNCS 2453)
[19] C-H Chang, S-C Lui, and Y-C Wu, *Applying Pattern Mining to Web Information Extraction*, Proc PAKDD 2001, 5th Pacific-Asia Conf on Knowledge Discovery and Data Mining, pp 4-16. LNAI 2035.
[20] J. Yang, H. Oh, K-G. Doh and J. Choi, *A Knowledge-Based Information Extraction System for Semi-structured Labeled Documents*. Proc IDEAL'02, 3rd Intl Conf Intelligent Data Engineering and Automated Learning, pp 105-110, 2002. LNCS 2412.
[21] J. Robinson, *Data Extraction from Web Database Query Result Pages via TagSets and Integer Sequences*, Proc IADIS WWW/Internet International Conference 2003.
[22] The RISE Repository of Online Information Sources Used in Information Extraction Tasks http://www.isi.edu/info-agents/RISE/index.html