Multi-threaded Data Mining of EDGAR CIKs (Central Index Keys) from Ticker Symbols

Douglas A. Lyon
Chairman, Computer Engineering Department
Fairfield University
1073 North Benson Rd.
Fairfield, CT 06824
lyon@docjava.com

Abstract

This paper describes how use the Java Swing HTMLEditorKit to perform multi-threaded web data mining on the EDGAR system (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 filings. These offers are filed with companies, using their Central Index Key (CIK). The CIK is used on the SEC's computer system to identify entities that filed a disclosure with the SEC. We show how to map a stock ticker symbol into a CIK.

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

The stop-and-wait nature of HTTP queries, as well as the non-deterministic nature of the response time, adversely impacts performance. We show that a combination of caching and multi-threading can improve performance by several orders of magnitude.

Introduction

According to our scanning program, there have been 6,842,333 filings made using the EDGAR system between 1996 and July of 2007. The massive nature of the database makes automation of the mining process desirable. Each of these filings is listed in a master record that consists of the CIK, a form type, and a company name. However, the company symbol is omitted from the index. EDGAR does, however, supply a search facility that will map a company name into a CIK. It does not supply a facility that maps a ticker symbol into a CIK (which is what we would like to do).

Thus, we are given an HTML data source, in EDGAR, and a ticker symbol. We would like to find a way to map the ticker symbol into a CIK number.

We are motivated to extract the CIK number because all filings are done via CIK number. This facilitates the look-up of information and helps in our conducting empirical studies, using data mining. 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.

We will show that the non-deterministic nature of the data sources gives rise to the need for a multi-threaded, cached, data-mining approach. Without multi-threading, we are bottlenecked by the stop-and-wait protocols that characterize HTTP queries.

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://ftp.sec.gov/edgar/docs/cik.txt.zip. The file is 231 KB, compressed and 1.2 MB uncompressed (i.e., not that big). The table stores only company names, not symbols. However, it was last updated in 2003 (which is a deal killer). The shift in company organization and ticker symbol requires a database that is maintained.

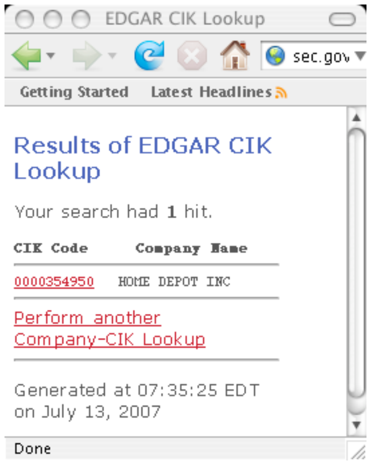
978-1-4244-1694-3/08/$25.00 ©2008 IEEE

Having out-of-date keys available for general use is both misleading and dangerous.

The keys are also available via a web-based GUI. The interface requires a company name (not a ticker symbol). However, we are used to entering ticker symbols and prefer it for human interface to our system. 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

Yields an output on the screen that looks Figure 2-1.



Results of EDGAR CIK Lookup

Your search had **1** hit.

CIK Code	Company Name
0000354950	HOME DEPOT INC

Perform another Company-CIK Lookup

Generated at 07:35:25 EDT on July 13, 2007

Figure 2-1. 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 replaced 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))". The term "TA" indicates that the CIK refers to a *transfer agent*. A transfer agent is an agency (usually a bank) that is appointed by a corporation to keep records of its stock and bond owners and to resolve problems about certificates. We typically skip transfer agents (as they are not primary filers).

Thus *getUrlCIK2* becomes the backup query engine, with the primary query formulated with (pseudo code):

```
getUrlCIK(String CompanyName){
  try Url1..try url2…try url3…etc.
}
```

This non-deterministic multiple-trial attempt at mapping a company name into a CIK is needed to provide a more robust means of doing the mapping. Thus, the approach of our algorithm is to automatically fallback to alternative sources. This results in unpredictable data mining performance.

3. Analysis

The *ParserCallBack* class uses HTML data to identify relevant data 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">
```

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) {

DataMiningUtils.mineParser(this, url);
}

/*
```

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

and:

```
<a href="browse-
edgar?action=getcompany&CIK=354950">000
0354950
```

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

    private void getCik1(){
        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:

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

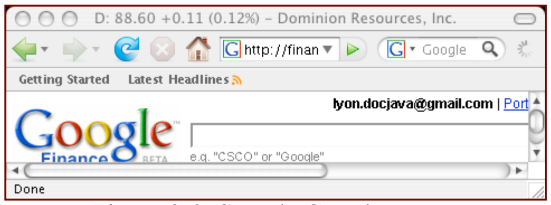


Figure 2-2. Sample Google output

Figure 2-2 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 (pseudo code):

```
public void setCompanyName(String
companyName) {
Convert to upper case.
```

```
Trim the spaces
Replace USA, with blank.
Replace CAPTITAL with blank.
Replace CORPORATION with CORP.
Replace COMPANY with CO.
  Etc.
```

Basically, we need a canonical company name for the mapping. 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". 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 really messy string manipulations that look like (pseudo code):

```
If cik is unset
    Replace INC with "", "-" with ""
    Replace "\INTERACTIVE\" with ""
    Execute an alternative CIK mining algorithm
```

For example:

```
testCik("asml");
```

ASML is a Dutch company that has an ADR in this country. Thus, it represents one of the more difficult companies to perform a lookup on. For example, when you search using *finance.yahoo.com* you get a response:

'ASML' is no longer valid. It has changed to ASMLD

Stockholders of ASML are surprised to learn of this apparent symbol change in their holdings (I have ASML employees in my class who were shocked by the news!). Apparently Yahoo changed its policy about listing tickers that represent ADRs'. Thankfully, *finance.google.com* still works, and you get a response:

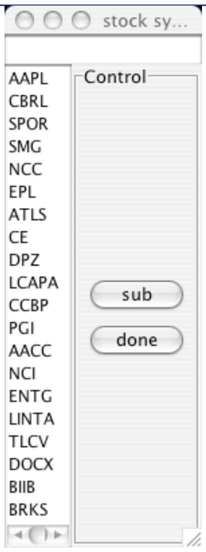
ASML Holding N.V. (ADR) (Public, NASDAQ:ASML)

The test code prints the CIK and some company info:

```
void testCik(String tckr) {
EdgarGoogle e = new EdgarGoogle(tckr);
System.out.println(
"------"+e.getCompanyName()+"------");
System.out.println(
    "cik:" + e.getCik());
System.out.println(
   "url cik:" + e.getUrlCik());
}
```

4. Building the Interface

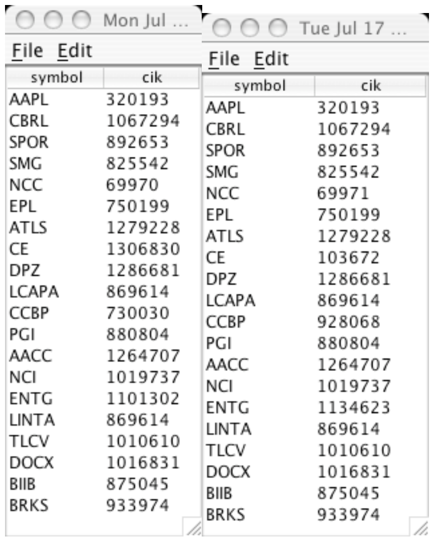
We are interested in a new "killer application" for development, called the *JAddressBook* program. This program is able to chart historic stock volumes (and manage an address book, dial the phone, print labels, do data-mining, etc.). The program can be run (as a web start application) from:
http://show.docjava.com:8086/book/cgij/code/jnlp/addbk.JAddressBook.Main.jnlp



AAPL
CBRL
SPOR
SMG
NCC
EPL
ATLS
CE
DPZ
LCAPA
CCBP
PGI
AACC
NCI
ENTG
LINTA
TLCV
DOCX
BIIB
BRKS

Figure 4-1. The Stock Symbols Dialog

Figure 4-1 shows entry of stock symbols into the stock symbols dialog. Once the user selects "done" a table of EDGAR CIK numbers is constructed.

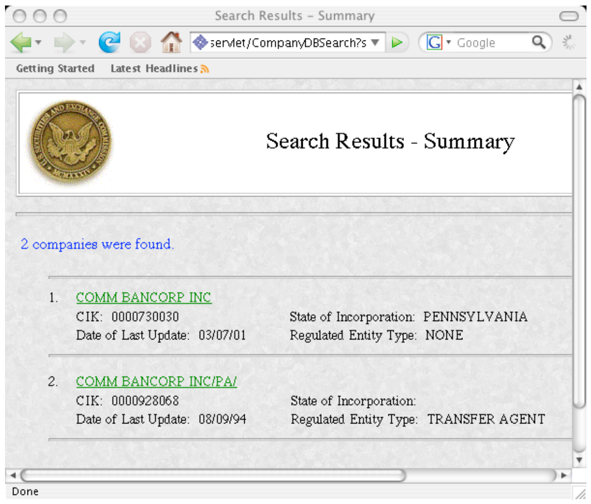


symbol	cik
AAPL	320193
CBRL	1067294
SPOR	892653
SMG	825542
NCC	69970
EPL	750199
ATLS	1279228
CE	1306830
DPZ	1286681
LCAPA	869614
CCBP	730030
PGI	880804
AACC	1264707
NCI	1019737
ENTG	1101302
LINTA	869614
TLCV	1010610
DOCX	1016831
BIIB	875045
BRKS	933974

symbol	cik
AAPL	320193
CBRL	1067294
SPOR	892653
SMG	825542
NCC	69971
EPL	750199
ATLS	1279228
CE	103672
DPZ	1286681
LCAPA	869614
CCBP	928068
PGI	880804
AACC	1264707
NCI	1019737
ENTG	1134623
LINTA	869614
TLCV	1010610
DOCX	1016831
BIIB	875045
BRKS	933974

Figure 4-2. The CIK table Comparison

Figure 4-2 shows an image of the CIK table comparison for a series of different symbols. The old algorithm is on the left. The backup algorithm, is on the right. The CCBP (COMM BANCORP, INC.) has a disparity (along with NCC, and CE). The correct answer (for CCBP) is 730030 (i.e., the original algorithm). In fact, if we scan the primary source, we find that the CIK for the transfer agent is listed.



Search Results - Summary

2 companies were found.

1. COMM BANCORP INC
CIK: 0000730030 State of Incorporation: PENNSYLVANIA
Date of Last Update: 03/07/01 Regulated Entity Type: NONE

2. COMM BANCORP INC/PA/
CIK: 0000928068 State of Incorporation:
Date of Last Update: 08/09/94 Regulated Entity Type: TRANSFER AGENT

Figure 4-3. Image showing transfer agent

So, the program is not wrong, per-se, it just need to know if the how to disambiguate the primary filers and the transfer agent. Transfer agents generally appear lower on the list (another ad-hoc observation!). Thus, we program the algorithm to always take the first result from the list (another hard-coded business rule!). Since CIK numbers are listed in order of "date of last update", we can be assured of the most recent CIK number. This is very important for companies that change state of incorporation. Our benchmarks show that we can map a ticker symbol into a CIK number in about 1 second. Our benchmark ran 20 symbols through our lookup facility. This is slow, but acceptable unless you want to build a large CIK/symbol database in advance (a reasonable alternative).

5. Multi-threading

In order to further speed our search for CIKs, we have resorted to creating a multi-threaded program that can resolve all the NASDAQ, NYSE and AMEX symbols into a single compressed CSV (Comma Separated Value) flat file. This can be used for fast lookups, when the symbol is already present in our cached data. We have placed the cached data on a web site, for ease of programmatic access.

Building a large cache, one symbol at a time, is very time consuming. Without multi-threading, the process would have taken days (rather than minutes). An easy quick hack was to make the lookups multi-threaded (and it makes the updates fun to watch!). The *MineAllCiks* class shows a fragment that gets a list of all the NASDAQ, NYSE and AMEX symbols (but not the pink sheets).

```
 public class MineAllCiks {
    private String symbols[];
    private Ciks ciks = new Ciks();
    private int numberOfJobsDone = 0;
    private int numberOfJobsRunning =
0;

public MineAllCiks(){
  Nasdaq nt = new Nasdaq();
  NyseAmex na = new NyseAmex();
  symbols = StringUtils.merge(
   nt.getTickers(),
    na.getTickers());
```

Once we have a long list of symbols (over 5,000). We typically have 200 threads running concurrently (a number that we arrived at empirically, in order to load-balance our systems queries so as not to time-out):

```
private void runAddCik(final int i) {
        numberOfJobsRunning++;
        new RunJob(1, false, 1) {
            public void run() {
                try {
                    addCik(symbols[i]);
                } catch (IOException e) {
                e.printStackTrace();

               } catch
(BadLocationException e) {
              e.printStackTrace();

              }
              numberOfJobsRunning--;
           }
       };
   }
```

Thus, the *RunJob* is a thread that runs only once and starts right away. Before the thread starts, it increments the *numberOfJobsRunning*. The thread decrements the *numberOfJobsRunning, upon termination*. This global variable is used to cause the launching to pause for 10 seconds if the number of threads exceeds 200:

```
private void addCiks(){
for (int i = 0;
     i < symbols.length; i++) {
           runAddCik(i);
// if the number of jobs running is
//greater than 200, sleep 10 seconds.
while (numberOfJobsRunning >200){
   int n =  numberOfJobsRunning;
   sleep(10);
System.out.println(
         "before sleep:"+n+
         " after sleep:" +
         numberOfJobsRunning);
System.out.println(
         "#ofCiks:"+ciks.getSize());
       }
   }
}
```

The act of performing a query is very time-consuming. Presently, our algorithm (unthreaded) takes

1 second per symbol to map a symbol into a CIK. However, in the multi-threaded mode, the algorithm is nearly 3 orders of magnitude faster, on average. Still, it is too slow for interactive speed when the number of symbols is high. Thus, we have taken to building a database in advance by mining the EDGAR site and creating a compressed CSV file that is posted on the authors web page. The trade-off is that the database must be built off-line. Thus we trade-off space for time and pay now, rather than pay later. These are quickly downloaded to populate our cache of CIK-symbol pairs, using:

```
public static String[]
getSymbolCikCompany()
throws IOException {
return UrlUtils.getTxtGz(new
URL("http://show.docjava.com:8086/book/
cgij/code/data/symbolCikCompany.txt.gz"
));

}.
```

The compressed contains the vast majority of the symbols, CIKs and company names, greatly speeding our lookup.

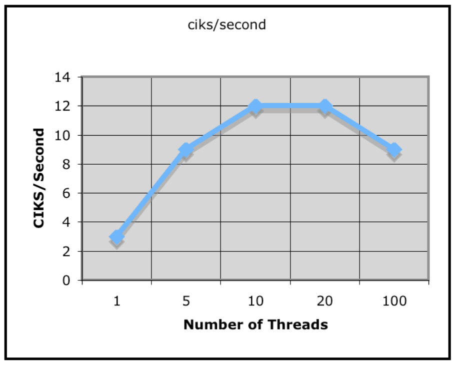


Figure 5-1. Non-cached Performance

Figure 5-1 shows diminishing returns when data mining many CIKS with multiple thread. Clearly, after only 10 threads, we reach diminishing returns, and adding threads only slows down the entire system. After we build our database, we establish a cache on a local web-server. This is downloaded into memory and is used to speed up lookups. Once this cache is built, we are able to increase the number of threads and improve our look-up rate. If all the symbols are cached (from a pre-built flat-file that is downloaded) we are able to sustain a rate of 30,000 CIKS/second (on a 2.4 Ghz Intel Core 2 Duo). When the cache is missed, for a single thread, we only get 1 or 2 CIKS/second. Thus, the question of how many threads to use is a function of how many bad symbols there will be in the list. Our experiments show that most of the symbols will be good, but that a few bad symbols can slow down the search by 4 orders of magnitude (for any given thread) Thus, with a cache in place, the number of threads that can be run has been found to be about 200. This is a comment more on the number of cache misses (on average) than the diminishing return performance shown in Figure 5-1. Thus, the limiting factor in the algorithm is a function of the average number of cache misses in the symbol look-up. This is something that is hard to predict, but can certainly be measured as a function of program use. Basically, we build the cache and look at the number of misses as a function of time. We call this *cache rot*. As the cache decays beyond a given point, it will need to be rebuilt. The question of how to implement this remains a topic of future work.

6. Previous Work

The idea of mining other forms for CIK data is not new either. The InsiderNewsWire.com approach is to make use of the 4K filings to obtain CIK data [1]. The main drawback of this approach is that not all companies' files 4K's every year. Also, the database is held static and, when a symbol cannot be found, no look up is performed. This leads to an incorrect null result.

Our approach is a hybrid approach, where most of the symbols are already mapped and, for those that do not appear, we make use of our multi-threading to perform parallel mining of the CIK data.

The use of the *RunJob* to provide a command design pattern and a façade design pattern to threads is not new [2,3]. Nor, for that matter, is the use of *HtmlEditors* for data mining [4-7]. What is new is the use of the *HTMLEditorKit* for multi-threaded data mining. Also new is the application of data mining in finance. The use of financial data mining, via the *HtmlEditorKit*, to discover CIK symbol relationships is, as far as we know, a new contribution.

Another theme in the data mining area, that appears new, is the notion of repeated attempts to learn of specific data. That is, when our first attempt to learn the CIK symbol map failed, we made use of a secondary source.

The idea of mining EDGAR for data is not new, however, in the work of Grant and Conlon, the use only the header in the 10K filing to perform CIK extraction. Their focus was on the new XBRL format and natural language processing [8].

7. 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.

This paper also described a multi-threading approach to data mining that enables our data-mining algorithm to over-come the stop-and-wait limits to throughput that plague most data mining programs.

A critical bottleneck ensues when the worst-case scenario develops and the cache misses increases. The question of how to adapt to cache misses and enables the program to learn new symbol-CIK pairs, remains open.

The messy ad-hoc string manipulations of data mining seem to creep in, no matter how high-level the data-mining framework. Regular expression parsers, HTML editors, etc, all ease the burden, but there really should be something better.

The question of why the *HtmlEditorKit* has not been used more often for the purpose of financial data mining remains open. Presently, we are exploring its use in processing financial narratives.

The notion of *cache rot* was introduced to describe what happens to the cache as time marches on. The decay occurs because of symbol retirement, creation and reuse. The cache decay causes cache misses that harm performance. The question of how often the cache will need a rebuild remains open.

8. References

1. Email correspondence with Robert Bruce Carleton, of insidernewswire.com, July, 2007.

2 "Project Imperion: New Semantics, Facade and Command Design Patterns for Swing" by Douglas A. Lyon, Journal of Object Technology, vol. 3, no. 5, May-June 2004, pp. 51-64.

3 "The Imperion Threading System" by Douglas A. Lyon, Journal of Object Technology. vol. 3, no. 7, July-August 2004, pp. 57-70.

4 "Displaying Updated Stock Quotes", by Douglas A. Lyon, Journal of Object Technology, vol. 6. no. 8. September-October, 2007, pp. 19-31.

5 "Data Mining Historic Stock Quotes in Java", by Douglas A. Lyon, Journal of Object Technology, vol. 6. no. 8. November-December, 2007, pp. 17-23.

6 "Data Mining Address Book", by Douglas A. Lyon, Journal of Object Technology, vol. 7. no. 1. January-February, 2008, pp. 15-26.

7 Alex Wing On Wong "Colleague Discoverer", MS Thesis, The University of Strathclyde in Glasgow Strathmore University, 2004, http://www.cis.strath.ac.uk/~mdd/misc/cit/projects/library/04/Wong_A.pdfhttp://www.cis.strath.ac.uk/~mdd/misc/cit/projects/library/04/Wong_A.pdf

8. Gerry Grant and Sumali Conlon, "EDGAR Extraction Systems: An Automated Approach to Analyze Employee Stock Option Disclosures", Journal of Information Systems, vol. 20, no. 2, 2006, pp. 117-142.