An Evolving Hybrid Mechanism for Stable Partial Information Extraction

Peng GAO a, Hao HAN b and Takehiro TOKUDA a

a Tokyo Institute of Technology, Japan
b National Institute of Informatics, Japan

Abstract. The Internet is providing a huge amount of information/knowledge through Web pages. For personal and effective use of such resources, the partial information extraction technology breaks a new path to enable end-users to obtain and integrate only needed information from various Web pages into original compositions. However the traditional XPath-only extraction method would fail in case Web sites use different templates to construct Web pages or change the layout of Web pages, which we call as the stability problem. In this paper, we propose a novel hybrid extraction mechanism for stably extract the partial information. We compare the original and changed Web pages to get the unchanged nodes as a keyword list and use them to generate new paths. Since the list will be re-ranked after new keywords are found, the success rates of extraction can be self evolving and correspondingly reduce manual intervention. We show the usefulness of our approach by experiment on real Web sites in practice.

Keywords. partial information extraction, stability, XPath

1. Introduction

1.1. Background

Nowadays the Internet provides access to an extraordinary and still rapidly growing amount of information through Web pages. The content of Web pages also have changed significantly from passive static documents into interactive dynamic Web applications developed using substantial amounts of server-side and client-side programs (e.g. Web-based email, social networking site, etc.). Such pages generated by program more likely have similar structured template that makes it possible to develop same rules by client-side to partially extract information/knowledge from all of them in the Web sites for personalized and professional consumption. Actually recent researches have indicated an increasing interest in technologies that automatically extract only-preferred resources for such advantages as data analysis, interactive Web service generation, information integration and reuse.

The nature of partial extraction make this kind of technology different with the Web crawler[6] that focuses on gathering whole Web pages. The integration and interaction intention are distinct it from simply wrapping and remixing partial HTML code to generate a new Web page. The typical process of partial extraction and composition is briefly introduced in four steps as following. The first step is selecting the target Web
page/application and then searching for the desired information. The next is to locate/log all data (e.g. image, video) and program (e.g. HTML, JavaScript) of the chosen contents through specific mechanisms which universally base on the XPath[12] or other similar position-record methods. The third is to extract the designated node of the target part using client-side programming. And the last step, which is optional, is to integrate and interact the extracted information with other parts to realize original content and service. For instance, the customized news extraction systems use the tree structure based approach to scrape wanted contents(e.g. text of title and body) from the semi-structured document of Web pages in target sites. As another typical example, the client-side mashup technology has evolved to enable end users(non-programmers) to design and quickly construct Web applications by just clipping necessary user interfaces (UIs) from any general Web applications. Suppose that a general User very like online shopping. With an UI-component based mashup tool, the User can construct a price comparison application of his/her own format by simple click-copy-paste level operation. Furthermore even the review/comment contents for which most Web sites do not provide application programming interface (API) can be shown in an integrated way for purchase reference.

1.2. Research Motivation

Though above introduction we can find that an important factor to complete the composition is the partial information extraction process. In this process any part of Web pages, which are not supported by Web services (APIs) (e.g. article text, JavaScript UI widgets such as the text input field, drop-down list of a form), will be considered as potential candidates of extraction targets. For example, in order to search for target information, Web applications usually provide request-submit functionality for users. A common assumption is that most of applications use the same layout show the search resulting for the same request-submit functionality. However, with our experience this assumption is receiving challenges. Firstly more Web sites use similar but different templates to construct even the same topic pages (e.g. different layout of profile page of basketball and football players), which will lead to the extraction failure or wrong result when reusing the same rule to extract contents of interest from other similar Web pages. Secondly, it is common and reasonable that users always hope to repeatedly use their compositions to retrieve, partial extract and integrate information from various Web resources. Therefore the user experience will largely influenced by the life-time of the composed applications. They suffer from a critical problem that the structure of a Web page which contains the extraction targets may change any time with no notification from Web sites. For example a web page may present data in a different sequence such as insert a volume into a table, or not include it at all. If the response Web pages use different layouts, the extraction precision would become low because paths of target parts vary with layouts of Web pages. Actually Web sites update the layout of web pages irregularly which will make the corresponding analysis, record of path and the integration processes have to be done again from the beginning. We call these two factors as stability problem which causes the XPath-only method largely lose effectiveness.

There have been a lot of research efforts addressing automated and robust extraction of specific content from Web sites by so called Wrapper technology[8,14,2,3], nevertheless they cannot work well in our case. The reason is that firstly wrappers focus on the page’s content of one or some particular Web sites. For example, a wrapper that extracts
the price’s value of products in Amazon Web site cannot work on the Yahoo Web site. However we cannot know in advance which source users would have interest to extract, because users may choose any part of any page from any site on the Web corresponding to desired components for the final composition. Even in the same page, different users are still probably choosing different parts. These make us cannot induct a sets of highly accurate rules suit all Web pages beforehand. Secondly, wrappers usually need to spend a time in training some data sets. However in the situation of partial information extraction and composition (PIEC), users usually need to accomplish their composition simply and get response quickly. Even for those programming experienced users, we cannot assume they will wait to download not a few pages from the same Web site to only get a single favorite component. Currently the common way to solve the stability problem is to relocate the path manually, which has proved to be a time-consuming task and also requiring a high level of programming expertise. Although we can use some specific system to make the acquisition process of XPath easier, as we mentioned before we still need to remake the composition from a fresh head. To sum up, we believe the stability problem is really important because it make add-value of PIE and composition technology to lost-value. We want to automatically find out the designated partial information stably with less manual intervention even if layout is updated irregularly by Web sites.

1.3. Our Contribution

As we introduced above, if the layout of the page that contains the designated part changed, the corresponding XPath will probably expired. Our main purpose of this paper is to develop a mechanism that can automatically mitigate the stability problem with less manually intervention for the client-side PIEC technology. Specifically, in this paper we make three main contributions:

- We analyze the stability problem of PIEC technology in two aspects.
- We propose a hybrid extraction mechanism that includes the customized XPath method and a novel keyword based method which uses unchanged parts of the page as keywords to dynamically update suitable paths.
- We conduct experiment on real Web sites, and demonstrate that our mechanism is effective in practice in coping with layout changes in Web sites.

In Section 2 we address the background knowledge of XPath and analyze two aspects of the stability problem. We describe our self-evolving hybrid extraction mechanism in Section 3. We present the experiment data, analysis of our findings in Section 4, discuss related work in Section 5 and conclude in Section 6.

2. Problem Statement

2.1. XPath-only based Extraction

XPath (XML Path Language [12]) expression is initially proposed as a query language for selecting nodes from an XML document. It is based on a tree representation of the XML document, and provides the ability to navigate around the tree, select nodes by a variety of criteria and compute values (e.g., strings, numbers, or boolean values). Com-
monly, an XPath expression is often referred to simply as an XPath and also used to target a node in HTML document by parsers such as HTML Parser \(^1\).

```html
<HTML>
  <head>
    <title>a simple HTML document</title>
  </head>
  <body>
    <table border=4 width=250 align=center id="teacher_list">
      <caption>Tutor List</caption>
      <tr>
        <th>Name</th>
      </tr>
      <tr align=center>
        <td>Prof. White</td>
      </tr>
    </table>
    <table border=4 width=250 align=center id="student_list">
      <caption>Student List</caption>
      <tr>
        <th>Name</th>
        <th>Gender</th>
        <th>Age</th>
        <th>ID</th>
      </tr>
      <tr align=center>
        <td>Tom</td>
        <td>Male</td>
        <td>18</td>
        <td>001</td>
      </tr>
      <tr align=center>
        <td>Mary</td>
        <td>Female</td>
        <td>19</td>
        <td>002</td>
      </tr>
    </table>
  </body>
</HTML>
```

**Figure 1.** Figure SampleHTML

As a simple student&teacher information table shown in (Figure SampleHTML), if we want to target and extract the names of students, we can use the XPath as follows:

- `/html/body/table[2]/tr/td[1]`
- `//*[@id="student_list"]/tr/td[1]`

The first XPath fully describes path information of target nodes from top node `<html>` and the second XPath uses node having id attribute as a relative referred node. They can run well in this example page and have their own merits in extraction when the page is changed. For example, if a new table is added over the table Tutor List, the first XPath would lose the effectivity of extraction but the second XPath could still extract success-

\(^1\)http://htmlparser.sourceforge.net
fully. However, if value of attribute id is changed or removed and no new table is added over table Student List, the second XPath could not target the node but the first XPath could keep the right extraction. Therefore XPath-only method is not sufficient in complex node searching and extraction during long period of time, especially for frequently updated websites/webpages.

2.2. Layout Change Model

In this paper we deal with such two scenarios illustrated in 2 that both cause the stability problem and make the XPath-only method fail.

Figure 2. Change Scenarios

Suppose that the Web site has \( i \) template(s) for the a content topic. For each template at time \( t = t_0 \), it consists of a set of entity units \( u_{11}, u_{12}, \ldots, u_{1m} \) which can be extracted in accordance with users’ choice. The two aspects of stability problem are presented as following. The first change scenario is that at time \( t = t_0 \), the user select and extract entity units (e.g. \( u_{11}, \ldots, u_{1m} \)) in template 1 and use the corresponding XPath to extract desired units \( u_{i1}, \ldots, u_{im} \) in template \( i \) which is similar but different with template 1. The second change scenario is that at time \( t = t_0 \), the user select and extract entity units (e.g. \( u_{11}, \ldots, u_{1m} \)) in template 1 based on XPath 1. Then at time \( t = t_1 \), the user reuse the XPath 1 directly to extract the desired entity units in the template 1’ which is a changed version of template 1 consists of \( u_{11}', u_{12}', \ldots, u_{1m}' \).

In scenario 1, even the Web site holds a huge number of pages, the amount of templates corresponding to the same topic are still limited and they usually have a very a-like structure. Therefore the extraction mechanism should be fast and have the ability to withstand small structural variations. In scenario 2, the page’s layout may change visibly in a future time point, accordingly a more stable mechanism is needed to deal with such change. And after getting extraction rules in the future version template 1’ we can use the same method with scenario 1 to handle the extraction of \( u_{i1}', u_{i2}', \ldots, u_{im}' \) in template \( i' \).
3. Our Solution

3.1. Approach Overview

In this section, we give an overview of our evolving hybrid mechanism for stable partial information extraction illustrated in Figure 3. We explain the main components here and describe them in details in the following sections.

Step 1. Selection of only needed parts (e.g. units 11 - 1m in template 1 and units i1 - in in template i) from Web pages in the target Web sites.
Step 2. Use of Extraction of the selected parts by XPath-only method
Step 3. Judgement of whether the extraction is fail: if fail, go to Step 4; if not fail, go to Step 6
Step 4. Choice of a stable keyword
Step 5. Generation of a new XPath then return to Step 2
Step 6. Finish of extraction then locate the new keywords
Step 7. Re-rank of keyword list

3.2. Data type Definition

There are various kinds of information in Web pages such as text string, images, and links. In order to extract information accurately and save the selected parts conveniently, we define data type for the extraction of partial information besides XPath for each target part. Data type represents “What kind of information is needed?” For example, in a Web page, usually each link contains an anchor text associated with a URL. Without the specified data type, we cannot get the right information because we do not know which one is needed between text and URL.

The data type includes two kinds of information: property and structure. Property is text, object or link. Text is the character string in Web pages such as an article. Object is one instance of the photo, video and other multimedia file. Link is a reference in a hypertext document to another document or other resource. Structure is single occurrence or continuous occurrence. A single occurrence is a node without similar sibling nodes such as the title of an article, and the continuous occurrence is a list of nodes with similar paths such as result list in a search result page. An example is showed in Figure 4

3.3. Extraction Mechanism (XPath)

3.3.1. XPath Definition

Different parsers, browsers and extractors support different types of XPath. In this paper, we use the following customized format to describe and save a XPath: $T_1[O_1][A_1_{-0} = "V_{1-0}"]...[A_0_{-m} = "V_{0-m}"]...[/T_n][O_n][A_n_{-0} = "V_{n-0}"]...[A_n_{-m} = "V_{n-m}"].../T_{n+1}...$, where $T_n$ is name of the n-th node/tag, $[O_n]$ is order of the n-th node/tag among its siblings having $T_n$ as names, $A_{n-m}$ is name of the m-th attribute of n-th node/tag, $V_{n-m}$ is the corresponding value of attribute $A_{n-m}$, and $T_n$ is parent node/tag of $T_{n+1}$ ($O_n$ and $A_{n-m}$ are neglected if there are no siblings or attributes or they are not fixed in the node searching).
For solving the path expiration problem showed in (Figure SampleHTML), the deviation is introduced into calculation of similar XPaths for stronger and robust node searching. Two paths are similar to each other, if these two paths have the same forms ignoring the difference of orders of nodes among sibling nodes, and the difference of orders is within a defined deviation range as follows:

$$T_1[R_1][@A_{1-0} = V_{1-0}][...][@A_{0-m} = V_{0-m}][...]/T_n[R_n][@A_{n-0} = V_{n-0}][...][@A_{n-m} = V_{n-m}][...]/T_{n+1}... R_n \in (O_n - H, O_{n + H})$$

where, $H$ is deviation value and $|R_n|$ is order. For example, /html/body/table[2]/tr/td[1] is similar to /html/body/table[3]/tr/td[1]. However attribute value is character string and concept of deviation is not suitable, which make XPath-only method is not sufficient in complex node searching and extraction during long period of time.

3.3.2. Extraction Process

Among the information extraction from two kinds of data structures, the extraction from a single node is the most easy and basic. For example, the extracted information of a photo is the value of attribute “src” of node $<img>$, and the extracted information of a link is the value of attribute “href” of node $<a>$. The extraction from continuous nodes...
Figure 4. Data type and Keyword Example

is based on the tree structure of HTML document. We use the following steps to extract
the corresponding node list of node $N$ if the data type is continuous occurrence:

1. We use the path to find the corresponding node $N$.
2. We get the parent node $P$ of $N$.
3. We get the subtree $S$ whose root node is $P$.
4. We get the node list $L$ of which each has the same path as $N$ without considering
   the orders of child nodes of $P$ under $S$.
5. If we find two or more than two nodes in $L$ and these nodes are different from
   the nodes of other selected parts, or $P$ is $<$body$>$, then $L$ is the final node list.
   Otherwise, we set the parent node of $P$ as the root node of $S$; then go to Step 4.

3.4. Extraction Mechanism (keyword)

XPath reflects a logically structural information of selected parts such as used nodes and
attributes in HTML document. It is invisible for Web users if they do not view HTML
codes of Web pages. Actually, general users recognize target parts from Web page with-
out viewing each part shown in Web page because there are some common characteristics
of presentation of page in Web browsers. For example in Figure 4, in a searching result
page there are usually a text string showing message about sum of matching results over
result list, which indicates the position of results and could not disappear even if layout
of pages is changed. Some of these characteristics, called as “keyword” by us, are more
steady than XPath as a relative refer when Web sites update the layout of Web pages.
We use the following information set to describe a keyword for further usage as extraction index: P, X, K, T, R. P is the Web page, X is XPath of keyword, K is text string shown in keyword if it is a text string or text link, T is the data type of keyword, R is the relative position such as previous or next to target node in HTML document, which represents the node distance between the keyword and the target information. The keyword is a very important element during our searching and works as an index of our target information. We need to find a suitable keyword because a suitable keyword can give a more precise index information.

3.4.1. New Path Generation

Each time the user chooses an entity unit of interest from a Web page, the corresponding URL, XPath, id and data type of the unit and the page will be logged. In case the path expired, we compare the web pages of the former one and the latest one to find unchanged nodes as candidate keywords. The keywords will be used to re-locate the possible paths of the originally selected unit. Many research results address technologies of the change detection for XML documents[15,9,1]. We can take some advantages of their merits as references to suit our requirements.

The tree structure of a Web page can be changed by three edit operations: insertion, deletion of a node, and substitution of the content or label of a node. The case is out of our scope that the deletion of a node and assigning all of its children as its parent’s children. Because no matter what parts were chosen by the user at beginning, if there are not included in other templates or deleted in the future version we cannot expect any method to recover the original contents. About substitution of the content of a node, we also do not count it as a target because that’s the way the the chosen nodes are supposed to be. For example, if the user has chosen and integrate a node that contains daily news, of course the content should dynamically changed everyday corresponding to the Web site’s updating.

Suppose that the chosen node is P the red node shown in Figure5. The blue nodes are original path. The green nodes are changed nodes by inserting or deleting.

Each operation of inserting a node corresponding to the chosen node can be done in three cases. The grey nodes are most suitable keyword selected from keyword list.

- Insertion of a node to right side columns (Figure 5(b)). Original XPath will still work because the order of rendering is from left to right.
- Insertion of a node to left side columns (Figure 5(c)Fig.sub.4). We can only locate the parent node of p (the "table" node of second branch) base on the render order, therefore the XPath-only method will fail. In this situation, we choose a keyword from the keyword list for re-locating the path of the target node P. We do a reversal search and choose the node that does not belong to the same level with p and has the shortest distance to the p, which will be the "table" node of third branch. The backward distance is 2 and we can know this value in advance by computing the original tree (Figure 5(a)).
- Insertion of a node to upper side within the same branch (Figure 5(e)). The solution is the same with insertion of a node to left side columns.

Each operation of deleting a node (not the target node) corresponding to the chosen node can also be done in three cases.
• Deletion of a node to right side columns or child node of the left side columns

Figure 5. Edit operation and keyword choice
(Figure 5(g)). Original XPath still works because the order of rendering is from left to right.

- Deletion of a node to left side columns (Figure 5(h)). We choose a keyword from the keyword list for re-locating the path of the target node $P$. We do a backward search and choose the node that does not belong to the same level with $p$ and has the shortest distance to the $p$, which will be the "table" node of third branch. The backward distance is 2 and we can know this value in advance by computing the original tree (Figure 5(a)) and generate the new XPath.

- Deletion of a node to upper side within the same branch (Figure 5(f)). The solution is the same with deletion of a node to left side columns.

3.4.2. Discussion

We circularly use the tree structure. For example if the target information has been in the position of the rightmost of the tree structure, correspondingly we use the leftmost branch as its "right" branch.

In practical page change scenarios the page change may be much more complex, which means a combination of changes happened in the tree structure. For example, in case of inserting nodes. The page of future version containing target node may consist of both insertion of nodes to right side and left side columns. When only using reverse search to compute the new XPath of target node we still cannot get the result directly. Therefore we use not only one keyword to locate the target information. For example, we combine the cases in Figure 5(b) and Figure 5(c). If the date type of target information is continuous then it will be still work because the new inserted content is the same with the original ones. In case the date type is single, we use two keywords to narrow the range that may contains target information (in this example the keywords will be the two "table"). Then we use other tag information such as "id" contents or others to search the target information. The "id" content maybe have been deleted or changed therefore we do not directly enforce the string matching for retrieving it. Just as the Google searching result example we introduced above the time cost of searching contains a string "results" and "seconds". Therefore the following nearest list structure will be the searching results we want to extract.

We may get a lot of unchanged nodes by comparing two similar Web pages. How to choose the appropriate candidate is very important to locate the target information. Also is a key point to enforce self-evolving intention. We rank the nodes that have shortest tree edit distance with the node of target information. After several circle, the keyword will frequently appear and have some specific format or relationship with other entity units, which can get a high credit for reuse in future.

4. Experiment and Evaluation

We collected history pages from Internet Archive\(^2\) which is a Web site storing historical versions of Web pages on the Internet. We obtained Web pages from this archive data source for three different domains:

\(^2\)www.archive.org
For testing we need to confirm whether an extraction is correct. Because each result is directly showed to users, normally we manually inspect each extraction result to check if the target information is correct. However as if we use more large test set, this approach will be tedious. Therefore in future we want to develop a test scheme that semi-automatically test the chosen parts one by one across all template or versions. The experiment result shows that for each one site of the test set, because the page layout changed largely XPath-only method only get average 9% success rates. And with our methods we can get nearly 39% success rate and with light weight manual adjusting we get nearly 60% success rate.

5. Related Work

**XPath Oriented Related Work** Taking advantage of the fact that a great and increasing number of Web pages are rendered by template system, a number of approaches have been proposed to analyze the structure of Web pages with the purpose of manual or semiautomatic example-based information extraction.

XSLT [13] uses the defined path patterns to target nodes that match with given paths repeatedly and outputs data by using information of the nodes and values of variables. ANDES [7] is a XML-based methodology to use the manually created XSLT processors to realize the data extraction from Web pages. Similarly, Marmite [16], implemented as a Firefox plug-in using JavaScript and XUL, uses a basic screen-scraping operator to extract the content from Web pages and integrate it with other data sources. The operator uses a simple XPath pattern matcher and the data is processed in a manner similar to Unix pipes.

PSO [10] is an approach to extract the parts of webpages. It keeps the view information of the extracted parts by using the designated paths of tree structures of HTML documents.

Crunch [4] is a HTML tag filter to retrieve content from DOM trees of Web pages after analyzing HTML documents of Web pages.

All these methods have the stability problem as we mentioned above. When Web sites update Web pages by modifying templates irregularly, extraction process would fail because paths are no longer available or wrong results are extracted. The extraction mechanism employed by these methods makes it difficult and impossible to detect the new and suitable paths automatically or dynamically.

Dapper [17] is a screen scrap tool that allows users to extract information from Web page. It has a powerful GUI that allows users to scrap data off a Web page without programming or even technology like regular expression or XPath. However, it is not very intelligent to recognize a long list. For a list, users may have to select each individual item. Moreover, users have to provide sample Web pages for analysis and execute selec-
tion separation many times for enhancing precision, which still could not keep extraction procedure steady after Web sites update layout of Web pages.

**Web clipper** [5] is a Web authoring environment that enable end-user to dynamically extract information from other Web resources. An advantage of the approach is that it enforces a robust mechanism based on multiple extraction patterns defined in advance. The system could only provide users an environment to extract content and construct a personalized Web page that make it weak than now popular UI-component based mashup systems. The training process carried out manually can be tedious if users want to get a ideal extraction result. For each component from Web page, the Web clip need to do at least once the training process even for the pages that generated by very similar templates. As the extraction patterns increasing, the training process goes to be long that will drop user experience and affect system performance.

**Regular express** Regular expression provides a concise means for matching strings of text, such as particular characters or patterns of characters. A regular expression is written in a formal language that can be interpreted by a regular expression processor. The resulting pattern can then be used to create a matcher object that can match arbitrary character sequences against the regular expression. Regular expression runs more flexibly with various logical, intersection, and union operators. However, generating a set of regular expressions [11] costs more time than XPath. If web sites update the layout of web pages, regular expressions become weak and the regeneration of expression needs more manual work.

6. Conclusion

In this paper we focus on the stability problem of the partial information extraction and composition. We propose a novel hybrid extraction mechanism for stably extract the partial information. We compare the original and changed Web pages to get the unchanged nodes as a keyword list and use them to generate new paths. Since the list will be re-ranked after new keywords are found, the success rates of extraction can be self evolving and correspondingly reduce manual intervention. We show the usefulness of our approach by experiment on real history page of Web sites in practice and get six times success rate than traditional XPath-only methods.

References


