RC 10245 (A0205-025) 23 May 2002 **Computer Science**

IBM Research Report

Robust Web Data Extraction with XML Path Expressions

Jussi Myllymaki, Jared Jackson

IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

Robust Web Data Extraction with XML Path Expressions

Jussi Myllymaki **Jared Jackson**

IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120, USA
`{jussi,jjared}@almaden.ibm.com`

Abstract

Automated extraction of structured Web data has attracted considerable interest in both the academia and industry. A particularly promising approach is to employ XML technologies to translate semi-structured HTML documents to “pure” XML documents. In this approach, HTML documents are first normalized into XHMTL and then mapped to the desired XML application format by using XML path expressions and regular expressions.

In this paper we describe a methodology for creating XML path (XPath) expressions that are capable of extracting data from virtually any HTML page, while placing an emphasis on the persistent integrity of these expressions. This robustness is critical given the vulnerability of extraction technologies to the continually changing content, structure, and formatting of pages on the Web. We define categories of extraction rules in terms of their dependence on content, structural, or formatting features, and provide practical tips on how to create dependable data extraction patterns for the Web.

1 Introduction

Automated extraction of structured Web data has attracted considerable interest in both the academia and industry in recent years. Despite this attention it is still unclear which tools, technologies, and standards should be employed for Web data extraction. Whether extracting data from a source as disorganized and chaotic as the Web makes sense at all has also been questioned. However, the general consensus appears to be that the Web is just too rich a data source to be excluded from any data analysis task. Yet, the problem remains that HTML, as the dominant document format on the Web, is intended for design and layout purposes and not data exchange. This makes it difficult to use Web pages as a general data source for data-driven computation and applications.

We note that, in the future, much of the content on the Web may be available in formats suitable for automated processing, in particular through the Extensible Markup Language (XML) [XML]. Despite being a relatively new technology, XML has become absolutely essential for enabling data interchange between otherwise incompatible systems. However, the volume of XML content available on the Web today is still miniscule compared to that of HTML. It is therefore reasonable to study ways of translating existing HTML content to XML, and thereby expose more Web sites to automated processing by end users and application programs.

A particularly promising approach for extracting structured data from HTML documents is to employ XML technologies to translate HTML to “pure” XML. In this approach, HTML documents are first normalized into Extensible HTML (XHMTL) [XHTML] and then mapped to the desired XML application format by using XML path (XPath) [XPATH] expressions and regular expressions. An increasingly popular method for programming such expressions is to use XML Stylesheet Language Transformations (XSLT) [XSLT] documents.

For Web data extraction to be tractable and reliable, one needs to ensure that data extraction expressions continue to function properly even if the structure or formatting of a source HTML page changes. In this paper we describe a methodology for creating XPath expressions that are capable of extracting data from virtually any HTML page, while maintaining the robustness of these expressions. We define categories of extraction rules in terms of their dependence on content, structural, or formatting features, and provide practical tips on how to create dependable data extraction patterns for the Web.

2 Related Work

The early work on query languages for the World Wide Web exploited the graph structure of the Web and resulted in the definition of *regular path expressions* over graphs of hyperlinked documents [FLO98]. Examples of these SQL-like query languages include W3QL [KON95] and WebSQL [MEN97].

More recently, the attention has shifted to exploiting the *internal structure* of an individual Web page. The goal in this line of work is to decompose an HTML document and translate its semi-structured content into a well-structured format, or better yet, a precise database schema. Research on this problem typically follows one of two paths. In database systems research, the focus is usually on wrappers that translate a database query to a Web request and transform the resulting HTML page to a relational dataset (the concept of “compact skeletons” [RAJ01] could be used to deduce the relational schema automatically). The extraction patterns are typically generated either manually or through the user of a graphical tool and require continual monitoring and maintenance to respond to changes in Web site formatting.

In contrast, machine learning techniques developed in the artificial intelligence field analyze a set of sample Web pages that share a common format and automatically generate rule sets that extract data from other similar pages. High success rates have been reported in learning such rule sets with very few sample pages and limited human involvement [KNO00, KUS99, RIB99]. These automated techniques also make extraction failure detection and recovery more feasible.

Instead of maintaining a strict categorization to manual “wrapper development” (or “knowledge engineering”) tools and automated “wrapper induction” tools, we argue that AI-based information extraction tools and user-driven wrapper development tools are a perfect complement to each other. Ideally, automated machine learning tools would generate a large fraction of extraction rules that would otherwise be written manually (including those described in this paper), leaving the user with the power to define extraction rules that cannot be learned or need manual fine-tuning.

In our research on Web data extraction, we emphasize the importance of middleware solutions that extract entire databases from target Web sites and make these datasets available for data mining and other analysis (similar to the Junglee system [GUP97]). This involves crawling target Web sites periodically, extracting structured data, and performing domain-specific feature extraction. Hence, our work encompasses a middle ground between database systems, AI, and Web systems research in general.

Our ideas have been implemented in ANDES, a software framework that merges crawler technology with XML-based data extraction technology [MYL01]. ANDES is similar to other extraction systems in that it defines a wrapper for each Web site of interest. The underlying data extraction method uses XSLT, which combines templates, path expressions, and regular expressions into a concise package. Templates can be used to decompose the data extraction process hierarchically, as is done in TSIMMIS [HAM97] and STALKER [KNO00]. An XML path expression can traverse an HTML document recursively and express predicates (WebLog [LAK96]), context and delimiter patterns (WHISK [SOD98]), and token features (SRV [FRE98]). Finally, regular expressions, which are an extension to XSLT, permit decomposition of plain-text fields (leaf nodes) of an HTML tree.

Note that many other data extraction systems process documents *linearly;* consider the forward and backward token rules in STALKER, the head-left-right-tail delimiters in the HLRT wrapper class [KUS00], and the prefix/infix/postfix expressions in [SAT99]. In contrast, XML path expressions take full advantage of the *tree structure* of HTML documents, making it easy to visit ancestors, siblings, and children before and after the current position in the document. For instance, finding the nth *top-level* TABLE element in a document is trivial using an XPath expression, but may be impossible using linear expressions due to the possibility of TABLE elements containing an unknown number of nested TABLE elements.

Below we briefly describe the details regarding a few Web data extraction systems and pattern languages proposed in the literature.

The WysiWyg Web Wrapper Factory (W4F) is a toolkit for generating Web wrappers [SAH99]. It contains a language for identifying and navigating Web sites (retrieval rules) and a declarative language for extracting data from Web pages (extraction rules). It also provides a mechanism for mapping extracted data to a target structure. As its name suggests, W4F provides a graphical user interface for generating retrieval, extraction, and mapping rules. While W4F and ANDES are similar in many respects, their main difference is that whereas W4F uses a proprietary language for data extraction and mapping rules, ANDES is based on the XSLT and XPath standards.

The goal of WIDL is to define a programmatic interface to Web sites [ALL97]. As such, it focuses more on the mechanics of how to issue a request to a Web site, retrieve the result, and bind the input and output variables to a host programming language, than the process of extracting data from the retrieved result page. WIDL allows data to be extracted using absolute path expressions, but, as we explain in Section 4.1, this falls short of building robust data extractors.

The Web Language (formerly WebL) from Compaq is a procedural language for writing Web wrappers [WEBL]. While it provides a powerful data extraction language (similar to recursive path expressions combined with regular expressions), the language is not tuned to XML inputs and outputs and lacks the power of XSLT templates and XPath axes and operators.

XWRAP [LIU00] is a semi-automatic wrapper-generator that builds on the semantic meaning of specific HTML tags (e.g. headings and tables) and how they are used for data layout. Heuristics are used to determine the parent-child relationships between data items, for instance table names, field names, and values. The resulting wrappers depend on the nesting and orientation of table and other elements, which works well with tabular Web sites but not with sites that have less structure. For instance, some Web sites concatenate several data items into a single plain text field, which requires regular expressions or similar text analysis tools to decompose the field back into the original data items.

Informia [BAR98] is an information mediation system whose Common Access Interface (CAI) is configured with retrieval and extraction rules, like W4F. The retrieval component was designed primarily to handle Web sites that contain repetitive data such as search result lists. Informia produces extraction rules automatically for pages that contain repetitive elements (analogous to *rel-infons* in [LAK96]) and employs induction algorithms similar to those described in [LER01]. However, the pattern language is proprietary and extractors would need to be created manually for sites with no repetitive data.

3 Extracting XML Data from HTML

3.1 On the Evolution of HTML Page Design

From its infancy in the early 1990's until just a few years ago, the HTML language evolved continuously, introducing increasingly complex design elements. Design elements such as tables within tables, frames, and image maps, were among the early additions. Later came client-side scripting, including its handling of mouse events (e.g. mousing-over an image), which improved the interactivity of Web sites and allowed them to function more like real applications.

In recent years, however, the makeup of HTML has stabilized and Web developers have shifted their focus to more programmatic Web standards such as XML and Web services. Today, HTML no longer evolves as a language and, apart from incompatibilities that exist between the HTML features supported by different browsers, it is fair to say that HTML itself and related development and design tools are mature and produce consistent output.

As a result of these developments, it is reasonable to assume that certain design paradigms in Web sites are programmed using a consistent and predictable set of HTML constructs. For instance, excepting complex graphics and client-side scripting, there is only one way to create pull-down menus on a Web page: using <select> and <option> tags. Similarly, text input boxes, radio buttons, and check boxes are specified using a well-established set of HTML tags. We refer to this predictability as "forward-looking robustness" in Section 4 where we discuss robustness of data extraction rules in detail.

The predictability of some HTML features doesn't remove the inherent uncertainty that accompanies the most critical aspect of page design for data extraction: page layout. Page layout is defined almost exclusively with nested <table>, <tr>, <td>, and <frame>

tags. As anyone who has observed the evolution of the Web in recent years can attest, Web pages have become increasingly more complex, with whole sections of a page being assigned to serve various business and technical needs; consider the wealth of advertising, standard headers/footers, navigation sidebars, polls, and search bars on Web pages.

As a consequence of the increasing structuredness of Web pages, over time the “main content” of a page is pushed deeper down the HTML tree. This means that more often that not, the “interesting” data is found somewhere in a deeply nested table, and raises the question: how do you find that “interesting” data in a robust manner?

We also note that today’s Web servers and application servers make it easy to combine the output from different applications and databases into a single HTML page. On a typical Web portal site, each section of the page (weather, news, stock quotes, etc.) may be produced by an independent “portlet.” In this distributed approach, the design process has the additional burden of ensuring overall compatibility between the different sections. Unless a total overall control is maintained, there is a possibility of generating aggregate HTML that is simply invalid (e.g. two <body> elements, two forms with the same name, conflicting client-side scripts, or binary characters emanating from incorrectly translated include files).

3.2 From HTML to XHTML

The stability of the HTML syntax doesn’t mean that all HTML documents have the proper syntax. On the contrary -- most HTML content on the Web is “broken” or ill-formed in some way. An increasingly popular method for making HTML processing easier is to normalize it into XHTML before attempting any other processing on it. XHTML is the format of choice for two reasons. First, it may one day be widely used as a source format on all Web sites and, as XHTML Basic, even extend to pervasive devices with limited browser support. Also, tools such as the HTML Tidy [TIDY] package already exist for converting bad HTML to proper XHTML. Secondly, and perhaps most importantly, XHTML is XML and therefore compatible with the broad range of XML tools and technologies developed in the past several years.

In XHTML, annoying errors in HTML content are fixed. Table elements are nested correctly, tags are not missing, duplicate attribute values cannot exist, and binary characters are tolerated only when properly encoded. This consistency removes interpretation ambiguities, which helps in constructing robust data extraction expressions and mimicking the processing rules implemented in browsers. In essence, what the Web designer intended the human user to see in a browser, is exactly what the data extractor sees at the source code level.

3.3 Data Extraction Process

Since XHTML is based on XML, any XML tool can be used to further process XHTML pages. A powerful approach for processing an XHTML document tree is to use XML path expressions (XPath) and, when necessary, combine it with regular expressions. The template mechanism in XSLT stylesheets provides a convenient method for associating data extraction patterns (XPath and regular expressions) with matching elements in an XHTML document. Using a compact notation, the stylesheet defines what

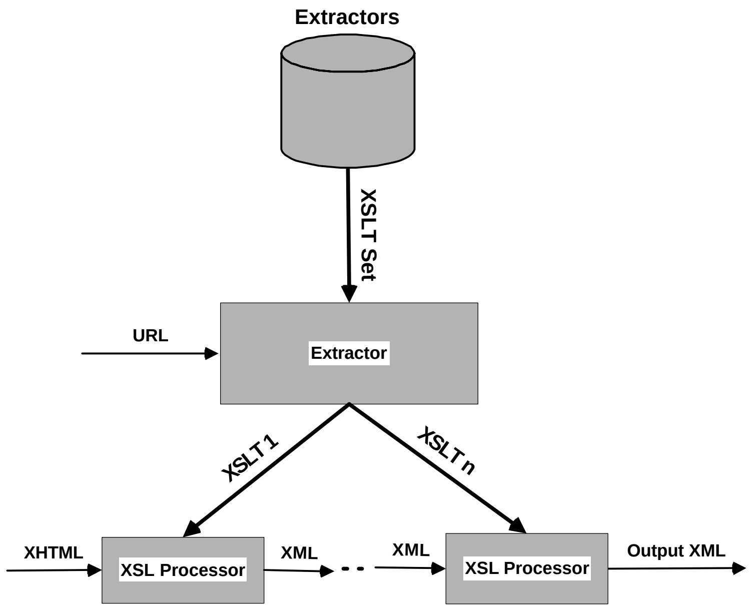


Figure 1. Extractor identifies XSLT files to be used. A pipeline of XSLT processors extract and refine data, yielding a domain-specific markup file as the ultimate output.

to extract from an XHTML document and what to output as a resulting XML document. A simple data extraction task can be accomplished with data extraction rules contained in a single XSLT file. However, in more complex situations it may be preferable to conceptually divide the tasks into a sequence of data extraction steps.

Assume a broader context for data extraction where a sequence of XHTML documents flow through a data extractor and for each XHTML document a set of XSLT files are identified and executed. The URL of an XHTML page is the most likely candidate for determining which files to apply. The XHMTL document is passed through the first XSLT file and the output is pipelined through other XSLT files defined for that URL (Figure). The final output is an XML file whose structure and content is determined by the last XSLT file in the pipeline. The exact markup produced is of course domain-specific, for example NewsML could be used for extracted news articles.

4 Building Robust Data Extraction Expressions

4.1 What Makes a Good Extraction Rule

We now shift our focus to data extraction expressions and consider the various factors that make a particular expression robust. By robustness we mean that the expression continues to extract the intended data item from an HTML page, even when the page design or structure changes. The main criticism directed towards existing Web data extraction efforts is a lack of robustness, i.e. that it fails miserably when the design or structure of a Web site changes. While total isolation from these changes is difficult to

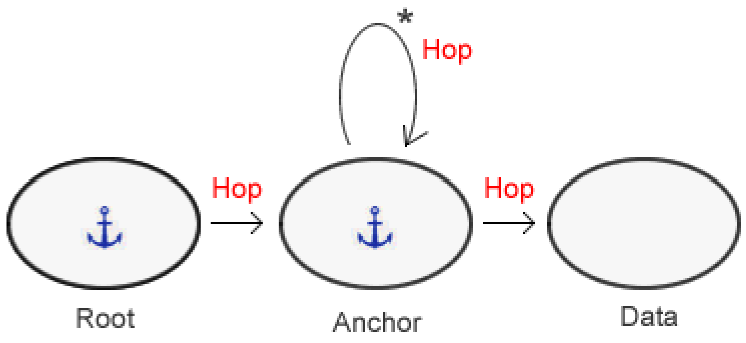


Figure 2. A traversal graph is constructed by following paths of hops and anchors down to the desired data.

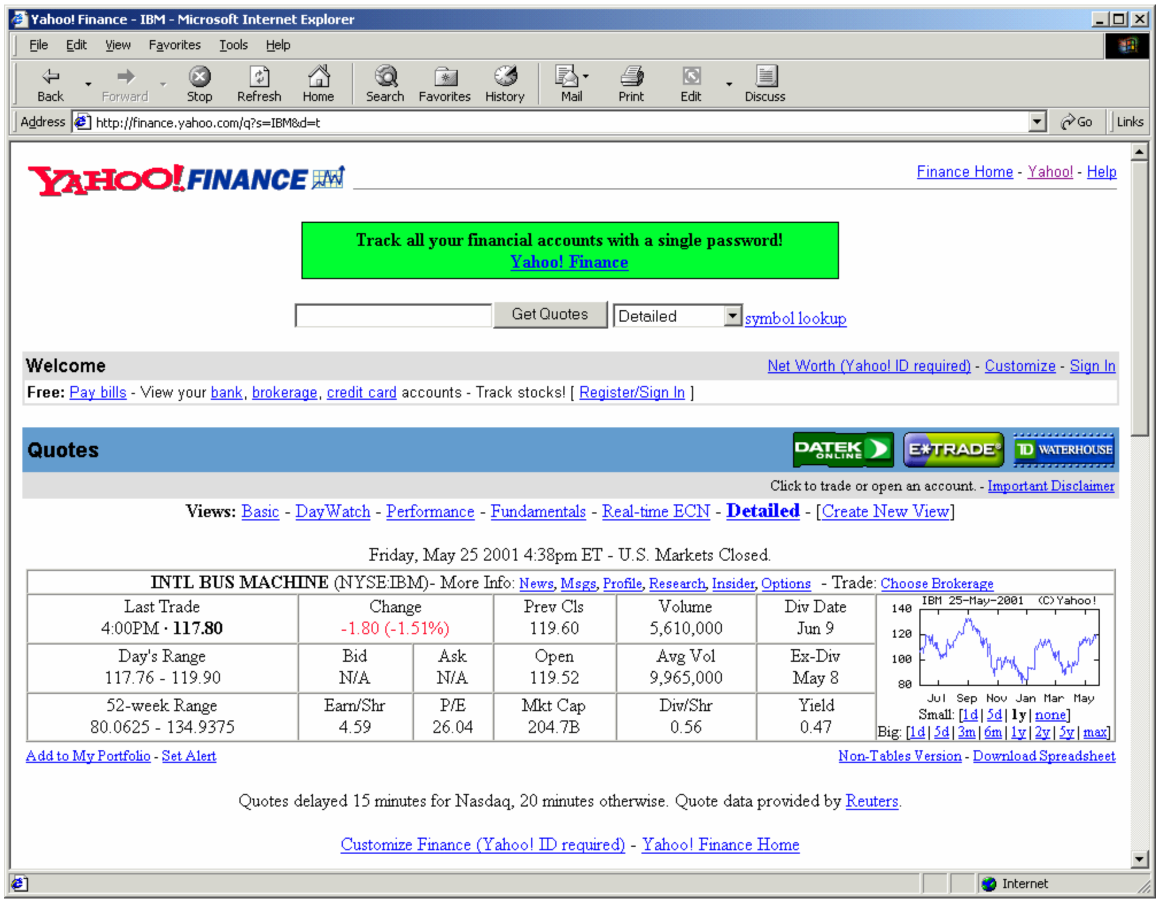
achieve, we believe that it is possible to create solid and robust data extraction rules. This is achieved by relying less on page structure and more on content.

We define two metrics, distinct in their approach, that can be used to formally evaluate the robustness of an expression. The first is an empirical measure of the performance of an expression over time. The simplest measure is to record the frequency of failure and subsequent correction of an expression applied to a specific Web source over a lengthy period of time, perhaps six to twelve months. This metric provides a good reality check of the usefulness of an expression. It does, however, fail to account for the frequency and magnitude of changes to a particular Web site. A particularly poor (i.e. structurally dependent) expression used on a site that never changes might receive a better rating than a more flexible expression used on a site whose structure changes significantly and often.

The second approach is to examine robustness outside the context of its specific use, instead considering trends in site updates and dependencies of the expression on the structure of the data source. This metric examines the number of dependencies an expression or a chain of expressions has to the document, the depth of the expression, and the method of 'anchoring' the expression. We call the former metric *a posteriori* robustness and the latter *a priori* robustness.

Some wrapper languages (e.g. HTML Extraction Language in W4F) require the use of absolute HTML paths that point to the data item to be extracted. An absolute path describes the navigation down an HTML tree, starting from the top of the tree, the <HTML> tag, and proceeding towards child nodes that contain the data to be extracted. The path is made absolute by the fact that it lists tag names expected to be seen in the tree and their absolute positions. For instance, an absolute path to the third table, first row, and second column in an HTML document could be expressed as the XML path expression /HTML/BODY/TABLE[3]/TR[1]/TD[2].

The absolute path approach is almost certain to fail when the design or structure of an HTML page changes (it has low *a priori* robustness). In contrast, a "content-based" or "attribute-based" approach yields more robust results by keying in on the actual content to be extracted, or other content near the content to be extracted. At an abstract level, the content-based method looks for an occurrence of a word or phrase that is known to stay


Yahoo! Finance - IBM - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address http://finance.yahoo.com/q?s=IBM&d=t
Go Links
Internet

YAHOO! FINANCE

Finance Home - Yahoo! - Help

Track all your financial accounts with a single password!
Yahoo! Finance

Get Quotes Detailed symbol lookup

Welcome Net Worth (Yahoo! ID required) - Customize - Sign In

Free: Pay bills - View your bank, brokerage, credit card accounts - Track stocks! [Register/Sign In]

Quotes DATEK ONLINE E*TRADE TD WATERHOUSE

Click to trade or open an account. - Important Disclaimer

Views: Basic - DayWatch - Performance - Fundamentals - Real-time ECN - **Detailed** - [Create New View]

Friday, May 25 2001 4:38pm ET - U.S. Markets Closed.

INTL BUS MACHINE (NYSE:IBM)- More Info: News, Msgs, Profile, Research, Insider, Options - Trade: Choose Brokerage

Last Trade	Change		Prev Cls	Volume	Div Date
4:00PM · **117.80**	-1.80 (-1.51%)		119.60	5,610,000	Jun 9
Day's Range	Bid	Ask	Open	Avg Vol	Ex-Div
117.76 - 119.90	N/A	N/A	119.52	9,965,000	May 8
52-week Range	Earn/Shr	P/E	Mkt Cap	Div/Shr	Yield
80.0625 - 134.9375	4.59	26.04	204.7B	0.56	0.47

IBM 25-May-2001 (C) Yahoo!
140 120 100 80
Jul Sep Nov Jan Mar May

Small: [1d | 5d | **1y** | none]
Big: [1d | 5d | 3m | 6m | 1y | 2y | 5y | max]

Add to My Portfolio - Set Alert

Non-Tables Version - Download Spreadsheet

Quotes delayed 15 minutes for Nasdaq, 20 minutes otherwise. Quote data provided by Reuters.

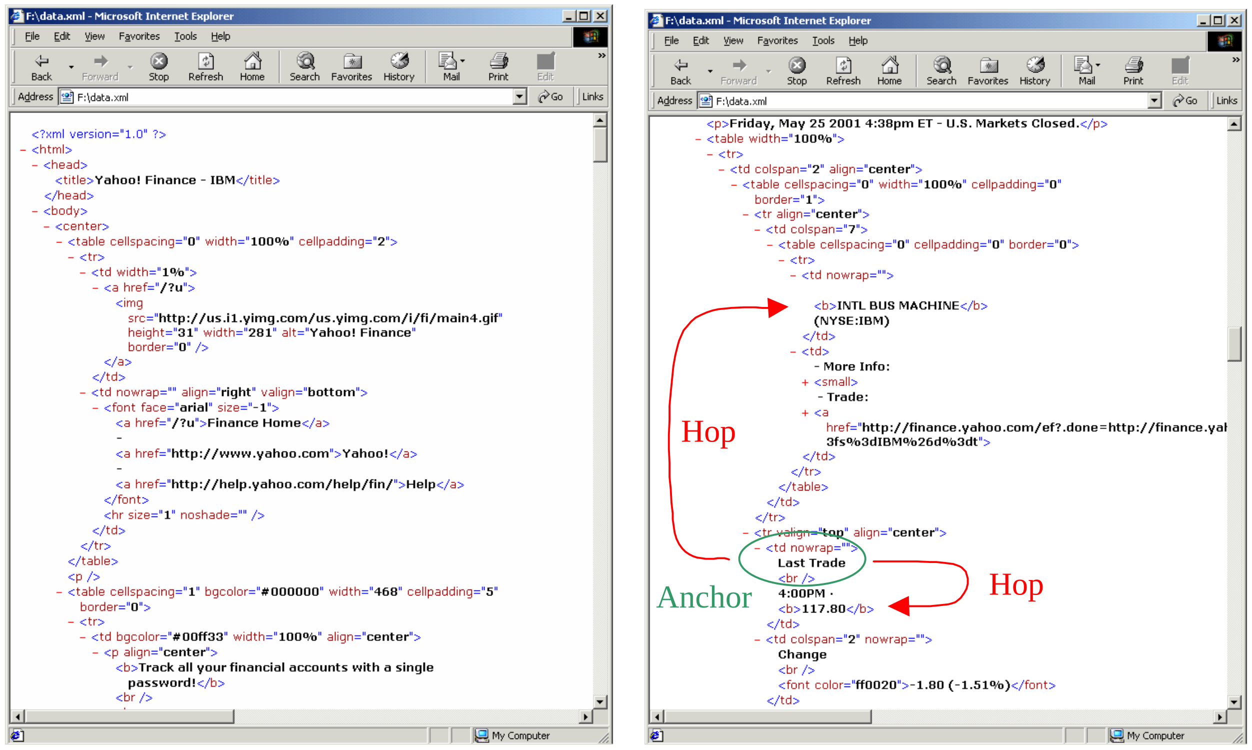
Customize Finance (Yahoo! ID required) - Yahoo! Finance Home

Figure 3. Screen shot of Yahoo Finance page for IBM stock.

constant even when the page design changes. Consider the occurrence of the phrase “Last Trade” on a Web page containing stock quote information. From the location where this phrase occurs in the HTML tree, it is an easier task to find the actual stock trade price.

Sometimes such static text is nowhere to be found and therefore a better cue can be derived from attribute values on the page. It is not uncommon to see important data items highlighted in a certain font size or other text attribute. Sometimes content is displayed using precise design parameters contained in a @class attribute. Consider a news article page where the title is contained in an HTML tag with the @class value “title.” Data extraction is simplified and made more robust by the fact that the name of the HTML element containing the title is unimportant.

We now introduce the concept of *forward-looking robustness* of data extraction patterns. By this methodology, expressions are carefully crafted in anticipation of the likely changes that the HTML page might go through in the months and years to come. In essence, it involves understanding the purpose of the page and its implications for the design and layout. We ask the question “What does the designer want to say on this page?” and consequently, “What HTML design paradigms do they have at their disposal to accomplish the task?” Understanding the thinking behind the page design and the tools used to create it will make data extraction expressions immensely more robust (*a priori*



```
   <?xml version="1.0" ?>
 - <html>
   - <head>
       <title>Yahoo! Finance - IBM</title>
     </head>
   - <body>
     - <center>
       - <table cellspacing="0" width="100%" cellpadding="2">
         - <tr>
           - <td width="1%">
             - <a href="/?u">
                 <img
                   src="http://us.i1.yimg.com/us.yimg.com/i/fi/main4.gif"
                   height="31" width="281" alt="Yahoo! Finance"
                   border="0" />
               </a>
             </td>
           - <td nowrap="" align="right" valign="bottom">
             - <font face="arial" size="-1">
                 <a href="/?u">Finance Home</a>
                 -
                 <a href="http://www.yahoo.com">Yahoo!</a>
                 -
                 <a href="http://help.yahoo.com/help/fin/">Help</a>
               </font>
               <hr size="1" noshade="" />
             </td>
           </tr>
         </table>
         <p />
       - <table cellspacing="1" bgcolor="#000000" width="468" cellpadding="5"
           border="0">
         - <tr>
           - <td bgcolor="#00ff33" width="100%" align="center">
             - <p align="center">
                 <b>Track all your financial accounts with a single
                   password!</b>
                 <br />
```

```
        <p>Friday, May 25 2001 4:38pm ET - U.S. Markets Closed.</p>
      - <table width="100%">
        - <tr>
          - <td colspan="2" align="center">
            - <table cellspacing="0" width="100%" cellpadding="0"
                border="1">
              - <tr align="center">
                - <td colspan="7">
                  - <table cellspacing="0" cellpadding="0" border="0">
                    - <tr>
                      - <td nowrap="">

                          <b>INTL BUS MACHINE</b>
                          (NYSE:IBM)
                        </td>
                      - <td>
                          - More Info:
                        + <small>
                           - Trade:
                        + <a
                            href="http://finance.yahoo.com/ef?.done=http://finance.yal
                            3fs%3dIBM%26d%3dt">
                        </td>
                      </tr>
                    </table>
                  </td>
                </tr>
              - <tr valign="top" align="center">
                - <td nowrap="">
                    Last Trade
                    <br />
                    4:00PM ·
                    <b>117.80</b>
                  </td>
                - <td colspan="2" nowrap="">
                    Change
                    <br />
                    <font color="ff0020">-1.80 (-1.51%)</font>
                  </td>
```

Figure 4. XHTML tree showing top of document and "interesting" content.

sense). For example, as mentioned earlier, pull-down menus can be defined in only one way, a fact that isn't likely to change in the future. We note that automated generation of data extraction patterns based solely on training samples and machine learning will not yield this kind of insight.

4.2 Anchors and Hops

We formalize the process of data extraction by creating a traversal graph on the source document composed of two parts: anchors and hops. Anchors are select nodes of the source document tree and represent the non-leaf nodes of the traversal graph. A hop is a relative "jump" from one anchor to another, or eventually to the sought data. Each hop is represented as directed edge on the traversal graph. As suggested in Section 3.1, we emphasize that "interesting" content is typically found nested deep down the HTML tree and therefore the primary task of an anchor is to find that location so that relative hops can be made more safely. The construction of the traversal graph is shown in Figure 2.

Let us use Yahoo Finance stock price pages, a popular source of financial information for many people, as an example for anchors and hops. Specifically, consider the IBM stock price page at http://finance.yahoo.com/q?s=IBM&d=t (shown in Figure 3) for the following discussion. Bypassing the various navigational and advertising materials on the page, we notice that the "interesting" content is contained in a table near the middle of the page. Inspecting the XHTML code for the page (shown in Figure 4), we observe two things. First, given the XML nature of XHTML code, it is very easy to browse the code in an XML-enabled tool such as Internet Explorer 5, which allows us to expand and

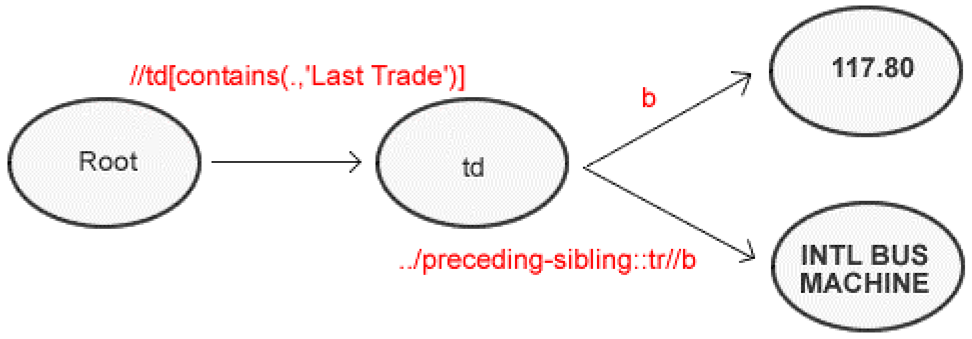


Figure 5. The traversal graph of the Yahoo! Finance page.

shrink parts of the tree as needed (notice the plus and minus signs next to each XHTML element containing child elements).

Secondly, we notice that the last trade price (117.80) is indeed located near the phrase "Last Trade" and further inspection of the code shows that other data values reside in the same table as the last trade price. We are inclined to define a hop expression of type "global search" in the tree that looks for the occurrence of the phrase "Last Trade." From the anchor defined by that hop, we can hop again to the actual stock price value by using a "relative structure" expression that takes us to the bold (<b>) element right underneath.

Now assume that we would also like to extract the name of the company shown on this page. The previously defined anchor serves as a good starting point for this hop also. Since the company name is located in a table row (<tr>) element immediately preceding the row of the anchor, we're inclined to define the hop as follows: find the preceding table row, then take the first bold (<b>) value.

The traversal graph for this extraction is shown in Figure 5. It is important to note that while the nodes in the traversal graph must be taken directly from nodes in the HTML source tree, they are not restricted by the hierarchical order of the source. Notice that from the first anchor defined in this example one hop traverses further down the source tree, while the other first ascends the tree then descends down a different branch. The elimination of the dependency of the hop on the HTML document hierarchy of its source anchor gives increased scope to the range of the hop and substantially differentiates this system of extraction from its peers.

Note that both the last trade price and the company name were highlighted using a bold font. This is not an anomaly or coincidence. Rather, in practice it is quite a frequent sight, and it makes robust data extraction more probable. In essence, what is intended to catch the eye of the human user by highlighting (different or bigger font) also makes it easier for a data extractor to perform its task.

The next step is to embed the anchor and hops into an XSLT file that can extract the data from the XHTML page. Assume that the desired XML output is some stock price markup language where stock price is contained in a PRICE element and name in NAME. The XSLT code shown in Figure 6 does the job and can be combined with other XSLT code for a more elaborate data extraction.

```
<xsl:template match="td[contains(.,'Last Trade')]">

  <PRICE>
    <xsl:value-of select="b"/>
  </PRICE>

  <NAME>
    <xsl:value-of select="../preceding-sibling::tr//b"/>
  </NAME>

</xsl:template>
```

Figure 6. Sample XSLT extraction rule for Yahoo! Finance.

5 Anatomy of Anchors and Hops

We continue our discussion of anchors and hops by focusing on their expressiveness and introducing a classification that helps compare the relative power and robustness of two expressions. In Section 6 we use the classification and show results from an empirical study on actual Web sites and data extractors.

As discussed earlier, a hop is used to find an anchor in an XHTML tree by traversing it and looking for an element that matches a given predicate. The traversal can start at the root of the tree, in which case we call the hop expression *global*. Otherwise, the hop expression starts from another anchor and is labeled *relative*. These two types of expressions define its *locality*.

The reach of the expression defines how deep into the subtree the expression traverses. An expression may be a recursive *search* (i.e. find an element nested arbitrarily deep in the tree) or a traversal along a specified *path*. The search and path alternatives, and their combinations at various levels of the tree, define the *elasticity* of the expression.

Locality and elasticity together define the *scope* of the expression.

Finally, at each level of the tree, the decision to pick one subtree or leaf node over another involves evaluating a predicate on the *structure*, *attribute value*, or *content* of that subtree or leaf node. A structure predicate compares element names or positions, while an attribute value predicate uses the value of an attribute to make the decision. A content-based predicate, the most powerful of the three, compares the content of text nodes. The three alternatives collectively define the *pattern type* of the hop expression. If the hop expression combines different patterns in a single expression, we tend to label the expression according to the most powerful pattern in the expression.

As an example, consider the three hop expressions shown in Table 1. Expression 1 is the one shown earlier in Section 4.2. We label it a “global content search” because it involves recursively searching the entire tree for an occurrence of an instance of some literal text. Expression 2, on the other hand, looks recursively for a font element whose @class attribute has the value “title” and accordingly it is classified as a “global attribute search” expression.

Table 1. Sample hop expressions and their classification.

	Hop Expression	Classification
Expression 1	//td[contains(.,'Last Trade')]	Global Content Search
Expression 2	//font[@class='title]	Global Attribute Search
Expression 3	//tr[count(td) > 5]	Global Structure Search
Expression 4	EXP3/following::b[starts-with(.,'DATE:')]	Relative Content

Table 2. Scope distribution of hop expressions.

	Global Search	Relative Search	Global Path	Relative Path	Existing Anchor
Count	83	20	0	8	117
Percentage	36%	9%	0%	4%	51%

Expression 3 finds an anchor that is a table row anywhere in the XHTML tree that has more than 5 columns. Its class is “global structure search.” Expression 4 starts from the anchor defined by Expression 3. It looks for a bold (<b>) element that starts with the text content “DATE:” and is a following sibling or child element of the table row. We classify it as a “relative content” expression because it starts from an existing anchor and depends more heavily on the occurrence of the literal string “DATE:” than the presence of a bold element.

6 Empirical Results

In this section we briefly review results from an empirical study on actual Web data extraction patterns used over the past 2 years. XSLT files were defined manually for approximately 40 Web data sources and the hop expression types classified by hand. There were 231 data extraction patterns (XPath expressions) in total. Of these, 228 defined intermediate anchors, while the remaining 3 patterns consisted of direct references to well-defined elements like /html/title (the title of the XHTML document).

In Table 2 we show the scope distribution of the 228 hop expressions encountered. There were 111 unique anchors defined by these hop expressions. We observe that the “global search” variety was the most commonly used scope (83 instances) and that 117 hop expressions reused an existing anchor. This is explained by the fact that in many XHTML documents, it is sufficient to find one good anchor, from which short hops to several interesting data items can be made. Recall that in our example in Section 4.2, we used the same anchor to find both the last trade price and company name of a stock.

Next, we inspect the distribution of hop expression pattern types used. Table 3 shows that roughly half the expressions were based either on content or attribute values, while

Table 3. Pattern type distribution of hop expressions.

	Structure	Attribute	Content	None
Count	10	46	55	117
Percentage	4%	20%	24%	51%

pure structural expressions accounted for less than 5% of the total. The category "None" corresponds to the "Existing Anchor" category in Table 2 -- these are anchors that were simply reused.

Combining the results from hop expression scopes and pattern types reveals that by far the most common expression is a "global search" for either attribute value or content. These types of expressions are very powerful and we feel confident that they are representative of typical data extraction tasks in the Web today.

7 Conclusion and Future Work

In this paper we have discussed the problem of data extraction from Web sites and explored an XML-based approach for solving it. Although the HTML syntax itself has stabilized over the last few years, the way Web sites are designed continues to evolve. Powerful design and publishing tools make the creation of complex Web sites easier than before, introducing a level of difficulty in data extraction that extends beyond simple "screen scraping." We discuss the evolution of Web site design, and suggest an approach for building a data extraction process with commensurate power and extensibility. In particular, normalizing HTML content into XHTML simplifies data extraction and makes it possible to use XML technologies, notably XML path expressions (XPath), in the process.

Data extraction based on XPath expressions and XSLT templates as a compact format for defining them is gaining popularity. We propose a method for constructing such XML path expressions using *anchors* and *hops*. Given that the general tendency in Web page design is to visually and syntactically segment the page into its constituent parts (headers and footers, advertising, navigational aids, and main content), we propose a process where an anchor residing at the approximate location of the desired data (table, list, frame, etc.) is first determined. From the anchor, it is then relatively simple to make a short hop to the precise location of the data.

We define a classification of hop expression types that spans global (entire XHTML document tree) and relative (subtree) traversals, and covers matching patterns based on content (literal text), attribute values, and structure. Results from our empirical study on existing data extraction processes covering approximately 40 different Web data sources suggest that most hop expression types fall into the category "global search for content," which is also the most powerful. It involves looking for an occurrence of a literal text such as "Last Trade" which is also known to be unique in the document and remain in the document even when the Web page layout changes (*forward-looking robustness*).

Another important finding is that many individual data items can be extracted using the same anchor. This is due to the relative proximity of “interesting” data to each other and means that when a Web page layout changes and a previously defined anchor fails, it is sufficient to redefine one or a few new anchors instead of a new anchor for each data item to be extracted.

Our future work includes further analysis of data extraction patterns using the anchors-hops method, and translating these ideas to automated pattern generators. While complete automation of pattern generation appears impossible, we believe that the process can be made significantly easier and faster with appropriate software tools.

8 Bibliography

[ALL97] Charles Allen. WIDL: Application Integration with XML. World Wide Web Journal 2(4), November 1997.

[BAR98] Maria Luisa Barja, Tore Bratvold, Jussi Myllymaki, and Gabriele Sonnenberger. Informia: a Mediator for Integrated Access to Heterogeneous Information Sources. Proc. ACM Conference on Information and Knowledge Management (CIKM), Washington, DC, November 1998.

[DB2XML] DB2 XML Extender. http://www.ibm.com/software/data/db2/extenders/xmlext/index.html.

[FLO98] Daniela Florescu, Alon Levy, and Alberto Mendelzon. Database Techniques for the World-Wide Web: A Survey. ACM SIGMOD Record, vol. 27, no. 3, 1998.

[FRE98] Dayne Freitag. Information Extraction from HTML: Application of a General Machine Learning Approach. Proc. Conference on Artificial Intelligence (AAAI), pp. 517-523, September 1998.

[GUP97] Ashish Gupta, Venky Harinarayan, and Anand Rajaraman. Virtual Database Technology. ACM SIGMOD Record, vol. 26, no. 4, December 1997.

[HAM97] Joachim Hammer, Hector Garca-Molina, Junghoo Cho, Arturo Crespo, and Rohan Aranha. Extracting Semistructured Information from the Web. Proc. Workshop on Management of Semistructured Data, Tucson, Arizona, 1997.

[KNO00] Craig A. Knoblock, Kristina Lerman, Steven Minton, and Ion Muslea. Accurately and Reliably Extracting Data from the Web: A Machine Learning Approach. IEEE Data Engineering Bulletin, vol. 23, no. 4, pp. 33-41, 2000.

[KON95] David Konopnicki and Oded Shmueli. W3QS: A Query System for the World Wide Web. Proc. International Conference on Very Large Data Bases (VLDB), pp. 54-65, Zurich, Switzerland, September 1995.

[KUS99] Nicholas Kushmerick. Gleaning the Web. IEEE Intelligent Systems, vol. 14, no. 2, pp. 20-22, March/April 1999.

[KUS00] Nicholas Kushmerick. Wrapper Induction: Efficiency and Expressiveness. J. Artificial Intelligence, vol. 118, no. 12, pp. 15-68, 2000.

[LAK96] Laks V. S. Lakshmanan, Fereidoon Sadri, and Iyer N. Subramanian. A Declarative Language for Querying and Restructuring the Web. Proc. 6th International Workshop on Research Issues in Data Engineering (RIDE), February 1996.

[LER01] Kristina Lerman, Craig A. Knoblock and Steven Minton. Automatic Data Extraction from Lists and Tables in Web Sources. Workshop on Automatic Text Extraction and Mining (ATEM), Seventeenth International Joint Conference on Artificial Intelligence (IJCAI), Seattle, WA, August 2001.

[LIU00] Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. Proc. International Conference on Data Engineering (ICDE), San Diego, California, February 2000.

[MEN97] Alberto Mendelzon, George Mihaila, and Tova Milo. Querying the World Wide Web. Int. J. on Digital Libraries, vol. 1, no. 1, pp. 54-67, 1997.

[MYL01] Jussi Myllymaki. Effective Web Data Extraction with Standard XML Technologies. Proc. Tenth International World Wide Web Conference, Hong Kong, May 2001.

[RAJ01] Anand Rajaraman, Jeffrey D. Ullman: Querying Websites Using Compact Skeletons. Proc. ACM Symposium on Principles of Database Systems (PODS), pp. 16-27, May 2001.

[RIB99] Berthier Ribeiro-Neto, Alberto H.F. Laender, and Altigran S. da Silva. Extracting Semi-Structured Data Through Examples. Proc. ACM Conference on Information and Knowledge Management (CIKM), Kansas City, Missouri, November 1999.

[SAH99] Arnaud Sahuguet and Fabien Azavant. Building Light-Weight Wrappers for Legacy Web Data-Sources Using W4F. Proc. International Conference on Very Large Data Bases (VLDB), Edinburgh, Scotland, September 1999.

[SAT99] Kai-Uwe Sattler and Michael Höding. Adapter Generation for Extracting and Querying Data from Web Sources. Proc. ACM SIGMOD Workshop on the Web and Databases (WebDB), pp. 49-54, Philadelphia, Pennsylvania, June 1999.

[SOD98] Stephen Soderland. Learning Information Extraction Rules for Semi-structured and Free Text. Machine Learning, vol. 34, no. 1, pp. 233-272, 1999.

[TIDY] HTML Tidy. http://www.w3.org/People/Raggett/tidy/.

[WEBL] Compaq's Web Language, Compaq Computer,
http://www.research.digital.com/SRC/WebL/index.html.

[WIDL] Web Interface Definition Language, W3C Note. September 1997.
http://www.w3.org/TR/NOTE-widl.

[XHTML] XHTML: The Extensible HyperText Markup Language, W3C
Recommendation, January 2000. http://www.w3.org/TR/xhtml1.

[XML] Extensible Markup Language (XML), W3C Recommendation, February 1998.
http://www.w3.org/TR/REC-xml.

[XMLSCHEMA] XML Schema Part 0: Primer, W3C Working Draft, April 2000.
http://www.w3.org/TR/xmlschema-0/.

[XPATH] XML Path Language (XPath), W3C Recommendation, November 1999.
http://www.w3.org/TR/xpath.html.

[XSLT] XSL Transformations (XSLT), W3C Recommendation, November 1999.
http://www.w3.org/TR/xslt.html.