SEMI-SUPERVISED INFORMATION EXTRACTION FROM VARIABLE-LENGTH WEB-PAGE LISTS

Daniel Nikovski, Alan Esenther
Mitsubishi Electric Research Laboratories, 201 Broadway, Cambridge, U.S.A.

Akihiro Baba
Mitsubishi Electric Corporation, 5-1-1 Ofuna, Kamakura, Kanagawa 247-8501, Japan

Keywords: Service oriented architectures, System integration, Information extraction, Web mining.

Abstract: We propose two methods for constructing automated programs for extraction of information from a class of web pages that are very common and of high practical significance — variable-length lists of records with identical structure. Whereas most existing methods would require multiple example instances of the target web page in order to be able to construct extraction rules, our algorithms require only a single example instance. The first method analyzes the document object model (DOM) tree of the web page to identify repeatable structure that includes all of the specified data fields of interest. The second method provides an interactive way of discovering the list node of the DOM tree by visualizing the correspondence between portions of XPath expressions and visual elements in the web page. Both methods construct extraction rules in the form of XPath expressions, facilitating ease of deployment and integration with other information systems.

1 INTRODUCTION

Modern information systems integrate, aggregate, and manage information from a large number of sources, such as databases, user input, sensors, remote web services provided by other information systems, etc. Successful integration of all these data sources depends critically on the existence of data exchange protocols and application programmable interfaces (APIs) that have been agreed upon by both the providers and users of such information.

Service-oriented architectures (SOA) hold much promise for facilitating the building and re-configuration of flexible information systems that integrate many data sources. In order for a data source to be available for integration in an SOA application, it is typically converted to a web service: a software system designed to support interoperable machine-to-machine operation over a network. Interoperability is achieved by means of adherence to web service standards for describing and accessing the functionality of the service (e.g., WSDL), and data formats and protocols for transporting the data between clients and servers (e.g., the SOAP protocol).

However, most of the content of what is arguably the largest existing source of information — the World Wide Web (WWW) — is not accessible by means of web services. The reason for this is the intended audience of information published on the WWW: human readers, and not computers. The vast majority of existing web pages are intended to be read by people, and their encoding (HTML) is optimized for efficient rendering into graphical form by web browsers. As a result, neither the HTML encoding, nor the final graphical rendering are easy to interpret by machines.

There have been several principled attempts to make such information accessible by machines. One solution involves representing the information as an XML file which is either communicated directly to a remote machine using web-services protocols such as SOAP, or transformed to an HTML file by means of suitable presentation templates, such as XSLT files, and further rendered into graphical form to be read by humans. Although this is a convenient solution, a very small percentage of existing web applications provide their contents in both XML and HTML formats.

One major reason for this is technical — content in XML format is usually provided by means of a ded-

Nikovski D., Esenther A. and Baba A. (2009).
SEMI-SUPERVISED INFORMATION EXTRACTION FROM VARIABLE-LENGTHWEB-PAGE LISTS.
In *Proceedings of the 11th International Conference on Enterprise Information Systems - Databases and Information Systems Integration*, pages 261-266
DOI: 10.5220/0001858402610266

icated web service which takes time, effort, and resources to develop. Furthermore, apart from the cost issue, this can be an unsurmountable obstacle in cases when legacy web applications have to be integrated into a novel system, and modifications to the existing application cannot be made. This is, in fact, a very common situation in system integration projects, when the developers of the original application have often left the organization since the time the application was deployed.

Consequently, for the purposes of system integration of legacy IT systems, it is very desirable to find ways to extract information that is intended to be used by humans and encoded in HTML format, and present it to remote machines in a computer-interpretable format such as XML. This is a problem that has been under investigation in the field of information extraction. We review the existing approaches and their limitations in the next section, and propose two novel algorithms for semi-supervised information extraction from web pages with lists of variable length in Section 3. In Section 4, we describe one possible implementation of the proposed methods, and its role in a system integration solution and method. Section 5 proposes directions for expanding these solutions to a wider class of web pages, and concludes the paper.

2 WEB INFORMATION EXTRACTION

The field of information extraction is an area of information technology that is concerned with extracting useful information from natural language text that is intended to be read and interpreted by humans. Such text can be produced either by other humans (e.g., a classified ad), or generated by machines, possibly using the content of a structured database (e.g., a product description page on an e-commerce web site) (Laender et al., 2002). Although humans do not necessarily make a significant distinction between these two cases, as far as their ability to inteprete the text is concerned, the difference between them has enormous importance as regards the success of interpretation of such text by machines. Understanding free-form natural language that has been generated by humans is a very complicated problem whose complete solution is not in the foreseeable future. In contrast, if text has been generated by a machine, using a boilerplate template for page layout and presentation (such as an XSLT file), and a database for actual content, the rate of success of automated information extraction methods can be very high. This type of text is often called semi-structured data, and due to its high practical significance, is the focus of this paper.

The usual method for extracting information from web pages that are output by legacy applications is by means of programs called wrappers (van den Heuvel and Thiran, 2003). The simplest approach is to write such wrappers manually, for example using a general-purpose programming language such as Java or Perl. Since this can be difficult, tedious, and error-prone, various methods have been proposed for automating the development of wrappers. Although most of these methods focus on creating extraction rules that are applied to web pages by an extraction tool, they differ significantly according to how they apply the induced rules on web pages, and according to how they actually induce these rules.

Regarding the first difference, some methods apply the rules directly to the stream of tokens in the web page. In such cases, the rules can be encoded as regular expressions, context-free grammars, or using more advanced specialized languages. One advantage of these methods is that they can easily filter out irrelevant text, e.g. interstitial ads in web pages. However, finding the rules that would extract all the needed information and only the needed information is not a trivial problem.

Other methods use the fact that a web page encoded in HTML is not just a stream of tokens, but has a tree-like structure. This structure is in fact recognized by web browsers when they transform the HTML code into a Document Object Model (DOM) tree prior to rendering it on screen. When a needed data item can be found in one of the leaves of the DOM tree, an extraction rule for its retrieval can be encoded by means of a standard XPath expression that specifies the path that has to be traversed in the DOM tree to reach the respective leaf. Applying such rules to new web pages in a deployed system is very straightforward: an embedded web browser is used to retrieve the web page and create its DOM tree, after which the XPath expression is applied to retrieve the data. Fully automated tools such as W4F (Sahuguet and Azavant, 1999) and XWRAP (Liu et al., 2000) operate on the DOM tree. (Although, they use different languages for representing the extraction rules.)

Regarding the second difference among wrapper construction tools — how extraction rules are induced — there are several principal approaches. Supervised methods require explicit instruction on where the data fields are in a web page, in the form of examples. One practical way for a human user to do this is to point to the data items on a rendered web page, for example by highlighting them by means of a computer mouse, and after that the corresponding extraction rules are automatically generated. As noted, when the extrac-

tion rules are encoded using XPath expressions, they can be readily generated by means of the DOM tree of the web page.

Supervised method work very well when the web page has a fixed structure, and only the needed data items vary between different instances of this web page. However, when the structure of the DOM tree itself changes over time, a single example of where the needed data items are is not sufficient. Exhaustive enumeration of all possible cases is not possible, either — apart from being very time consuming, it would result in multiple extraction rules, each with limited validity.

In contrast, unsupervised methods do not require explicit labeling of data items within example web pages. Instead, they require only input of several instances of a web page, after which the methods can infer from the instances what part of the web page is data, and what part is formatting and labeling text. These methods usually compare multiple instances of a web page, and treat the variable part as data fields and the non-changing part as a formatting template. One shortcoming of those methods is that they require multiple example instances, and it is sometimes not clear how many examples would be sufficient for correct rule generation.

3 INFORMATION EXTRACTION FROM VARIABLE-LENGTH LISTS

The practical case we are considering is the extraction of data fields from web pages that contain lists of records formatted identically, but these lists can be of variable length. One example of such web pages are search results for products, articles, addresses, films, etc. — practically all search engines return such lists. Another example are tables that contain variable number of rows, such as train schedules, class rosters, sports rankings, etc. In this case, the table is the list, each row in the table is a record, and each cell in the table is a data field. What is common between these cases is that the structure of the web pages varies due to the variable number of records, but does so in a fairly regular manner: any new records have the same structure as the previous ones.

Writing extraction rules for this case manually can be challenging, because the structure of the DOM tree of the web page is variable. A supervised method would require labeling of all occurrences of data fields in a possibly unlimited number of instances of web pages that contain lists of progressively increasing length. Unsupervised methods would not need labeling, but would still require a dataset of web page examples of unknown size (possibly very large), and are also likely to return as data such fields that do change across page instances, but are not of interest.

To address these deficiencies, we propose two methods for information extraction from such pages that require only one instance of a data record to be labeled explicitly on only one instance of a web page. Both methods use XPath to encode the extraction rules, but neither of them requires manual coding of these extraction rules. The first method is computationally more difficult, but does not require any familiarity with XPath, whereas the second method is computationally simpler, but is suitable for users who are familiar with XPath expressions.

In both cases, a human user uses a web browser embedded in a wrapper construction application (WCA) to navigate to a target web page that contains a list of at least two records, and pinpoints several data fields on the web page that belong to the same record. Let the number of records on the web page be denoted by m, such that $m \geq 2$. If there are n such data fields, we will denote them by F_j, $j = 1,n$. We assume that each data field F_j contains the value of an output variable of interest V_j, such that the value of this variable $v_{i_0,j}$ for some record i_0, that is, $v_{i_0,j} = F_j$, $j = 1,n$. The goal is to be able to extract all values $v_{i,j}$, $i = 1,m$, and $j = 1,n$, for all web pages where m varies arbitrarily, by using only the pinpointed location of data field examples F_j, $j = 1,n$. Two methods for the generation of extraction rules that can accomplish this are described below.

3.1 Automated Induction of Rules

The first method starts by determining the absolute XPath expressions for all data fields. An absolute XPath expression consists of a sequence of tree nodes separated by slash signs, where nodes to the left of a slash sign represent parent nodes, and nodes to the right of a slash sign represent children nodes in the DOM tree. For example, the second cell in the first record of the first table of an HTML document could be identified by the absolute XPath expression "/html[1]/body[1]/table[1]/tbody[1]/tr[1]/td[2]" (see Fig. 1). Let the XPath expression of data field example F_j be denoted by E_j.

In the second step of the algorithm, the least common ancestor (LCA) node of all data field examples F_j, $j = 1,n$ is found. This can be achieved by traversing all path expressions E_j simultaneously from left to right, and stopping when a mismatch between the tags of any two paths is encountered.

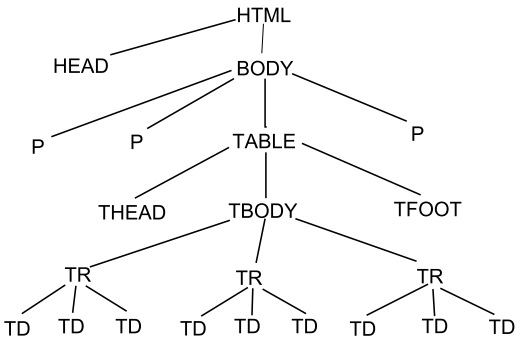

Figure 1: A typical DOM tree of an HTML document that contains a list (table) of variable number of records (rows) of the same structure (here, three data fields per record). Information to be extracted is contained in the leaves of the tree. Each node in the tree, leaf or not, can be identified by a unique XPath expression.

The longest portion of all paths E_j that matches is the path to the LCA node of all data fields in the DOM tree of the HTML document. (It may or may not correspond to the entire record i_0 that contains all example data fields.) For instance, if two cells in the first row of a table were designated as examples, e.g. "/html[1]/body[1]/table[1]/tbody[1]/tr[1]/td[2]" and "/html[1]/body[1]/table[1]/tbody[1]/tr[1]/td[4]", the path to their LCA node will be "/html[1]/body[1]/table[1]/tbody[1]/tr[1]", that is, the LCA node will be the one that corresponds to the first row of the table. Let this LCA node be denoted by L.

In the third step of the algorithm, the sub-tree whose root is node L is identified and stored in variable S. By definition, it does contain all nodes that correspond to example data fields, but, as noted, it does not necessarily correspond to an entire record.

In the fourth step of the algorithm, in order to find the DOM-tree node that corresponds to the entire list of records, all ancestors A of node L are traversed in a bottom up manner (starting from L and moving up to the root of the DOM tree), while at the same time matching the tree stored in variable S to all subtrees whose root nodes are siblings or cousins of L, that is, they are at the same level as L, and have A as their ancestor. The lowest ancestor A where a match is identified is the node that corresponds to the entire list of records.

In our example, if the path to L is "/html[1]/body[1]/table[1]/tbody[1]/tr[1]", then all ancestor nodes to L will be examined, starting with A="/html[1]/body[1]/table[1]/tbody[1]". In this case, the tree S will have a root and N children denoted by tags "td[1]", "td[2]", "td[3]", ..., "td[N]", where N is the number of columns in the data table, $N \geq n$. Now, since there is at least one other record (row) with the same structure (because of the requirement that $m \geq 2$), there will be a cousin of the node L ("/html[1]/body[1]/table[1]/tbody[1]/tr[1]") which is the root of a tree with the same structure as S. In this case, this will be another row in the table, for example L_2, "/html[1]/body[1]/table[1]/tbody[1]/tr[2]". At this point, a match will be established, and A="/html[1]/body[1]/table[1]/tbody[1]" will be declared the list node.

Note that this method of discovering the list node is robust to multiple nesting of tags. In this case, there exists a "tbody" tag between the "table" and "tr" tags in a table, so the list node will be the one corresponding to the "tbody" tag, and not the one corresponding to the "table" node. In other words, the list node is the deepest node in the DOM-tree (respectively, the rightmost tag in XPath) whose sub-trees correspond to individual records.

Once this node has been identified, an XPath for individual data field F_j can be constructed by simply omitting the qualifying index in the tag that follows A in the XPath expression E_j. For example, if E_1 ="/html[1]/body[1]/table[1]/tbody[1]/tr[1]/td[2]", then a suitable extraction path for field F_1 would be "/html[1]/body[1]/table[1]/tbody[1]/tr/td[2]". Its meaning is, as expected, to return the second cell in *any* row of the first table of the document.

3.2 Interactive Construction of Rules

The main computational effort in the automated construction algorithm described above is in finding the list node A whose immediate children correspond to the individual records in the list. For added accuracy and ease of implementation, this step can be completed manually by a system integration engineer in an interactive tool, if this engineer is familiar with XPath.

In practice, the engineer could select the node in the XPath expression that corresponds to the entire record in the web page that contains the data fields F_j, $j = 1,n$. To this end, the XPath expression E_j for one data field F_j, $j = j_1$ is displayed to the user simultaneously with the target web page. Then, when the user clicks on a node in the XPath expression, the corresponding visual element of the web page is highlighted in the web browser. When multiple nodes correspond to the same visual element, for example when there is nesting of multiple HMTL tags, the user should select the highest element in the DOM tree, i.e., the node that is farthest left in the XPath expression. The selected tree node that corresponds to the

entire first record will then be an immediate child of the list node *A*, per our definition. The construction of the extraction rules can then proceed as in the completely automated method.

4 IMPLEMENTATION OF THE INFORMATION EXTRACTION METHODS

The proposed idea for information extraction from web pages has been implemented as an extension of the Firefox browser. This extension operates by allowing the user to navigate to any web page using the normal web navigation facilities of the browser, and designate that page as a target for information extraction. By highlighting individual text elements on the web page and using a context menu, a set of data fields F_j, $j = 1, n$ are designated, and their XPath expressions E_j are obtained from the DOM tree of the target web page. All of these XPath expressions are displayed in the Firefox extension, next to the target web page.

By placing the mouse over individual tags in any XPath expression, the corresponding visual element (cell, row, table, etc.) on the target page is highlighted using a different background color (Fig. 2). In general, the user proceeds over tags right to left, stopping when the entire list (as opposed to a single record only) is highlighted. Clicking on the corresponding tag then identifies the list node A, and extraction rules can be constructed.

The implemented tool provides a high-quality method for extracting information from structured web pages. This is due to the use of XPath for specification of the information to be extracted; intuitively, when the data records have been generated from a specific data schema, the XPath expression generated by the tool uniquely identifies a field in that schema. In contrast, character matching methods do not provide such unique identification, and might result in a large number of false matches.

5 RELATED WORK

The proposed technique for automated construction of rules for information extraction from web pages is based on two main ideas: the use of XPath for rule specification and execution, and the analysis of the DOM tree of the HTML document. Individually, these two ideas have been proposed previously. Schwartz discussed the use of XPath for information extraction from web pages, and pointed out that the main challenge is to build the correct XPath expression that would remain robust to variations in page structure (Schwartz, 2007). Liu proposed a variety of algorithms for learning extraction rules from example HTML documents that operate by analyzing the DOM-tree of these examples (Liu, 2007). Some of these algorithms even allowed for imperfect matching between the sub-trees of the individual records. However, he did not use XPath as the language for encoding of the rules, and furthermore, these algorithms required multiple examples of web pages, whereas both algorithm proposed in this paper require only a single example.

6 CONCLUSIONS

We have described two methods that allow easy construction of wrappers for extracting information from one of the most frequent and useful sources of information — web pages with variable number of records of identical structure. The output of most search engines falls into this category, as well as most web-based interfaces for querying databases.

Even though a number of methods for wrapper generation have already been proposed (some of which fairly advanced), most of them do not provide a very convenient solution to this problem of high practical importance. Both supervised and unsupervised methods require a large number of examples to be able to learn the correct extraction rules. In contrast, the two methods proposed in this paper would require only one example web page, and need only data fields in a single record to be identified. As a result, the construction of wrappers is not substantially more difficult than browsing to a web page and highlighting the data fields of interest.

Once the data fields have been specified, the first algorithm proceeds by identifying the node in the DOM tree that comprises all data fields of interest. Under the assumption that all other records must include an identically structured sub-tree, the algorithm looks for such matching sub-trees elsewhere in the DOM tree. The node that is the lowest common ancestor of all such sub-trees is identified as the list node of the entire web page.

In contrast, the second proposed method relies on interactive identification of the list node of the DOM tree, with the help of a system integration engineer. The method is intuitive, and requires only basic familiarity with XPath. List node identification is performed by scanning the XPath expression for any single data field in a bottom-up manner, and providing

Figure 2: An interactive tool for construction of extraction rules for information extraction from web pages.

visual feedback to the engineer as to what visual element in the web page corresponds to the currently scanned tag. As soon as a tag is found that comprises the entire list of records, the corresponding DOM-tree node of this tag is identified as the list node of the web page. Both algorithms rely on the expressive capabilities and widespread use of the XPath language for querying XML documents. This also facilitates the deployment of wrappers when integrating them with other information systems.

While the described methods are convenient and handle well web pages of the described type (variable-length lists with records of fixed structure), there exist other variations in the structure of web pages that are of practical interest and could be addressed in the future by possible extensions of these algorithms. One such variation is interstitial ads — advertisements that appear among items (records) of interest. When such ads are placed as direct children of the list node of the DOM-tree, the XPath expression for a data variable can match accidentally some content from the ad. One possibility for eliminating such erroneous matches is to include the entire context of a data variable in its XPath expression, so that valid data records are identified first, and the extraction of individual data fields occurs only after that. We hope that this type of matching could possibly be implemented by modifying the XPath extraction rules in an appropriate manner, to be investigated in the future.

REFERENCES

Laender, A. H. F., Ribeiro-Neto, B. A., da Silva, A. S., and Teixeira, J. S. (2002). A brief survey of Web data extraction tools. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 31(2):84–93.

Liu, B. (2007). *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Data-Centric Systems and Applications. Springer.

Liu, L., Pu, C., and Han, W. (2000). XWRAP: An XML-enabled wrapper construction system for web information sources. In *Proceedings of the International Conference on Data Engineering*, pages 611–621.

Sahuguet, A. and Azavant, F. (1999). Building light-weight wrappers for legacy web data-sources using W4F. In *25th Conference on Very Large Database Systems*, pages 738–741, Edingurgh, UK.

Schwartz, R. L. (2007). HTML scraping with XPath. *Linux Magazine*, 2007(4).

van den Heuvel, W.-J. and Thiran, P. (2003). A methodology for designing federated enterprise models with conceptualized legacy wrappers. In *Proceedings of the Fifth International Conference on Enterprise Information Systems ICEIS'03*, pages 353–358.