9-2008

Mining Edgar Tender Offers

Douglas A. Lyon
Fairfield University, dlyon@fairfield.edu

Published Citation

Douglas Lyon, "Mining Edgar Tender Offers", Journal of Object Technology, Volume 7, no. 7 (September 2008), pp. 17-31

JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©JOT, 2008

Vol. 7, No. 7, September-October 2008

Mining Edgar Tender Offers

Douglas Lyon, Ph.D.

Abstract

This paper describes how use the *HTMLEditorKit* to perform web data mining on EDGAR (Electronic Data-Gathering, Analysis, and Retrieval system). EDGAR is the SEC's (U.S. Securities and Exchange Commission) means of automating the collection, validation, indexing, acceptance, and forwarding of submissions. Some entities are regulated by the SEC (e.g. publicly traded firms) and are required, by law, to file with the SEC.

Our focus is on making use of EDGAR to get information about company offers to purchase stock, known as *tender offers*. These offers are filed with companies, using their Central Index Key (CIK). The CIK is used on the SEC's computer system to identify corporations and individual people who have filed a disclosure with the SEC. We show how to map a stock ticker symbol into a CIK and how to extract tender offer data. Our example show how we extract the number of shares tendered, the price range for the auction, the honoring of "odd lots" and the initial termination date for the auction.

The methodology for converting the web data source into internal data structures is based on using HTML as input into a parser-call-back facility that builds up a data structure using a context sensitive table data. Screen scraping is a popular means of data entry, but the unstructured nature of HTML pages makes this a challenge.

1 THE PROBLEM

The Williams Act (named for New Jersey Senator, Harrison A. Williams of New Jersey) was passed into law in 1968. The act requires that a company who engages in a tender offer state all details of the offer in a filing to the SEC. The filing must include the terms, cash source, any plans for the company after takeover, etc. The law mandates a minimum offering period of 20 days and gives tendering shareholders 15 days to change their minds. If only a limited number of shares are accepted, they must be prorated among the tendering stockholders.

Since 1996, all public domestic companies have been required to make their filings with the SEC using the EDGAR system (except in the case of hardship). Since 2002 this ruling has also applied to those foreign companies that are subject to regulation by the SEC. In short, most filings are electronic and these filings constitute a domain-limited financial narrative that is available on the web. According to our scanning program, there have been 6,842,333 filings made using the EDGAR system between Q1 of 1996 and 7/15/07. Of these, 2,828 are of type "SC TO-I" (primary tender offers). There were third-party tender offers, and amendments to tender offers,

Cite this column as follows: Douglas A. Lyon "Mining Edgar Tender Offers", in *Journal of Object Technology*, vol. 7, no. 7, September-October 2008, pp. 17-31

but we did not count these. The massive nature of the database makes automation of the mining process desirable.

Thus, we are given an HTML data source, in EDGAR, and we would like to find a way to create an underlying data structure for describing tender offers that is both type-safe and well formulated.

We are motivated to study these problems for a variety of reasons. Firstly, for the purpose of conducting empirical studies, entering the data into the computer by hand is both error-prone and tedious. We seek a means to get this data, using free data feeds, so that we can build dynamically updated displays and perform data mining functions. Secondly, we find that domain limited data should be easier to parse, yet it is surprisingly difficult and these challenges enable us to hone our techniques for data mining. This example is used in a first course on network programming.

2 FINDING THE DATA

Finding the data, on-line, and free, is a necessary first step toward this type of data mining. We obtain EDGAR filings by using the SEC's CIK (Central Index Key). The keys are stored in a table that is available via anonymous FTP at ftp.sec.gov. The table is updated daily and is large. The keys are also available via a web interface lookup feature. The interface requires a company name (not a ticker symbol). However, we are used to entering ticker symbols and prefer that be used for the interface. Thus, our first step is to map the ticker symbol into a company name, then use the company name to query the CIK database. It is then a matter of constructing an HTML parser that is able to extract the CIK from the EDGAR reply. For example:

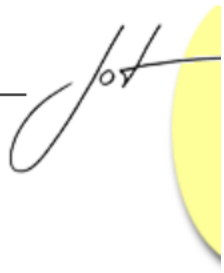
http://www.sec.gov/cgi-bin/cik.pl.c?company=home+depot

This creates an output on the screen that looks Figure 1-1.

Figure 1-1. The EDGAR CIK

To synthesize the URL needed to get the data, we use:

```
public static String getUrlCIK2(String companyName) {
        return "http://www.sec.gov/cgi-
bin/cik.pl.c?company=" +
```



```
                UrlUtils.conditionUrl(companyName);
        }
```

Where:

```
public static String conditionUrl(String s){
        return s.replaceAll(" ","%20");
    }
```

Is needed to make sure that illegal URL characters, like spaces, are replace with their decimal equivalents. After a great deal of development, we discover that the EDGAR system has bugs in its' query results. For example, a search for: "Dominion Resources Inc" results in:

```
0000314712   DOMINION RESOURCES INC /DE/
0000826613   DOMINION RESOURCES INC /TA/
/TA
0000715957   DOMINION RESOURCES INC /VA/
0000314712   DOMINION RESOURCES INC/DE/
```

The first (and last company) is "DIGITAL IMAGING RESOURCES INC". The second and third companies represent a change of location. For example, Dominion is listed as: "formerly: DOMINION RESOURCES INC /TA/ /TA (filings through 2006-03-27))". Thus "getUrlCIK2" becomes the backup query engine, with the primary query formulated with:

```
public static String getUrlCIK(String companyName){
        return

"http://www.edgarcompany.sec.gov/servlet/CompanyDBSearch?"
+
                "start_row=-1&end_row=-
1&main_back=1&cik=&company_name=" +
                UrlUtils.conditionUrl(companyName) +

"&reporting_file_number=&series_id=&series_name=&" +
                "class_contract_id=&class_contract_name=&"
+

"state_country=NONE&city=&state_incorporation=NONE&" +

"zip_code=&last_update_from=&last_update_to=&page=summary&"
+
                "submit_button=View+Summary";
}
```

We automatically fall-back to the secondary source, in the case of primary source failure.

3 ANALYSIS

We use a *ParserCallBack* class to process the HTML data. The goal is to identify the relevant data in the HTML responses from the primary and secondary sources. In the case of the primary source, we get:

```
<A
HREF="/servlet/CompanyDBSearch?page=detailed&cik=0000869614
&main_back=2">
```

And in the case of the secondary source we get *hrefs* in the form of:

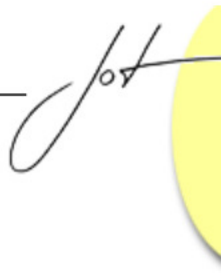
```
<a href="browse-
edgar?action=getcompany&CIK=354950">0000354950</a>
```

Thus, we are interested in anchor tags that contain *href* attribute "CIK=". This is done with a combination of standard callback features and ad-hoc string manipulations. Each time we approach the problem of parsing new data, our goal is to make the parser tool a little bit more general (and thus reusable):

```
public class EdgarParser extends
HTMLEditorKit.ParserCallback {
    private HTML.Tag startTag = null;
    private HTML.Tag endTag = null;
    private String lastText = "";
    private int cik = 0;

    public EdgarParser(URL url) throws IOException,
BadLocationException {
        DataMiningUtils.mineParser(this, url);
    }

    /*
    <A
HREF="/servlet/CompanyDBSearch?page=detailed&cik=0000869614
&main_back=2">
     */
    public void handleStartTag(HTML.Tag startTag,
MutableAttributeSet a, int pos) {
        this.startTag = startTag;
        if (startTag.equals(HTML.Tag.A)) {
            String href = ((String)
a.getAttribute(HTML.Attribute.HREF)).toUpperCase();
            if (href.contains("CIK=")) {
                String s = StringUtils.isolate(href,
"CIK=", "&");
                if (s != null)
                    cik = Integer.parseInt(s);
```



```
                else //secondary url is being used
                    cik = Integer.parseInt(

href.substring(href.indexOf("CIK=") + 4));
                }
            }
        }
```

We are using the *ParserCallBack* to look for the primary and secondary URLs:

```
<A
HREF="/servlet/CompanyDBSearch?page=detailed&cik=0000869614
&main_back=2">
```

and:

```
<a href="browse-
edgar?action=getcompany&CIK=354950">0000354950
```

For example, on the secondary URL, the *href* attribute that is returned is:

```
browse-edgar?action=getcompany&CIK=354950
```

Thus it is a simple (though data-specific) matter to isolate the CIK string and parse it. The CIK is now a unique key into the EDGAR database and is stored in:

```
public class Edgar {
    private String symbol;

    private int cik = 0;
    private URL urls[];
    private DarTo darTo;
    private String companyName;
    private String urlCik;
    GoogleSummaryData gd;

    public Edgar(String symbol) throws IOException,
BadLocationException {
        this.symbol = symbol;
        gd = new GoogleSummaryParser(symbol).getValue();
        companyName = gd.getCompanyName();
        getcik();
        darTo = new DarTo(symbol);
    }

    private void getcik() throws IOException,
BadLocationException {
        getCik1();
```

```
        if (cik == 0) {
            companyName = companyName.replaceAll("INC",
"").trim();
            companyName = companyName.replaceFirst("-", "
").trim();
            companyName =
companyName.replaceFirst("\\(INTERACTIVE\\)", "").trim();
            getCik1();
        }
    }

    private void getCik1() throws IOException,
BadLocationException {
        urlCik = getUrlCIK(companyName);
        EdgarParser ep = new EdgarParser(new URL(urlCik));
        cik = ep.getCik();
    }
```

We make use of the Google summary data because YAHOO finance tends to mangle the name of the company in its title. We obtain the Google summary in order to obtain the company name and then transform the name into a form that the Edgar system will recognize. The URL for Google finance is extracted from:

```
    public static URL getUrl(String ticker) throws
MalformedURLException {
        return new
URL("http://finance.google.com/finance?q=" +
                ticker);
    }
```

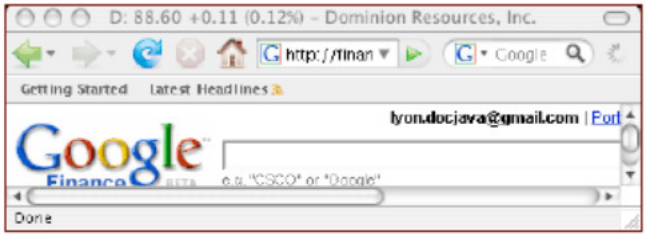
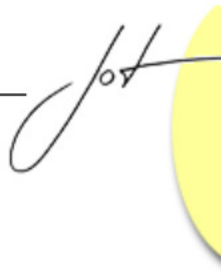


Figure 2-1. Sample Google output

Figure 2-1 shows the title in the sample Google output. The SEC wants the commas and periods removed from the title, in order to recognize the query. It also wants a series of other changes to *normalize* the form of the company name. This logic was incorporated, in an ad-hoc string manipulation procedure, embedded in the setter of the GoogleData:

```
public void setCompanyName(String companyName) {
        companyName =
companyName.toUpperCase().replaceFirst("THE", "").trim();
```



```
        companyName = companyName.replaceFirst("\\(USA\\)",
"").trim();
        companyName =
companyName.replaceFirst("\\(CAPITAL\\)", "").trim();
        if (companyName.endsWith("CORPORATION"))
            companyName =
companyName.replaceAll("CORPORATION", "CORP").trim();
        companyName = companyName.replaceFirst("COMPANY",
"CO").trim();
        this.companyName = companyName.replaceAll(",",
"").replaceAll("\\.", "").trim();
    }
```

For example, the Edgar system is confused by "The Home Depot, Inc.", it wants "Home Depot Inc". Also, "TLC Vision Corporation (USA)" must be "TLC Vision Corporation" and "Liberty Media Corporation" must be "Liberty Media Corporation". However, "Document Sciences Corporation "must be written as "Document Sciences Corp".

Even more special cases are needed with the Edgar search engine when things don't work the first time, thus accounting for:

```
if (cik == 0) {
            companyName = companyName.replaceAll("INC",
"").trim();
            companyName = companyName.replaceFirst("-", "
").trim();
            companyName =
companyName.replaceFirst("\\(INTERACTIVE\\)", "").trim();
            getCik1();
        }
```

4 BUILDING THE INTERFACE

We are interested in a new "killer application" for development, called the *JAddressBook* program. This program is able to chart historic stock volumes (and manage an address book, dial the phone, print labels, do data-mining, etc.). The program can be run (as a web start application) from:

http://show.docjava.com:8086/book/cgij/code/jnlp/addbk.JAddressBook.Main.jnlp

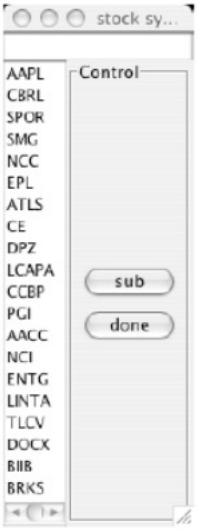


Figure 3-1. The Stock Symbols Dialog

Figure 3-1 shows entry of stock symbols into the stock symbols dialog. Once the user selects "done" a table of Edgar CIK numbers is constructed.

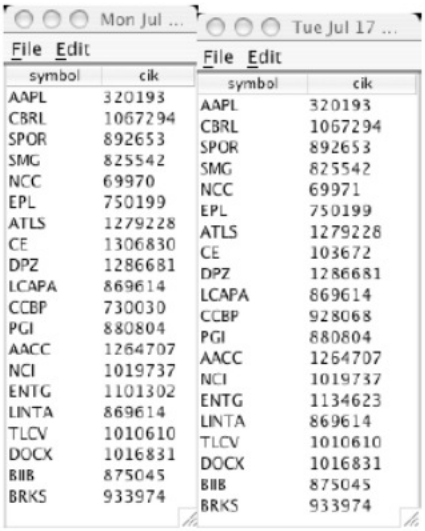


Figure 3-1. The CIK table

Figure 3-1 shows an image of the CIK table for a series of different symbols. CIK numbers can help to obtain information, not only from the query system, but also from the FTP file system that mirrors the Edgar filings.

5 MINING THE FTP SITE

There are a series of ftp files that are available on the Edgar site. They are text files that have been compressed, using GZIP. The database is too large to fit into RAM, however, it can be scanned, gradually. Since 1996, there were 6,842,333 electronic filings made to the EDGAR system. These are indexed in compressed text files. There are 4 files per year (one per quarter). The indexed files show the CIK number and the

form type. The form type is coded according to a standard. For example, we are interested in tender offers, so we can scan for form type "SC TO-I". Since 1996 there were 2,828 "SC TO-I" filings on EDGAR.

6 QUERY EDGAR TENDER OFFER FILINGS

In the previous section, we showed how to use EDGAR to obtain CIK data on a stock, given its symbol. In this we, how to process EDGAR tender offer filings on a stock, given its symbol. Queries can be formulated without a CIK and typed directly into a browser. For example, for Home Depot, the CIK is 353950, thus:

> http://www.sec.gov/cgi-bin/browse-edgar?type=sc+to&dateb=&owner=include&count=40&action=getcompany&CIK=354950

Will return an output, in the browser that looks list Figure 4-1.

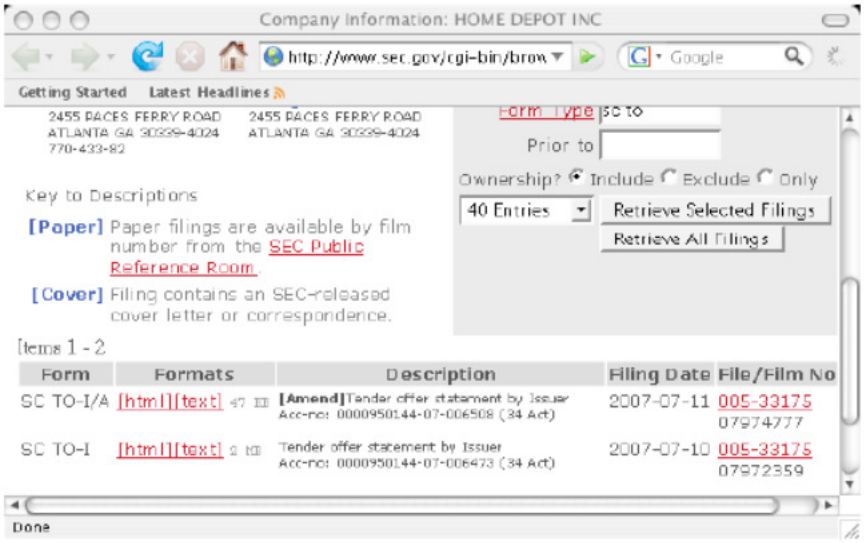


Figure 4-1. Tender Offer Filings with the SEC

Our first step is to scan the document for links, reformulate the relative links so that they are absolute links, sort them and remove the duplicates. We formulate the tender offer URL using:

```
public URL getUrlTo() throws MalformedURLException {
        return new URL("http://www.sec.gov/cgi-bin/browse-
edgar?type=" +
                "sc+to-i&" +
                "dateb=&" +

"owner=include&count=10&action=getcompany&CIK=" +
                cik);
    }
```

And we can test our *href* link parser using:

```
public void test() throws IOException, BadLocationException
{

        System.out.println(cik);
        LinkParser lp = new LinkParser(getUrlTo());
        urls = lp.getURLs();
        System.out.println(lp.getBase());
        PrintUtils.print(urls);
    }
```

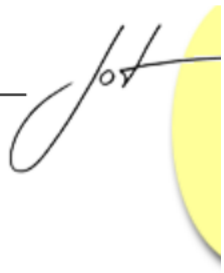
Where the heart of the links parser is given by:

```
public class LinkParser extends
HTMLEditorKit.ParserCallback {
    private HTML.Tag startTag = null;
    private HTML.Tag endTag = null;
    private String lastText = "";
    private SortableVector links = new SortableVector();
    private URL base;

    public LinkParser(URL url) throws IOException,
BadLocationException {
        base = url;
        DataMiningUtils.mineParser(this, url);

    }

    /**
     * sorts the links and removes the duplicates.
     * @return properly formatted absolute urls, no e-
mails, no javascript.
     */
    public URL[] getURLs() {
        links.removeDuplicates();
        URL urls[] = new URL[links.size()];
        for (int i=0; i < urls.length; i++){
            try {
                urls[i]= new
URL((String)links.elementAt(i));
            } catch (MalformedURLException e) {
                e.printStackTrace();
```



```
            }
        }
        return urls;
    }

    public void handleStartTag(HTML.Tag startTag,
MutableAttributeSet a, int pos) {
        this.startTag = startTag;
        if (startTag.equals(HTML.Tag.A)) {
            String href = (String)
a.getAttribute(HTML.Attribute.HREF);
            if (href != null) {
                processHref(href);
            }
        }
    }

    private void processHref(String href) {
        if (href.startsWith("javascript:")) return;
        if (href.startsWith("mailto:")) return;
        try {
            URL url = new URL(base, href);
            if (url != null)
                links.addElement(url.toString());
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }
```

We selected to ignore the *mailto* and *javascript* based *hrefs* as these are not important to our application.

The output follows (the CIK leads the way):

354950

http://www.sec.gov/cgi-bin/browse-edgar?type=sc+to-i&dateb=&owner=include&count=10&action=getcompany&CIK=354950

http://www.sec.gov/Archives/edgar/data/354950/000095014407006473/0000950144-07-006473-index.htm

http://www.sec.gov/Archives/edgar/data/354950/000095014407006473/0000950144-07-006473.txt

http://www.sec.gov/Archives/edgar/data/354950/000095014407006508/0000950144-07-006508-index.htm

http://www.sec.gov/Archives/edgar/data/354950/000095014407006508/0000950144-07-006508.txt

http://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&CIK=0000354950&owner=include&count=10

http://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&SIC=5211&owner=include&count=10

http://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&State=GA&owner=include&count=10

http://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&filenum=005-33175&owner=include&count=10

http://www.sec.gov/cgi-bin/browse-edgar?action=getcurrent

http://www.sec.gov/edgar/searchedgar/webusers.htm

http://www.sec.gov/index.htm

http://www.sec.gov/info/edgar/forms/edgform.pdf

http://www.sec.gov/info/edgar/ofis.shtml#access

Since we are only interested in the links to tender offers, we isolate the *hrefs* that end with *.txt*:

http://www.sec.gov/Archives/edgar/data/354950/000095014407006473/0000950144-07-006473.txt

http://www.sec.gov/Archives/edgar/data/354950/000095014407006508/0000950144-07-006508.txt

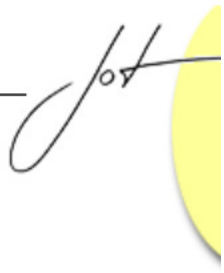
7 PARSING TENDER OFFERS

The previous section shows how to obtain all the relevant links that the SEC has in Edgar for tender offers. The next bit is a tad tricky. The first step is to construct the target data structure of the parsing operation:

```
public class DarTo {
    private String symbol=null;
    private double min=-1;
    private double max=-1;
    private Date initialTerminationDate=null;
    private boolean oddLotHonored = true;
```

In a Dutch tender offer, managers select a price range for tendering and announce the number of shares to be repurchased, the termination date of the offer and whether or not the odd-lot rule is in effect. Where an odd lot is defined as a number of shares that is less than 100. When the "odd lot rule" is in effect, the company will purchase any lot of shares less than 100, without proration. Proration occurs when more stock is tendered than the company is authorized to purchase.

Using Edgar filings on tender offers is surprisingly difficult. There are no standards for the statement and there is a good deal of variation from one statement to



the next. Still, using ad-hoc string manipulation techniques, we were able to derive the following information from the Edgar database:

```
symbol, min, max, term date, odd lot
hd,39.0,44.0,8/16/2007,true
d,82.0,92.0,8/07/2007,true
schw,19.5,22.5,6/29/2007,true
brks,16.5,19.0,5/31/2007,true
biib,47.0,53.0,6/26/2007,true
entg,11.0,12.25,6/08/2007,true
```

The high-level API for doing this makes the job looks easy and reliable. It is neither.

```
System.out.println(DarTo.getHeader());
        String symbols[] = {
            "hd","d","schw","brks","biib",
              "entg"
        };
        for (int i=0; i < symbols.length; i++)
            testEdgar(symbols[i]);
        System.out.println("done");
```

The heart of the parser is given by:

```
private void parseDar(URL url) {
        //System.out.println(url);
        parseDar(UrlUtils.getOneBigUrlString(
                        url.toString()));
    }

    private void parseDar(String text) {
        text =
StringUtils.removeOddCharacters(StringUtils.removeNewLines(
text.toUpperCase()));
        if (text.contains("ODD LOTS"))
            darTo.setOddLotHonored(true);
        darTo.setMax(StringUtils.getMax(text));
        darTo.setMin(StringUtils.getMin(text));

darTo.setInitialTerminationDate(StringUtils.getTerminationD
ate(text));
    }
```

We obtain the contents of the URL as one large string, convert it to upper case, strip out the control characters and some of the HTML tags. We then set to work with ad-hoc string manipulations:

```
public static double getMax(String text) {
```

```
            Pattern pattern = Pattern.compile("NOT GREATER THAN
" +
                    usDollarCurrencyPattern);
            Matcher maxMatcher = pattern.matcher(text);
            if (maxMatcher.find()) {
                return Double.parseDouble(maxMatcher.group(1));
            }
            return -1;
        }
```

Where the simplified currency pattern is given by:

```
    // a non-comma delimited currency with two decimal points
        private static final String usDollarCurrencyPattern =
"\\$([0-9]*.[0-9]{2})";
```

For the termination date of the auction, we use:

```
    public static Date getTerminationDate(String s) {
            String iso1 = isolate(s, "WITHDRAWAL RIGHTS ",
                    "UNLESS THE OFFER IS EXTENDED");
            if (iso1==null) return null;
            final String iso2[] = iso1.split("ON");
            return DateUtils.getLongDate(iso2);
        }
```

The *isolate* method strips off the prefix and the postfix string from the source, leaving a smaller, string of interest:

```
    public static String isolate(String source, String prefix,
    String postfix) {
            if (!source.contains(prefix)) return null;
            if (!source.contains(postfix)) return null;
            int start = source.indexOf(prefix) +
    prefix.length();
            int end = source.lastIndexOf(postfix);
            if (start > end) return null;
            return source.substring(start, end);
        }
```

8 CONCLUSION

In this paper we disclosed techniques that make use of the *HTMLEditorKit* and ad-hoc parsing to extract numeric, context-sensitive table data, from the web. This technique presents some reusable code, along with a plug-in style callback-parsing framework that is sensitive to changes in URL protocol and presentation data.

A new set of problems appears when attempting to parse unstructured financial narratives (even in a domain restricted area). The question of how to best approach this problem remains open. After a Dutch auction is over, we should be able to mine the results and summarize them in a semi-automatic manner. This remains a topic of current research.

Inconsistency in the wording of the company filings, along with inconsistency of the search results provided by the Edgar system conspired to make data mining a difficult task.

REFERENCES

[Lyon 04D] *Java for Programmers*, by Douglas A. Lyon, Prentice Hall, Englewood Cliffs, NJ, 2004.

[Lyon 08C] "Multi-threaded Data Mining of EDGAR CIKs (Central Index Keys) from Ticker Symbols", by Douglas A. Lyon, 1st International Workshop on Parallel and Distributed Computing in Finance (PDCoF) 2008 Technical Program Friday April 18, 2008, Proceedings 22nd IEEE International Parallel and Distributed Processing Symposium.

About the author

Douglas A. Lyon (M'89-SM'00) received the Ph.D., M.S. and B.S. degrees in computer and systems engineering from Rensselaer Polytechnic Institute (1991, 1985 and 1983). Dr. Lyon has worked at AT&T Bell Laboratories at Murray Hill, NJ and the Jet Propulsion Laboratory at the California Institute of Technology, Pasadena, CA. He is currently the Chairman of the Computer Engineering Department at Fairfield University, in Fairfield CT, a senior member of the IEEE and President of DocJava, Inc., a consulting firm in Connecticut. Dr. Lyon has authored or co-authored three books (Java, Digital Signal Processing, Image Processing in Java and Java for Programmers). He has authored over 30 journal publications. Email: lyon@docjava.com. Web: http://www.DocJava.com.