Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 6.017

IJCSMC, Vol. 6, Issue. 6, June 2017, pg.245 – 253

Efficient Web Data Mining with Standard XML Technologies

P.N. Santosh Kumar[1], Dr. V. Radha[2], Dr. C. Sunil Kumar[3]
[1]Associate Professor, ECM Dept, SNIST, Hyderabad, pnsk47@gmail.com
[2]IDRBT. Hyderabad, vradha@idrbt.ac.in
[3]Professor of IT, SNIST, Hyderabad, ccharupalli@gmail.com

ABSTRACT: The problem of Web data extraction and XML-based methodology whose goal extends far beyond simple "screen scraping are discussed." An ideal data extraction process is able to digest target Web databases that are visible only as HTML pages, and create a local, identical replica of those databases as a result. What is needed in this process is much more than a Web crawler and set of Web site wrappers. A comprehensive data extraction process needs to deal with such roadblocks such as session identifiers, HTML forms, and client-side JavaScript, and data integration problems such as incompatible datasets and vocabularies, and missing and conflicting data. Proper data extraction also requires a solid data validation and error recovery service to handle data extraction failures, which are unavoidable.

In this paper we describe NDES, a software framework that makes significant advances in solving these problems and provides a platform for building a production- quality Web data extraction process. Key aspects of NDES are that it uses XML technologies for data extraction, including XHTML and XSLT, and provides access to the "deep Web."

Keywords: Wrappers, Crawling, Data Extraction, Semi structured Data, Deep Web.

1. INTRODUCTION

Given the rapid growth and success of public information sources on the World Wide Web, it is increasingly attractive to extract data from these sources and make it available for further processing by end users and application programs. Data extracted from Web sites can serve as the springboard for a variety of tasks, including information retrieval (e.g. business intelligence), event monitoring (news and stock market), and electronic commerce (shopping comparison).

Extracting structured data from Web sites is not a trivial task. Most of the information on the Web today is in the form of Hypertext Markup Language (HTML) documents which are viewed by humans with a browser. HTML documents are sometimes written by hand, sometimes with the aid of HTML tools. Given that the format of HTML documents is designed for presentation purposes, not automated extraction, and the fact that most of the HTML content on the Web is ill-formed ("broken"), extracting data from such documents can be compared to the task of extracting structure from unstructured documents.

In the future, some if not most Web content may be available in formats more suitable for automated processing, in particular the Extensible Markup Language (XML) [17]. Despite being a relatively new development, 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 (and profitable) 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. The tools and techniques that we collectively known as Web data extraction are key to making this possible.

In this paper we focus on systems-oriented issues in Web data extraction and describe our approach for building a dependable extraction process. Our ideas are manifested in NDES (Nifty Data Extraction System), a crawler-based Web data extraction framework and the backbone of several Web data extraction systems in production use at IBM.

2. RELATED WORK

Several research groups have focused on the problem of extracting structured data from HTML documents. Much of the research is in the context of a database system, and the focus is on wrappers that translate a database query to a Web request and parse the resulting HTML page. Our focus is on batch-oriented data extraction: crawling target Web sites, extracting structured data, performing domain-specific feature extraction and resolution of missing and conflicting data, and making the data available to local database applications. Monitoring the quality of the extracted data and providing alerts when failures occur is an important issue, as is the capability to synthesize hyperlinks dynamically in order to retrieve data from the "deep Web" [6].
Our ideas have been implemented in NDES, a software framework that merges crawler technology with XML-based data extraction technology to form a dependable, robust process. NDES is similar to other extraction systems in that it defines a wrapper for each Web site. The precise mechanism how wrappers are defined, implemented, and used is different in each system, however. Below we compare the NDES wrapper mechanism with others proposed in the literature.

The Web Wrapper Factory (W4F) is a toolkit for generating Web wrappers [9]. 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 NDES are similar in many respects, their main difference is that whereas W4F uses a proprietary language for data extraction and mapping rules, NDES is based on XHTML [16] and XSLT [20] and can exploit templates, (recursive) path expressions, and regular expressions for more effective data extraction, mapping, and aggregation. Hyperlink synthesis, which allows data to be extracted from the "deep Web," is also unique to NDES.

The goal of WIDL is to define a programmatic interface to Web sites [1][15]. 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 3.2, this falls short of building robust data extractors. Feature extraction and structure synthesis would be difficult to implement in WIDL and would be relegated to some higher-level program.

The Web Language (formerly WebL) from Compaq is a procedural language for writing Web wrappers [14]. 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.

The Ariadne [2][7], Garlic [13], and TSIMMIS [4] systems are mediators that facilitate querying multiple heterogeneous sources. While Garlic and TSIMMIS support a wide range of sources, including Web sources, database systems, and file systems, Ariadne focuses on Web sources exclusively. In each system, a modeling process produces an integrated view of the data contained in the sources and a query planning process decomposes queries on the integrated view into a set of subqueries on the sources.

In Garlic and TSIMMIS, wrappers are written in a procedural programming language and are compiled into executable code, whereas in Ariadne, an induction-based wrapper generation mechanism is used. It uses regular expressions and includes mapping tables to resolve vocabulary differences between Web sources, but lacks path expressions. We note that path expressions are important in extracting data from an HTML tree because hierarchical navigation between nested HTML elements is frequently needed. In NDES, a combination of XPath axes and operators with regular expressions provides for more robust data extraction rules than what is possible with regular expressions alone.

XWRAP [8] 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 [3] 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 provides a toolkit for automatically producing extraction rules for pages that contain repetitive elements. However, the language is proprietary and extractors that are created manually for sites with no repetitive data (e.g. some Yahoo! Finance pages) are difficult to maintain.

3. EXTRACTING STRUCTURED DATA FROM WEB SITES

Extracting structured data from Web sites requires solving five distinct problems: finding target HTML pages on a site by following hyperlinks (navigation problem), extracting relevant pieces of data from these pages (data extraction problem), distilling the data and improving its structuredness (structure synthesis problem), ensuring data homogeneity (data mapping problem), and merging data from separate HTML pages (data integration problem). We discuss each problem in the following sections.

3.1 Web Site Navigation

In the NDES data extraction framework, we view Web sites as consisting of two types of HTML pages: target HTML pages that contain the data we want to extract and navigational HTML pages that contain hyperlinks pointing to target pages or other navigational pages. An automated crawler is used to retrieve target pages from a Web site. The crawler is guided by a rule-based configuration file that tells it where to start, which hyperlinks to follow (and which ones not), and the desired crawling depth. These instructions collectively define the navigation rules of a given Web site.

The crawler starts the navigation by retrieving the seed page (or pages) from the Web site and, based on its URL, determines whether the page is a target page or a navigational page. If it is a target page, it is forwarded to the data extractor for subsequent processing. Hyperlinks from both types of pages are analyzed and a decision to follow them is made on a link-by-link basis. A crawling depth parameter determines how many links away from the seed page the crawler can move.

Besides handling static hyperlinks (<A>, <FRAME>, and <IMG> tags), our approach is designed with the "deep Web" [6] in mind. The deep Web is crawled by analyzing HTML forms and JavaScript code and producing "synthetic hyperlinks" that the crawler can follow. This is described in more detail in Section 3.3.

```
<gcs-config>
<group name="Reviews"> <url-pattern-list>
<url-pattern recursion-depth="2"> <seed-list>
<li>http://www.epinions.com/cmhd_Notebooks-IBM</li> </seed-list>
<include-pattern-list>
<url-obj-pattern host="www.epinions.com" file="/cmhd_Notebooks-IBM*"/> <url-obj-pattern host="www.epinions.com" file="/cmd-review*"/>
</include-pattern-list> </url-pattern>
</url-pattern-list> </group>
</gcs-config>
```

Figure 1. Sample GCS configuration file.

Web site navigation rules are typically written by hand after a careful analysis of the target Web site. Semiautomatic tools may assist in the process, but largely due to the requirement to have extremely fine-tuned navigation rules, the use of automation is limited except in the simplest cases. We point out that it is important to minimize the number of pages retrieved from a Web site while maintaining a safeguard against potential changes occurring on the Web site over time. This balancing act is a difficult (but rewarding) challenge for which no practical, automated procedures have been devised.

NDES is independent of the specific crawler used, as crawlers in general provide a very similar service and are configured using the same general principles outlined above. The current implementation of NDES uses as its crawler the Grand Central Station (GCS) system [10][11], a flexible and extensible crawler framework developed at the IBM Almaden Research Center. In GCS, navigation rules are expressed as XML, an example of which is shown in Figure 1. The sample navigation rules direct GCS to retrieve reviews of IBM's laptop computers from epinions.com.

3.2 Data Extraction

Target HTML pages are subjected to a sequence of data extraction steps. As mentioned earlier, much of the HTML content on the Web today is ill-formed because it does not conform to HTML specifications. Therefore, the first step in data extraction is to translate the content to a well-formed XML syntax because this helps in subsequent data extraction steps. The specific approach taken in the NDES framework is to pass the original HTML page through a filter that "repairs" the broken syntax and produces well-formed HTML, or what is today known as Extensible HTML (XHTML) [16]. Toolkits for this step exist already, including the Tidy package [12].

Since XHTML is based on XML, any XML tool can be used to further process target HTML pages. Given that the goal of NDES is to produce XML as output, we view the task of converting XHTML to XML as an XML transformation problem. The data transformation mechanism chosen for NDES is Extensible Stylesheet Language Transformations (XSLT) [20], a language that provides powerful XML path expressions (XPath) [19] combined with regular expressions through the XSLT extension mechanism.

As shown in Figure 2, the URL of an XHTML document is used to determine which set of XSLT files to apply to it. The XHMTL document is passed through the first XSLT file and the output is pipelined through other XSLT files defined for that

URL. The final output is an XML file whose structure and content is determined by the last XSLT file. This is typically an XML application[2], for instance iCalendar XML or NewsML. The pipeline approach fits well with the goals of domain-specific NDES applications; the first XSLT file merely extracts data from an XHTML page, while subsequent XSLT files in the pipeline can refine the data and fill in missing data from domain knowledge (more on structure synthesis in Section 3.4).

The main criticism directed towards HTML data extraction projects is that the approach essentially amounts to "screen scraping" and fails miserably when the design (structure and content) of a Web site changes. While total isolation from these changes is difficult to achieve, we believe the NDES approach is solid and produces very robust wrappers. This is achieved by relying less on HTML structure and more on content.

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 (<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 in XPath as /HTML/BODY/TABLE[3]/TR[1]/TD[2].

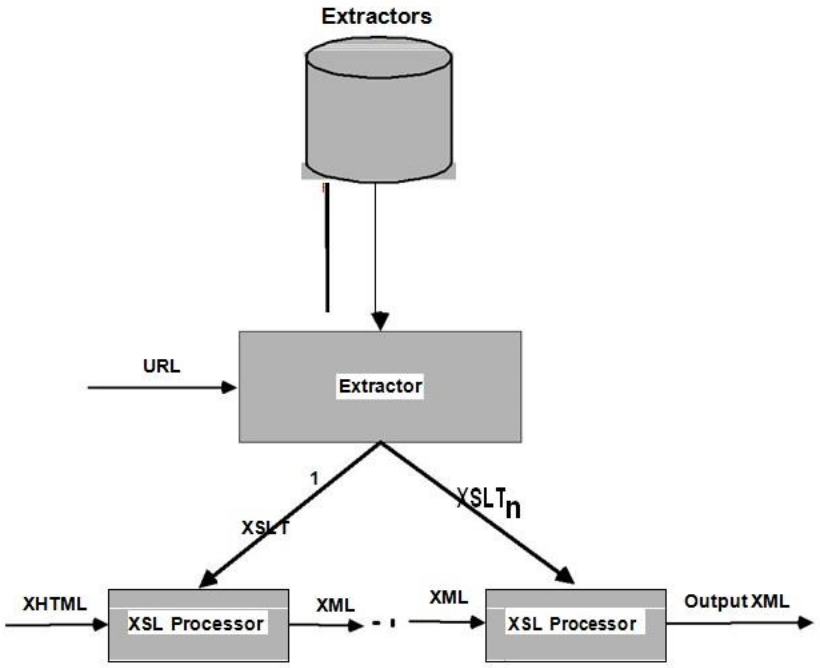

Figure 2. Extractor identifies XSLT files to be used. A pipeline of XSLT processors extract and refine data, yielding an XML application file as the ultimate output.

The absolute path approach is likely to fail when the target HTML page changes. The most common change in HTML design is changing the positioning of items on the page. Layout is typically performed by using tags like <TABLE>, <TR>, and <TD>, as seen in the example above. When new content (e.g. advertising) is added to a page or when existing content is moved around, the absolute location of tags changes. For this reason, it is important to establish the location of data items independently of their absolute paths.

Our approach involves finding anchors within the page that serve as starting points for data extraction. Ideally, anchors are established based on the content of a data item, not on its HTML path. For instance, a page that contains the price of a book probably has the word "Price" somewhere near the price value. By looking for the word "Price," we can establish an anchor for the price value and be independent of its absolute location.

An example of XSLT code that extracts the last stock quote from a Yahoo! Finance page is shown in Figure 3. Note that we look for a table cell containing the words "Last Trade" and extract the value contained in the B (bold) tag. The XSL processor starts from the root of the XHTML tree and recursively looks for a matching table cell. Once the table cell is found, the instructions contained in the template are executed, in this case the production of a PRICE element in the output XML document.

3.3 Hyperlink Creation

One shortcoming of today's crawlers is that they can typically only follow static hyperlinks (such as those contained in <A>,

<FRAME>, and <IMG> tags) but not dynamic hyperlinks that are a result of HTML forms and JavaScript code. Dynamic hyperlinks are typically computed based on user input but may involve arbitrary computation in JavaScript.

It has been argued that a large fraction of Web content is "hidden" this way, whether the hiding is a side effect or an explicit goal of the Web site owner. The notion of "deep Web" [6] has been used to describe the hidden data on the Web. NDES provides access to the deep Web by analyzing HTML forms and JavaScript code and extending the crawler's reach with "synthetic hyperlinks," or static copies of dynamic hyperlinks.

This is accomplished by passing each Web page through one or more XSLT filters that analyze HTML forms and JavaScript code and produce a list of static hyperlinks that mimic the selections made by an imaginary user. The links are then inserted back into the page as regular <A> tags and the page is passed to the crawler. The advantage of this approach is that the crawler itself is not modified in any way. Figure 4 illustrates the HTML augmentation process used in hyperlink synthesis.

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

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

</xsl:template>
```

Figure 3. Sample XSLT extraction rule.

Special care is needed for handling HTML forms that require the use of the POST method, as crawlers typically cannot handle links other than those that use the GET method. A simple GET-to-POST method conversion proxy solves this problem. The proxy receives GET requests from the crawler and converts specially marked hyperlinks to use the POST method instead. The target Web site is accessed using the correct method and the resulting page is returned to the crawler.

3.4 Structure Combination

It is fairly easy to extract simple data items such as the stock quote shown in Figure 3. The extracted data is simple in structure and its presentation on the HTML page maps directly to a corresponding XML structure (essentially a flat database record).

Structure synthesis may be required in more complex situations. Consider the task of aggregating product catalog data from several Web sites. Here, it is essential to represent catalogs and the products they contain in as much detail and fine granularity as possible so that integration of the catalogs can be successfully made. What makes this difficult is that a Web site may not provide enough structure to make direct mapping to an XML structure possible. For instance, on many online shopping sites product features are embedded in plain text paragraphs, and some data may be omitted because it is implicitly understood by the user viewing the page or is available elsewhere (e.g. that a laptop computer has an LCD display as opposed to a CRT display).

In NDES, a knowledge engineer familiar with the target domain can provide regular expressions that extract structured data from snippets of unstructured text. For instance, the string "Mon Oct 30" is easy to convert to a timestamp structure with regular expressions. This analysis is potentially less accurate than a deep linguistic analysis but is still very powerful and quick to master.

Missing data can be filled in by XSLT code that encapsulates domain knowledge. For instance, an XSLT file that is specific to computer products can inspect the description of a computer and determine if it refers to one of the known laptop brands (e.g. IBM ThinkPad). If so, a new product feature can be added to the output XML structure, specifying that the computer has a worldwide three-year warranty, even if this fact is not listed in the product description.

3.5 Data Mapping

Aggregating data from several Web sites requires that the data be homogenized. Web sites may follow different conventions for naming things or for expressing measured units. Mapping discrete values (e.g. company names) into a standard format improves the quality of the extracted data. Web sites are prone to have misspelled names, leading to data pollution if not corrected during the extraction process. The unit of measured values (e.g. corporate earnings) needs to be indicated in the XML output, or the values need to be converted to a common unit (e.g. billions of US dollars).

Homogenization of discrete values and measured values is performed in NDES with a combination of conditional statements, regular expressions, and domain-specific knowledge encapsulated in the XSLT code. Arbitrary Java methods can also be invoked, for instance looking up a company name in a JDBC-compatible database.

3.6 Data Integration

The final task in Web data extraction is to integrate data from multiple, related Web pages. There are two reasons why this is necessary. First, some Web sites use HTML frames for layout, which breaks up a logical data unit into separate HTML documents. Second, some Web sites break up the data across multiple "sibling pages" so as not to overload a single page with too much information. For instance, Yahoo! Finance contains comprehensive financial information on each company in its database, but each of their Web pages contains only a fraction of the data (e.g. company background information, stock quotes, opinions of financial analysts, etc.).

Data integration in NDES is performed in two steps. First, the original HTML documents are crawled normally and data is extracted from them individually. The output XML from each extractor is a piece of the ultimate output XML. For instance, if the

output is in the NewsML format, one piece could contain a news article body, while another piece contains the publisher's contact information. Both pieces conform to NewsML but each piece has only part of the data.

The second step involves concatenating these partial outputs into one output and passing the resulting file through an XSLT filter that merges related data. Continuing the example above, the filter could identify related NewsML fractions based on the URL of the original HTML documents and "join" the data. The ultimate output has the data merged in an application-specific manner.

4. NDES ARCHITECTURE

In this section we describe some operational features of the NDES framework. All features described in Section 3 have been implemented in the framework and are in production use within IBM.

4.1 Overview

NDES has been written in pure Java and consists of five components: data retriever, extractor, checker, exporter, and scheduler/manager interface (Figure 5). As mentioned earlier, the default data retriever is the Grand Central Station (GCS) crawler. Crawls are scheduled to occur on certain days and times that depend on the target sites. When the scheduled time arrives, GCS is invoked and target HTML pages are retrieved from the Web sites. The pages are delivered to the extractor, which performs data extraction, structure synthesis, and data mapping and integration functions.

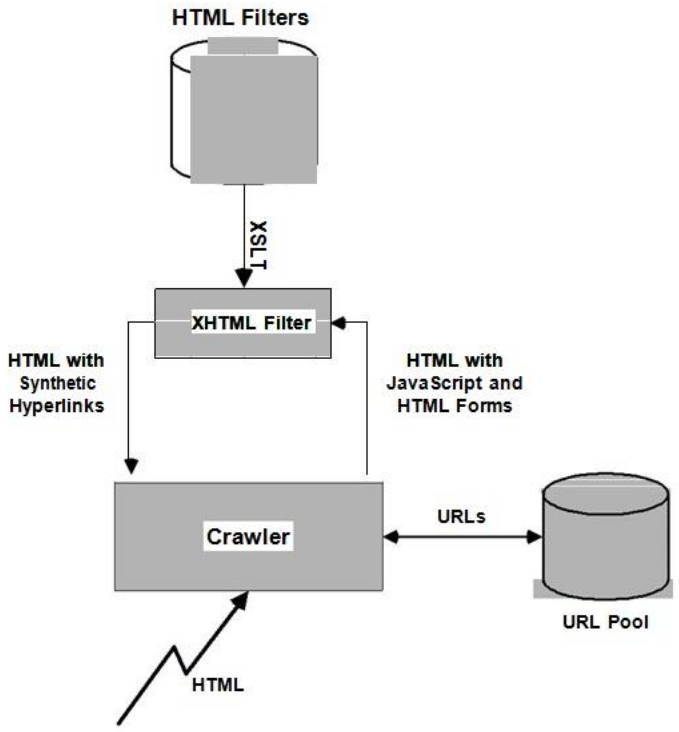

Figure 4. Hyperlink synthesis via HTML augmentation.

The output XML documents produced by the extractor are forwarded to a data checker and finally to an exporter. Different exporters may be written; the default exporter inserts the XML data into a relational database, thereby making the extracted data available to database applications and tools (e.g. for data mining).

A Web -based management interface allows the administrator to control NDES, making changes to the scheduled crawls, inspecting the status of a running crawl, and viewing statistics of completed crawls. NDES has been designed to run with little operator intervention, and it is configured to alert the administrator by email if problems are encountered. Several NDES installations can co-exist on the same host computer, all managed through one Web interface.

4.2 Scheduling

The scheduler is responsible for triggering the data extraction from different Web sites at prespecified times and repeating the extraction periodically. The periodicity depends largely on the frequency of data change, but also on domain-specific and corporate requirements. Data extraction is usually run during periods of low network activity, for instance at night.

4.3 Web-Based Management Interface

We have designed a comprehensive, Web-based management interface that a system administrator can use for monitoring and controlling NDES. It is also essential that NDES be capable of monitoring itself and alerting the administrator when problems are encountered. Given that Web sites are autonomous and can change at any time, NDES monitors the coverage of each crawl and issues an alert when significant changes in the coverage are noticed. For instance, if the navigation rules of a Web site change, NDES may get only half the number of target HTML pages compared to a previous crawl. When this happens, the administrator is notified, who will then take appropriate action.

4.4 Data Validation

As Web sites change, an XSLT file may fail to correctly extract data from the pages that changed. NDES continuously monitors the quality of data extraction and alerts the administrator when adjustments to XSLT files are required. Data validation is performed on the XML output of XSLT filters, not on the HTML source files. This means that if a Web site changes but the XSLT filter continues to correctly extract data from the changed pages, the new pages pass the data validation check and no alerts are generated.

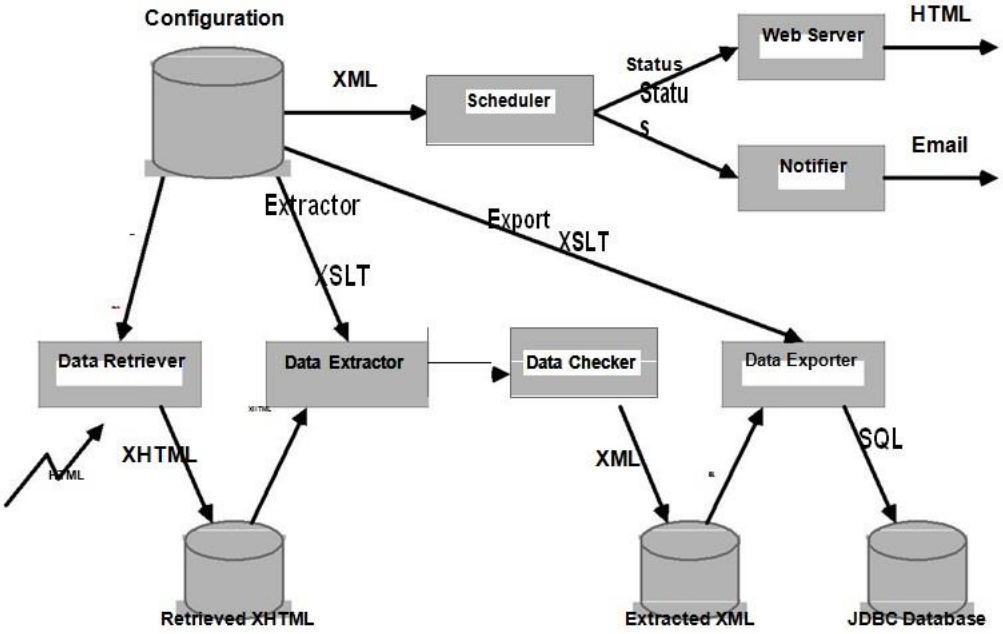

Figure 5. Overview of NDES architecture.

Data validation is performed on several levels of abstraction. Syntactic checks are performed first: they verify that each XML element is present in the output and that values match their expected types (numeric vs. string). This is followed by semantic checks which spot incorrect values. This is domain-specific but very powerful. For instance, if it is known that stock prices are usually less than $ 1000 (Berkshire-Hathaway shares being the notable exception), this can be described to the data validator which then separates the "bad" data from "good." The bad data is moved to a staging area and the administrator is asked to decide what to do with it. The administrator can accept the data as-is, the boundary conditions can be automatically modified, the data can be ignored as a one-time error, or the data can be manually corrected.

4.5 Data Export

The default data exporter in NDES converts XML data to relational tuples and inserts them into a JDBC database. The normalization into tabular data is performed using XSLT and the resulting Comma Separated Values (CSV) files are loaded into the database. Alternative approaches are currently being studied, including the use of an XML extension to relational databases, such as the DB2 XML Extender product [5].

5. CONCLUSIONS AND FUTURE WORK

In this paper we have discussed the problem of data extraction from Web sites and suggested an XML-based approach for solving it. We view the task of data extraction as a multi-step process where the goal extends far beyond simple "screen scraping." Our ultimate goal is to be able to extract semistructured data from given Web sites and transform the data into a well-structured, feature-rich representation. Managing the heterogeneity of data retrieved from different Web sites is an integral part of this process, as is domain-specific processing of missing and conflicting data.

Since many Web sites are driven by a set of HTML templates and a database backend, an ideal Web data extractor would be able to "see through the templates" and create an identical copy of such databases even though it only has a limited view of the data. Our experience with the NDES framework has shown that production-quality Web data extraction is quite feasible, and that incorporating domain knowledge into the data extraction process can be effective in ensuring the high quality of extracted data.

Our work continues in several areas. Navigation rules and extraction rules are currently optimized by hand, a burden that automatic or semiautomatic tools may ease. Likewise, automatic tools will help in building and managing domain-specific rules for handling missing or conflicting data. It is not uncommon to see incorrect data in Web sites (misspelled names and incorrect quantities or units). Our current, manually written data validation rules will be replaced by rules generated by semiautomatic tools; data classification and machine learning techniques appear to be promising solutions. In the future, we also expect to use the XML Schema syntax [18] for expressing data validation rules.

6. ACKNOWLEDGEMENTS

The authors thank to the principal and the management for constant support and encouragement.

REFERENCES

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

[2] Naveen Ashish and Craig Knoblock. Wrapper Generation for Semi-structured Internet Sources. In Proc. ACM SIGMOD Workshop on Management of Semistructured Data, Tucson, Arizona, May 1997.

[3] 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.

[4] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. The TSIMMIS Project: Integration of Heterogeneous Information Sources. Proc. IPSJ Conference, 1994.

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

[6] BrightPlanet.com DeepWeb White Paper.
http://www. completeplanet.com/Tutorials/DeepWeb/index.asp.

[7] Craig Knoblock, Steven Minton, Jose Luis Ambite, Naveen Ashish, Pragnesh Modi, Ion Muslea, Andrew Philpot, and Sheila Tejada. Modeling Web Sources for Information Integration. In Proc. National Conference on Artificial Intelligence (AAAI), July 1998.

[8] 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.

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

[10] Bruce Schechter. Information on the Fast Track, IBM Research Magazine, 35(3): 18–21, 1997.

[11] Marc Songini. IBM: All Searches Start at Grand Central, Network World, November 11, 1997.

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

[13] Mary Tork Roth and Peter Schwartz. Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. Proc. International Conference on Very Large Data Bases (VLDB), Athens, Greece, August 1997.

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

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

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

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

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

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

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

