# EFFICIENT RSS FEED GENERATION FROM HTML PAGES

Jun Wang
*Fujitsu R&D Center Co., Ltd., B306, Eagle Run Plaza No.26 Xiaoyun Road, Beijing 100016, China.*

Kanji Uchino
*Fujitsu Laboratories, Ltd., 4-1-1 Kami-kodanaka, Nakahara-Kawasaki, Kanagawa 211-8588, Japan*

Keywords: RSS, Metadata, Information Extraction, Knowledge Management

Abstract: Although RSS demonstrates a promising solution to track and personalize the flow of new Web information, many of the current Web sites are not yet enabled with RSS feeds. The availability of convenient approaches to "RSSify" existing suitable Web contents has become a stringent necessity. This paper presents EHTML2RSS, an efficient system that translates semi-structured HTML pages to structured RSS feeds, which proposes different approaches based on various features of HTML pages. For the information items with release time, the system provides an automatic approach based on time pattern discovery. Another automatic approach based on repeated tag pattern discovery is applied to convert the regular pages without the time pattern. A semi-automatic approach based on labelling is available to process the irregular pages or specific sections in Web pages according to the user's requirements. Experimental results show that our system is efficient and effective in facilitating the RSS feed generation.

## 1 INTRODUCTION

The knowledge workers who strive to keep up with the latest news and trends in the field have to frequently revisit specific Web pages containing list-oriented information such as headlines, "what's new", job vacancies and event announcements. The above information can certainly help enterprises and individuals track competitions and opportunities, and understand markets and trends, however it becomes not easy for workers to keep current when information sources exceed a handful.

Rich Site Summary (RSS), a machine-readable XML format for content syndication (Hammersley, 2003), allows users to subscribe to the desired information and receive notification when new information is available. RSS feeds send information only to the parties that are truly interested, thereby relieving the pressure on email systems suffering from spam (Miller, 2004). Since virtually almost any list-oriented content could be presented in RSS format, RSS demonstrates a promising solution to track and personalize the flow of new Web information. Furthermore, enterprises can take advantage of the simplicity of the RSS specification to feed information inside and outside of a firewall. Today RSS has become perhaps the most widely used XML format on the Web. However, much of the current Web content is not yet enabled by RSS feeds. For example, in some big enterprises there are hundreds or even thousands of Web sites belonging to different departments, and many of these sites are equipped with old systems which are rigid and difficult to update for supporting RSS feed. It would be cumbersome and cost prohibitive to replace or reconstruct all these legacy service systems. For small organizations and non-technical individuals, they are often lack of the expertise and budget to update their sites to support RSS feed. Even for the sites providing RSS feeds, only a small fraction of suitable content is RSS-enabled.

In order to evangelize RSS application and leverage the Web's valuable contents, the availability of convenient approaches to "RSSify" suitable Web content has become a stringent necessity. The point is to translate existing semi-structured HTML pages into structured RSS feeds. The simplest way is to observe HTML pages and code extraction rules manually (Hammer, 1997; Huck, 1998; Sahuguet, 1999). However, writing



Wang J. and Uchino K. (2005).
EFFICIENT RSS FEED GENERATION FROM HTML PAGES.
In *Proceedings of the First International Conference on Web Information Systems and Technologies*, pages 311-318
DOI: 10.5220/0001230103110318

rules requires a certain knowledge of programming. In addition, it is time-consuming, error-prone and not scalable. Therefore, we need more efficient approaches for RSS feed generation, which should be automated to the largest extent possible, in order to allow for large scale extraction tasks even in presence of changes in the underlying sites.

In this paper, we introduce EHTML2RSS, a system for converting list-oriented information in HTML pages to RSS feeds. For the information items with release time, the system provides an automatic approach based on time pattern discovery. Another automatic approach based on repeated tag pattern discovery is applied to translate the regular pages without the time pattern. At the same time, a semi-automatic approach based on labelling is also available to process the irregular pages. Figure 1 shows the component diagram of EHTML2RSS.

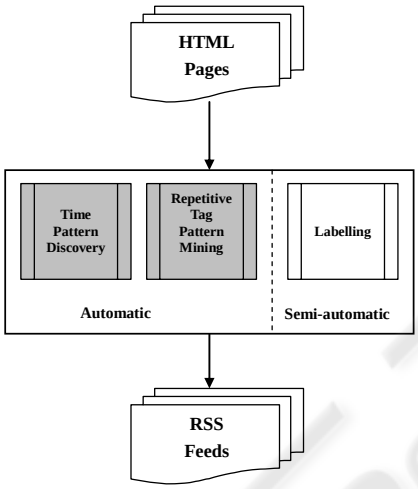



Figure 1: The EHTML2RSS Architecture

# 2 RSS FEED GENERATION

Since core content of different versions of RSS are very similar in general structure and consistent in concept and our work is independent of version, related RSS tags are presented in RSS 2.0 format in this paper. At the most basic level, a RSS feed consists of a *channel* with its own metadata (e.g. *title*, *link*, *description*, *pubDate*, *language* etc) and a number of *items*, each of which has its own metadata (e.g. *title*, *link*, *pubDate*, *category* etc). The *title* in the RSS *channel* can be easily extracted from the content of the *title* in the HTML *head*. The *url* of the HTML page can be treated as the *link* in the RSS *channel*. When the metadata of the HTML *head* contain *description* or *keywords*, we can convert them to contents of the *description* in the RSS *channel*. If the HTML page is static, we can convert the *last-modified* time in the HTTP *head* to *pubDate* in RSS *channel;* otherwise we just set current time as the *pubDate*. The *language* of RSS *channel* can be extracted from the *content-language* or *charset* metadata of the HTML *head*.

The primary contents of the information items in list-oriented pages are the *title*, *url* and *release time* which are the counterparts of the *title*, *link* and *pubDate* in the *item* of RSS specification. The *url* of a news item in HTML pages is in the *href* attribute of a <*a*> tag and the corresponding *title* usually resides in texts in or near the <*a*> tag. Therefore, the primary task of EHTML2RSS is to locate suitable <*a*> tags and texts in HTML pages. However, Web pages often contain multiple topics and a number of irrelevant pieces of information from navigation, decoration, and interaction parts (Gupta, 2003). It is not easy for the machine to automatically locate and convert target items since HTML is designed for presentation instead of content description (Berners-Lee, 2001). EHML2RSS proposes efficient and effective approaches to solve this problem based on different features of list-oriented information in HTML pages.

## 2.1 Automatic Approach Based on Time Pattern Discovery

In news or "what's new" pages, the news item is often published with the corresponding release time. This feature is a prominent and useful clue for locating and extracting the target news items. Figure 2 shows a company press release page and a conference news page. Since the formats of date and time are simple and limited, the release time is easily identified and we can easily construct a database of time patterns represented in regular expressions. In our current experiment, only about 20 time patterns are required to cover almost all the time and date formats we have met on Japanese and Chinese sites. In figure 3, there are some typical date and time formats.

Firstly, we create a DOM tree for the HTML page. We use the number to represent the address of nodes in DOM tree. The address consists of numbers joined by a '.', starting with '0', and followed by the order (index of the node in its parent's children nodes list) of the nodes in the tree that are ancestors of current node, starting from the top. As a bit of a special case, the address of the root is simply '0'.

Secondly, we need to extract all text nodes containing the release time of news items in DOM tree. By pre-order traversing of the DOM tree, each text node matching the time pattern in the database is named a time node *TN*, and its address and corresponding time pattern are recorded in an

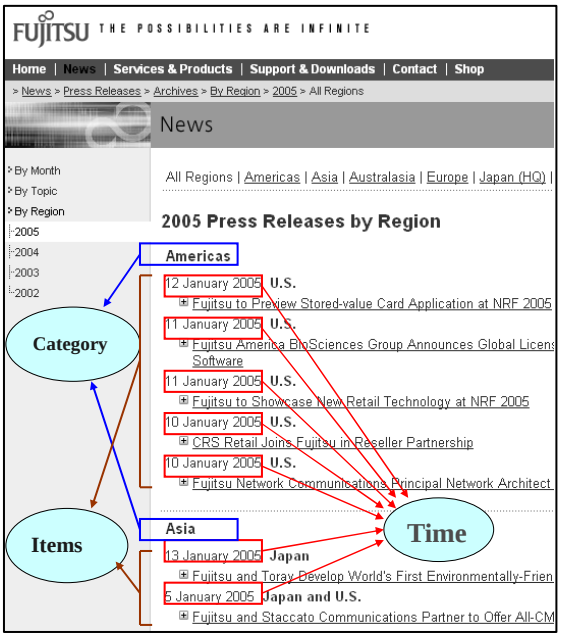
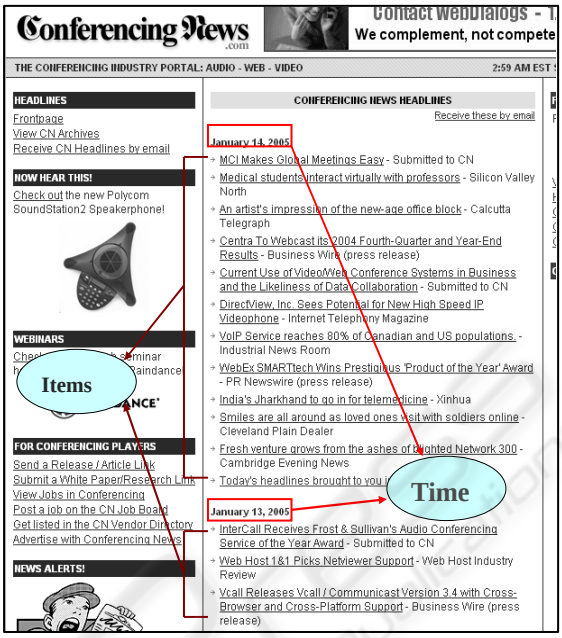



Figure 2: (a) A press release page




(b) A conference news page

address list *AL* and a time pattern list *TPL* respectively. In some cases, there are multiple time patterns in a Web page, and we can output the time nodes of all time patterns, or just time nodes belonging to specific patterns selected by a heuristic rule, or just time nodes matching patterns designated by the user. It is dependent on the concrete application requirement.

***2004-06-28 03:26 PM***
***20040518 14:50***
***2004/06/28***
***13 January 2005***
***Oct. 1, 2004***
***2003 年 5 月 1 日***

Figure 3: Examples of the time formats

| | |
|---|---|
| 0.1.9.0.0.0.2.0.0.0.0.0.4.0.0.1.0.0.0.0.0.0 | 12 January 2005 |
| 0.1.9.0.0.0.2.0.0.0.0.0.4.0.0.1.0.0.0.2.0.0 | 11 January 2005 |
| 0.1.9.0.0.0.2.0.0.0.0.0.4.0.0.1.0.0.0.4.0.0 | 11 January 2005 |
| 0.1.9.0.0.0.2.0.0.0.0.0.4.0.0.1.0.0.0.6.0.0 | 10 January 2005 |
| 0.1.9.0.0.0.2.0.0.0.0.0.4.0.0.1.0.0.0.8.0.0 | 10 January 2005 |
| --- | --- |
| 0.1.9.0.0.0.2.0.0.0.0.0.6.0.0.1.0.0.0.0.0.0 | 13 January 2005 |
| 0.1.9.0.0.0.2.0.0.0.0.0.6.0.0.1.0.0.0.2.0.0 | 5 January 2005 |

Figure 4: Example of Address Array M

Since the syntax structure of a HTML page is usually consistent with its semantic structure, based on the DOM tree structure, *AL* can be segmented into sections in each of which time nodes keep spatial locality. Each address in *AL* can be split into a 1-dimension array based on the separator '.', and *AL* finally is converted to a 2-dimension array *M*. Figure 4 shows the M corresponding to the release time listed in figure 2. We can segment *AL* by partitioning *M*. A triple <*r*, *c*, *n*> defines a section in *M*. *r* and *c* are the row number and the column number, respectively, of the top left element in the section. *n* is total number of rows contained in the section. *R[i]* is said to be $i^{th}$ row of *M* and corresponds to a *TN* in DOM tree, and *C[j]* represent the $j^{th}$ column of *M*. *M[i, j]* is said to be the element in the $i^{th}$ row and the $j^{th}$ column of *M* and also corresponds to a node in DOM tree. Let the total row number of *M* be *TR* and present full section of M as *<0, 0, TR>*. Figure 5 shows the recursive segmentation algorithm.

```
Segment (M, <r, c, n>) {
  do {
    j ← c;
    isAllValuesSame ← checkValueInArray(C[j]);
    if (isAllValuesSame == TRUE) {
      j++;
    }
  } until (isAllValuesSame == FALSE);
  SectSet = {<r_p, j, n_p>|0 ≤ p ≤ k-1} ← splitByValues(<r, j, n>);
  if (in ∀<r_p, j, n_p> ∈ SectSet, n_p == 1) {
    InfoExtract (M, <r, c, n>, j, TPL);
  }
  else {
    for each <r_p, j, n_p> in SectSet {
      Segment (M, <r_p, j, n_p>);
    }
  }
}
```

Figure 5: Segmentation Algorithm

*checkValueInArray(C[j])* checks if all the values

in the $j^{th}$ column of the *M* are same or not. If the values are not same, *splitByValues(<r, j, n>)* will segment the section *<r, j, n>* into *k* sub-sections in each which the values in the $j^{th}$ column are the same. When each sub-section contains only 1 row, the segmentation process will be stopped and we can extract the information items in the current section.

Although HTML pages containing the time pattern have diverse contents and structures, they can be classified into two types in terms of the layout. In the first type, each news item has an individual release time, and the page showed in figure 2(a) is a typical example. The page in figure 2(b) is an example of the second type, in which multiple news items follow every release time. The algorithm in figure 6 describes the details of information extraction based on the structure and layout analysis.

*getTimeNode(R[i])* returns the time node *TN* corresponding to the $i^{th}$ row of *M*. *getNode(M[i, j])* returns a node *TBN* corresponding to *M[i, j]*, which defines the border of current *TN*. *getTim*e extracts the time information from *TN* based on the corresponding time pattern stored in *TPL* and output *pubDate* in the standard format such as '*Tue, 18 Jan 2005 07:27:42 GMT*'. *searchInBorder* searches and outputs all *<a>* nodes under *TBN* to a node set *NodeSetB*. *checkPostion* checks if all the *<a>* nodes in a set are presented in the same line in a browser or not. For the list-oriented information, each item is usually displayed in an individual line. This is an important layout feature. The line presentation relies on the DOM tree structure and specific tags such as *<ul>*, *<li>*, *<tr>*, *<p>*, * * and *<br>*, which cause a new line in the display. *extractTitleLink* uses heuristic rules to select the *href* attribute of a suitable *<a>* node and a proper title text in the current line as the *title* and *link* in RSS feeds. *searchInLine* searches *<a>* nodes in the line in which *TBN* is presented, and outputs to a set *NodeSetL*. *ExtractInMultiline*, described in figure 7, extracts information items from a *<a>* node set in which the nodes are displayed in multiple lines. *devideByLine* is used to divide a node set into multiple sub-sets in which all the nodes are displayed in the same line. For some pages, like the example in the figure 2(b), we detect the position of two adjacent *TBN*s and search target nodes between them by *searchArea*. But for last *TBN* in *M* there is no next *TBN* as the end border *detectSearchBorder* is used to decide the end border of search. In general, the structure of each section is similar, so we can use the structure in the last section to deduce the current end border. Obviously, *ResultList* can be easily translated to a RSS format.

After recognition of all the items in a section, we can decide the complete border of this section. In some pages, such as the page in figure 2(a), each section has a category title for summarizing content in the section, which corresponds to the *category* in the RSS *item*. The *category* data is usually presented in a line above and adjacent to the first item of the section, and contained in continuous text nodes on the left part in the line. If category is presented in an image, we can use a similar method to check the *alt* attribute of the appropriate *<img>* node. If necessary, we can also extract this information automatically.

The idea of the time pattern discovery can be easily extended to mine other distinct format patterns, such as price patterns, which can be used to extract pairs of the product name and price from pages in e-commercial sites.

```
InfoExtract (M, <r, c, n>, j, TPL) {
  TR ← total row number of M
  for (i = r, i< r+n, i++) {
    TN ← getTimeNode(R[i]);
    TBN ← getNode (M[i, j]);
    pubDate ← getTime(TN,TPL[i]);
    NodeSetB ← searchInBorder( TBN);
      if (NodeSetB ≠ NULL) {
        isInSameLine ← checkPostion(NodeSetB);
        if (isInSameLine == TRUE) {
          <TITLE, LINK> ← extractTitleLink(NodeSetB);
          Push(ResultList, <TITLE, LINK, pubDate>);
        }
        else (isInSameLine == FALSE) {
          ExtractInMultiLine(NodeSetB);
        }
      }
      else {
        NodeSetL ← searchInLine(TBN);
        If (NodeSetL ≠ NULL) {
          <TITLE, LINK> ← extractTitleLink (NodeSetL);
          Push(ResultList, < TITLE, LINK, pubDate>);
        }
        else {
          if (r+i ≠ TR-1) {
            NextTBN ← getNode (M[i+1, j]);
          }
          else {
            NextTBN ← detectSearchBorder;
          }
          AreaSet ← searchArea (TBN, NextTBN);
          ExtractInMultiLine(AreaSet);
        }
      }
  }
}
```

Figure 6: Algorithm for information item extraction

```
ExtractInMultiLine(NodeSet) {
  LineSets ← divideByLine (NodeSet);
  for each LineSet in LineSets {
    <TITLE, LINK> ← extractTitleLink(LineSet);
    Push(ResultList, <TITLE, LINK, pubDate>);
  }
}
```

Figure 7: Information Extraction in multiple lines

## 2.2 Automatic Approach Based on Repeated Tag Pattern Discovery

Although the approach based on time pattern discovery can generate RSS feeds conveniently, there are still many pages containing no time pattern. In HTML pages containing list-oriented information, information items are usually arranged orderly and compactly in a coherent region, with the same style of presentation and a similar pattern of HTML syntax. We call this kind of coherent region *InfoBlock*. Information items in an *InfoBlock* usually share a repetitive tag pattern and have a same parent node. Figure 8 shows a repeated tag pattern and its corresponding instances (occurrence of the pattern) in a music news page. Therefore mining the repeated tag patterns in HTML pages provides guidance for the effective extraction of information items and generation of RSS feed.

Figure 8: Example of repeated tag pattern in HTML page

Since it is more convenient to discover repetitive patterns by token stream, we generate tag token stream by pre-order traversing DOM tree. We also create a mapping table between each tag token in the stream and the corresponding node in the DOM tree. We use the<*text*> tag to represent a text node. A Suffix Trie (Gusfield, 1997; Ukkonen, 1995) is constructed for the tag token stream and applied to induce repetitive patterns. We apply "non-overlap" (The occurrences of a repeated pattern cannot overlap each other) and "left diverse" (The tags at the left side of each occurrence of a repeated pattern belong to different tag types.) rules to filter out the improper patterns and generate suitable candidate patterns and associated instance sets. For RSS feed generation, the target items are located in the <*a*> and <*text*> nodes, so the patterns containing no <*a*> and <*text*> will also be removed. Finally more than 90% of the repeated patterns are discarded.

By a method similar to that used to segment *AL* in section 2.1, we can partition the instance set of each repeated tag pattern into sub-sets based on structure of DOM tree. Here the basic unit is a series of nodes belonging to a repeated pattern instance instead of one time node. After the partition, the instances in each sub-set will present spatial locality. For the instances in a sub-set, we can find corresponding nodes in DOM tree, and the root node of the smallest sub-tree containing all these nodes is called *RST* (the root of the smallest sub-tree) node, which represent a page region, i.e. *InfoBlock*.

Since sometimes a RST node associated with a specific information item format may correspond to multiple instance sets belonging to different patterns discovered previously, each of which represents the information item format wholly or partly, we need to assess and select the best qualified set for identifying the correct border of information items under the current RST node. We create a series of criteria such as the frequency of occurrences, length, regularity and coverage of the repeated pattern for the assessment. Regularity of a repeated instance set is measured by computing the standard deviation of the interval between two adjacent occurrences. It is equal to the standard deviation of the intervals divided by the mean of the intervals. Coverage is measured by the ratio of the volume of the contents contained by repeated instance set to the volume of the all contents under the *RST* node. Each of the criteria has a threshold that can be adjusted by the user. An assessment usually applies one or more of above criteria, either separately or in combination. Since the each news item usually is displayed in an individual line, this feature also can be helpful to identify and information items and their borders.

The desired part i.e. list-oriented information for the RSS feed generation, usually occupies notable regions in a HTML page. Therefore, we can select the pattern whose instance set contains the maximum contents or occupies the maximum area in the HTML page. We also can list candidate patterns and show their corresponding regions in the page, and let the user to select the pattern compatible with his requirements. After selecting the right pattern and identifying the border of each information item, it is easy to extract the *title* and *link* from target items due to the simple structure of news items. If necessary, we also can employ the similar method used in section 2.1 to extract the *category* information based on the border of each *InfoBlock*.

## 2.3 Semi-automatic Approach Based on visual labelling

No automatic approach can process all list-oriented HTML pages well, and there are always some exceptions for a fraction of irregular or complicated pages during automatic RSS feed generation. Sometimes a HTML page contains several suitable regions, but user wants to select only one specific section to generate the RSS feed. In order to solve above problems, we design a semi-automatic labelling GUI tool to process pages with unsatisfying result in automatic approaches.

As shown in figure 9, the GUI tool contains two part of labelling interfaces: a DOM tree in the left side and a browser in the right side. The user can label RSS metadata on appropriate parts of HTML page directly and intuitively in the browser interface. When the user clicking the hyperlinks or selecting the texts displayed in the browser interface, the tool can help the user to locate the corresponding nodes in DOM tree automatically and associate RSS metadata with the nodes conveniently. The user can also select and mark the nodes in the DOM tree interface to define a region in the Web page or associate the nodes with corresponding RSS metadata. When a DOM tree node is selected, the corresponding region in the HTML page can be identified and displayed at the same time. As we mentioned before, the information items in HTML pages, as discerned in their rendered views, often exhibit spatial locality in the DOM tree, and we also exploit this feature to optimize the labelling operations. After we label an item in a list, the tool can automatically deduce other items in this list based on the structure of the current item in the DOM tree. After we finish the labelling on an item list of first category, the tool can automatically deduce the lists of other categories similarly. During the deducing process, the user can simultaneously adjust labelling process and range according to the result displayed in a visual interface.

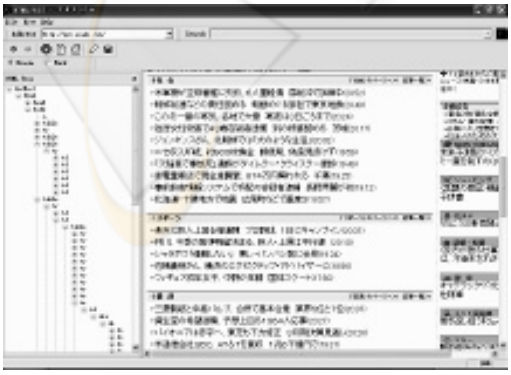


Figure 9: GUI interface for labelling

```
[Encoding]
Encoding=EUC-JP
[Paths]
Channel_Date_Path=/HTML[0]/BODY[0]/TABLE[1]/TR[4]/TD[2]
Subject_Object_Path1=/HTML[0]/BODY[0]/TABLE[3]/TR[0]/TD[1]/
TABLE[0]/TR
Subject_Title_Path1=/TH[0]/A[0]
Item_Object_Path1.1=/TD[0]/DIV[0]/UL[0]/LI
Item_Title_And_Link_Path1.1=/A[0]
Item_Date_Path1.1=/
[Date Format]
Channel_Date_Format=yyyy 年 MM 月 dd 日 HH 時 mm 分
Item_Date_Format=HH:mm
```

Figure 10: Example of extraction rule

After labelling the page and verifying the converting result, we can induce an extraction rule automatically. The rule is represented in a simple format similar to XPath and can be reused to process the new contents of current page in the future. Figure 10 shows a rule example generated from the asahi.com.

But for some irregular pages whose semantic structure are not consistent with the syntax structure, above automatic deducing process will fail, and we have to mark the items or lists manually one by one, however, even in this poor situation the tool is still useful especially for the non-technical, because the user just need click mouse instead of writing complicated extraction programs.

Actually, for the above two automatic feed generation approaches, it is also possible to induce the reused rule from extraction result, and reduce the computing work of the RSS feed generation in the future.

# 3 EXPERIMENTS

EHTML2RSS has been tested on a wide range of Japanese and Chinese Web sites containing the news or other list-oriented information, including country wide news paper sites, local news paper sites, TV sites, portal sites, enterprise sites and i-mode (the most popular mobile Internet service in Japan) sites for cellular phones. We measure two performance metrics: precision and recall.

Firstly, we investigated about 200 Japanese and Chinese sites and found that about 70% of news sites and almost all “what’s new” or press release pages in the enterprise sites contain the release time of news items. We also checked lots of intranet sites in our company and found 90% of news information list are provided with the release time. We selected 217 typical pages with time pattern from various sites as the representative examples. Following table 1 presents the experimental result based on time pattern discovery. In table 1 each page in the local news set is collected from an individual Japanese

local news paper site. Since the time pattern has the distinct feature for the recognition, the extraction of the *pubDate* in target items has very high performance. The time pattern is a useful and accurate clue for locating the target item, therefore, as shown in the table 1 the extraction result of other data is also very good. The errors in *pubDate* extraction occur in only very few conditions, for example, there are multiple occurrences of current time pattern in one target item. We can solve this problem by checking the global structure of the item list in the future. The *category* extraction depends on the partitioning information item list into the appropriate sections, however, in some irregular cases the syntax structure of the page is not consistent with its semantic structure and consequently the partition will be misled. In some other cases, the partition result is correct, but there are some advertisements or recommendations information between the *category* title and news items, and the extraction also fails. Therefore the extraction result of the *category* is not as good as *title*, *link* and *pubDate*.

Table 1: Experimental result for the approach based on time pattern discovery

| site/ page | title | | link | | pubDate | | category | |
|---|---|---|---|---|---|---|---|---|
| number | Pre. | Rec. | Pre. | Rec. | Pre. | Rec. | Pre. | Rec. |
| asahi/13 | 0.974 | 0.946 | 1 | 0.974 | 1 | 1 | 1 | 0.958 |
| yomiuri/10 | 0.971 | 0.964 | 1 | 0.993 | 1 | 1 | 0.882 | 0.838 |
| nikkei/9 | 0.998 | 0.867 | 1 | 0.867 | 1 | 1 | 0.893 | 0.655 |
| sankei/10 | 1 | 1 | 1 | 1 | 1 | 1 | 0.778 | 1 |
| peopledaily/10 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| yahoo/10 | 1 | 1 | 1 | 1 | 1 | 1 | 0.571 | 0.529 |
| nifty/15 | 0.976 | 0.932 | 0.99 | 0.946 | 1 | 1 | 0.9 | 0.82 |
| sina/10 | 0.989 | 0.967 | 0.989 | 0.967 | 0.991 | 0.991 | 0.864 | 0.841 |
| fujitv/3 | 1 | 1 | 1 | 1 | 1 | 1 | N/A | N/A |
| phoenixtv/9 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| localnews/16 | 0.998 | 0.973 | 1 | 0.976 | 1 | 1 | 1 | 0.963 |
| beijingnews/12 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| fujitsu/15 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| nec/15 | 1 | 1 | 1 | 1 | 0.982 | 0.982 | 1 | 1 |
| hitachi/10 | 1 | 1 | 1 | 1 | 1 | 1 | N/A | N/A |
| canon/5 | 1 | 1 | 1 | 1 | 0.989 | 0.989 | N/A | N/A |
| haier/10 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| huawei/5 | 1 | 1 | 1 | 1 | 1 | 1 | N/A | N/A |
| i-mode/10 | 1 | 1 | 1 | 1 | 1 | 1 | N/A | N/A |
| intranet/20 | 0.963 | 0.975 | 0.981 | 0.972 | 0.974 | 0.974 | 0.863 | 0.822 |

Table 2: Experimental result for the approach based on repeated tag pattern discovery

| site/page | title | | link | | category | |
|---|---|---|---|---|---|---|
| number | Pre. | Rec. | Pre. | Rec. | Pre. | Rec. |
| nifty/20 | 0.9 | 0.893 | 0.9 | 0.893 | 0.333 | 0.35 |
| localnews/14 | 0.929 | 0.879 | 0.929 | 0.879 | 0.717 | 0.717 |
| i-mode/20 | 1 | 1 | 1 | 1 | N/A | N/A |

Furthermore we tested another automatic approach based on repeated tag pattern discovery. Since most of news-like pages in big sites we investigated contain time patterns, we selected test pages without time pattern from the some small local news paper sites. We also found that some sites such as nifty.com (one of the top portal sites in Japan) have many pages containing list-oriented information without time pattern, so test pages also selected from them. Most of i-mode pages have no time pattern associated with target items, so they are also good test candidates. Table 2 shows the corresponding experimental result. Compared with the time pattern based approach the complexity of this approach is much bigger and the performance is also lower due to the complicated repeated tag pattern mining. In some cases, some irrelevant *InfoBlocks* share the same repeated pattern with target items, so the precision decreases. In the future we plan to analyze the position of each section of the HTML page in the browser, which can help us to locate data-rich regions correctly. Since most of the data-rich sections are usually displayed in the centre part of the page, and top, bottom, left and right side of the page are the noise information such as navigation, menu or advertisement (Chen, 2003). We can remove the redundant *InfoBlocks* containing the same time pattern according to the display position. Because i-mode page structure is very succinct and contains the evident repeated pattern, the corresponding extraction result is very good.

According to the above experiments, we know the automatic extraction of *category* is not easy due to its irregularity. If the target section is small or displayed in a special position, the automatic approaches do not work too. Therefore we need complement our system with a semi-automatic interactive tool. Since the tool is based on the manual labelling, the generation result can be under control and the result is always correct. The point is the complexity of the operation which is dependent on the regularity of the target page. Currently, we need 4-10 clicks to label common pages, but the operation highly depends on concrete requirements.

# 4 RELATED WORK

Since RSS feeds have great potential to help knowledge workers gather information more efficiently and present a promising solution for information share and integration, recently more and more attentions are paid to approaches for translating legacy Web contents authored in HTML into the RSS feeds. There has been some existing services and tools to “RSSify” HTML pages. FeedFire.com provides an automatic “Site-To-RSS” feed creation that allows the user to generate RSS feed for Web sites. But the FeedFire is only extracting all hyperlinks and corresponding anchor

texts in the page and does not identify the data-rich regions and desired information items for RSS generation. Therefore, the results of the RSS feed are often full of noises such as links in the regions for navigation, menu and advertisement. MyRSS.JP also provides an automatic RSS feed generation service similar to FeedFire.com, which is based on monitoring the difference between the current contents and previous contents of a Web page. The new hyperlinks emerge in current contents are extracted with corresponding anchor texts. This approach can reduce part of the noise, but the results are not good enough due to complexity of Web pages. The above two services cannot extract the release time of the information items. xpath2rss (Nottingham) is a scraper converting HTML pages to RSS feeds and uses XPath instead of using regular expressions. However its converting rule in XPath has to be coded manually.

RSS feed generation from HTML pages is a kind of specific information extraction, and there is a large body of related research on the Web information extraction. IEPAD (Chang, 2001) uses repetitive HTML tag pattern to extract the information items in a page, but it only treats the HTML page as a sequential text and does not apply the hierarchical structure of HTML page, which is useful for refining the identification of information items. (Mukherjee, 2003; Wang, 2003) also did work related to repetitive patterns, but the authors addressed a different topic, discussing how to segment HTML pages according to the semantic structure instead of information item extraction. Compared with existing work, our work focuses on the efficient information extraction for RSS feed generation and provides adaptive approaches based on the distinct features of the list-oriented information in HTML pages, consequently reaching a better result.

## 5 CONCLUSION

In this paper we present EHTML2RSS, an efficient system for converting legacy HTML pages to RSS feeds. We use two automatic approaches based on time pattern and repeated tag pattern discovery, and a semi-automatic approach, based on interactive labelling. The experimental results show that our system is highly efficient and effective for RSS feed generation. Currently EHTML2RSS has widely been used for knowledge management in our company.

## REFERENCES


Berners-Lee, T., 2001. The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. Scientific American, May 2001 Issue.

Chang, C., 2001. IEPAD: Information Extraction based on Pattern Discovery, *In the 10th International Conference on World Wide Web*, Hong Kong.

Chen, Y., 2003. Detecting Web Page Structure for Adaptive Viewing on Small Form Factor Devices, *In the 12th International Conference on World Wide Web*, Budapest, Hungary.

Gupta, S., 2003. DOM-based Content Extraction of HTML Documents, *In the 12th International Conference on World Wide Web*, Budapest, Hungary.

Gusfield, D., 1997. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge University Press; 1st edition

Hammer, J., 1997. Extracting Semistructured Information from the Web, In *Workshop on the Management of Semistructured Data*, 1997

Hammersley, B., 2003. Content Syndication with RSS, Oreilly & Associate, Inc. 1st edition.

Huck, G., 1998. Jedi: Extracting and Synthesizing Information from the Web, In *CoopIS1998, 3rd International Conference of Cooperative Information Systems*. New York.

Miller, R., 2004. Can RSS Relieve Information Overload? EContent Magazine, March 2004 Issue.

Mukherjee, S., 2003. Automatic Discovery of Semantic Structures in HTML Documents, In *7th International Conference on Document Analysis and Recognition*, Edinburgh

Nottingham, M., XPath2rss, http://www.mnot.net/

Sahuguet, A., 1999. Web Ecology: Recycling HTML pages as XML documents using W4F, In *WebDB 99.*

Ukkonen, E., 1995. On-line construction of suffix trees. Algorithmica, 14(3):249-260, Sept. 1995.

Wang, J., 2003. On Intra-page and Inter-page Semantic Analysis of Web Pages, In *the 17th Pacific Asia Conference on Language, Information and Computation*, Singapore