Proceedings of _______:
Conference Name
Date and Location

Reference ID: 1071

EASE – A SOFTWARE AGENT THAT EXTRACTS FINANCIAL DATA FROM THE SEC'S EDGAR DATABASE

Özkan Cetinkaya, Detlef Seese, Ralf Spöth, Thomas Stümpert
Institute AIFB, University Karlsruhe (TH), 76128 Karlsruhe, Germany
{seese, stuempert}@aifb.uni-karlsruhe.de
(+49)721/608-0, Fax:(+49)721/693717

ABSTRACT

In this paper we discuss text mining approaches for financial data from the Electronic Data Gathering, Analysis and Retrieval (EDGAR) database of the Securities and Exchange Commission (SEC) which contains filings including financial statements of about 68,000 companies. The structure of these filings varies between companies, and changes over time for individual companies as well. Moreover, their technical specification is comparably weak. Altogether, this makes automated data extraction a great challenge for software agents.

The focus of this paper was the recognition of balance sheets, that is, how to find relevant sections in a large document. A filing consists of HTML or plain text. With respect to this distinction, we followed two different approaches for the respective types.

Regarding HTML encoded content the agent builds a DOM (Document Object Model) instance on top of non-standard filing, which allows very convenient data access. This DOM-based approach revealed additional potential for navigation in these filings in order to detect financial information faster and more reliably even when filings do not adhere to syntactical conventions strictly. For plain text, a modified vector space model has been developed. We succeeded to extract key financial information at a reasonably high level for conventional text files.

INTRODUCTION

Since financial statements are vital for decision makers in the professional investment world, quick access and automated import of this data is essential for the finance industry. Unfortunately, the data available in the SEC EDGAR database (see [SEC]) is weakly structured in technical terms. Filings contain mixed content of natural language text and semi-structured financial tables (for data extraction methods in general, see e.g. [HFAN98]).

The SEC is the regulatory authority for securities markets in the United States established for the protection of investors and to maintain fair, honest and efficient markets. Registered companies are required to submit certain financial reports in prescribing formats. Form 10-K is the annual report that most reporting companies file with the SEC, containing annual financial statements. Filings consist of plain text and/or HTML. As long as these filings are the standard and most up-to-date source of information for professional and private investors, software agents (see [Klu99]) that transform them into an exchangeable format are of greatest interest.

We continued prior explorations of data extraction using agents, formerly named Edgar2xml (see [LSSS01]), leading to two different approaches that take into account the changing document formats (see EDGAR Filer Manual, [SECFM]). Before we delve into the details of our EASE (Extraction Agent for SEC's EDGAR database) project, we take a look at existing implementations with similar objectives.

Currently several EDGAR agents exist, e.g. EDGAR online (see [FREDG] and [EDGOL]), 10-K Wizard (see [10KW], no free online services available) and FRAANK (Financial Reporting and Auditing Agent with Net Knowledge, see [FRAANK], cf. [KNSVL98], and cf. [KNSVL00]). EdgarScan (see [EDGSC]) from PriceWaterhouse Coopers pulls filings from the SEC's servers and parses them automatically to find key financial tables and normalize financial positions to a common set of items that is comparable across companies. The normalization makes EdgarScan superior compared to other implementations. When filings are available as HTML flavored only, EdgarScan obviously converts the HTML documents into plain text and parses these documents. The most important effort in standardizing exchange of financial information is the Extensible Business Reporting Language (XBRL, see [XBRL]), which is comprised of a set of XML schemas, describing how to present items, and taxonomies, describing which items make up a certain type of financial report. Traditional EDGAR agents do not always detect a balance sheet in a 10K filing correctly. We describe an algorithm which extracts key financial information at a high level for conventional text files.

The paper is organized as follows: The first section describes content and structure of SEC 10-K filings. In the following main part we describe our text mining approaches for extraction of financial data from these filings. It is divided into two subsections, one of which is dedicated to the traditional, textual filings, and the other for the newer HTML encoded documents. The final section contains results and conclusion.

STRUCTURE OF 10-K FILINGS

A filing is a text document that contains *tags* in order to structure the parts it contains. The specification for filings can be found in the filer manual (see [SECFM]). These tags are special to SEC filings, but similar to HTML tags in terms of syntactical rules. A filing consists of a single root element named "SEC-DOCUMENT", which contains a single child named "SEC-HEADER" and one or more children named "DOCUMENT".

Structure of document elements

The common structure of all embedded documents is a sequence of named properties followed by a single text node. An example:

```
<TYPE>10-K
<DESCRIPTION>ANNUAL REPORT
<TEXT>
    actual contents
</TEXT>
```

In this example, the first two lines specify two document properties "TYPE" and "DESCRIPTION". The text node encloses the actual contents which is an embedded document of type HTML, plain text with tags, or even some graphical type. Parsing filings in order to obtain the structure up to this level is comparably easy.

Embedded documents which contain financial information are nearly always either HTML or plain text, and sometimes a mixture. The format of these documents varies a lot between companies, and may change over time for a single company.

Embedded Plain Text

Plain text documents are best viewed with browsers that display the contents formatted with a monospaced font. Line and page breaks as well as the length of character runs including white space are significant for the visible structure of sections, paragraphs and tables. The following excerpt illustrates this.

```
           2001     2000
------------------------
<S>        <C>      <C>
ASSETS
Cash       27       85
```

Representing this example using a true type font, not respecting line breaks, produces something like

2001 2000 -------------------------------- <S> <C> <C>ASSETSCash 27 85

which clearly illustrates the importance of the properties stated above. That said, we could easily draw a table using pipe character immediately in front of the "smaller than" characters of the columns, and would obtain this visible result:

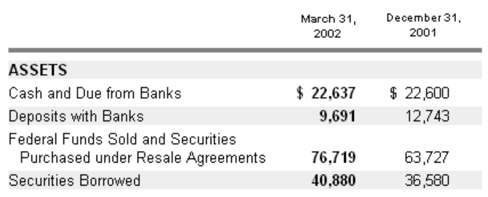
```
          |2001    |2000
----------|--------|----
ASSETS    |        |
Cash      |27      |85
```

We refer to documents with embedded plain text of this kind as text flavored documents hereinafter.

Embedded HTML

Embedded HTML documents are in no way special to SEC filings when compared to other real-world HTML content published on the web. They are enclosed in HTML tags, using "HEAD" and "BODY" tags, "TABLE" tags and others. These documents should be viewed with contemporary HTML browsers only. The following figure shows the beginning of a balance sheet.

The tabular structure is clearly visible. As in the textual example before, we can identify three



	March 31, 2002	December 31, 2001
ASSETS		
Cash and Due from Banks	$ 22,637	$ 22,600
Deposits with Banks	9,691	12,743
Federal Funds Sold and Securities Purchased under Resale Agreements	76,719	63,727
Securities Borrowed	40,880	36,580

Figure 1: Balance Sheet in HTML

columns with item names on the left and financial numbers in the second and third column. In contrast to the visible three columns, this document consists of as much as nine table columns in the HTML source, some of which are used for alignment and spacing purposes only. Formatting is targeted towards human readers, not machines. – We refer to documents with embedded HTML simply as HTML documents hereinafter.

PROCEDURES IN MORE DETAIL

Information Extraction (IE) is a technology dedicated to the extraction of structured information from texts to fill pre-defined templates. While most contemporary work focuses on either machine learning for IE patterns or wrapper generation (cf. [AS99], [HFAN98] and others), and is geared towards web search engines, we deal with a pre-defined structure for the information, namely the financial statements of US-GAAP compliant financial reports. The guidelines in the filer manual (cf. [SECFM]) specify what information must be present, but the technical specification is comparably weak.

Text Mining in Traditional Filings

Here is an example of how a balance sheet should appear in a filing (we call this well-formed or standard):

```
...
<PAGE>
...
<TABLE>
         CONSOLIDATED BALANCE SHEET
<CAPTION>
IN MILLIONS OF DOLLARS,
EXCEPT PER SHARE AMOUNTS

                         2001         2000
-------------------------------------------
<S>                      <C>          <C>
ASSETS
Cash                     27           85
Investments              0            1,487
...
Total Assets             9,456        9,154

LIABILITIES
Debts                    812          756
Treasury stock           (1.2)        (.8)
...
Total Liabilities        9,456        9,154
<FN>
notes to the consolidated balance sheet at
p 16
</TABLE>
...
```

This example illustrates a kind of canonical table in a traditional document. First of all, the document uses `page` tags to separate pages. Second, the table is enclosed in an opening and a closing `table` tag. Third, the table uses the caption to identify its purpose. Fourth, it uses a sequence of `S` and `C` tags which determines the column specification. Finally, the table entries are placed correctly within the text, that is, they are starting at the their respective horizontal position. Filings often deviate in a series of points from the example above, and some tags are frequently omitted. This makes it difficult to generalize the parsing process.

We introduce a modified version of the vector space model (cf. [Sal68]) for the purpose of the identification of financial tables. We will refer to the following three text excerpts from real-world filings in order to compare the results of the standard vector space model to our extended version. The second excerpt contains the balance sheet.

d_1:

```
<page>
The status of the pension plans follows.

<table>
                            Assets exceed
                              accumulated
                       benefit obligation
-------------------------------------------
                        1997          1996
-------------------------------------------
Assets, primarily
stocks and bonds at
market             $ 5,074.5     $ 4,327.6
...
```

d_2:

```
<page>
       Balance Sheet
<table>
                           1997        1996
-------------------------------------------
Assets
Current assets:
Cash and cash equivalents  800.8     598.1
...
```

d_3:

```
<table>
                     1998     1997     1996
                  -------  -------  -------
Assets:
   Current assets  $1,569   $1,949   $1,995
...
</table>

Balance Sheet from December 31, 1998 is on
page 12.
```

The standard vector space model creates a weighted mapping of terms to documents so that similarity of documents and search queries can be measured. We split a filing into segments, which we use as document equivalents. This is necessary since we are dealing with a single document, the parts of which need to be valuated. The query is defined as the search for the balance sheet, the formal details will be explained below. Aim of our modifications is to obtain a value which ranks the document segments in terms of likelihood for the contention of a balance sheet (or other financial table).

First, we define a set of search terms $T = (t_1, \ldots t_n)$, each of which is a literal term like "`balance sheet`" or "`assets`" or a tag like `<table>` or `<page>` etc. Document segments are defined as set $D = \{d_1, \ldots d_m\}$. Vector $q = (q_1,\ldots, q_n)'$ is the vector of query weights associated with the terms of T, and $w_j = (w_{j,1}, \ldots, w_{j,n})' \in R^n$ a binary vector with components $w_{j,k} = 1$ if term t_k exists in d_j, or 0 otherwise.

The similarity measure s_j for document d_j is defined as

$$s_j = \sum_{k=1}^{n} w_{j,k} q_k \ .$$

Higher values of similarity s_j indicate a higher probability that a segment contains the balance sheet. The weights used for the terms in this model are taken based on observations of dozens of tests. Examples with an arbitrary sample of weights are shown below:

Query	Term	Weight
q	`<page>`	10
	`<table>`	15
	balance sheet	55
	Assets	20

Figure 2: Example for term weights for the query

Calculating the similarity measure, we get:

$s_1 = 1\cdot10 + 1\cdot15 + 0\cdot55 + 1\cdot20 = 45$
$s_2 = 1\cdot10 + 1\cdot15 + 1\cdot55 + 0\cdot20 = 80$
$s_3 = 0\cdot10 + 1\cdot15 + 1\cdot55 + 1\cdot20 = 90$

The highest similarity d_3 does not lead to a successful detection of the balance sheet. We observed that the order of the occurrence of the individual terms matters. To take the order into account we replace the vectors of the equation with matrices, where the components of the matrix reflects the order of the terms. This leads to the following modification:

Be Q an $n{\times}n$ term-term-matrix:

$$\begin{pmatrix} q_{1,1} & \cdots & q_{1,n} \\ \vdots & \ddots & \vdots \\ q_{n,1} & \cdots & q_{n,n} \end{pmatrix}$$

where $q_{i,k}$ denotes the weight of subsequent occurrences of term t_i and t_k. W_j denotes the matrix of occurrences in document j,

$$W_j = \begin{pmatrix} w_{1,1} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{n,1} & \cdots & w_{n,n} \end{pmatrix} \in \mathrm{R}^{\mathrm{nxn}},$$

where $w_{i,k}$ equals 1 if term t_i precedes t_k, or 0 otherwise.

As similarity measure for d_j we use the sum of weighted occurrences, i.e.

$$s_j = \sum_{i=1}^{n} \sum_{k=1}^{n} w_{j,i,k} q_{i,k} \ .$$

A high value for the similarity measure again represents a high likelihood for the occurrence of a balance sheet. We apply the modified model to the document segments above (see below).

Q	`<page>`	`<table>`	balance sheet	assets
`<page>`		25	65	30
`<table>`			70	35
balance sheet		70		75
assets				

W1	`<page>`	`<table>`	balance sheet	assets
`<page>`		1		
`<table>`				1
balance sheet				
assets				
W2	**`<page>`**	**`<table>`**	**balance sheet**	**assets**
`<page>`		1		
`<table>`			1	
balance sheet				
assets				
W3	**`<page>`**	**`<table>`**	**balance sheet**	**assets**
`<page>`				
`<table>`			1	
balance sheet				
assets			1	

We get a similarity measure of $s_1 = 60$ for segment d_1, $s_2 = 95$ for segment d_2, and $s_3 = 70$ for segment d_3. As stated before, the valuation performs better in terms of identification. As you can see in section Results, observations indicate a high degree of suitability for the identification of document segments.

The extraction algorithm now tries to capture individual items by their names, and by splitting rows into columns based on both column specification, number of consecutive separating white space characters, and appearance of isolated numbers. First we identify the relevant table and determine the start of the table. Next we extract multiplier and currency of the numbers. After that we start extracting the data for the different item labels. The extraction of the details is beyond the scope of this document.

Extraction of HTML Documents

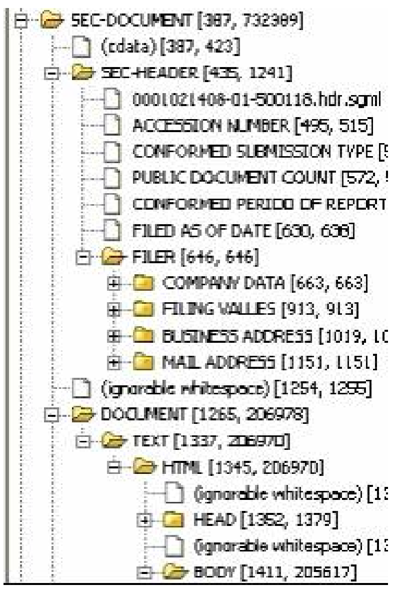
Since SEC filings consists of HTML parts and other predefined tags mixed up with plain text, the filing is not conform to the XML specification and for the extraction of HTML flavored documents requires the use of absolute HTML paths that point to the data item to be extracted. The extraction process of HTML flavored documents is two fold: In the first stage, a *parser* builds a document tree based with an HTML DOM root node for each embedded HTML document. This structure is used by the *extractor* in the second stage to locate financial statements and extract them into the target object.

Input for the document builder is the raw character sequence containing the entire document. Output is an object which contains a reference to the raw text and a root node, which represents the root of the document hierarchy.

As stated before, a filing consists of multiple parts with varying syntax. The main parser splits the document into a header and a number of documents first, and delegates the actual parsing to specialized parsers for headers and documents, a header parser and a document parser, respectively. The header and each document are added as nodes to the root node.

Splitting the entire document into its constituent parts is realized with simple regular expressions, searching the respective opening and closing tags of "SEC-HEADER" and "DOCUMENT" in a single call.

The header can be split into lines, and these lines can be split into a property name and its value simply finding the first occurrence of a colon. Each property is added as a single node to the document tree. The document parser uses a single regular



```
SEC-DOCUMENT [387, 732389]
  (cdata) [387, 423]
  SEC-HEADER [435, 1241]
    0001021408-01-500118.hdr.sgml
    ACCESSION NUMBER [495, 515]
    CONFORMED SUBMISSION TYPE [5
    PUBLIC DOCUMENT COUNT [572, !
    CONFORMED PERIOD OF REPORT
    FILED AS OF DATE [630, 638]
    FILER [646, 646]
      COMPANY DATA [663, 663]
      FILING VALUES [913, 913]
      BUSINESS ADDRESS [1019, 10
      MAIL ADDRESS [1151, 1151]
  (ignorable whitespace) [1254, 1255]
  DOCUMENT [1265, 206978]
    TEXT [1337, 206970]
      HTML [1345, 206970]
        (ignorable whitespace) [13
        HEAD [1352, 1379]
        (ignorable whitespace) [13
        BODY [1411, 205617]
```

Figure 4: Screen of DOM Tree

expression in order to split the document contents into the leading property values and the final text element. The former are put as properties to the document node, and the text node is added as the only child to the document node. The content of this text element is then passed on to an implementation of an HTML parser the details of which need to be explained in more detail. That parser adds the document as child named "HTML" to the text node. The resulting document tree can be viewed with a special end-user application implemented for demonstration purposes (see screenshot in figure 4).

Navigation through the entire document is possible using the methods which the node implementation allows for, among them searching by content and/or attribute values, and addressing nodes using path expressions. The address of the "CONFORMED SUBMISSION TYPE" for example would be "/SEC-DOCUMENT/SEC-HEADER/CONFORMED SUBMISSION TYPE". Moreover, the entire document can be processed in consistent way regardless of the input type.

The main challenge for the HTML parser stems from the fact that most real world HTML documents do not follow the W3C recommendations strictly. In contrast to XML, which enforces strict adherence to standards, HTML documents have always been quite sloppy in terms of standardization. Web browsers have been very tolerant towards incorrect syntax since their invention, which lead to the proliferation of HTML editors that produce code not in compliance with recommendations.

From a theoretical point of view, XML is a type-2 grammar in Chomsky's hierarchy. As such, it requires a finite state engine with a stack and thus exceeds the capabilities that pure regular expressions provide. HTML is only a special incorporation of XML or SGML when perfectly in compliance with HTML recommendations. In practice, most HTML documents violate the recommendations frequently. Capturing the document tree reliably despite potential violations almost certainly requires a Turing-complete engine. And in fact, writing such a parser proved to be challenging.

The basis for the HTML parser is a skeletal XML parser with special precautions for various expected and unexpected violations. The parser contains two regular expressions. The first expression is used to find comments, opening tags, closing tags, and character data tags in the parsers main loop. The second is used to parse attributes in opening tags.

The parser keeps a reference to the node which has been opened most recently. Every time an opening tag is detected, a new node is created. This node will become the current top node. When a closing tag is detected, the current top node is closed, and its parent is made the current top.

The reference to the current top node resembles the stack required for type-2 parsers. The current top node is the top element of the stack. Creating a new node and making it the current top node resembles the "push" operation of a stack, and making the current top nodes' parent the current top node resembles the "pop" operation of a stack.

As stated before, this approach must fail for real-world HTML documents. First of all, the broken tags (opening tags without a closing pendant) must not become the top node. Second, in some cases closing tags are omitted. An opening tag then forces previously found opening tags to be closed. Third, a closing tag may be misplaced at some later position in the document, in which case it will be ignored.

The extraction process is similar to the concept of *XPath*, which describes how to address a node in an XML document. For example, in order to find out the type of a filing (10-K or 10-Q), the extractor searches for the header node, and in this the node named *submission type*. The following code illustrates this:

```
Node root; // given
Node header = root.find("SEC-HEADER");

Node submissionType =
  header.find("CONFORMED SUBMISSION TYPE");
String typeName = submissionType.getText();
```

The path and search expressions vary from company to company, while there is still a set of default or standard methods of how to identify a certain node. The extractor looks into its configuration database whether it finds a specialized path for a given item and company, and defaults to a generic implementation if not available. This procedure makes it possible to extend the range of covered companies by simply adding specialized entries to the configuration database for that particular company. Our implementation differs from XPath in that it supports regular expressions, a feature which is scheduled for a future version, not earlier than version 2.0 accompanied with XSLT 2.0.

CONCLUSION

Basis for the following observations was a sample of 10-K filings for the 30 Dow Jones Industrial Average companies for at least the last 3 years. Performance benchmark was the former Edgar2xml (see [LSSS01]) implementation, which used about

	YEAR							
	1996	**1997**	**1998**	**1999**	**2000**	**2001**	**2002**	**2003**
CIK								
1001039	ok	ok	ok	ok	ok	ok		
831001	ok	ok	ok	ok	ok	ok	ok	ok
789019	ok	ok	ok	ok	ok	ok		
773840	ok	ok	ok	ok	ok	ok	ok	ok
764180	ok	ok	ok	ok	ok	ok	ok	
732717	X	ok	ok	ok	ok	ok		
354950	ok	ok	ok	ok	ok	ok	ok	ok
200406	ok	ok	ok	ok	ok	ok	ok	ok
104169	ok	ok	ok	ok				
101829	ok	ok	ok	ok	ok	ok		
80424	ok	ok	ok	ok	ok	ok	ok	
66740	ok	ok	ok	ok	ok	ok	ok	ok
64978	N	ok	ok	ok	ok	ok	ok	
63908	ok	ok	ok	ok	ok		ok	
51434	ok	ok	ok	ok	ok	ok	ok	ok
51143	ok	ok	ok	ok	ok	ok	ok	
50863	ok	ok	ok	ok	ok			
47217	ok	ok	ok	ok	ok	ok	ok	
40730	X	ok	ok	X	ok	ok	ok	ok
40545	ok	ok	ok	ok	X	ok	ok	ok
34088	ok	ok	ok	ok	ok	ok	ok	
31235	ok	ok	ok	ok	ok	ok	ok	ok
30554	ok	ok	ok	ok	ok	ok	ok	ok
21344	ok	ok	ok	ok	ok	ok	ok	ok
19617	ok	ok	ok	ok	ok	ok		
18230	ok	X	ok	ok				
12927	ok	ok	ok	ok	ok	ok	ok	
5907	ok	ok	ok	N	N	ok	ok	ok
4962	ok	ok	ok	ok	ok	ok	ok	ok
4281	ok	ok	ok	ok	ok	ok	ok	

Table 1: Overview of investigated balance sheets

X = balance sheet not found
N = no balance sheet in filing

2 minutes to extract financial data from comparable filings. This package utilized the GNU regular expression implementation.The DOM parser was able to create the DOM instance for all filings in the sample which contained embedded HTML.

Addressing individual items in financial statements needs a lot of configuration work still to be done, yet the results are promising since extraction works incredibly fast and reliably. Experiments show that items can be extracted provided that a proper configuration has been defined. The parsing process took only about 1.5 seconds for an average 1 MB filing on a 1.5 GHz single processor machine with 512 MB of RAM. Navigation within the document tree proved to be very fast.

The plain text parser recognizes 201 balance sheets of 206 text-based filings of the sample (see table below). Processing took about 0.5 seconds per filing with an average size of 400 kB.

The filing for CIK 732717 in 1996 contained no tags at all. The occurrence of consolidated balance sheets for CIK 40730 included those of subsidiaries, which confused the agent. The filing for CIK 40545 for year 2000 contained a mix of HTML and plain text formatting means, which the agent cannot deal with. For CIK 18230 in year 1997, the agent found too similar patterns in other segments of the filing.

REFERENCES

[10KW] http://www.tenkwizard.com, last visited at Oct 8th, 2003

[AS99] Azavant, F., Sahuguet, A. 1999, "Web Ecology: Recycling HTML pages as XML documents using W4F", in *WebDB'99.*

[EDGOL] http://www.edgar-online.com, last visited at Oct 8th, 2003

[EDGSC] http://edgarscan.pwcglobal.com, last visited at Oct 8th, 2003

[Fri02] Jeffrey E.F. Friedl, "Mastering Regular Expressions", 2nd Edition, O'Reilly, pp. 365

[FRAANK] http://fraank.eycarat.ukans.edu, last visited at Oct 8th, 2003

[FREDG] http://www.freeedgar.com, last visited at Oct 8th, 2003

[HFAN98] Huck, G., Fankhauser, P., Aberer, K., Neuhold, E. 1998. "JEDI: Extracting and Synthesizing Information from the Web.", in Michael Halper (Ed.), *Proc. 3rd IFCIS Intl. Conf. on Cooperative Information Systems*, Los Alamitos, California, IEEE Computer Society Press, pp. 32-43

[Klu99] Klusch, M. 1999, "Intelligent Information Agents - Agent-Based Information Discovery and Management on the Internet", Berlin-Heidelberg, Springer.

[KNSVL00] Kogan, A., Nelson, K., Srivastava, R., Vasarhelyi, M., Lu, H. 2000. "Virtual Auditing Agents: The EDGAR Agent Challenge." in *Decision Support Systems,* Vol 28 (3), pp. 241-253.

[KNSVL98] Kogan, A., Nelson, K., Srivastava, R., Vasarhelyi, M., Lu, H. 1998. "FRAANK: Financial Reporting and Auditing Agent with Net Knowledge." In *Collected Abstracts of the American Accounting Association Annual Meeting.*

[LSSS01] Leinemann, C., Schlottmann, F., Seese, D., Stuempert, T., 2001, "Automatic Extraction and Analysis of Financial Data from the EDGAR database" in *South African Journal of Information Management*, Vol 3 No. 2.

[Sal88] Salton, G. 1988., "Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.", Addison Wesley, pp 313.

[SEC] http://www.sec.gov, last visited at Oct 8th, 2003

[SECFM] http://www.sec.gov/info/ edgar/filermanual.htm, last visited at Oct 8th, 2003

[Sko91] Skousen, F. 1991, "An introduction to the SEC", Cincinnati: South-Western Publ.

[XBRL] http://www.xbrl.org, last visited at Oct 8th, 2003