AN AUTOMATIC NEWS ARTICLE FILTERING ENGINE

By

Hong Wan

B.S., Beijing Institute of Information Technology, 1991

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Computer Science

Acadia Univeristy
Fall Convocation 1998

0-612-33832-0

Table of Contents

List of Tables

List of Figures

Abstract

With the rapidly growing amount of information on the internet, many readers face the increasingly serious problem of overwhelming incoming news data (text, photos, and video). The ability to filter away irrelevant information is becoming critical to a news delivery system.

This thesis introduces a new methodology which binds a news representation object to a news document. When users perform a search to retrieve news documents, they can acquire similar ones from other resources. First this thesis presents an algorithm for creating a representation of news object. Based on the analysis of actual collections of newspaper articles, a news representation object was created which captured the regularities of the news documents. Four types of descriptive features can be extracted from news documents. They are: Person, Event Data, Event Location and Organization. Second this thesis presents an algorithm to calculate the similarity between two news objects. We assume that two documents are related if their objects overlap.

Results indicate that the representation of news objects can be used to quickly sieve through news documents for meaningful information. The algorithms for calculating a relationship between two news objects can be used as a fast way to filter away irrelevant news documents.

Acknowledgments

I would like to take this opportunity to thank all those who have contributed to this thesis, directly or indirectly.

I would first like to express my sincere gratitude to my supervisor Dr. Carolyn Watters for her constructive criticism, guidance, support and for sharing her immense wealth of knowledge, all of which have had a strong influence on this thesis. I'd like to thank Dr. Robert Korfhage , Dr. Ke. Qiu, Dr. André Trudel and Dr. Craig Bennett for providing comments on this thesis.

I want to thank my parents Zehua Jia and Qing Wan for helping me to get this far. I could never have made it without them.

Chapter 1
Introduction

In this chapter, we discuss the purpose and goal of this thesis and give the outline of this thesis. This thesis presents a new filtering engine which binds a news representation object to a news document. When users perform a search to retrieve news documents, they can acquire similar ones from other resources. This will enhance the functionality of personalized newspaper projects.

1.1 Purpose and Goal of this Thesis

This thesis presents a system that can rank a news document using a set of scalable, adaptable, and dynamic phrases. It associates each news document with a news representation object. It uses this news representation object to find related documents.

This thesis focuses on solving the following problems that readers may face in personalized newspaper systems.

1. Precision: The system selects the documents deemed to be related to the customer and eliminates the rest. The proportion of irrelevant documents delivered to the user should be as low as possible.

2. Dynamic: The system should be capable of exploring new additional information when it becomes available on a specific event topic. This will allow users to see updated reports of an event.

3. Unique: The system should be able to identify redundant information. The same event is usually reported by different news sources. The system should be able to eliminate redundant ones.

We developed a three-step process to address these problems:

1. Select appropriate parameters to describe a news class.

2. Create a news object associated with a news document.

3. Evaluate the similarity between two news objects.

"Who, where, when, what, why and how" are basic tasks of reporters in describing events. We use FullName and Organization, EventLocation and EventDate feature name-phrases to describe three components of an event (who, where and when). So members of the news class are a set of FullName name-phrases and Organization name-phrases, EventLocation name-phrases and EventDate name-phrases (see Chapter 3). An algorithm is presented which extracts important features from news documents and groups features into different classes.

Features are defined by name-phrase, where a *name-phrase* is a phrase which can present a meaning unit, such as the name of a person or the name of a location.

News objects represented by the news class contain information extracted from original news articles. We can then calculate the similarity between two news objects (see Chapter 3). We assume that two documents are related if their objects overlap.

1.2 Overview of this thesis

Chapter 1 states the problems in news document-filtering and then gives the outline of this thesis.

Chapter 2 provides a review of research into news delivery.

Chapter 3 focuses on the methodology. This includes the construction of the news representation object. It also explains the formulas used to calculate the similarity between two news objects.

Chapter 4 presents the experimental performances and evaluates the methodology described in Chapter 3.

Chapter 5 describes the general architecture of "News Document-Filtering Agent". This includes the user interface elements, such as dialog windows, HTML layouts, etc. It also describes the modules which the agent uses to manage the searches for the user.

Chapter 6 offers the conclusion and directions for future research.

Chapter 2 Background

In this chapter we present an overview of previous work related to news delivery services and relevant information retrieval methods. First we review the research in the news domain. Second we introduce the World-Wide-Web (WWW) and related tools which help the user locate interesting pages in the WWW. Then we introduce the use of agents which facilitate the user's search for interesting information in the WWW. Finally we introduce some information retrieval and information filtering methods.

2.1 News Research

News is information about recent events of general interest, especially as currently reported by newspapers, periodicals, radio, or television [1]. Gigantic archives of news data (text, photos, and video) in digital format are generated from different news resources every day. *News data* is often used in text analysis and querying research largely because of its general interest and copious supply [2]. Since a few years ago, a number of research groups have done much related work in designing electronic news service systems or addressing specific news delivery problems.

The electronic news delivery project [3] at Dalhousie, Acadia, and Waterloo Universities developed an overall architecture integrating text, photographs, video and audio into personalized multimedia news presentations. The architecture has three layers. The first is News Resource Layer. The multiple news resources are stored in this layer. The second is News Management Layer. It provided the query engine to link related resources together. The last is News Reader Layer. It described how the news resources are presented to the user. Based on this project, a number of research results are published. They are involved in the variety of news delivery domains. Carrick and Watters [2] presented an algorithm to efficiently link a story and its related photos together. Watters [4] described the metaphor for the presentation of electronic news.

Some experts worked on other news delivery problem domains. Newhagen [5] presented the relationship between feedback and news from sociological view. Janne, Marko and Tuomas [6] presented a logical structure of a hypermedia newspaper to locate news information efficiently and get more controls for the news resource. Kenrick and Rao [7] described the INFOS (Intelligent News Fitering Organizational System) which aims to reduce a user's search burden while browsing a large number of messages.

Recent research efforts on news retrieval effectiveness and efficiency include statistical methods, natural language processing (NLP) based approaches and even artificial intelligence approaches. Yan and Garcia-Molina point out [8], "In a large-scale wide-area system where the number of information providers and seekers are large, efficiency in the dissemination process is an important issue and must be addressed."

Rau described a method that focused on extracting names from free text and considered two parameters of information retrieval, effectiveness and efficiency as well. She said:

"Current methods generally start by identifying key artifacts in the text, such as proper names, dates, times, and locations, and then use a combination of linguistic constraints and domain knowledge to identify the important content of each relevant text. For example, in news stories about joint ventures, a system can usually identify joint venture partners by locating names of companies, finding linguistic relations between company names and words that describe business tie-ups, and using certain domain knowledge, such as understanding that ventures generally involve at least two partners and result in the formation of a new company."[9].

Rau [10] presented a detailed description of the algorithm that extracts company names automatically from financial news. She said: " Extracting company names from text is one problem; recognizing subsequent references to a company is another." She addressed both problems in an implemented, well-tested module that operates as a detachable process from a set of natural language processing tools. Her algorithm combines heuristics, exception lists and extensive corpus analysis. The algorithm generates the most likely variations that those names may go by, for use in subsequent retrieval. Tested on over one million words of naturally occurring financial news, the system has extracted thousands of company names with over 95% accuracy compared to a human., and succeeded in extracting 25% more companies than were indexed by a human.

The ANES (Automatic News Extraction System) [11] system constructed by GE Research and Development was compared to the Searchable Lead system developed by Mead Data Central, in the largest evaluation of automatic summarization undertaken to date. The ANES approach utilized a combination of statistical and heuristic techniques to determine key sentences for extraction and inclusion in summaries. Statistics based on the relative frequency of terms in a document as compared to the corpus as a whole determined topical words. The sentences containing these words were selected using constraints that incorporated preferences such as location within a document and the presence of anaphoric references that could interfere with the readability of the final summary. The Searchable Lead [12] technology chooses the first sentences in the news story for inclusion up to the target length for the summary. Three lengths of articles were evaluated for 250 documents by both systems, totaling 1,500 suitability judgements in all. The results of this evaluation were totally unexpected. The lead-based summaries outperformed the "intelligent" summaries significantly, achieving acceptability ratings of just over 90%, compared to the 84% acceptability of the ANES system.

Some specialists worked in very constrained domains to programs that can perform useful information extraction from a very broad range of text.

The SCISOR (System for Conceptual Information Summarization, Organization, and Retrieval) [13] is a prototype system that performs text analysis and question answering in constrained domains. Developed over the last four years, SCISOR operates on financial news, selecting and analyzing stories about corporate mergers and acquisitions from an on-line financial service. It uses both bottom-up and top-down processing to analyze and summarize financial news articles. *Bottom-up* analysis starts with a parse of each sentence, identifying linguistic structures and mapping these linguistic structures into a conceptual framework. *Top-down* analysis starts with conceptual expectations, such as the knowledge that takeovers involve two companies, and tries to fill these expectations given partial information from the text. It utilizes lexical, syntactic, semantic, and domain-specific knowledge in a coherent manner. SCISOR is a robust system that can process approximately six stories a minute. The program has been tested on 729 stories taken directly from the newswire source, and achieved an averaged recall and precision of slightly more than 90% in the determination of which stories were about mergers and acquisitions (69 stories). Key features have been extracted with an accuracy of 80-90%.

C. Carrick. and C. Watters [2] presented an approach that automatically associated text and photo news items into personalized multimedia news presentations. A frame representation is associated with a news story. They use a method that extracts the capitalized words from text and parses them into the different slots. Based on the equation for document similarity used for the vector space model, they present an approach to calculate the similarity between news story and photos. It can be used to automatically group the same issues with different media types together. For example, the method can be used to generate a link between a photo and text about the same story. An averaged recall is more than 75%. An averaged precision is more than 57%.

2.2 The World-Wide Web

The *World-Wide Web* (WWW) was designed to share information on the Internet. Berners-Lee created the WWW at CERN in 1989. The architecture of the WWW combines simple

HyperText document links with the Client-Server principle. *HTML* (HyperText Markup Language) is used to format documents and generate HyperText links. *CGI* (Common Gateway Interface) is a way of using a programming or scripting language on a server, to respond to requests from a web client. In other words, a CGI script is called from a web client, the script is executed by the web server, and the script returns HTML to the web client as the output of its execution. Java, Perl, Javascript and Vbscript are usually used as CGL script languages to turn any request from the WWW client into a GUI (Graphic User Interface). A Web GUI browser, such as Mosaic, Netscape and Microsoft Explorer, can become an interface to server resources on any network.

Because readers now face seriously increasing amounts of information at the WWW sites, some WWW search engines have been produced to help the user navigate interesting pages in the WWW, like "Yahoo!" and "AltaVista". *Term Weight models* are used as IR (information retrieval) engines. Generally, each term in a document is weighted differently according to its likely importance, and the document is retrieved according to the sum of all the weights. The three sources of weighting data used are as follows [14]:

1. Collection Frequency – assign more important value when the terms occur in only a few documents

2. Term Frequency – assign more important value when the terms more frequently appear in a document

3. Document Length – assign more important value for the term in the short document when it occurs the same number of times as in a long one.

However, these new tools have one important missing element. They lack a mechanism for continuously filtering and informing the user of new information [8]. For example, news is a kind of dynamic information. News posted on the Internet has attracted much attention from millions of Internet users because it is of general interest and copious supply. The statistics show that there are now 31.3 million adult online users in the United States and most people go online for news (66%) [15]. At the same time its megabytes of daily traffic creates an information overload.

2.3 Agents

The term *agent* [16] is usually defined as "A computing entity (piece of software) that performs user delegated tasks autonomously." The technology of agents focuses on [16]:

1. Putting intelligence into user interfaces to enable unskilled users to get more out of computing applications.

2. Personalizing applications and services to meet users' preferences, goals, and desires.

3. Managing the retrieval, dissemination, and filtering of the vast amounts of information available on enterprise networks, value-added networks, and especially the Internet.

4. Enabling electronic commerce in various forms.

5. Managing flexible manufacturing cells (robotics).

Information-Filtering Agents are agents that return the best "match" to a user using different information sources. Because an information-filtering agent may find the content instead of only finding Web site, this characteristic is useful for the user who is interested in a specific news data domain. There are several commercial products on the web that personalize newspapers including:

1. NewsHound [17] is a personalized newspaper that searches the articles in the San Jose Mercury News as well as several other newspapers to find the articles that match a user's profile. Verity® information applications [18] implemented by Verity, Inc. are used to catalogue news resources. With the Verity Query Language as a tool, users can express with precision a focussed area of interest so that they can access the information they need. The news resource retrieval is based on concept-based retrieval technology that allows people to ask questions and find items on the right concept, thus going beyond keyword systems.

2. The Krakatoa Chronicle [19] is an experimental system which implements an interactive, personalized newspaper on the WWW. The architecture differs from conventional Web-based newspapers in that it applies a flexible layout control mechanism. This allows users to only receive articles based on their interests. Its indexing engine is based on the SMART system [20] that is used to convert articles into document vectors and compute the weight of each term. The batch process to index all 100 text files takes about half an hour. The size of each text file ranges from 10-200 Kbytes. This system also uses other methods to build the indexing or a user profile, such as adding a score to each article or selecting explicit keywords.

These kinds of agents all have an indexing engine that binds keywords to each article. The agent returns related information based on a user profile and the index database. In an information-filtering system, the user expresses his interests in a number of long-term, continuously evaluated queries, called a *profile* [8]. The user then passively receives documents filtered according to the profiles. Such a service will become increasingly important and form an indispensable tool for the dynamic environment of wide-area information systems [8].

2.4 Information Retrieval and Information Filtering

The Internet is one of the largest publicly available information resources. Totally chaotic, random and unstructured information is on web pages. Methods of filtering effectiveness are attracting more attention in Internet developments. Information retrieval and information filtering technologies present different approaches.

Information retrieval (IR) is concerned with the representation, storage, organization and accessing of information items. In reference [14], Salton describes the use of statistical schemes such as probabilistic and vector space models for document representation and retrieval. The use of probabilistic methods goes back as far as the early sixties but the trend of using probability model for information retrieval has recently decreased [21].

The *vector space model* means that a document is represented by a vector space of keywords. The vector space method is experimentally tested in the SMART system [20]. This system supports the methods of the term-frequency/inverse-document frequency and relevance feedback of the user. *Term-frequency* is the frequency of constructs (words, phrases, word groups) in a document [14] *Inverse-document frequency* is the number of documents in a collection in which the term occurs divided by the number of documents in the collection[14]. The SMART system uses these calculations to assign an importance value of terms. Hence it provides the possibility to return documents containing the terms with high importance values. *Relevance feedback* means that the user is able to classify the retrieved documents as to whether they are relevant or not. These classified documents are then examined and extracted information is used to improve the original query.

A number of commercial filtering agents are done using the SMART system, such as Lira [22]. Lira locates Web pages with the help of existing index pages and search engines. These pages are analyzed further with methods of IR for improving the retrieved result. It uses a term-frequency/inverse-document frequency weighting to extract terms from Web pages.

The semantic approach focuses on natural language processing (NLP). It attracts non-professional searchers for two reasons. First, it can handle full-text or unstructured text. Second, it allows computers to better retrieve relevant documents, and to understand natural language questions [23]. There are a number of such systems in commercial use. Much of the recent progress in this area has come from U.S. government-sponsored programs and evaluation conferences, including the TIPSTER Text Program, TREC and the MUC.

One group that collaborated on the TIPSTER text program from the University of Massachusetts at Amherst experimented with expansion of their state-of-the-art INQUERY retrieval system so that it was able to handle the 3 gigabyte test collection [24]. This included research in the use of query structures, document structures, and extensive experimentation in the use of phrases. In general, the use of phrases as opposed to the use of single terms for retrieval did not significantly improve performance, although the use of noun phrases to expand a query shows much more promise. This group has found phrases to be useful in retrieval for smaller collections, or for collections in a narrow domain. The TIPSTER project has progressed to a second phase that allows standardized communication between document retrieval modules (usually statistically based) and natural language processing modules (usually linguistically based).

In the series of Text REtrieval Conferences (TREC) [25], some research groups representing very diverse approaches used text from lexicons, dictionaries, thesauri and databases as sources for the IR NLP systems. For example, the group in New York University discovered relationships between key words using traditional statistical methods. They then useed these terms to expand or modify the queries. Another group using natural language processing was the group from General Electric Research and Development Center. They used natural language processing techniques to extract information from the training texts.

The Message Understanding Conference (MUC)[26] is a gathering of researchers in natural language processing. Conference participants must develop NLP systems that perform a variety of information extraction tasks. Each system's performance is evaluated by comparing its output with the output of human linguists. Two important metrics for assessing the performance of an NLP system are recall and precision. *Recall* is defined as the number of relevant documents retrieved in response to a query divided by the total number of relevant documents in the collection. *Precision* is defined as the number of relevant documents retrieved in response to a query divided by the total number of retrieved documents. There are four different task-oriented evaluations. First is the *Named Entity task* which consists of three subtasks (entity names, temporal expressions, number expression). The expressions to be annotated are "unique identifiers" of entities (organizations, persons, locations), times (dates, times), and quantities (monetary values, percentages). According to the results shown in MUC-6, the majority of systems evaluated on Named Entity had recall and precision over 90%, which was judged to be comparable to human performance on the task. The second are *Conference tasks* that recognize alternative ways of identifying an entity tagged by Standard Generalized Markup Language (SGML) annotations. The third is the *Template Element task* that designs the composite of elements for a specific data domain. The highest-scoring system had a recall of 70% and a precision of 80%. The final task is the *Scenario Template* that concerns changes in corporate executive management personnel; the extracted information includes answers to the basic questions of "Who is creating or filling what vacancy at what organization?". The top scoring system has 40%-50% recall, and 60%-70% precision.

Information filtering (IF) technology is becoming popular because it focuses on analyzing dynamic information rather than the static data used in IR. A filtering system uses profiles to represent the user interests after repeated interactions with the user. Some learning processes are necessary. *Learning* means that the system uses feedback information to improve the precision of the system. *Neural networks* and *Genetic Algorithms* (GA) are two prominent methods. The first one focuses on seeing how important various keywords or combinations are in connection with a particular person or information need after training. The second one focuses on finding a better query based on information from various other queries and retrieved documents. To a lesser extent, these technologies are used in artificial intelligence research.

Maillist and the USENET news system are very similar kinds of information filtering services that are available on the Internet. The user subscribes to a Maillist server with one or more topics of interest and receives messages via mail. The USENET news system is an electronic bulletin board system on the Internet. These two systems have been successfully used by millions of users, but often create information overload [8]. The coarse classification of topics leads to some relevant articles posted to different Maillist and USENET news groups and some irrelevant ones in the same Maillist and USENET news groups. This will cause users to miss interesting information if they don not subscribe to lists properly.

Chapter 3
Methodology

This chapter is divided into two parts. In the first part, a representation of the news document is shown. In the second part the algorithm to calculate the similarity between two documents is presented.

The data domain of this thesis is news documents. News is information about recent events of general interest, especially as reported by newspapers, magazines, radio, or television [3]. The characteristics of a news document include short lifetime, large number, widespread content, overlapping information and non-archival use. First, the life of a news document is very short. News documents are updated at least once a day. Second, everyday thousands of news sources around the world produce gigabytes of news data. Third, the content of news documents touches upon wide-ranging topics, such as politics, sport, and entertainment. Fourth, different media sources write their own reports about the same event, so overlapping information is available for the reader. Finally, the reader is more interested in current events than past ones.

The main objective of this thesis is to simplify the process for a user to find related documents for a given news story. The algorithms described in this chapter are focused on creating a news representation object. These objects are then used by a news document-filtering engine for selecting related items.

First we define a news representation class. Then some rules are given that are used to create a news representation object. Exception handling is also considered to improve the accuracy of extracting name-phrase, where *name-phrase* is a meaning unit. It is composed of a set of related capitalized words. Finally we introduce how to calculate the similarity between two news representation objects.

3.1 Definition of News Representation Class

Blair [27] suggests that a retrieval model must have better document content representations:

"The central problem of Information Retrieval is how to represent documents for retrieval. The most intricate or carefully designed retrieval algorithm cannot compensate for inappropriately represented documents. ... The central task of Information Retrieval research is to understand how documents should be represented for effective retrieval. This is primarily a problem of language and meaning."

News readers can ask six questions about an event [28]:

1. Who – It answers a reader's questions of who did what, or to whom something happened. It may be a single person, a group of people or an organization.

2. What – It tells briefly what happened at a planned or unplanned event.

3. When – It refers to the time or date of an event.

4. Where – It gives the location of the news event.

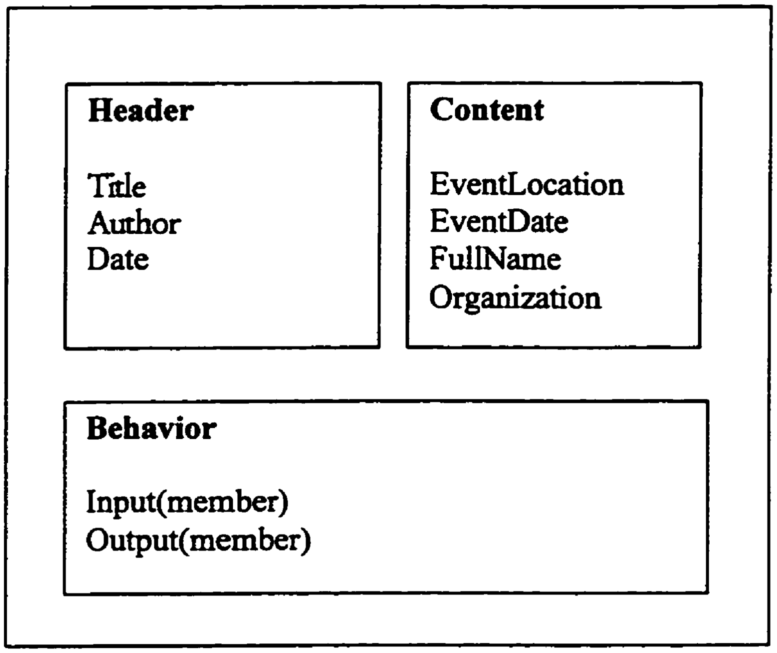
5. Why – It concerns the cause of the event and is frequently implied or unavailable.

6. How – It relates the circumstances or manner in which something is accomplished in the story.

The *ideal* news document representation should answer the above six questions for an event. Four types of descriptive features can be extracted relatively easily from news documents, *FullName*, *EventDate*, *EventLocation* and *Organization*. The features, FullName and Organization, will provide answers for question 1. The feature EventDate will provide answers for question 3. The feature EventLocation will provide answers for question 4. The algorithms needed to capture the regularities for the above four features are reasonably simple and can be encoded as a computer simulation program. On the contrary, the algorithms needed to capture the regularities for answering questions 2, 5 and 6 are complex and computationally intensive. They must mimic human logic and thinking. Currently no computer program can even produce a general summary, one that contains answers satisfactory to everyone [29]. Consequently algorithms for questions 2, 5 and 6 are not considered in modeling a news representation in our study. Since the main objective of this thesis is to simplify the retrieval procedure while maintaining a high retrieval efficiency, we do not investigate the amount of computing needed to actually understand the article. Secondly, currently text understanding is possible but only within a specific application domain [30]. News data covers very wide domains.

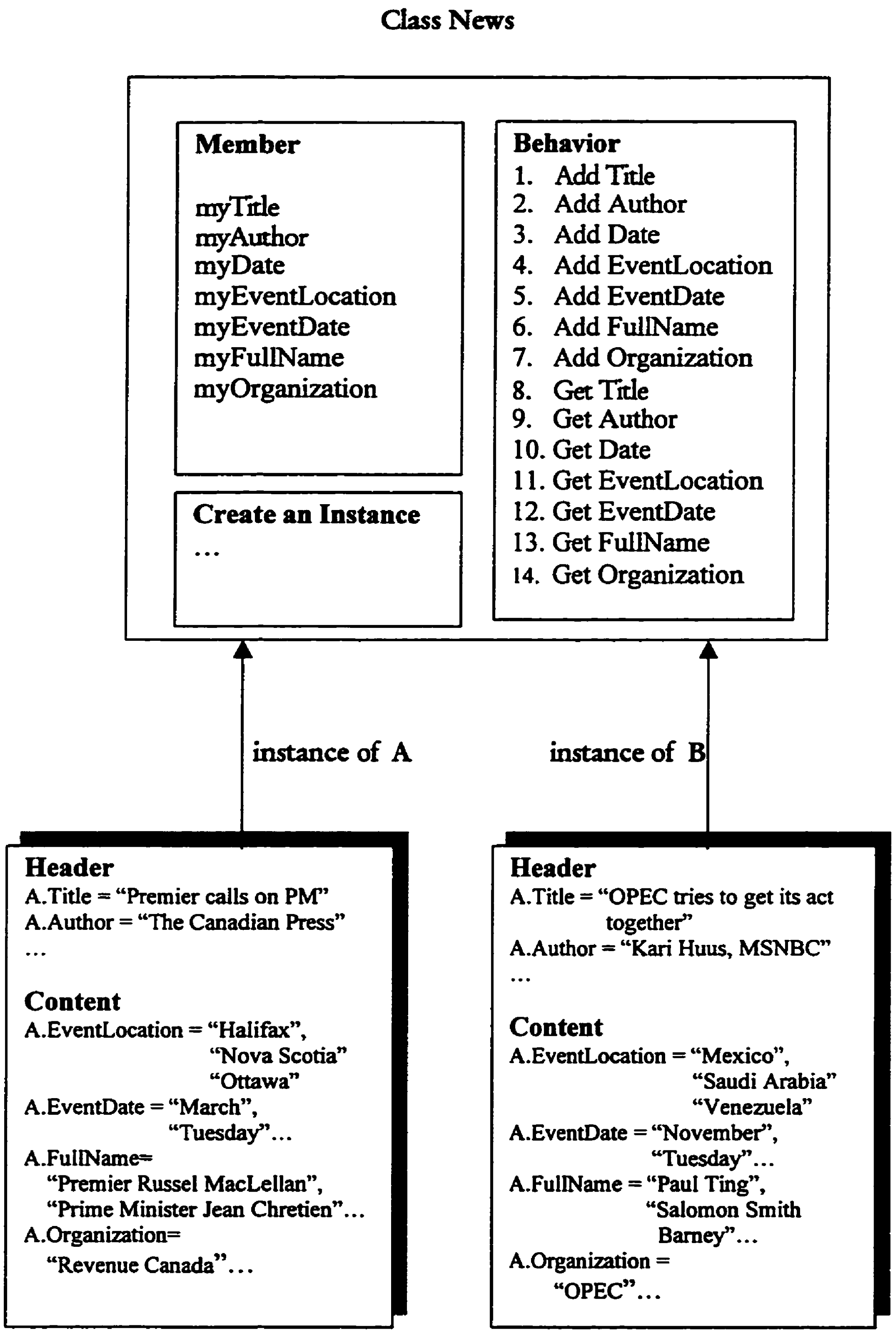
In this study we consider each representation of a news document as an object (Figure 3.1). What we mean by an *object* in this description is an entity able to save information about the news document and which offers a number of behaviors related to this information. There are two types of information: header and content. The *header* contains information about the structure of the news document, such as author, publisher, title, dates, and so on. There is no restriction on the number of members in the header part. There are four members in the *content* part. They are EventLocation, EventDate, FullName and Organization.

Figure 3.1 A news object



Next we define a class News, and each object that represents a news document becomes an instance of this class. In our example in Figure 3.2, A and B are instances of this class.

Figure 3.2 A and B are instances of the class News


Class News
Member
myTitle
myAuthor
myDate
myEventLocation
myEventDate
myFullName
myOrganization
Create an Instance
...
Behavior
1. Add Title
2. Add Author
3. Add Date
4. Add EventLocation
5. Add EventDate
6. Add FullName
7. Add Organization
8. Get Title
9. Get Author
10. Get Date
11. Get EventLocation
12. Get EventDate
13. Get FullName
14. Get Organization
instance of A
instance of B
Header
A.Title = "Premier calls on PM"
A.Author = "The Canadian Press"
...
Content
A.EventLocation = "Halifax", "Nova Scotia" "Ottawa"
A.EventDate = "March", "Tuesday"...
A.FullName= "Premier Russel MacLellan", "Prime Minister Jean Chretien"...
A.Organization= "Revenue Canada"...
Header
A.Title = "OPEC tries to get its act together"
A.Author = "Kari Huus, MSNBC"
...
Content
A.EventLocation = "Mexico", "Saudi Arabia" "Venezuela"
A.EventDate = "November", "Tuesday"...
A.FullName = "Paul Ting", "Salomon Smith Barney"...
A.Organization = "OPEC"...

Arrays of name-phrases can be assigned separately to each member of the class. Generally speaking, a *name-phrase* can be either a capitalized word or an adjacent phrase which is composed of capitalized words. For example, "Ottawa", "Tuesday", and "Acadia Univeristy" are all name-phrases. Name-phrases are categorized as EventLocation, EventDate, FullName or Organization name-phrase types. The algorithms used to further refine the name-phrase identification process are presented in Section 3.4.

3.2 Creation of a News Representation Object

There are five different tasks in the process of creating a news representation object shown in Figure 3.3. The first task is to extract information defining the header and content from different parts in a marked up file. A *marked up file* means that a file is divided into different parts by tags. In our study we use all or parts of tags, <PUBDATE>, </PUBDATE>, <HEADLINE>, </HEADLINE>, <BYLINE> and </BYLINE> to identify header information. We use <CONTENT> and </CONTENT> to identify content information. We borrowed these tags from the tag set used for the raw files of the Halifax Herald. A *rawfile* is a plain text file with a set of tags. It is used to present news items on a newspaper web site. An example of a marked up file is shown in Figure 3.4. The second task is to instantiate header members in the news object. It is described in section 3.2.1. The third task is to create a proper name set. A *proper name* is a capitalized word or a phrase including successive capitalized words. A proper name is extracted from the marked up file directly. This task is described in Section 3.3. The fourth task is to categorize each proper name in the proper name set into the different feature classes. It is described in Section 3.4. The final task is to instantiate content members of the news object.

3.2.1 Extracting Header Information and Content Information

A news document can contain information such as authors, publishers, dates, and so on. It is therefore necessary to extract this information and associate it with a news representation object. This information supplies potential index terms for further queries and news document management. Generally the header members of an object can be instantiated directly from a marked up file. We use the marked up file shown in Figure 3.4 as example. After running the program, the title "Premier calls on PM" is instantiated to the Title of the news object, and the author "THE CANADIAN PRESS" is instantiated to the Author of the news object.

The content information can also be extracted directly from a marked up file. We also use the marked up file shown in Figure 3.4 as example. After running the program, the content "Premier Russell MacLellan came to town Tuesday to drop in on some old friends - like Prime Minister Jean Chretien and ... " is extracted from the document as the content information.

3.2.2 The process of creating a news representation object

Figure 3.3 shows the data flow of the process of creating a news object. From the content information a proper name set is generated using special rules (see Section 3.3). After the process of feature name-phrase identification (see Section 3.4), the feature name-phrases are instantiated as different feature members of the object. Using the "get" method defined in the news class, the feature information can be output from a news object.

The Process in PseudoCode

```
NEW an object
READ a marked up file
FOR each line of the header:
    SWITCH the content of line:
        CASE Title
                object->addTitle
        CASE Author
                object->addAuthor
        CASE Content
                ADD this content into a content array
    END SWITCH
END FOR
FOR each member of a content array:
    IF proper name then ADD it into the proper name array.
END FOR
FOR each member of a proper name array:
    SWITCH the proper name
        CASE EventLocation
                Object->addEventLocation
        CASE EventDate
                Object->addEventDate
        CASE FullName
                Object->addFullName
        CASE Organization
                Object->addOrganization
        CASE Others
                NEXT
    END SWITCH
END FOR
```

Figure 3.3 The process of creating a news representation object

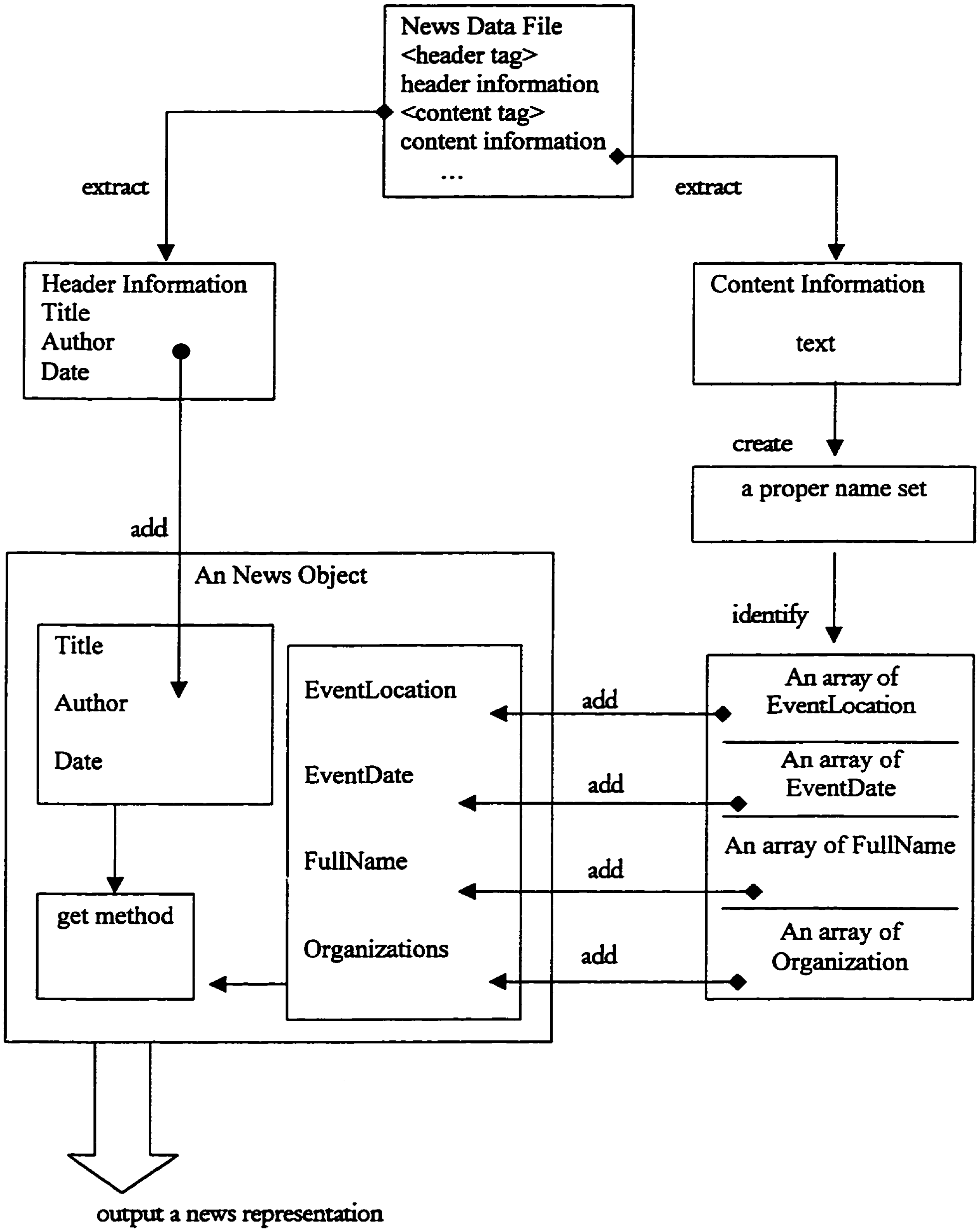


Figure 3.4 An example of a marked up file

(Halifax Herald March 17, 1998)

```
<PUBDATE>
1998/04/01
</PUBDATE>
<HEADLINE>
Premier calls on PM
</HEADLINE>
<BYLINE>
By THE CANADIAN PRESS
</BYLINE>
<CONTENT>
Premier Russell MacLellan came to town Tuesday to drop in on some old friends - like Prime Minister Jean
Chretien and Finance Minister Paul Martin.

 MacLellan, heading a precarious Liberal minority government, has been under  pressure from opposition New
Democrats and Conservatives over the blended sales  tax the province shares with Ottawa.

 The premier has also been pressing the federal government for extra funding for post-secondary education to
take account of the large number of out-of-province students in Nova Scotia universities.

 When spotted on Parliament Hill, however, he was close-mouthed about his visit. He did acknowledge he'd
"probably" see Chretien before catching an evening flight back to Halifax.

 A spokesman for the prime minister confirmed a meeting was planned but said it was private and there would
be no further comment.

 An aide to Martin similarly confirmed MacLellan would be meeting the finance minister but said there would
be "no formal agenda."

 MacLellan and Martin have met before - once last October and once in February - to discuss the blended sales
tax.

 Then came the March 24 election that reduced the Liberals to 19 seats, a flat-footed tie with the NDP, while
the Tories took 14.

 Both parties have threatened to withhold support for MacLellan's government, and the budget he wants to
bring in this spring, unless there is movement on the tax front.

 The NDP campaigned on a platform of scrapping the BST while the Conservatives want significant changes.

 MacLellan promised, during his run for the Liberal leadership last year, to offer tax rebates for home heating
oil and electricity. But once in power he found the lost revenue would endanger another key promise - to
balance the provincial budget.

 The premier said after his last meeting with Martin that there was no prospect for Ottawa directly funding any
BST relief.

But he did ask for help expediting a Revenue Canada decision on alleged tax overpayments by Nova Scotia
Power. The province is seeking refunds that would help ease the $16-million burden the BST added to
electricity bills.
</CONTENT>
```

3.3 Proper Name Extraction

Proper Name Extraction is the third task in the process of creating a news representation object. Proper Names include information about news features. In English writing, a single capitalized word often represents the name of a location, person, or organization, such as "Beijing", "Hong", or "AP". In other cases, a sequence of capitalized words presents the name of a location, person and organization, such as "North American", "Ms. Hong Wan", or "Diana Foundation". Single or multiple proper words are first extracted from the content information and stored in a proper name set.

In practice, the first step is to generate a pre-proper name set of all capitalized words. A *stop word* is a high-frequency word. Most documents are comprised of 40-50% percent stop words [14]. A stop word does not present any meaning and should be eliminated. The second step is then to delete stop word occurrences from the pre-proper name set. In this step duplicated information is deleted as well. We only keep one copy of any proper name in the proper name set. The output, a proper name set, is created for the feature match processes. The data flow chart is shown in Figure 3.5. The algorithm for this process in pseudocode follows.

The process of Proper Name Extraction in pseudocode

- *Create a content array*

```
FOR each line of the content information
    BREAK INTO words.
    SAVE each word into a content array.
END FOR
```

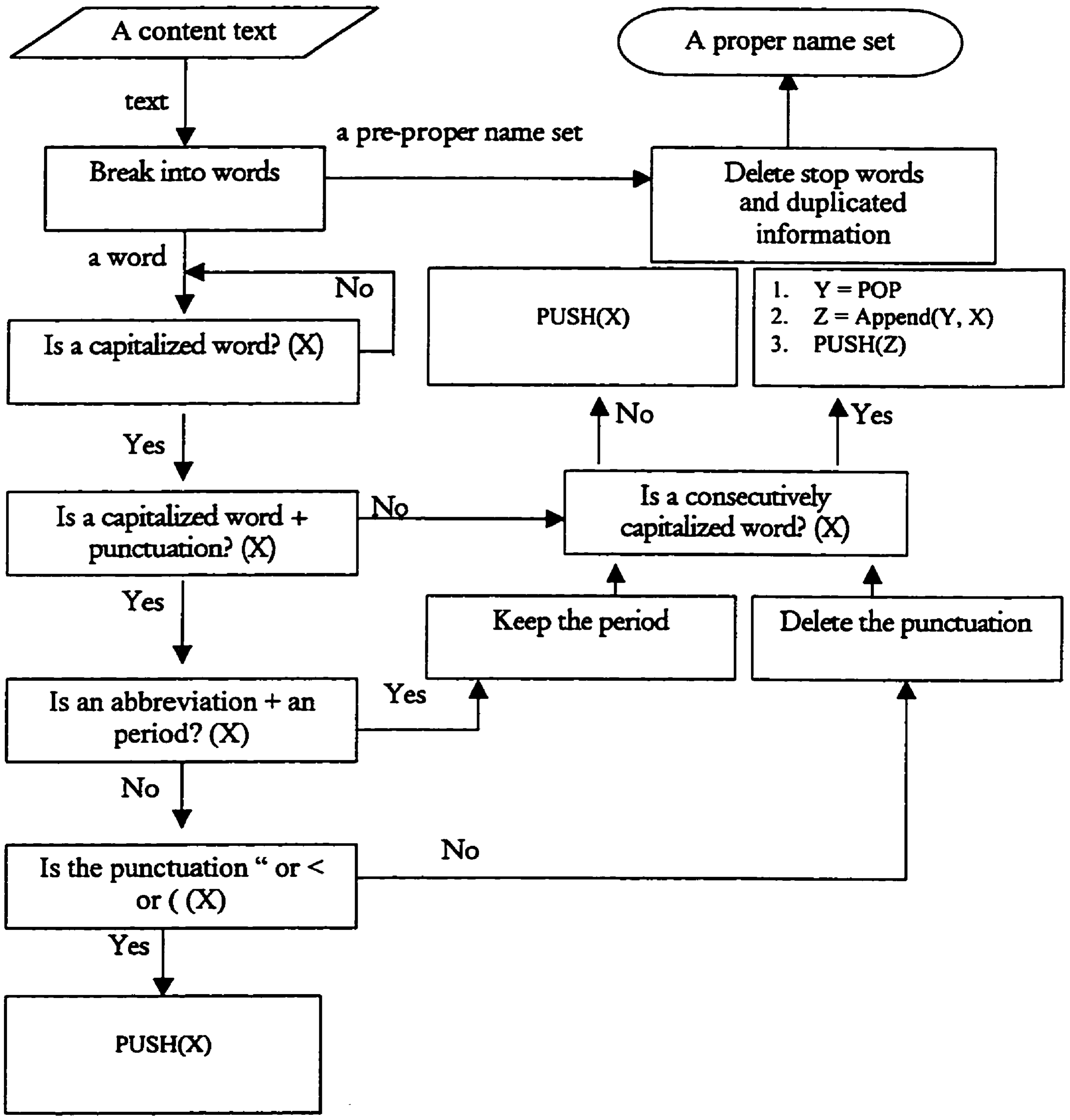
- *Create a pre-proper name set*

```
FOR each member of a content array
    IF a capitalized word then
      IF a capitalized word with a punctuation then
        IF an abbreviation word with an period then
          IF a consecutively capitalized word then
            POP the pre-proper name stack
                 and get previous capitalized string
            APPEND this word and period to the previous capitalized string
                 and get a new capitalized string
            PUSH the new capitalized string into the pre-proper name stack
          ELSE
            Push this word and period into the pre-proper name stack.
          END IF
        ELSE
          SWITCH punctuation
                CASE " or < or (
                      PUSH this word into the pre-proper name stack
                CASE others
                      IF a consecutively capitalized word then
                        POP the pre-proper name stack and
                             get previous capitalized string
                        APPEND this word without punctuation to
                             the previous capitalized string and get a new
                             capitalized string
                        PUSH the new capitalized string
                      ELSE
                        PUSH this word and period into the pre-proper name
                             stack
                      END IF
            END SWITCH
        END IF
      ELSE
        IF a consecutively capitalized word then
          POP the top of the pre-proper name stack and
               get previous capitalized string
          APPEND this word to the previous capitalized string
               and get a new capitalized string
          PUSH the new capitalized string into the pre-proper name stack
        ELSE
          PUSH this word into the pre-proper name stack.
        END IF
      END IF
    ELSE
    NEXT
  END IF
END FOR
```

- *Create a proper name set*

```
FOR each member of a pre-proper name set
    IF stop word then DELETE this stop word
    ELSE IF duplicated information then DELETE this information
        ELSE SAVE this pre-proper name into a proper name array
    END IF
END FOR
```

Figure 3.5 The data flow chart of proper name extraction process


A content text
text
Break into words
a pre-proper name set
A proper name set
Delete stop words and duplicated information
a word
No
Is a capitalized word? (X)
PUSH(X)
1. Y = POP
2. Z = Append(Y, X)
3. PUSH(Z)
No
Yes
Yes
Is a capitalized word + punctuation? (X)
No
Is a consecutively capitalized word? (X)
Yes
Keep the period
Delete the punctuation
Is an abbreviation + an period? (X)
Yes
No
Is the punctuation " or < or ((X)
No
Yes
PUSH(X)

We use the sample document (shown in Figure 3.4) as input data. After running the programs, the result of the pre-proper name set is shown as table 3.1. The content of the proper name set is shown as Table 3.2.

Table 3.1 The pre-proper name set

(using document presented in Figure 3.4 as input data)

Premier Russell MacLellan	Tuesday
Prime Minister Jean Chretien	Finance Minister Paul Martin
MacLellan,	Liberal
New Democrats	Conservatives
Ottawa	The
Nova Scotia	When
Parliament Hill	He
Chretien	Halifax
A	An
Martin	MacLellan
MacLellan	Martin
October	February
Then	March
Liberals	NDP
Tories	Both
MacLellan	The NDP
BST	Conservatives
MacLellan	Liberal
But	The
Martin	Ottawa
BST	But
Revenue Canada	Nova Scotia Power
The	BST

Table 3.2 The proper name set

(using the document presented in Figure 3.4 as input data)

Premier Russell MacLellan	Tuesday
Prime Minister Jean Chretien	Finance Minister Paul Martin
MacLellan	Liberal
New Democrats	Conservatives
Ottawa	Nova Scotia
Parliament Hill	Chretien
Halifax	Martin
October	February
March	Liberals
NDP	Tories
BST	Revenue Canada
Nova Scotia Power	

The Special Cases

We now give several examples to explain how the algorithm deals with special cases.

- Punctuation and special characters are normally considered to separate proper names.

1. Examples with period

"... Spokane. Wash... "

Because "Spokane" is not an abbreviation, the period separates the above words into two proper names: Spokane and Wash.

<TYPE= "PROPER NAME" >Spokane

<TYPE= "PROPER NAME" >Wash

Another example,

"... Mr. Truman... "

Because "Mr." is an abbreviation, the period is not used to separate proper names. The phrase "Mr. Truman" is extracted as one proper name.

<TYPE= "PROPER NAME" >Mr. Truman

2. Examples with dash

If the dash connects two words with a space, the dash separates the proper names. If the dash connects two words without a space, the dash is not used to separate the proper name. So "U.S. – Based" will be extracted as two proper names: U.S. and Based

<TYPE= "PROPER NAME" >U.S.

<TYPE= "PROPER NAME" >Based

"U.S-Based" will be extracted as one proper name: U.S.-Based.

<TYPE= "PROPER NAME" >U.S-Based

3. Examples with other punctuation

The double quote, angle bracket, and parenthesis are used to separate proper names. For example,

" Mr. Clinton is visiting Beijing (AP Report). "

The left parenthesis separates the above phrase "Beijing AP Report" as two proper names: Beijing and AP Report.

<TYPE= "PROPER NAME" >Beijing

<TYPE= "PROPER NAME" >AP Report

- The apostrophe is not used to separate proper names. The apostrophe is simply deleted, if it occurs in a proper name.

"California's Brown"

The word "California" has an apostrophe, so we ignore the apostrophe and extract the proper name as:

<TYPE= "PROPER NAME" >California Brown

- Multi-names containing conjunctions, such as "and", "or" are treated as separate proper names.

"North and South America"

After a run, the proper names are extracted as followings:

<TYPE = "PROPER NAME" >North

<TYPE = "PROPER NAME" >South America

Stop words

One way to improve information retrieval performance is to eliminate stop words. A list of words filtered out during automatic indexing because they make poor index terms is called a *stoplist* or a negative dictionary [31]. In our study the stoplist was based on Carrick's [32] and Francis and Kucera's [31]. The stoplist was increased to 343 words after running the program 150 times to remove additional high frequency terms. The improved stoplist is shown in Table 3.3.

Table 3.3 An improved stoplist

A	Able	About	Above	According
Across	Added	After	Again	Against
Ago	Air	All	Almost	Alone
Along	Already	Also	Among	An
AND	And	Another	Any	Anybody
Anyone	Anything	Anywhere	Are	Area
Areas	Around	ARTICLE	As	Asked
Assembly	At	Away	Back	Bad
Based	Be	Because	Been	Before
Behind	Being	Best	Better	Between
Big	Body	Both	But	By
Byline	Called	Calls	Came	Can
Can't	Caption	Cat	Cent	Certain
Certainly	Clear	Clearly	Come	Could
Date	Day	Days	Despite	Did
Didn't	Different	Do	Does	Doesn't
Doing	Done	Don't	Down	During
Each	Earlier	Early	Eight	Either
End	Enough	Even	Evenly	Eventually
Ever	Every	Everybody	Everyone	Everything
Expected	Fearing	Feel	Few	Find
Finished	First	Five	Fname	Following
For	Former	Four	Fourth	From
Full	Fully	Further	Furthered	Furthering
Get	Give	Go	Going	Good
Got	Group	Had	Happened	Has
Have	He	Held	Her	Here

He's	High	Him	His	Hopefully
How	However	I	IF	If
I'm	In	Include	Including	Instead
Into	Involved	Is	It	Its
It's	Just	Keep	Keywords	Knew
Know	Largest	Last	Late	Later
Least	Left	Like	Little	Long
Look	Looking	Made	Make	Making
Many	Matter	May	Me	Meanwhile
Meet	Metres	Might	Month	More
Most	Mostly	Move	Much	Must
My	Myself	Near	Need	Neither
Never	New	News	Next	Nine
No	Nobody	Normally	Not	Nothing
Now	Number	Obviously	OF	Of
Off	Often	OK	On	Once
One	Only	Or	Other	Others
Our	Out	Over	Own	Part
People	Per	Photo	Photographer	Place
Plus	Points	Possible	Possibly	Previously
Probably	Provide	Put	Really	Recent
Recently	Regardless	Released	Said	Same
Say	Says	Second	Section	See
Set	Seven	Several	She	Should
Show	Showed	Since	Six	Small
So	Some	Someone	Something	Sometimes
Special	Start	Started	Stay	Still
Such	Suddenly	Take	Taking	Ten
Than	That	That's	THE	The
Their	Them	Then	There	There's
These	They	They'll	They're	Thing
Things	This	Those	Though	Thought
Three	Through	Tif	Title	To
Told	Too	Took	Traditionally	Turned
Two	Typically	Under	Unlike	Until
Up	Us	Use	Very	Want
Wants	Was	Way	We	Week
Well	Went	Were	We're	We've
What	WHEN	When	Where	Which
While	Who	Why	Will	With
Without	Would	Year	Years	Yet
You	Your	You're		

3.4 The feature name-phrase identification process

The basic idea in the feature pattern-matching process is that proper names in the proper name set are classified as names, dates, locations or organizations. Dictionaries of pre-defined feature names help to assign each proper name to a feature class. The attributes of each feature are defined as follows:

- *EventLocation:* the name of a politically or geographically defined location (cities, provinces, countries, international regions, bodies of water, mountains, etc). In practice, a dictionary of 2444 names is used. It was obtained from Energy, Mines and Resources Canada. For example, the proper name "Northern California" would be identified an EventLocation by a match in the dictionary and represented as:

  <TYPE = "EVENTLOCATION" >Northern California

- *EventDate:* a name of days or months. A dictionary of 37 words with different spellings for days or months is used. It is shown in Table 3.5 and 3.6. For example, the proper name "April" would be classified as an EventDate name-phrase as follows:

  <TYPE = "EVENTDATE" >April

- *FullName:* a name of a person, family or organization. In practice, a dictionary of 3190 first names, a dictionary of 3200 last names and a dictionary of 33 common titles (or occupations) are used to identify this feature. These dictionaries are from C. Carrick and C. Watters [2]. For example, the proper name "Mips President John Hime" would be identified as a FullName name-phrase as follows:

<TYPE = "FULLNAME" ><O>Mips<T>President<F>John<M><L>Hime

Note: <O>, <T>, <F>, <M>, and <L> represent attributes of a FullName name-phrase: Other, Title, FirstName, MiddleName, and LastName. In our study a multiple FullName name-phrase must have at least a last name.

- *Organization:* name corporate, governmental, or other organizational entity. For example, the proper name "Treasury " would be identified as a Organization name-phrase as following:

  <TYPE = "ORGANIZATION" >Treasury

  In our implementation, we did not create an Organization dictionary to identify Organization name-phrases. Since the topics of our test set are about World events, Local events and Canadian events, a wide-range of organization names are used inside the test set. It is difficult to pre-define organization names in a dictionary. We put those proper names which are non-EventLocation, non-EventDate, or non-FullName name-phrase into this class.

The feature name-phrase set after this process for the document shown in Figure 3.4 is shown in Table 3.4.

Note: [O] presents Others attribute in a FullName name-phrase. [T] presents Title attribute. [F] presents First Name attribute. [M] presents Middle Name attribute. [L] presents Last Name attribute.

Table 3.4 The feature name-phrase set

(using the document shown in Figure 3.1 as input data.)

Title

Premier calls on PM

Author

By THE CANADIAN PRESS

Date

1998/04/01

Event Location

Halifax
Nova Scotia
Ottawa

Event Date

March
February
October
Tuesday

Full Names

[O]Premier	[T]	[F] Russell	[M]	[L]	MacLellan
[O]Prime Minister	[T]	[F] Jean	[M]	[L]	Chretien
[O]Finance Minister	[T]	[F] Paul	[M]	[L]	Martin

Others

Parliament Hill
Conservatives
Liberals
NDP
Tories
BST
Revenue Canada
Nova Scotia Power

3.4.1 EventDate identification

Event data is seldom the most important element of a story, but it must be included since it orients the reader in time. In news writing, there are some rules [28]:

- Days within the newspaper dateweek are referred to by the day, i.e., Monday, Tuesday, Wednesday.
- Days outside the dateweek are referred to by month and date (July 1).
- Yesterday, tomorrow, next and last (next Monday, last Monday).

The terms for relative time expressions are not identified as EventDate name-phrases. For example, "yesterday", "tomorrow", "spring" and so on. The first reason is that it is ambiguous in time specification. The second reason is that there is not a uniform writing rule. For example, they are written as words beginning with capital letters only on the beginning of a sentence. In our study, numbers were not deemed likely to be as important as words, so the dates are not appended in a EventDate term. For example, "July 1, 1998" would be extracted as:

<TYPE = "EVENTDATE" >July

Table 3.5 and 3.6 show the dictionaries used to identify EventDate name-phrases.

Table 3.5 The month dictionary

Jan.	Feb.	Mar.	Apr.
Jun.	Jul.	Aug.	Sept.
Oct.	Nov.	Dec.	January
February	March	April	May
June	July	August	September
October	November	December	

Table 3.6 The day dictionary

Sun.	Mon.	Tues.	Wed.
Thurs.	Fri.	Sat.	Sunday
Monday	Tuesday	Wednesday	Thursday
Friday	Saturday		

When a proper name appears in above tables (Table 3.5 and 3.6), it can be identified as an EventData name-phrase . The term "May" needs special mention, since it is also a stop word. The following special process is used to identify the term "May". If the term "May" is the first word in a sentence, it is identified as a stop word. In other cases, the term "May" is identified as an EventDate term. For example, the term "May" is extracted as a stop word from the text, "I was in Ottawa last summer. May I do it again this year?" as following:

<TYPE = "Stop Word">May

because "May" is the first word in the sentence "May I do it again this year?".

The "May" is extracted as a EventDate name-phrase from the text, " I will go to Ottawa on May 12.", as follows:

<TYPE = "EventDate"> May

3.4.2 EventLocation identification

According to the rules to generate a proper name set described earlier, the EventLocation name-phrases can be identified in the following special cases:

- The phrase "of EventLocation name-phrase" may or may not be part of an organization name-phrase, such as Hyundai of Korea. The EventLocation name-phrase would be identified as a EventLocation name-phrase. For example, "Korea" would be extracted as a EventLocation name-phrase from the phrase "Hyundai of Korea."

  <TYPE = "EventLocation" > Korea

- If two location proper names are separated by punctuation, they are to be identified as two separate EventLocation name-phrases. For example, "Washington, D.C." would be identified as two EventLocation name-phrases and the result would be as follows:

  <TYPE = "EventLocation" >Washington

  <TYPE = "EventLocation" >D.C.

- Location-related words beginning with a lower case letter are not identified as EventLocation name-phrases. Because words beginning with a lower case letter are not extracted into the proper name set. For example, "Canadian exporters" would be identified as:

  <TYPE = "EventLocation" >Canadian

- Miscellaneous characters after a location-name are to be identified as an EventLocation name-phrase. For example, "Beijing's " would be extracted as follows:

<TYPE = "EventLocation" >Beijing

3.4.3 FullName identification

The FullName is the most important feature in a representation, since people and organizations are generally the central actors in an event. In this study, a FullName name-phrase has five attributes: others, title, first name, middle name, and last name entity. The title dictionary is used to identify title attribute. It includes 33 usual words to entitle a person. The first name dictionary is used to identify the first name attribute. It includes 3190 first names. The last name dictionary is used to identify last name attribute. It includes 3200 last names.

In formal writing, all people must be fully identified once in the body of a news story. It is possible to distinguish different persons with the same first names or last names. The steps of the algorithm used to identify FullName name-phrases is given below, where <O> represents others attribute, <T> represents title attribute, <F> represents first name attribute, <M> represents middle name attribute, and <L> represents last name attribute.

The algorithm in two cases

1. *A single proper name*

   If it is a single proper name (X), checks are made to see if it is in the last name dictionary. If a match is found, then it is identified as:

<TYPE = "FullName"> <L>X

In English writing, a last name occurring in a document is usually with a title. For example, "Mr. Clinton". A last name may occur independently, such as "Clinton". In our study, the single proper name is not identified as a first name.

2. *A multiple proper name*

Suppose the composite of a proper name is $A_1A_2 \ldots A_n$ where A_i represents a word. The process of identification is presented as follows:

- *Check the title*

  Checks are first made to see if there are titles (k) inside it. If the match is found (Am), then the proper name is to be identified as:

  <TYPE = "FullName"><O>$A_1 \ldots A_{m-1}$ <T>A_m <F>A_{m+1} <M> $A_{m+2} \ldots A_{n-1}$ <L>A_n.

  At exception handling, a check is made to make sure that A_n is not an organization entity. Otherwise, the attribute of last name will be A_{n-1} and of middle name will be $A_{m+2} \ldots A_{n-2}$. Multiple titles can be extracted from a FullName name-phrase.

- *Check the last name*

  If there is no title, then checks are made to see if there is a last name. The search begins with A_n toward back A_1. If the match (A_m) is found, then the proper name is to be tagged as:

  <TYPE = "FullName" ><O>$A_1 \ldots A_{m-3}$ <F>A_{m-2} <M>A_{m-1} <L>A_m <O>$A_{m+1} \ldots A_n$

In this case, A_{m-2} is automatically identified as a first name. Furthermore, A_{m-2} will be added to the first name dictionary if it is not in it.

- *Check the first name*

If no last name was found above, checks are made to see if there is a first name. The search goes forward from A_1 to A_n. If a match (A_m) is found, then the proper name is identified as:

<TYPE = "FullName"><O>$A_1 \ldots A_{m-1}$ <F>A_m<M>$A_{m+1} \ldots A_{n-1}$ <L>A_n

In this case, A_n is automatically identified as a last name. Furthermore, A_n will be added to the last name dictionary if it is not in it. Or as exception handling, a check is made to make sure that A_n is not an organizational entity. Otherwise, the attribute of last name will be A_{n-1} and of middle name will be $A_{m+1} \ldots A_{n-2}$.

- *The last step*

If no first name was found above, the proper name will be identified as a non-FullName.

The Special Cases

According to the rules to generate a proper name set described above, the FullName name-phrases are identified as follows for some special cases:

- Special Titles

Titles such as "Mr." and role names such as "President" are considered to be title attributes of a FullName name-phrase. However, appositives such as "Jr." are not considered part of a FullName name-phrase. For example, "Mr. Harry Schearer" is to be identified as:

<TYPE = "FullName"> <T>Mr. <F>Harry <L>Schearer

"John Doe, Jr." is to be identified as:

<TYPE = "FullName"> <F>John <L>Doe

There are actually two proper names "John Doe" and "Jr." extracted from the phrase "John Doe, Jr". In our study we do not provide a function to group related proper names together.

- Person name with apostrophe

Person name would be identified as a FullName name-phrase without any apostrophes. For example, "Hong's" is to be identified as:

<TYPE = "FullName"> <F>Hong

"Hong" is extracted as a proper name from the phrase "Hong's".

- Person name related phrase

  If it is a person name related phrase, the person name is identified as a FullName name-phrase.

  If the original text is "the Nobel prize" then "Nobel" is identified as a FullName phrase.

  "Alex Wan School" would be identified as:

  <TYPE = "FullName"> <F>Alex<L>Wan<O>School

3.5 Reduce redundant name-phrases

The idea of this process is to ensure that every feature term is unique. For example, if both the terms "Wed." and "Wednesday" are in the EventDate class, only one of them is kept in the representation object. Generally there are few multi-word name-phrases in either EventDate or EventLocation class, so an exact match process is used to reduce the redundant name-phrases from these two type classes. If the composite of two name-phrase is the same, they are considered to be the same name-phrase. For example, the term "North China" and "China" are treated as two different name-phrases. But "China" and "China" are the same name-phrase. As for the FullName feature, a partial match process is used to reduce redundant information. The rules applied for implementation are as follows:

1. If two FullName name-phrase have the same attributes of first name and last name, the shorter one is deleted from the class. For example, "Mr. Bill Clinton" and "Bill Clinton", the shorter phrase "Bill Clinton" is deleted.

2. If a single FullName name-phrase has the same last name attribute as others, it is deleted from the class. For example, given both "Mr. Bill Clinton" and "Clinton", "Clinton" is deleted.

For Others feature, a loose match process is used to reduce redundancy. If a name-phrase is part of another name-phrase, it is deleted from the class. For example, the name-phrase "Respite Care" is part of name-phrase "Respite Care Intervention Association" and it would be deleted from the class.

The advantage of this process is that it helps to keep the value of a document-to-document similarity between 0 and 1. The frequency of a name-phrase in a document is not taken into consideration in this study.

3.6 A document-to-document similarity method

Once we have created a representation of a news object, we can look for other news objects which are similar.

We have adopted the similarity methods introduced by Carrick and Watters [2]. According to our equations, the similarity value is between 0 and 1. Equation 3.1 was used to calculate the document-to-document similarity.

$$sim(A,B) = \alpha \sum_{i=1}^{4} a_i \qquad (3.1)$$

where a_i is the similarity of the i-th feature between news document A and B, and α is the importance value as calculated by

$$\alpha = \frac{1}{\sum_{i=1}^{4} \min(|A_i|,|B_i|)} \qquad (3.2)$$

where $\min(|A_i|, |B_i|)$ is the length of the smaller of the two feature name-phrase sets. The value a_i can be calculated using the following equation

$$a_{i}(X,Y) = \sum_{j=1}^{m}\sum_{k=1}^{n}\theta(x_j, y_k) \qquad (3.3)$$

which is the sum of the common terms between two name-phrase set X and Y. And m is the length of X and n is the length of Y. The value for θ is calculated using the following formula

$$\theta(x_j y_k) = \begin{cases} 1 & \text{if } x_j = y_k \\ 0 & \text{if } x_j \neq y_k \end{cases} \tag{3.4}$$

In the case of the EventDate or EventLocation feature, x_j and y_k are deemed to be the same when they precisely match each other. For example, "Beijing China" is the same as "Beijing China", but it is not the same as "China". In the case of the Organization feature, the x_j and y_k are deemed to be the same when one is a substring of the other. For example, "Beijing KL Technology Co." is the same as "KL Technology Co.".

There are some special rules to calculate the similarity between two FullName name-phrases. Every FullName term has five attributes and the importance value assigned to each attribute is shown in Table 3.7.

Table 3.7 The importance value of FullName attributes

Attribute	Importance Value
Title	0.20
First Name	0.30
Middle Name	0.05
Last Name	0.40
Other	0.05
Threshold	0.80

If the last name attributes are not the same between two FullName name-phrases, the similarity will equal 0.

- Equation 3.5 is used to calculate the same FullName name-phrase.

$$Sam(N_1, N_2) = \sum_{i=1}^{5} w_i \theta(n_{1i}, n_{2i}) \qquad (3.5)$$

where W_i is the importance value of ith attribute shown in the table 3.7. $\theta(n_{1i}, n_{2i})$ is calculated using

$$\theta(n_{1i}, n_{2i}) = \begin{cases} 1 & if\ n_{1i} = n_{2i} \\ 0 & if\ n_{1i} \neq n_{2i} \end{cases} \qquad (3.6)$$

- When the value of *Sam* is more than the threshold value (≥0.8 in practice), these two name-phrases N_1 and N_2 are deemed to be the same.

Table 3.8 is a summary of results of the FullName match method. It lists some special cases as examples. (A and B represent FullName name-phrases. A is presented using bold font. The result column shows the similarity between A and B.)

- A Title+LastName

  For example, A is "Mr. Smith". B is deemed the same as A only if their LastName attributes are the same.

  For example B is "Justice. Smith". A and B can be deemed as the same. According to Equation 3.5, the similarity equals 0.8.

- A FirstName+LastName

  For example, A is "John Smith". B is deemed the same as A only if their LastName attributes are the same and B's FirstName is empty or the same as A's.

  For example, B is "John Smith" or "Mr. Smith". B is the same as A. According to Equation3.5, the similarity equals 1.0.

  For example, B is "George Smith". B is not the same as A. According to Equation 3.5, the similarity equals 0.7.

- A Title+FirstName+LastName

For example, A is "Mr. John Smith". B is deemed as the same as A only if their LastName are the same, B's FirstName is empty or the same as A's, and B's Title is empty or the same as A's.

For example, B is "Mr. George Smith". B is not the same as A. According to equation 3.5, the similarity equals 0.7.

Table 3.8 Summary of results of the FullName match method

Other	Title	First Name	Middle Name	Last Name	Result
	Mr.			**Smith**	
	Justice.			Smith	0.8
		John		**Smith**	
		George		Smith	0.7
	Mr.			Smith	1.0
	Mr.	**John**		**Smith**	
	Mr.	George		Smith	0.7

Chapter 4 Results

This chapter discusses the evaluation of the quality of the news representation making programs and the document-document similarity algorithms described in Chapter 3. Two different methods are used. First a quantitative evaluation of a test set shows how accurate and efficient the association is. The test set included 78 articles from one day randomly chosen from the Halifax Herald. The other method of testing was using the users. Recall and Precision are used to evaluate the result. The test set included alternate data from the Halifax Herald, data from different newspapers, the MSNBC, Reuters and Yahoo! News on Asia site. The methodology of the evaluation and the results derived are presented in the following sections.

4.1 Evaluation of the associated name-phrases

Before we present the test results, we will define the words used in this section. A *term* here means any single word. *Important terms* mean the terms covering special information in an article. For example, "Halifax", "Nova" and "Scotia" are *important terms*. *Important phrase* means that a phrase represents a meaning unit, such as "Nova Scotia". Important terms and phrases are selected manually by an IR expert. *Total terms* mean all terms found in an article. *Capital terms* mean terms beginning with capital letters. *Noise terms* mean terms without meaning in an article. Generally they are stop words.

The test set

The collection of news stories used to evaluate the quality of the programs is a set of 78 news articles from five different categories:

- Nova Scotia News (28)
- Canada News (15)
- World News (14)
- Business (10)
- Entertainment (11)

These were chosen from the April 1, 1998, edition of the Halifax Herald newspaper. Further information on the collection that we used is given in Table 4.1.

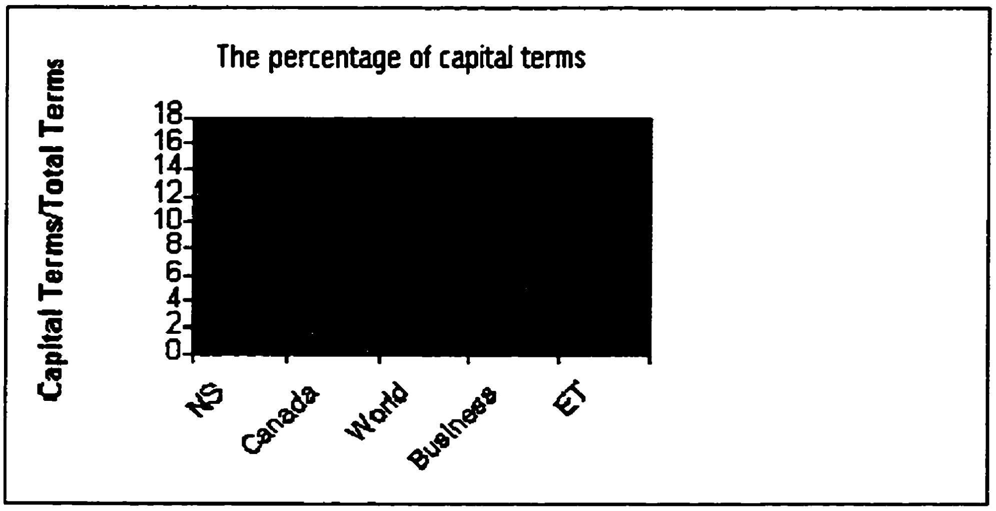
Table 4.1 The information of the test set

Categories	# of Total Terms	# of Capital Terms	# of Important terms
Nova Scotia News	8376	1315	1067
Canada News	6075	748	575
World News	4533	612	472
Business	3857	607	498
Entertainment	3477	557	439

The number of Important Terms obtained was by manual inspection of the original documents. The chosen standard was based on human behavior in a hypothetical news reading situation. How well the programs can find and associate the important terms with a news story is important. Before testing programs, we first analyze our test set.

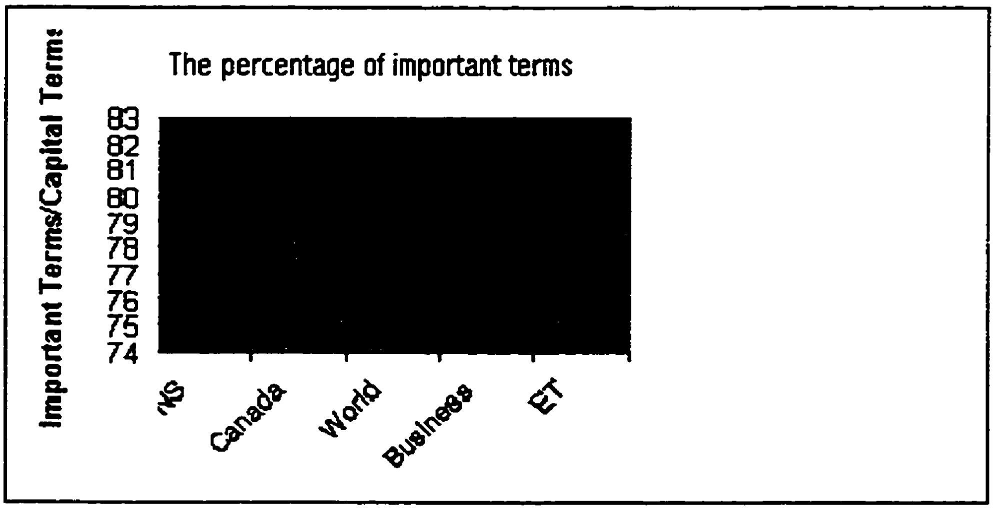
According to the occurrences shown in Figure 4.1, the capital terms are 14.6% of the total terms. We found that the percentage of capital terms were lower for long article than for short ones. The average total terms are 299 terms per article in a Nova Scotia category and its percentage of capital terms is 15.7%. The average total terms are 405 terms per articles in a Canada News category and its percent is 12.3%.

Figure 4.1 The percentage of capital terms



The percentage of important terms in the capital terms was found to be between 77% and 82% as is shown in Figure 4.2.

Figure 4.2 The percentage of important terms



The program performance

How well the feature name-phrases can be extracted is important for the document-document similarity programs, because different features have different calculations. The wrong feature classification will directly affect the precision of a system. Further it may cause an unrelated match to be returned to the user in an electric newspaper system. We use accuracy to evaluate this performance. *Accuracy* refers to the percentage of name-phrases that are correctly classified into different feature classes.

For example, there are 10 name-phrases in a news object. If there are 6 name-phrases in the FullName class and 5 of them are FullName name-phrases, the accuracy for the FullName is 83%.

In the test run, a news object was generated for each article in the collection. Based on manual inspection, the results obtained for the average accuracy for the EventDate, EventLocation and FullName name-phrases are shown in Table 4.2.

Table 4.2 The results of accuracy

	NS	CANADA	WORLD	BUSINESS	ET
# of EventDate	21	25	18	20	20
Accuracy	100%	100%	100%	100%	100%
# of EventLocation	20	32	18	49	7
Accuracy	100%	100%	100%	100%	100%
# of FullName	134	45	39	42	32
Accuracy	92%	93%	94%	94%	93%

Avg. of Accuracy on EventDate	100%
Avg. of Accuracy on EventLocation	100%
Avg. of Accuracy on FullName	93%

How well the feature name-phrases cover the important information from the original article affects the recall of a system. High recall means that we do not miss related documents to be returned to the users. We use association to evaluate this performance. *Association* refers to the percentage of important name-phrases associated with a given news story.

For example, there are 10 important phrases in the original articles and 9 of them are included in the news object, and the association is 90%.

In the test run, a news object was generated for each article in the collection. Based on manual inspection, the result obtained for the average of association is shown in Table 4.3.

Table 4.3 The results of association

	NS	CANADA	WORLD	BUSINESS	ET
# of important phrases	344	200	166	241	130
# of important features	317	182	149	212	116
Association	92%	91%	90%	88%	89%

Avg. of Association	90%

Based on manual inspection the correct percentage of EventData terms is 100% out of 104 EventDate phrases in 78 articles. The correct percentage of EventLocation terms is 100%. The algorithm of FullName phrase extraction use the first name, last name and title dictionaries as match templates. The associated terms in the FullName class has the 93% correct percentage of FullName terms. The average of association is 90%.

The problems

There are three reasons that cause problems in selecting important phrases within a news object:

- Informal English spellings – When a name-phrase is a foreign phrase beginning with a lower case letter, it cannot be associated with the news object. For example, the phrase "La du Ronge" will be associated with the news object as two name-phrase "La" and "Ronge". These two name-phrases cannot represent their original meanings in the article. As a result it causes some noise terms.

- Punctuation problems – Punctuation is normally considered to separate name-phrases in our algorithm. If the punctuation is not correct, it may cause important terms to be missed. For example, if a right apostrophe or left apostrophe is missed in an article, the name-phrases are extracted incorrectly. Sometimes the programs put unrelated terms together.

- Special terms – The programs do not provide a function which can process special terms, such as "&" and "of" and it may lead to missing important terms. For example, the name phrase "AT&T" can be extracted correctly, because it is one word. But the phrase "AT & T" cannot be extracted correctly, because it is three-word phrase with a special character "&".

4.2 Tests with the users

There are two standard measures in IR for retrieval success: recall and precision. *Recall* is defined as the number of relevant documents retrieved in response to a query divided by the total number of relevant documents in the collection. *Precision* is defined as the number of relevant documents retrieved in response to a query divided by the total number of retrieved documents. Recall is the ability of the system to present all relevant items. Precision is the ability to present only the relevant items [1].

The following testing was conducted with the aim of calculating the recall and precision after searches using first the selected name-phrases set and second the news object itself. Hence, we discuss the threshold of similarity used by different searches.

The document set

Based on the news data domain, a set of 25 documents was pre-selected from the Reuters, AP, MSNBC and Yahoo! Asian News web site. They are published between June 12, 1998 and June 20, 1998 from above news resources. There are news stories related to the war in Kosovo, Iraq oil and Asian economics, etc.

Experiments

The performance measures used are precision and recall. The document set does not change over the duration of this experiment. Recall is calculated by manually going through each of the stories in the document set.

For each scenario, each document is classified as relevant if the major topic of the news story was associated with Kosovo. We selected 0.4 as the similarity threshold.

Scenario 1

Suppose we have a user named A who is a graduate student writing a thesis on the present political situation in Kosovo. A is interested in gathering information about what kind of role the Russian government plays in Kosovo. So A selected some name-phrases shown in column S1 of Table 4.4.

Results

Of the 25 stories, 17 stories were returned as related and all of these were found to be relevant. Manual inspection found that no articles related to Kosovo in the collection were missed. So in this case the filter achieved 100% recall and 100% precision.

Scenario 2

Suppose we have a user named B who is a graduate student writing a term paper on the present political situation in Kosovo. B is interested in gathering information about what kind of role the American government plays in Kosovo. So B selected some name-phrases shown in column S2 of Table 4.4.

Results

Of the 25 stories, 13 stories were returned as related and all of them were found to be relevant. Manual inspection found that 2 articles on the Kosovo war were missed. So the filter achieved slightly over 87% recall and 100% precision.

Scenario 3

Suppose we have a user named C who is a graduate student writing a thesis on the present political situation in Kosovo. C wants to know how deeply related other documents are to the story titled "Contact group seeks Kosovo solution" in June 12, 1998 from MSNBC. In a run, the news object of this report is created to be used as a search. The name-phrases were shown in S3 columns of Table 4.4.

Results

Of the 25 stories, 11 stories were returned as related and all of them were found to be relevant. Manual inspection found that 6 articles on the Kosovo war were missed. So, the filter achieved slightly over 45% recall and 100% precision. If we selected the similarity threshold as 0.3, of the 25 stories, 14 stories were returned as related and 13 stories of them were found to be relevant. Manual inspection found that 4 articles on Kosovo were missed. So the recall is 76% and precision is 93%.

Summary

Because we used a very simple method to calculate the document-document similarity, noise terms have the same weights as important terms. It was observed that when a user uses a news object for a search the optimal similarity threshold was 0.3. Although the precision is lower, the recall goes up. As a result, we do not miss some related documents to be returned to the user. When a search uses a set of name-phrases selected by the user, the optimal similarity threshold was observed to be 0.4. Generally the name-phrases included in a selected name-phrase set are seldom noise ones. Without the noise phrase affection, the optimal similarity threshold is slightly higher than it is when using a news object in a search.

The selected name-phrase set directly affects the values of recall and precision. For example, the user A selected a good name-phrase set, so both recall and precision are satisfied.

During the experiments, the value of recall is not stable, but the value of precision keeps high when it uses the selected name-phrase set as search. A poorly selected name-phrase set will cause lower recall. For example, if user B's profile is used to search related documents about Kosovo. The recall is 76%. It missed two stories about the relationship of Russia and Kosovo in the history. These two stories are related to the Kosovo topic, but do not address the role of the American government in the Kosovo war. So in scenario 2, the recall is 87% higher.

Table 4.4 The name-phrases selected by the users in three searches

S1	S2	S3
<EVENTLOCATION>	<EVENTLOCATION>	<EVENTLOCATION>
Yugoslavia	Yugoslavia	Moscow
Moscow	Washington	Washington
Russia	Albania	Pakistan
	United States	India
		Italy
		Germany
		France
		Russia
		United States
		Albania
		London
		Yugoslavia
<EVENTDATE>	<EVENTDATE>	<EVENTDATE>
		Thursday
		Friday
<FULLNAME>	<FULLNAME>	<FULLNAME>
Slobodan Milosevic	Slobodan Milosevic	Serbian President Slobodan Milosevic
Boris Yeltsin	President Bill Clinton	State Madeleine Albright
Igor Sergeev		Defense Secretary William Cohen
		British Foreign Secretary Robin Cook
		Russian Defense Minister Igor Sergeev
<OTHERS>	<OTHERS>	<OTHERS>
NATO	NATO	G-8
U.N.	U.N.	Representatives
Albanians	Albanians	Contact
Bosnian	Bosnian	Albanians
		NATO
		Macedonia
		Kosovo Albanian
		U.N.
		Bosnian
		Reuters
		BBC
		Britain

Chapter 5
A News Document-Filtering Agent

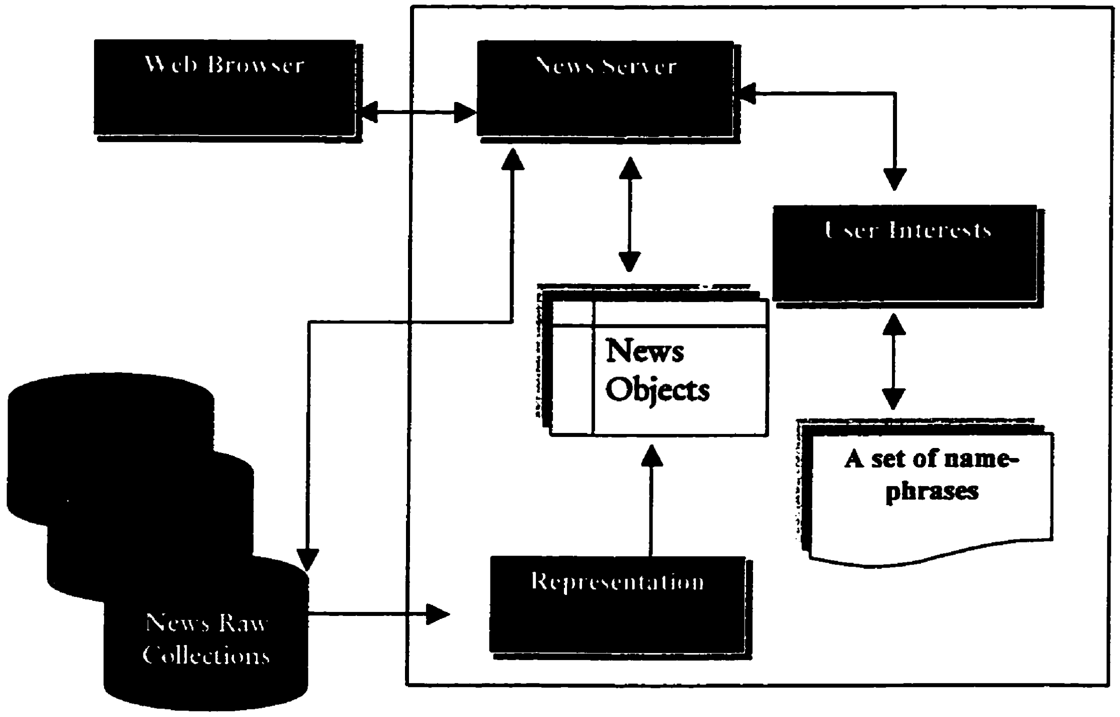
The *news document-filtering agent* is a computer program that filters on-line news from the Internet according to the personal interests of the user, and presents the "strongest" match to the user. The agent interacts with the user to retrieve documents from the on-line news based on name-phrase similarities. The name-phrases used for searches are those associated with a given news objects. The filtering service supplies a set of tools to quickly create a news object, display information about a news object and calculate the similarity between two news objects. They let the user pick a topic, set parameters for similarity and return a set of related articles.

First we describe the architecture of the news document-filtering agent. Second we describe a simple example to show how the service works. Finally we discuss how to create a good search.

5.1 The architecture of news document-filtering agent

As shown in Figure 5.1, the News Document-Filtering Agent consists of four main modules – Representation, User Interests, Web Browser, and News Server. The Representation module generates an object for every news document gathered from a fixed number of resources. The User Interests module allows the user to express his or her interests in finer granularity (using a set of name-phrases) than the documents themselves. The Web Browser module provides a Web interface to present HTML documents. The News Server module establishes a connection with the internal filtering engine by the client CGI program. The user submits an interest search by selecting name-phrases. The News Server sends a CGI request to the filtering engine and receives the response. The internal filtering engine supports two kinds of searches. In the first case, the user can request the documents based on name-phrases searches. Say a user is interested in UN news reports and he or she submits a search consisting of two words "UN and UN General Assembly". The engine then returns a set of documents including these name-phrases. Second the user can request documents related to a news object search. Say a user is interested in documents similar to a news report titled "Israel adopts UN resolution on Lebanon withdrawal". The engine then returns a set of documents based on a threshold of similarity, which the user can set.

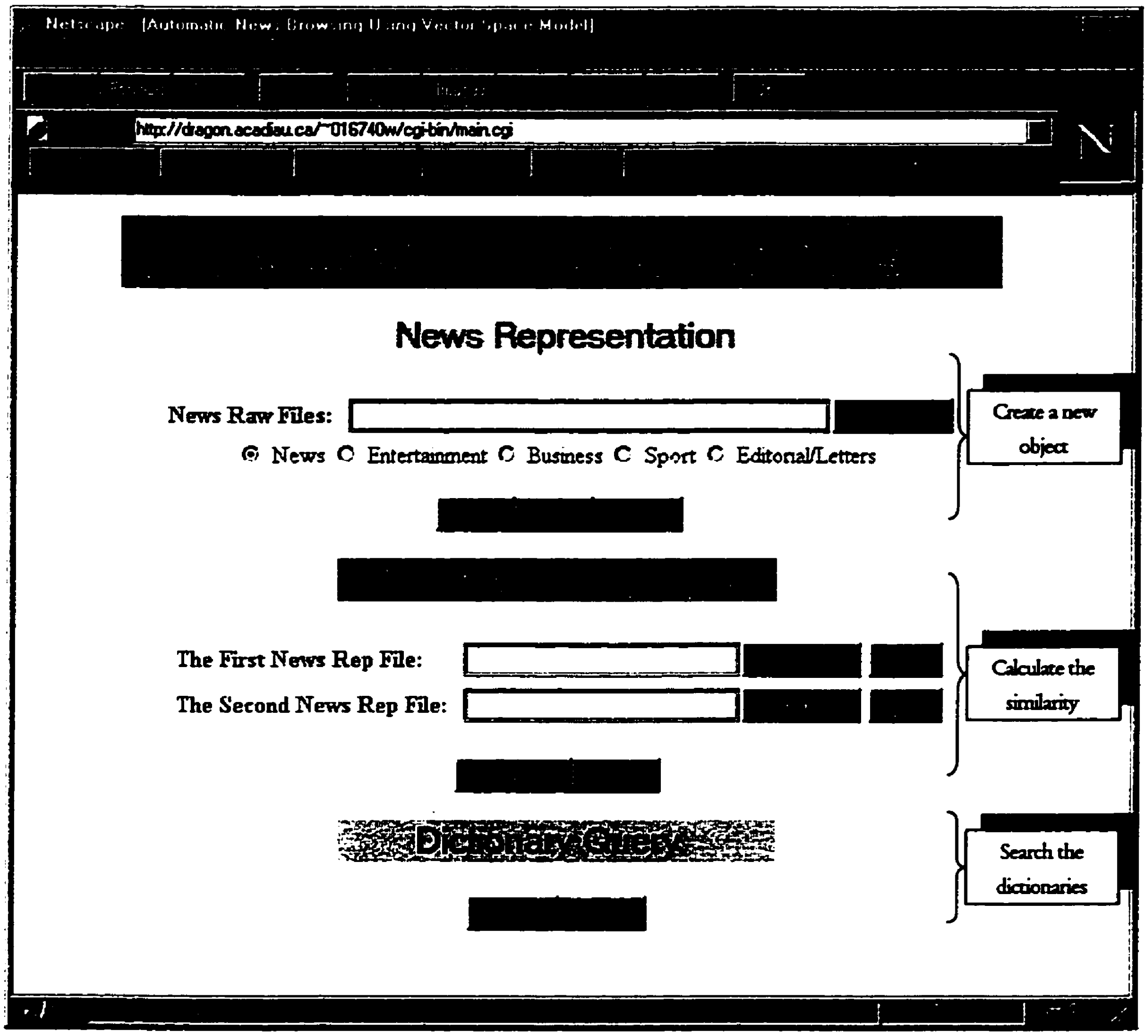
Figure 5.1 The architecture of a news document-filtering agent



5.2 News document-filtering agent prototype development

We describe a simple example to show how a news object is created, how the document-document similarity is calculated and how a user search is processed. There are two types of WWW interfaces used for two different tasks. The "Automatic News Filtering Tools" shown in Figure 5.2 is used for the internal filtering engine which helps the user create news objects and calculates the document-document similarities. The "Today's Headlines" shown in Figure 5.3 is a demo of the news filtering system which allows the user to browse today's headlines and search for documents based on our filtering algorithms (described in Chapter 3).

Figure 5.2 The interface of "Automatic News Filtering Tools"



Netscape - [Automatic News Browsing Using Vector Space Model]

http://dragon.acadiau.ca/~016740w/cgi-bin/main.cgi

News Representation

News Raw Files:

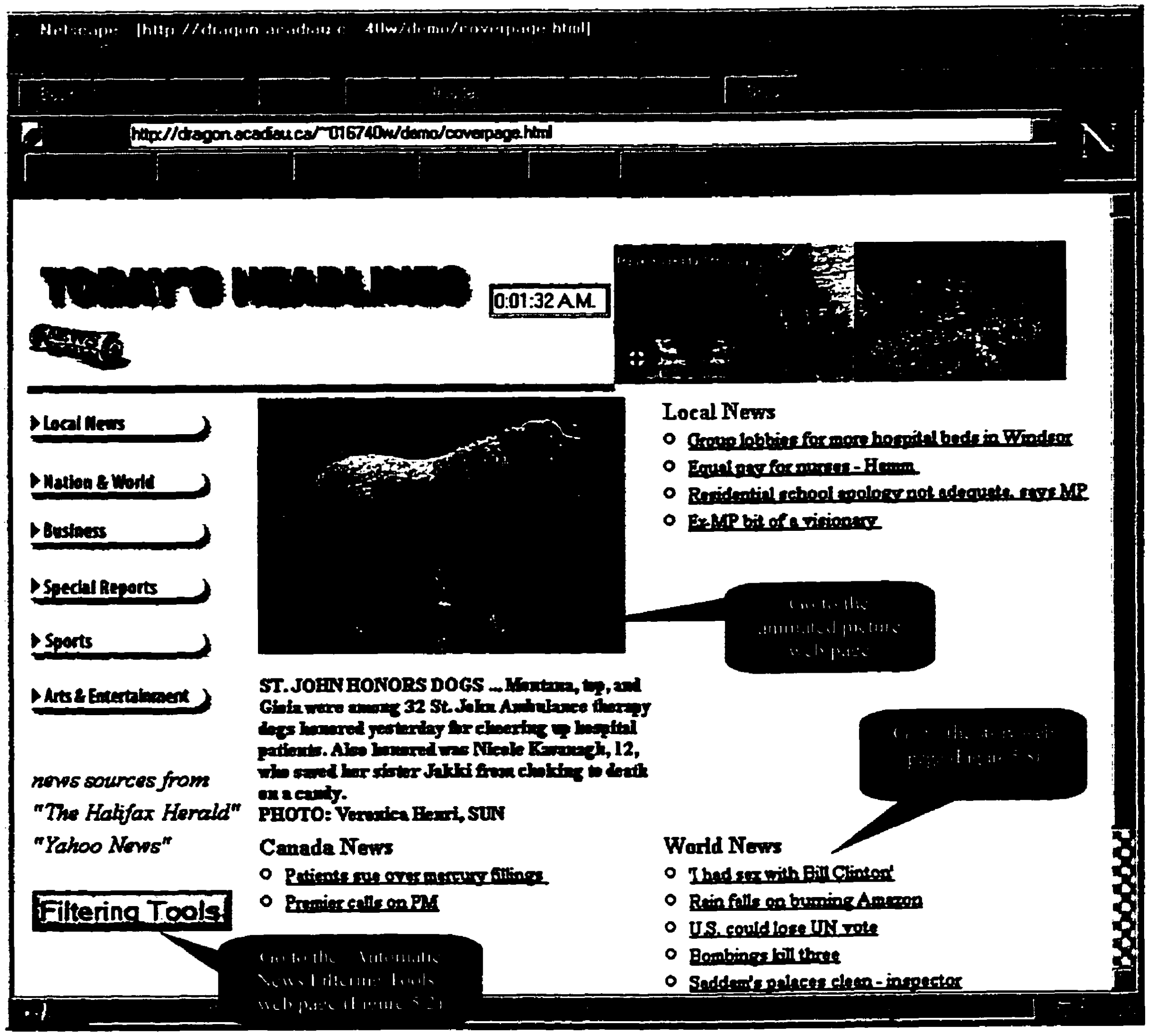
[✓ News] Entertainment Business Sport Editorial/Letters

The First News Rep File:

The Second News Rep File:

Dictionary Query

Figure 5.3   The interface of "Today's Headlines"



http://dragon.acadiau.ca/~016740w/demo/coverpage.html

TODAY'S HEADLINES 0:01:32 A.M.

Local News
- Group lobbies for more hospital beds in Windsor
- Equal pay for nurses - Hmm.
- Residential school apology not adequate, says MP
- Ex-MP bit of a visionary

ST. JOHN HONORS DOGS ... Montana, top, and Gizia were among 32 St. John Ambulance therapy dogs honored yesterday for cheering up hospital patients. Also honored was Nicole Kavanagh, 12, who saved her sister Jakki from choking to death on a candy.
PHOTO: Veronica Henri, SUN

news sources from
"The Halifax Herald"
"Yahoo News"

Canada News
- Patients sue over mercury fillings
- Premier calls on PM

World News
- 'I had sex with Bill Clinton'
- Rain falls on burning Amazon
- U.S. could lose UN vote
- Bombings kill three
- Saddam's palaces clean - inspector

5.2.1 Automatic News Filtering Tools

The WWW interface, "Automatic News Filtering Tools" shown in Figure 5.2, is used by the internal filtering engine. It includes three modules: create a news object from a news story, calculate document-document similarities, and conduct dictionary queries.

- **Create a news object and save its information as a rep file**

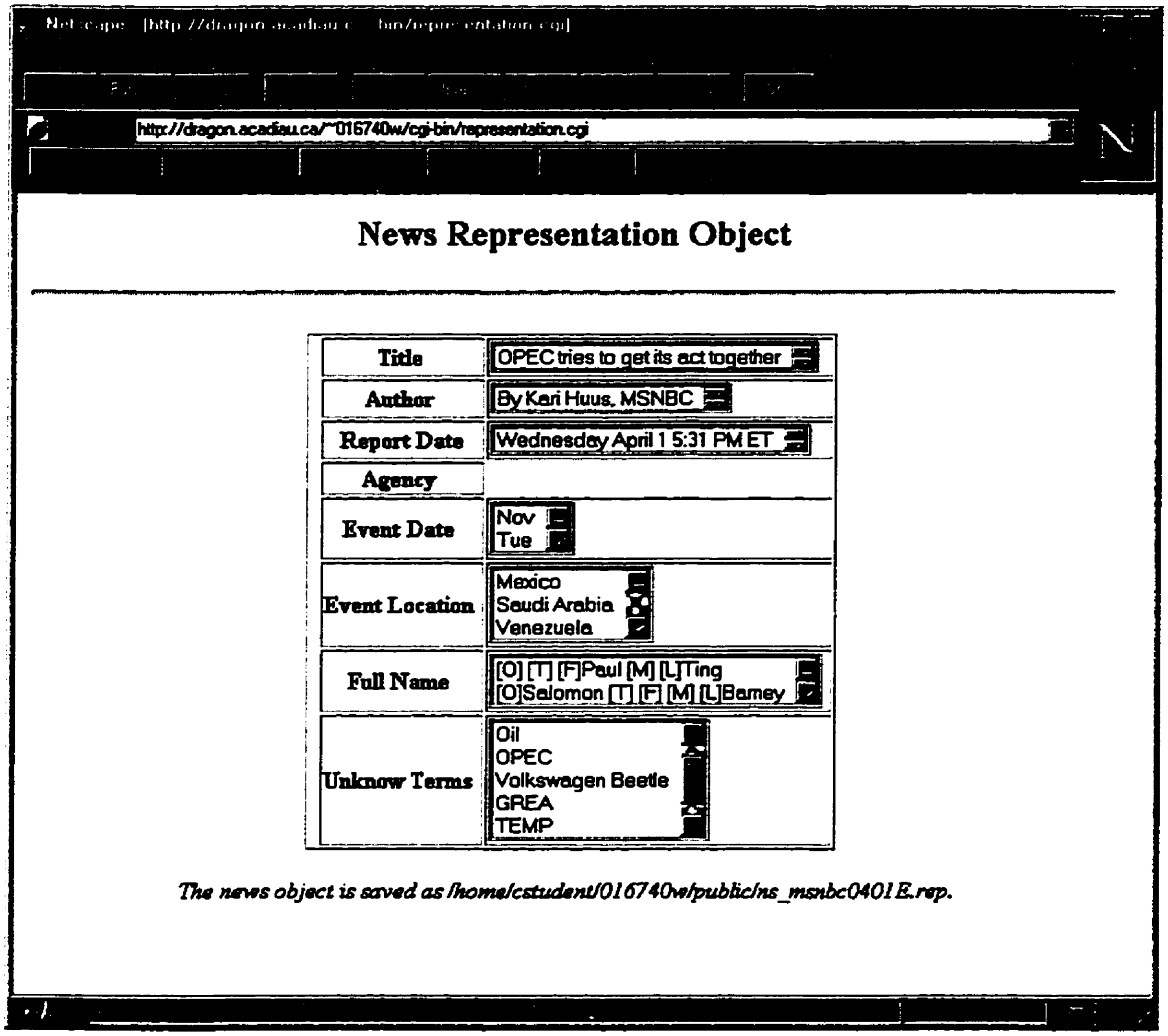
Suppose we have a raw news file named "msnbc0401E.raw", as follows. It is a marked up file using tags which are described in Chapter 3.

```
<PUBDATE>
Wednesday April 1 5:31 PM ET
</PUBDATE>
<HEADLINE>
OPEC tries to get its act together
</HEADLINE>
<BYLINE>
By Kari Huus, MSNBC
</BYLINE>
<CONTENT>
World oil producers have banded together to limit supply, but cooperation
is likely to prove short-lived once again.
...
</CONTENT>
```

After the user fills in the News Raw File field (using "msnbc0401E.raw" as example) and selects the "Build" button (shown in Figure 5.2). The result is shown as Figure 5.4.

Figure 5.4 The result of a news object creation

(using "msnbc0401E.raw" as example)



http://dragon.acadiau.ca/~016740w/cgi-bin/representation.cgi

News Representation Object

Title	OPEC tries to get its act together
Author	By Kari Huus, MSNBC
Report Date	Wednesday April 1 5:31 PM ET
Agency	
Event Date	Nov Tue
Event Location	Mexico Saudi Arabia Venezuela
Full Name	[O] [T] [F]Paul [M] [L]Ting [O]Salomon [T] [F] [M] [L]Barney
Unknow Terms	Oil OPEC Volkswagen Beetle GREA TEMP

The news object is saved as /home/cstudent/016740w/public/ns_msnbc0401E.rep.

The information of object is saved as "ns_msnbc0401E.rep" as follows.

```
<TITLE>
OPEC tries to get its act together
<AUTHOR>
By Kari Huus, MSNBC
<DATE>
Wednesday April 1 5:31 PM ET
<AGENCY>
<EVENTLOCATION>
Mexico
Saudi Arabia
Venezuela
<EVENTDATE>
Nov
Tue
<FULLNAME>
 Paul  Ting
Salomon  Barney
<OTHERS>
World
Oil
OPEC
Volkswagen Beetle
GREA
TEMP
Typically
```

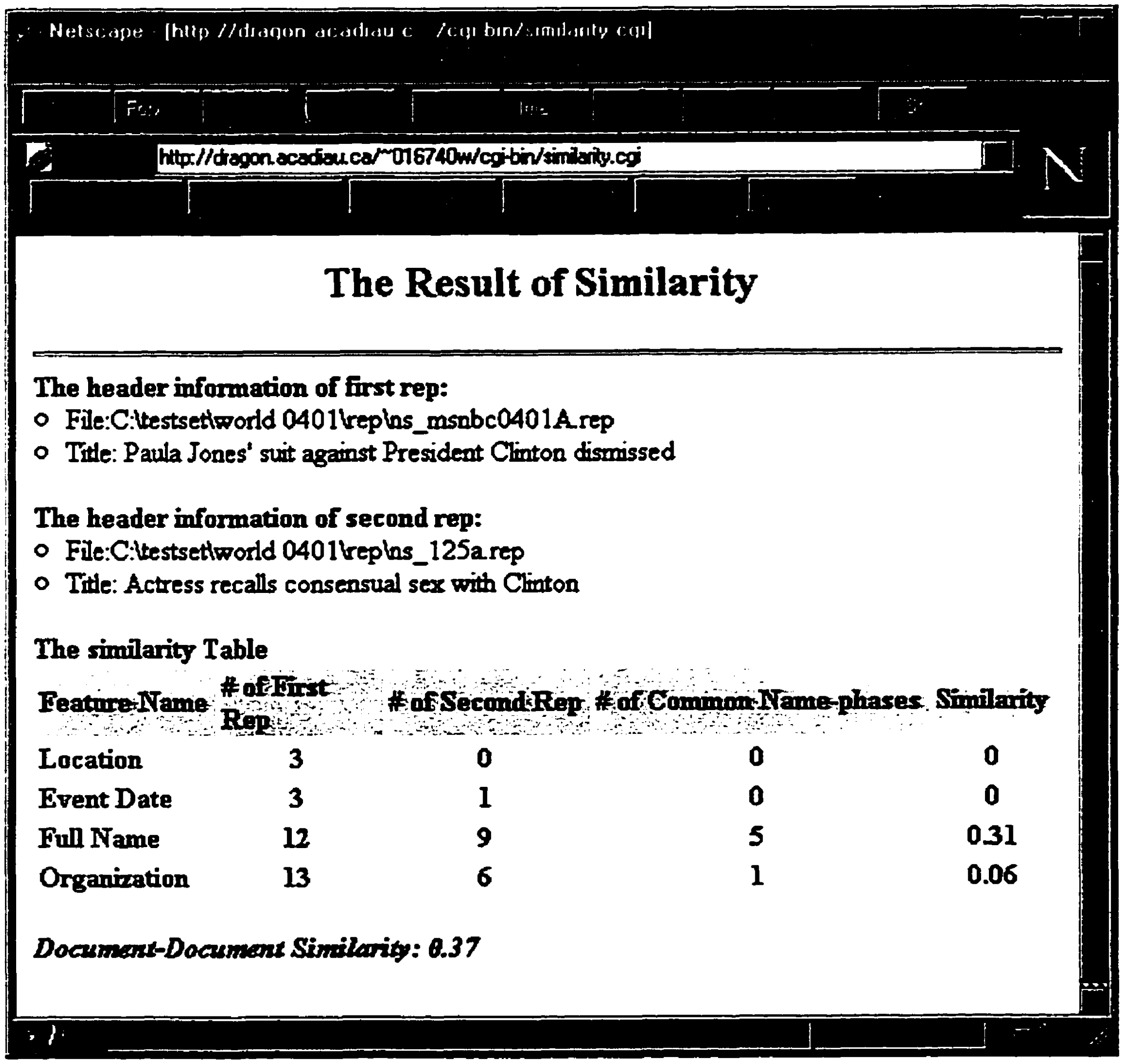
When the user selects "Read" button, he or she can check if the format of the file is correct. The module only accepts a valid file as described in Chapter 3.

This process is fast. After running the program 26 times using twenty-six different marked up files as input, it takes 2-3 seconds per document. These marked up files are obtained from Halifax Herald, MSNBC, and Reuters. They are total 90.4KB. The average space of each document is 3.48KB.

- **Calculate a document-document similarity**

Suppose the user wants to calculate the similarity of two news documents. After he or she fills in the First Rep File and the Second Rep File fields (using ns_msnbc0401.rep and ns_125a.rep for example) and selects the "Similarity" button (shown in Figure5.2), the result is shown as Figure 5.5.

Figure 5.5 The result of a document-document similarity


Netscape - [http://dragon.acadiau.c /cgi-bin/similarity.cgi]
http://dragon.acadiau.ca/~016740w/cgi-bin/similarity.cgi

The Result of Similarity

The header information of first rep:
- File:C:\testset\world 0401\rep\ns_msnbc0401A.rep
- Title: Paula Jones' suit against President Clinton dismissed

The header information of second rep:
- File:C:\testset\world 0401\rep\ns_125a.rep
- Title: Actress recalls consensual sex with Clinton

The similarity Table

Feature Name	# of First Rep	# of Second Rep	# of Common Name-phases	Similarity
Location	3	0	0	0
Event Date	3	1	0	0
Full Name	12	9	5	0.31
Organization	13	6	1	0.06

Document-Document Similarity: 0.37

It returns information about how many common feature name-phrases exist between two objects, and the value of similarity. For example, there are 5 common FullName name-phrase and 1 Organization name-phrase existing between the two objects (ns_msnbc0401A.rep and ns_125a.rep) shown in Figure 5.5. According to Eq. 3.2 described in Chapter 3, the important value X is calculated as:

$$X=1/(0+1+9+6)=1/16$$

According to Eq. 3.1 described in Chapter 3, the similarity value Y is calculated as:

$$Y= (1+5)/16 = 0.37$$

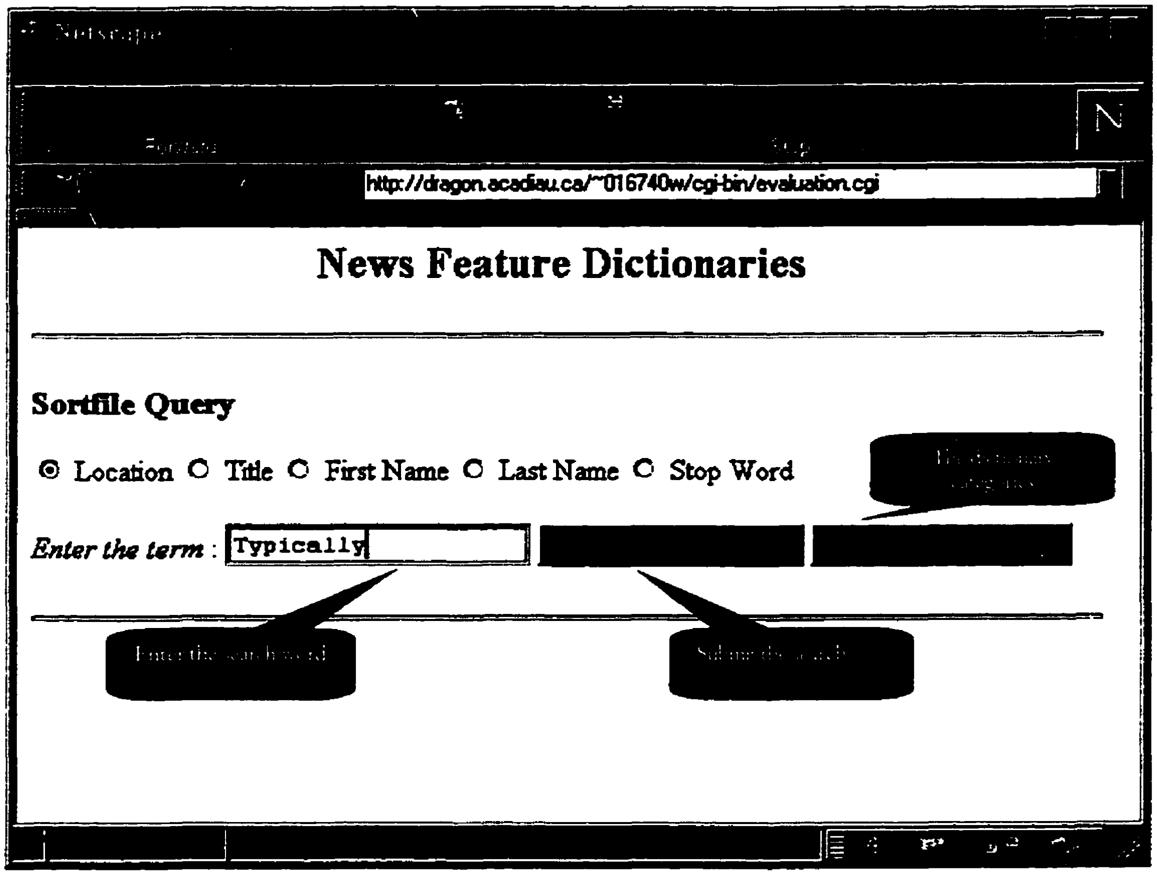
- Query dictionaries

This module supplies an interface (Figure 5.6) to improve the dictionaries. These dictionaries are used to match feature name-phrases and delete stop words. There are five different dictionaries used in our algorithms. They are: stoplist, place, title, first name and last name dictionaries. Each type of dictionary, except the stoplist, includes 26 sortfiles. A *sortfile* is a file using the first letter of its name as an index and all terms in this file begin with its index.

The user can query a dictionary by selecting a radio button. For example, the user may query if the word "Typically" is a stop word. As the example shown in Figure 5.4, the word "Typically" is associated with the news object. But it is not a meaningful phrase for describing an event. After getting the query result shown in Figure 5.7, the user then decides if the word "Typically" should be added to the stoplist. The user can analyze information associated with numerous news objects, and then improve dictionaries based on his experiences.

Here we do not provide add or delete function to dynamically modify dictionaries, because we do not want any internet user to modify them without protection. The user sends the message asking for modification to the server. The server is responsible for modifying these dictionaries.

Figure 5.6 The interface of the query dictionary



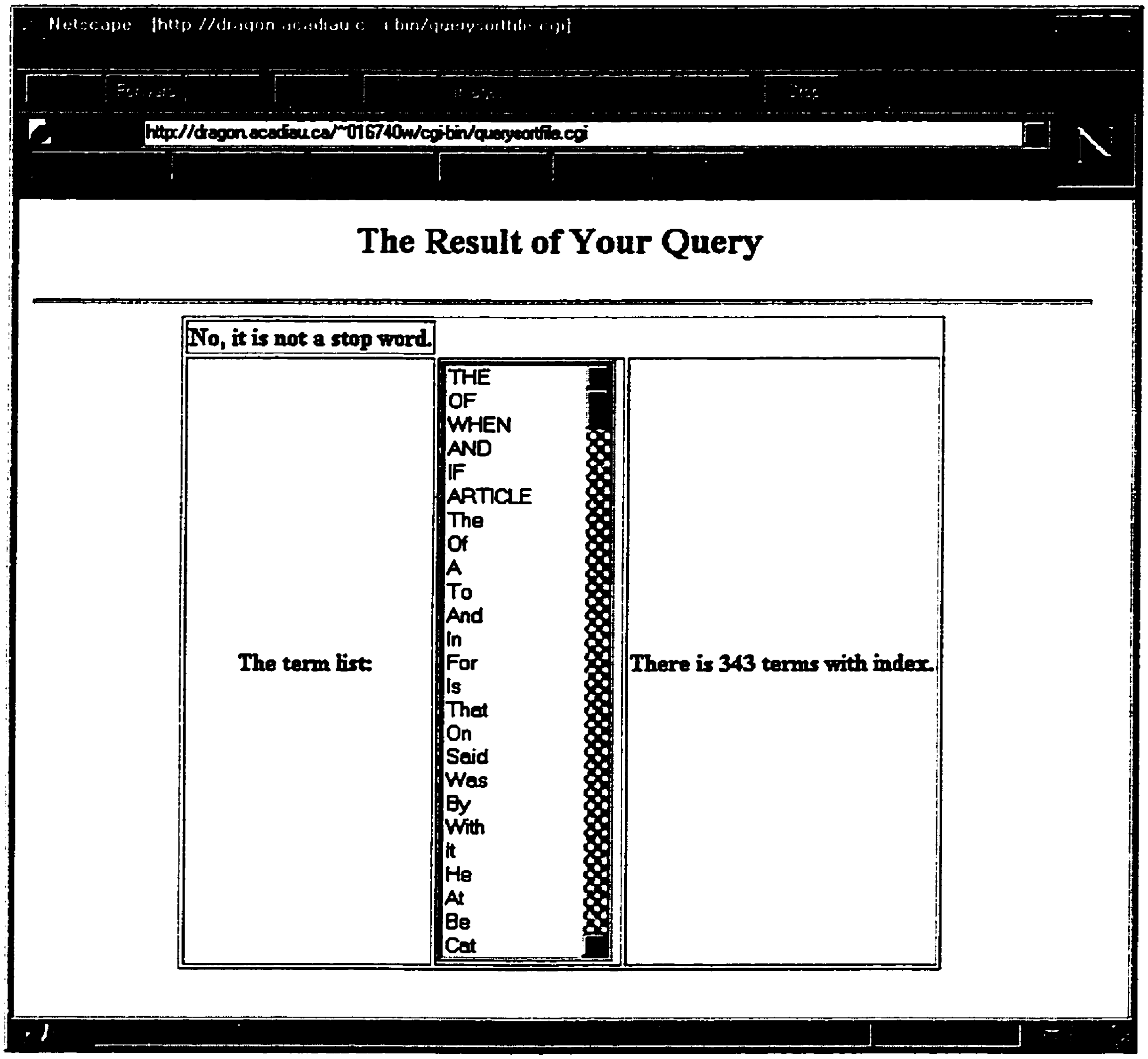
News Feature Dictionaries

Sortfile Query

[✓ Location] Title First Name Last Name Stop Word

Enter the term : Typically

Figure 5.7 The result of a dictionary query


Netscape - [http://dragon.acadiau.ca/~016740w/cgi-bin/querysortfile.cgi]

http://dragon.acadiau.ca/~016740w/cgi-bin/querysortfile.cgi

The Result of Your Query

No, it is not a stop word.

The term list:

THE
OF
WHEN
AND
IF
ARTICLE
The
Of
A
To
And
In
For
Is
That
On
Said
Was
By
With
It
He
At
Be
Cat

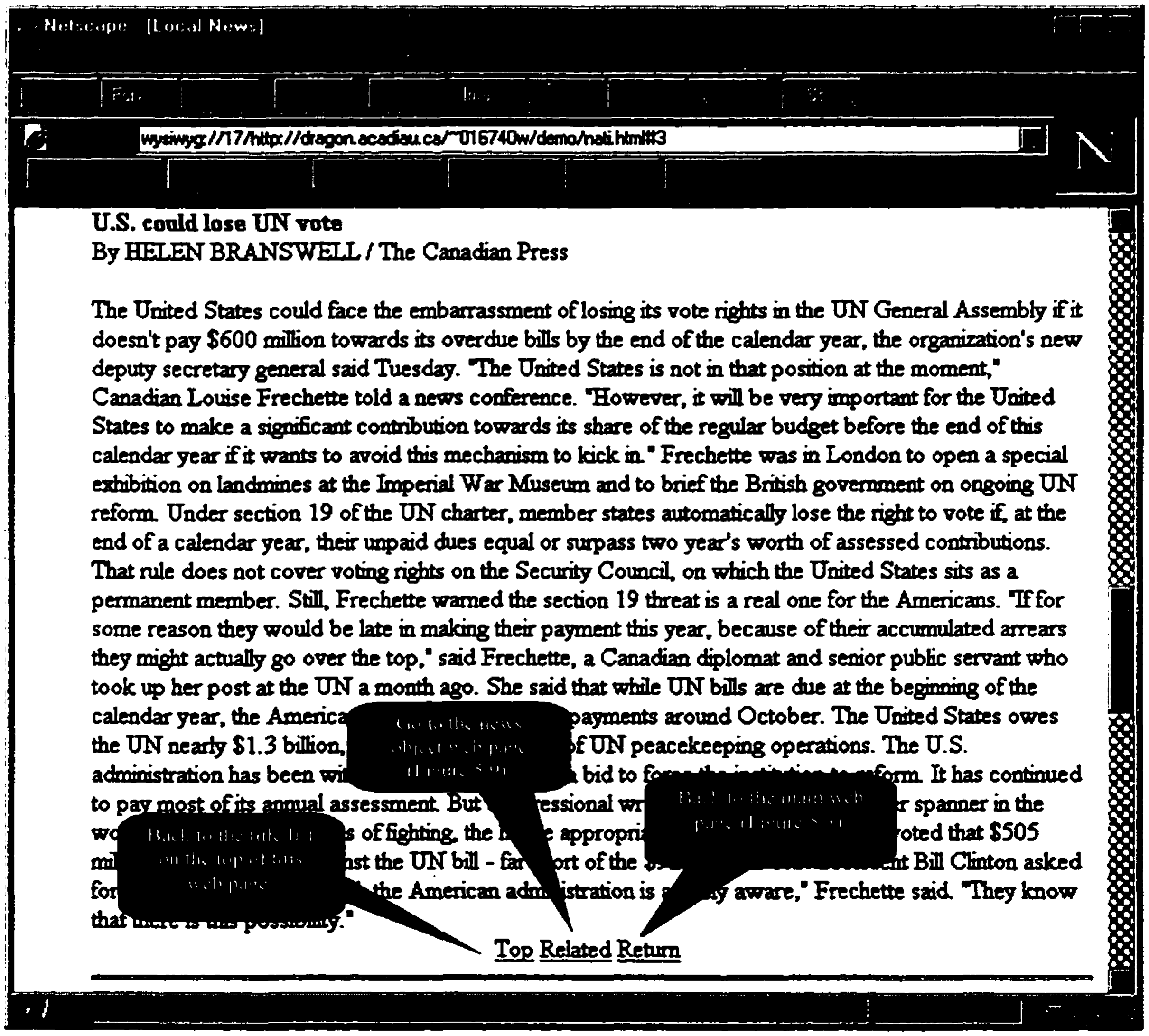
There is 343 terms with index.

5.2.2 Today's Headlines

The "Today's Headlines" WWW interfaces shown in Figure 5.3 supplies a set of services to let the user select news stories and find other related ones.

Suppose a user selects a news story titled " U.S. could lose UN vote" by clicking the hypertext link in Figure 5.3, as shown in Figure 5.8.

Figure 5.8 The news story web page


Netscape - [Local News]
wysiwyg://17/http://dragon.acadiau.ca/~016740w/demo/nati.html#3
Go to the news object web page (Figure 5.9)
Back to the title list on the top of this web page
Back to the main web page (Figure 5.3)

U.S. could lose UN vote
By HELEN BRANSWELL / The Canadian Press

The United States could face the embarrassment of losing its vote rights in the UN General Assembly if it doesn't pay $600 million towards its overdue bills by the end of the calendar year, the organization's new deputy secretary general said Tuesday. "The United States is not in that position at the moment," Canadian Louise Frechette told a news conference. "However, it will be very important for the United States to make a significant contribution towards its share of the regular budget before the end of this calendar year if it wants to avoid this mechanism to kick in." Frechette was in London to open a special exhibition on landmines at the Imperial War Museum and to brief the British government on ongoing UN reform. Under section 19 of the UN charter, member states automatically lose the right to vote if, at the end of a calendar year, their unpaid dues equal or surpass two year's worth of assessed contributions. That rule does not cover voting rights on the Security Council, on which the United States sits as a permanent member. Still, Frechette warned the section 19 threat is a real one for the Americans. "If for some reason they would be late in making their payment this year, because of their accumulated arrears they might actually go over the top," said Frechette, a Canadian diplomat and senior public servant who took up her post at the UN a month ago. She said that while UN bills are due at the beginning of the calendar year, the America payments around October. The United States owes the UN nearly $1.3 billion, f UN peacekeeping operations. The U.S. administration has been wi bid to form. It has continued to pay most of its annual assessment. But essional wr r spanner in the wo s of fighting, the e appropri oted that $505 mi nst the UN bill - far ort of the nt Bill Clinton asked for the American administration is y aware," Frechette said. "They know that"

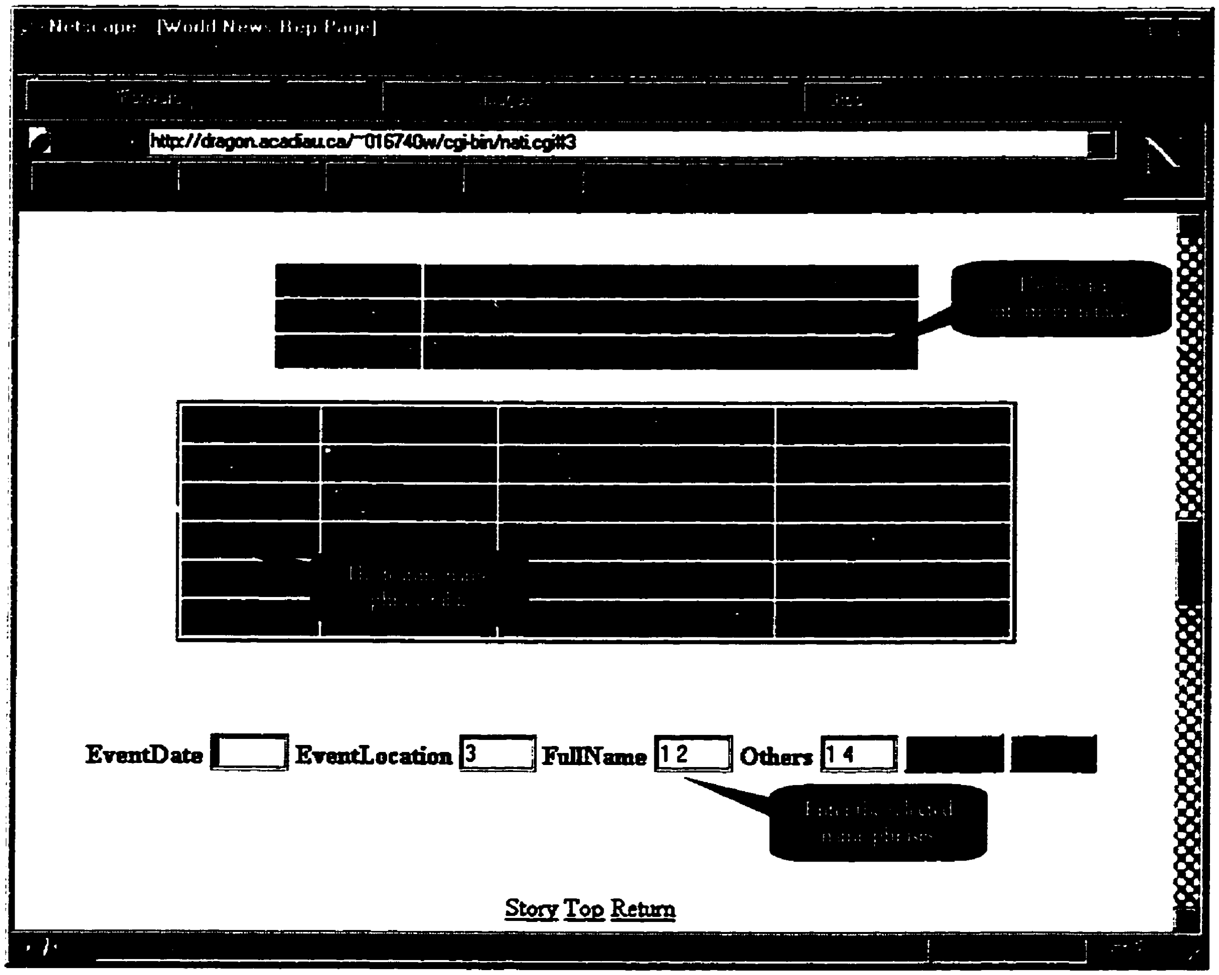
Top Related Return

The user then can retrieve documents from our Today Headlines news collection based on name-phrases search shown in Figure 5.9. The news server creates the news objects for every document in the collection and displays them in the news object Web page. The user clicks the "related" hyperlink shown in Figure 5.8 to navigate to the news object Web page. The header information associated with the news object is shown in the header table. The feature name-phrases associated with the news object are listed in the feature table. There are four columns, Event Date, Event Location, FullName and Others. Based on the name-phrase set, the user can pick some name-phrases to submit a search. The news server finds the documents in the collection including all or partial name-phrases and returns their hypertext links to the user. The user follows these links to find out the related stories.

In this demo the news collection includes 17 documents on April 1, 1998 from Halifax Herald, MSNBC and Reuters.

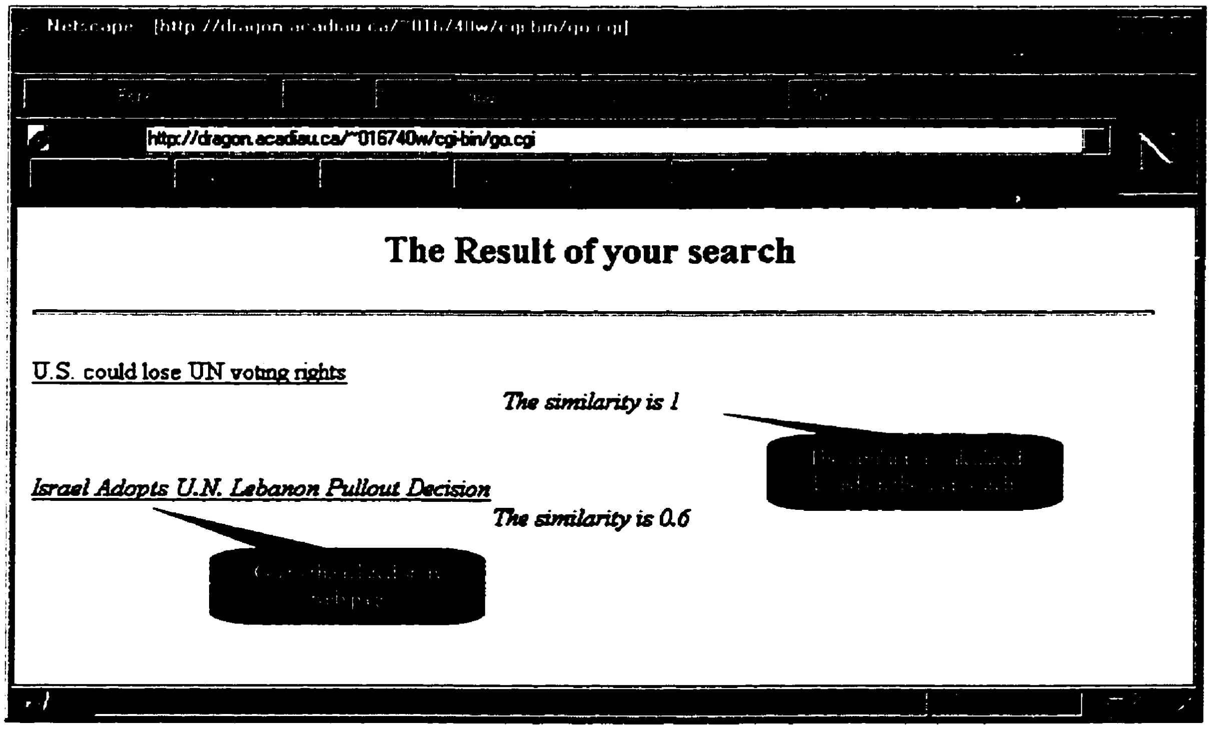
Figure 5.9 shows the input for a user search. In this example, the user picked the EventLocation name-phrase "London", the FullName name-phrase "Canadian Louise Frechette" and "President Bill Clinton", and the Others name-phrase "UN General Assembly" and "Security Council".

Figure 5.9 The name-phrase set search


Netscape [World News Rep Page]
http://dragon.acadiau.ca/~016740w/cgi-bin/nati.cgi#3
EventDate
EventLocation 3
FullName 1 2
Others 1 4
Story Top Return

The news server returns the result shown in Figure 5.10. The report titled "U.S could lose UN voting rights" is the best match. It includes all name-phrases appearing in the user search. According to Eq. 3.1, the similarity between this document and the search is 1 (5/5). The report titled "Israel Adopts U.N. Lebanon Pullout Decision" is also returned as the relevant documents. It includes three name-phrases appearing in the user search. According to Eq. 3.1, the similarity between this document and the search is 0.6 (3/5).

Figure 5.10 The return of a name-phrase set search



The Result of your search

U.S. could lose UN voting rights

The similarity is 1

Israel Adopts U.N. Lebanon Pullout Decision

The similarity is 0.6

5.3 How to pick a good search

To effectively filter the information, the user must be careful in the search phase. For example, if the user selects only one word such as "Tuesday", the word "Tuesday" is likely to match many articles that just happen to have an occurrence of that date value. If the user describes his or her interests in more detail, such as giving a fullname name-phrase or location name-phrase, the results will be far better.

Chapter 6
Conclusion

6.1 Results of the Study

The methodology presented in this thesis will assist news delivery system development in several different ways:

1. It provides a fast way of sifting through news documents for meaningful information that can be used in linking related stories.

2. It further assists an automatic text categorization system to classify news document into pre-defined categories, thereby grouping related ones together.

3. It presents a relationship (the same, similar or different) between two news documents. Hence it provides additional knowledge for the learning process.

This name-phrase based filtering engine works well for news articles which typically deal with a particular event, person, place, or thing and have enough name-phrase clues in the document that can be exploited.

There are shortcomings to a name-phrase based approach to filtering. First, the user cannot always get the concept of an event directly from feature name-phrases. Second, the threshold of similarity between two news objects needs to be kept low in order to improve the recall. The optimal threshold value was found to be approximately 0.3.

6.2 Future Work

There are two interesting directions for future work concerning this news document-filtering algorithm. One area of future work is to improve the quality of name-phrases selected from the original text. This includes improving the algorithm for extracting proper names and creating a word-pattern for dealing with synonyms. The other direction for future work is to weight name-phrases associated with a news object.

Algorithm optimization for extracting proper names and discovering word-pattern

According to different types of news stories, different algorithms may be used for extracting proper names. It was found that the algorithms used for business news stories should deal with numbers and company names, and the algorithms used for sport news should deal with scores of games. For example, the scores should be associated with one or more given person names or team names.

A *word-pattern* is a combination of words that deals with the variety of ways in which text can be written. For example, "ABC Bank", "ABC Bank Beijing" and "ABC Bank of Ottawa" refer to the same organization. In further implementation, the algorithms should consider a more sophisticated technique to discover some general word-patterns to improve the performance of calculating the news object overlap.

Algorithm optimization for improving the recall

An efficient way to improve the recall is for the algorithm to provide a set of rules to weight feature name-phrases either through calculating the frequencies of name-phrases or manually weighting feature name-phrases by inspection. Future news filtering system development should get feedback from the users, and then automatically adjust the weights of the name-phrases.

Bibliography

[1] C.R. Watters, F.J.Burkowski, Michael Shepherd: *Introduction*, Information Processing & Management Volume 33, No. 5, 1997.

[2] C. Carrick, and C.R. Watters: *Automatic association of news item*, Information Processing and Management Volume 33, No.5, 1997.

[3] C.R. Watters, M.A. Shepherd, and F.J.Burkowski: *Electric News Delivery Project*, Journal of the American Society for Information Science Volume 49, No.2, 1998.

[4] C.R. Watters, M.A. Shepherd, T. Chiasson, and L. Manchester: *An evaluation of two metaphors for electronic news presentation (Tech. Rep.)*, Halifax, Nova Scotia, Canada: Dalhousie University, Computing Sciences Division, Department of Mathematics, Statistics & Computing Science.

[5] John E. Newhagen: *The role of feedback in the assessment of news*, Journal of the American Society for Information Science Volume 49, No.2, 1998.

[6] Janne S, Marko T and Tuomas P: *Logical structure of a hypermedia newspaper*, Journal of the American Society for Information Science Volume 49, No.2, 1998.

[7] Kenrick J. M and V. Rao V: *Information filtering via hill climbing, wordnet, and index pattern*, Journal of the American Society for Information Science Volume 49, No.2, 1998.

[8] Tak W. Yan, Hector Garcia-Molina: *SIFT – A Tool for Wide-Area Information Dissemination*, in Proceedings of the 1995 USENIX Technical Conference, page 177-86, 1995.

[9] Lisa F. Rau: *Industrial Applications of NLP.*
http://www.cis.upenn.edu/˜cliff-group/94/lrau.html

[10] Lis F. Rau: *Extraction company names from text*, Seventh IEEE Conference on Artificial Intelligence Applications P.29-32, 1991.

[11] Lisa F. Rau: *Industrial Applications of NLP – Text clustering, Summarization and automatic extraction.*
http://www.cis.upenn.edu/˜cliff-group/94/lrau.html#ref-lfr:summarization

[12] *Domain-Independent Summarization of News.*
http://www.fh-hannover.de/ik/Dagstuhl/Abstract/Abstracts/Rau/Rau.html

[13] P.S. Jocobs, L.F. Rau: *SCISOR: extracting information from on-line news*, Communications of the ACM Vol: 33 ISS: 11 p.88-97, Nov. 1990.

[14] Gerard Salton, Michael J. McGill: *Introduction to Modern Information Retrieval*, McGraw-Hill Book Company 1983.

[15] John Consoli: *Who's using the Web? FIND/SVP survey result revealed*, Editor & Publisher Interactive Conference July 31 - August 2, 1997.

[16] Alper K. Caglayan, Colin G. Harrison: *Agent Source Book*, Wiley Computer Publishing, 1997.

[17] *NewsHound.*
http://www.hound.com/

[18] Verity Incorporated: *Introduction to Topics Guide V2.0 Jones, Mary.*
http://www.verity.com

[19] T. Kamba and M.C. Albers: *The Krakatoa Chronicle – An interactive, personalized, newspaper on the web,* in Proceedings of the Fourth International World Wide Web Conference. Boston, Mass. 1995.

[20] *SMART.*

http://www.cs.cornell.edu/pub/smart

[21] *Relevance Feedback and Probabilistic Models in IR.*

http://ai.bpa.arizona.edu/papers/mlir93/

[22] Marko Balabanovic, Yoav Shoham and Yeogirl Yun: *A Demonstration of the LIRA System.*

http://robotics.stanford.edu/people/marko/lira/

[23] *TAPESTRY NLP TOOLKIT.*

http://www.krdl.org.sg/RND/knowledge/MT/

[24] J. Broglio, J. Callan, and W.B. Croft: *Proceedings of the TIPSTER Text Program--Phase I,* the INQUERY system, R. Merchant, editor, San Mateo, California, 1993.

[25] *Text REtrieval Conferences (TREC).*

http://trec.nist.gov/

[26] *Named Entity Task Definition (MUC).*

http://www.cs.bu.edu/courses/cs545/F96/papers/ne-task-def.v2.1.txt

[27] D. C. Blair: *Language and Representation in Information Retrieval,* Elsevier, Amsterdam (Holland), 1990.

[28] *News writing Tips.*

http://www.dinfos.osd.mil/jwebsite/News/Leads.htm

[29] M.T. Maybury: *Generating summaries from event data,* Information Processing & Management Vol: 31 Iss: 5 p. 735-51.

[30] *When less means more – information extraction in practice*

http://www.iss.nus.sg/RND/MLP/Projects/TCA/ta.html

Institute of Systems Science National University of Singapore.

[31] Gerard Salton: *Automatic Text Processing.* MA: Addison Wesley, 1989.

[32] C. Carrick: *Automatic association of news items in a multimedia environment.* Unpublished master's thesis, Department of Mathematics, Statistics & Computing Science, Dalhousie University, Halifax, Canada. 1995.

Appendix A

The Explanation of the Core Program Modules

We supply a set of object-oriented core of PERL modules. The "buildrep.pm" and "fullname.pm" modules are used to extract feature name-phrases from a news document and instantiate them to the news object. The "newsitem.pm" module is used to create a news object. The "simcom.pm" module is used to calculate the similarity between two objects. The "topicdb.pm" module is used to save or restore an object. The "newprint.pm" module is used to present information associated with an object via the WWW interface. Based on methods inside these modules, it is easy for the user to develop a customer news filtering agent system using our filtering algorithms.

Following we describe how to call these methods.

1. Create a new object

```
use buildrep;

use newsitem;

use fullname;



$item = new newsitem;

@buildbody = &buildrep::newstitle(\@input,$item);

@left = &buildrep::newsitem(@buildbody, $item);

&fullname::fullname(@left, $item);
```

2. Get information associated with an object.

```
use newsitem;

@location = $object->getLocation;

@eventdate = $object->getEventdate;

@fullname = $object->getFullname;

@others = $object->getOthers;
```

3. Save an object to a rep file or restore an object from a rep file:

```
use topicdb;

&topicdb::save_newsitem($object, $repfile);

$object = &topicdb::restore_newsitem($repfile);
```

4. Calculate the similarity between two objects:

```
use simcom


%similarity = &simcom: sim_newsobject($object1, $object2)
```

5. Print information associated with an object via the WWW interface

```
do "newsprint.pl";


&table_terms($object);
```

Appendix B

A Perl Module for Creating a News Object

```
#!/loc/bin/perl -w

package newsitem;


#######################################################
##
##    CLASS: newsitem.pm
##  PURPOSE: To encapsulate the data and methods for
##           creating a news representation object.
##   AUTHOR: Hong Wan
##  CREATED: 18, Nov 1997
## MODIFIED: 20, Mar 1998
##
#######################################################

##------------ Constructor --------------------------##

sub new{
    my $self ={};
    #$self -> _initialize;
    bless($self);
    return $self;
}

sub _initialize{
    # Initialize data members
    $self ->{'title'} = undef;
    $self ->{'author'} = undef;
    $self ->{'date'} = undef;
    $self ->{'name'} = undef;
    $self ->{'location'} = undef;
    $self ->{'eventdate'} = undef;
    $self ->{'others'} = undef;
}

##---------------- Accessors ------------------------##
sub addFullname{
    my $self = shift;
    $other = $_[0];
    $title = $_[1];
    $first = $_[2];
    $middle = $_[3];
    $last = $_[4];
    $rec = {};
    $rec->{'other'} = $other;
    $rec->{'title'} = $title;
    $rec->{'first'} = $first;
    $rec->{'middle'} = $middle;
    $rec->{'last'} = $last;
    push @{$self ->{'fullname'}}, $rec;
}
```

```
sub getFullname{
    my $self = shift;
    return @{$self -> {'fullname'}};
}

sub addTitle{
    my $self = shift;
    if (@_) {@{$self ->{'title'}} = @_}
    #return @{$self ->{'title'}};
}

sub getTitle{
    my $self = shift;
    return @{$self ->{'title'}};
}

sub addAuthor{
    my $self = shift;
    if (@_) {@{$self ->{'author'}} = @_}
    #return @{$self ->{'author'}};
}

sub getAuthor{
    my $self = shift;
    return @{$self ->{'author'}};
}

sub addDate{
    my $self = shift;
    if (@_) {@{$self ->{'date'}} = @_}
    #return @{$self ->{'date'}};
}

sub getDate{
    my $self = shift;
    return @{$self ->{'date'}};
}

sub addAgency{
    my $self = shift;
    if (@_) {@{$self ->{'agency'}} = @_}
    #return @{$self ->{'agency'}};
}

sub getAgency{
    my $self = shift;
    return @{$self ->{'agency'}};
}

sub addLocation{
    my $self = shift;
    if (@_) {@{$self ->{'location'}} = @_}
    #return @{$self ->{'location'}};
}
```

```
sub getLocation{
    my $self = shift;
    return @{$self ->{'location'}};
}

sub addEventdate{
    my $self = shift;
    if (@_) {@{$self ->{'eventdate'}} = @_}
    #return @{$self ->{'eventdate'}};
}

sub getEventdate{
    my $self = shift;
    return @{$self ->{'eventdate'}};
}

sub addOthers{
    my $self = shift;
    if (@_) {@{$self ->{'others'}} = @_}
    #return @{$self ->{'others'}};
}

sub getOthers{
    my $self = shift;
    return @{$self ->{'others'}};
}

1;
```

Appendix C

A Perl Module for Extracting EventDate and EventLocation from

News Documents

```
#!/loc/bin/perl -w

package buildrep;

######################################################
##
##     CLASS: buildrep.pm
##   PURPOSE: To extract EventDate and EventLocation
##            from a news document
##    AUTHOR: Hong Wan
##   CREATED: 18, Jan 1998
##  MODIFIED: 20, Mar 1998
##
######################################################

use dbm_create;
use FileHandle;

sub newstitle{
    my($item) = pop @_;
    #local($file) = @_;
    my($titlelines) = @_;

    my @author = ();
    my @title = ();
    my @date = ();
    my @body = ();

    my $flag = 0000;

    Next: foreach $_(@$titlelines){    ## extract header information
       if(/<HEADLINE>/){
         $flag=1000;
         next Next;
       }
       if(/<\/HEADLINE>/){
         $flag=0000;
         next Next;
       }
       if($flag == 1000){
         push @title, $_;
         next Next;
       }
       if(/^<BYLINE>$/){
         $flag=0100;
         next Next;
       }
       if(/<\/BYLINE>/){
         $flag=0000;
         next Next;
       }
       if($flag == 0100){
         push @author, $_;
         next Next;
       }
       if(/^<PUBDATE>$/){
         $flag=0010;
```

```
          next Next;
        }
        if(/<\/PUBDATE>/){
          $flag=0000;
          next Next;
        }
        if($flag == 0010){
          push @date, $_;
          next Next;
        }
        if(/^<CONTENT>$/){
          $flag=0001;
          next Next;
        }
        if(/<\/CONTENT>/){
          $flag=0000;
          next Next;
        }
        if($flag == 0001){
          push @body, $_;
          next Next;
        }
     }
    ## set value to the object ##
    if(scalar(@title)){$item ->addTitle(@title);} ## in case null array
    if(scalar(@author)){$item ->addAuthor(@author);}
    if(scalar(@date)) {$item ->addDate(@date);}
    #if(scalari(@agency)){$item ->addAgency(@agency);}

    return @body;
}

sub newsitem{

## Read content information into array ##
local($item) = pop @_;
local(@lines) = @_;

## Split each line into single words and save them into an array ##
foreach $eveline(@lines){
   @chars = split / /, $eveline;
   WORD: foreach $_(@chars){
      if(/\s/) {chop $_; } ## del \n
      if(/^\W*$/){next WORD;} ## skip blank line or symbols
      #if(/^\W*(\w+)(\)\W*)$/) {push @bodywords, $1}
      #elsif(/^\((\w+\W*)/) { push @bodywords, $1}
      else {
            push @bodywords, $_;}
   }
}

my($mayflag) = 0;

## May
NEXTBODY: foreach $_(@bodywords){
   $mayflag++;
   if (/^May$/ || /^May\.$/) {
```

```
                    $pre_word = $bodywords[$mayflag-1];
                    if (($pre_word =~ /\w+\.$/) || ($pre_word =~ /\w+\;$/)){
                          Next NEXTBODY;
                    }
                    else { /(\w\w\w)/; push @eventdate, $1;}
     }
}

@phase = &cre_phase(\@bodywords);


## Read stoplist into an array ##
@stoplines = &locate_stopwords;

## Read month dictionary into an array ##
@monthpattern = &locate_month_patterns;

## Read week dictionary into an array ##
@weekpattern = &locate_week_patterns;

NEXT: foreach $word(@phase){

   ## dash
   if ($word =~ /(\w+-.*\w+)\W*$/){ push @forname, $1; next NEXT;}
   #@split = split /$$/, $word;
   ## incase B.C.
   if ($word =~ /([A-Z].[A-Z].).*$/) {
     push @checkplace, $1;
   }

  else{
   my($i) = 0;
   @split = split /$$/, $word;
NEXTTERM:    foreach $_(@split){
        $i++;
        /^\W*(\w+).*$/;
        $match = $1;   #
        $get = &extract_term(@stoplines, $match);
        if($get !~ /$match/)
          {
          if(/^(\w+\.).*$/){$match = $1;}    ## in case Feb.
          $get = &extract_term(@monthpattern, $match);
          if($get =~/$match/){
             $get =~ /(\w\w\w)/;
             push @eventdate, $1;}
          else{$get = &extract_term(@weekpattern, $match);
              if($get =~/$match/){ $get =~ /(\w\w\w)/; push @eventdate,
$1;}
               elsif($i > 1){ $save = join "$$", $save, $match;}
                    else{ $save = $match;}
              }
         }
         else { $i--; next NEXTTERM;}
   }
   push @checkplace, $save;
  } #else
}
```

```
## Define key set of place ##
@KEYNAME = ("A","B","C","D","E","F","G","H","I","J","K","L","M",
            "N","O","P","Q","R","S","T","U","V","W","X","Y","Z");

%place = &hash_sortfiles(@KEYNAME,"/home/cstudent/016740w/www/cgi-
bin/dbm/sortfiles/world_places_new");

## Extract EventLocation from the proper name set ##
foreach $_(@checkplace){
   /^\W*(\w)/;
   @places = @{$place{$1}};
   $get = &extract_place(@places, $_);
   if(/$get/){
      push @eventlocation, $get;
   }
   else {
         ## River, City and Mountain
         if((/River$/)||(/City$/)||(/Mountain$/)){
             push @eventlocation, $_;
          }
         else{
              push @nextstep, $_;}
         }
}


if(@eventdate != ()){ @finaldate = &del_duplicate(@eventdate);
                      $item ->addEventdate(@finaldate);}
if(@eventlocation != ()){ @finallocation =
&del_duplicate(@eventlocation);
                      $item ->addLocation(@finallocation);}

## process duplicated words and 's and symbol ##
FORNAME: foreach $_(@nextstep){
  if(/^\W*$/){ next FORNAME;}
  if(/^(.*?)'s$/){ push @forname, $1} ## incase Jan.'s
  else {push @forname, $_;}
}
@pre_final = &del_duplicate(@forname);
@final = &del_loose_duplicate(@pre_final);

return @final;
}

## Read stoplist into an array ##
sub locate_stopwords {
## Read stopwords into array ##
$stopfile = "/home/cstudent/016740w/www/cgi-bin/stoplist.txt";
open (STOPFILE, $stopfile) or print "Cannot open stoplist file!$! \n";
@stoplines = <STOPFILE>;
chop(@stoplines);           # remove newline
close(STOPFILE);

return @stoplines;
}
```

```
## Read week dictionary into an array ##
sub locate_week_patterns {

$weekfile = "/home/cstudent/016740w/www/cgi-bin/week.txt";
open(WEEKFILE, $weekfile) or print "Cannot open week.txt !$! \n";
@datepatterns = <WEEKFILE>;
close(WEEKFILE);

return @datepatterns;
}

## Read month dictionary into an array ##
sub locate_month_patterns {

$monthfile = "/home/cstudent/016740w/www/cgi-bin/month.txt";
open(MONTHFILE, $monthfile) or print "Cannot open month.txt !$! \n";
@datepatterns = <MONTHFILE>;
close(MONTHFILE);

return @datepatterns;
}

## Delete stop words from a pre-proper name set ##
sub del_stopwords {
local($word) = pop @_;
local(@stoplines) = @_;

foreach $stopword(@stoplines){
        $stopword =~  s/^\s*(.*?)\s*$/$1/;  # trim white space
        if ($word =~ /^\W*$stopword\W*$/) {
           return -1;
        }
}
return $word;
}

##  If two terms aren't the same, then it returns -1. Otherwise it
##  returns either one of these two terms.
sub extract_term{
my($term) = pop @_;
my(@pattern) = @_;
my($match);
my($upcase);

## incase U.S.
if ($term =~/U.S./) { return $term;}
foreach $_(@pattern){
   if ($term =~ /(\w*).?$/) {$upcase = $1;}
   else {$upcase = $term;}
   $match = $_;
   $match =~ tr/a-z/A-Z/;
   $upcase =~ tr/a-z/A-Z/;
   if ($match =~ /^$upcase$/) {return $term;}
}
$return = "-1";
return $return;
}
```

```
sub extract_place{
my($term) = pop @_;
my(@pattern) = @_;

my($match);
my($upcase);
my($sum) = 0;

$return = "-1";
@matchterm = split /$$/, $term;
$i = scalar(@matchterm);

if ($i == 1){
               $result = &extract_term(@pattern, $term);
               return $result;}

foreach $_(@pattern){
   @multiplace = split / /, $_;
   $j = scalar(@multiplace);
   if($i == $j){
       for($k=0; $k<$i; $k++){
           $multi_term = $matchterm[$k];
           $multi_place = $multiplace[$k];
           $multi_term =~ tr/a-z/A-Z/;
           $multi_place =~ tr/a-z/A-Z/;
           if($multi_term =~ /^$multi_place$/) {$sum++;}
        }
       if($sum == $i){return $term;}
       else {$sum = 0;}
   }
}
return $return;
}

sub hash_sortfiles{
local($filename) = pop @_;
local(@keyname) = @_;

foreach $key(@keyname){
   $gethash{$key} = ();
   $name = $filename.$key.".source";
   $fh = new FileHandle $name, "r";
   if(defined $fh){
      @$key = <$fh>;
      chop(@$key);
      $gethash{$key} = [@$key];
      undef $fh;
   }
   else { print "The name is $name. Sorry, cannot open sortfils!!\n";}
}
return %gethash;
}

## Delete the duplicated information for an array  ##
sub del_duplicate{
local(@list) = @_;
```

```
@result = ();
POPNEXT: while(@list != ()){
  $popone = pop @list;
  if(@list != ()){
     foreach $_(@list){
         if(/^$popone/){next POPNEXT;}
     }
  }
  push @result, $popone;
}
return @result;
}

## Delet loose duplicated information from an array. ##
sub del_loose_duplicate{
local(@list) = @_;

my(@del) = ();
my(@result) = ();
POPNEXT: while(@list != ()){
  $popone = pop @list;
  foreach $delone(@del){
          if($popone =~ /$delone/){ next POPNEXT;}## incase xxx A and A
  }
  if(@list != ()){
     foreach $_(@list){
         if(/$popone/){ next POPNEXT;}
         if($popone =~ /$_/){#print "$popone ==> $_ \n";
                             push @result, $popone;
                             push @del, $_;
                             next POPNEXT;
                            }
     }
  }
  push @result, $popone;
}
return @result;
}

## Create a pre-proper name set ##
sub cre_phase{
my($myinput) = pop @_;   ## input reference

my($flag) = -1;
my($pre_mark) = -1;
my($mark) = -10;
my(@capticalwords);
my(@captical);

## Defin key set of title ##
@TITLEKEY = ("A","C","D","F","G","L","M","P","R","S");

%titles = &hash_sortfiles(@TITLEKEY,"/home/cstudent/01674Ow/www/cgi-
bin/dbm/sortfiles/titles_new");

## Extract captical term from nonmonth news body ##
my($syflag) = 000000;
```

```
NEXTLOOP: foreach $_(@$myinput){
   $flag++;
   my($sin_nothing) = -1;
   my($mul_nothing) = -1;
   my($end_nothing) = -1;

   ## case 1  (...) or [...] or " " or '' or { } or < ..> ##
   ## A: (single)
   SWITCH:{
     if(/^\W*\((\w+.*)\).*$/) {
                              push @capticalwords, $1;
                              last SWITCH;
                                     }    ## (single)
     if(/^\W*\{(\w+.*)}.*$/)  {
                              push @capticalwords, $1;
                              last SWITCH;
                                     }    ## {single}
     if(/^\W*\[(\w+.*)].*$/)  {
                              push @capticalwords, $1;
                              last SWITCH;
                                     }    ## [single]
     if(/^\W*"(\w+.*)".*$/)  {
                              push @capticalwords, $1;
                              last SWITCH;
                                     }    ## "single"
      if(/^\W*'(\w+.*)'.*$/)  {
                              push @capticalwords, $1;
                              last SWITCH;
                                     }    ## 'single'
     if(/^\W*<(\w+.*)>.*$/) {
                              push @capticalwords, $1;
                              last SWITCH;
                                     }    ## <single>
     $sin_nothing = 1;
   }#switch

   if($sin_nothing != 1){ next NEXTLOOP;}
   ## B: (multiple)
   if($sin_nothing == 1){
   SWITCH:{
     if(/^\W*\((\w+.*)$/)   { $syflag = 100000;
                              $phase = $1;
                              last SWITCH;
                            }    ## (mutiple)
     if(/^\W*\{(\w+.*)$/)   { $syflag = 010000;
                              $phase = $1;
                              last SWITCH;
                            }    ## {single}
     if(/^\W*\[(\w+.*)$/)   { $syflag = 001000;
                              $phase = $1;
                              last SWITCH;
                            }    ## [single]
     if(/^\W*"(\w+.*)$/)    { $syflag = 000100;
                              $phase = $1;
                              last SWITCH;
                            }    ## "single"
```

```
    if(/^\W*<(\w+.*)$/)  { $syflag = 000001;
                           $phase = $1;
                           last SWITCH;
                         }    ## <single>
    $mul_nothing = 1;
  }#switch
  }#if multiple

  if($mul_nothing != 1){ next NEXTLOOP;}
  if($mul_nothing ==1){
     ## check ) } ] ...set syflag
     SWITCH:{
    if(/^\W*(\w+.*)\).*$/) { if($syflag == 100000)
                                {$phase = join "$$", $phase, $1;
                                 push @capticalwords, $phase;}
                                last SWITCH;
                           }    ## (mutiple)
    if(/^\W*(\w+.*)}.*$/)  { if($syflag == 010000)
                                {$phase = join "$$", $phase, $1;
                                 push @capticalwords, $phase;}
                                last SWITCH;
                           }    ## {multiple}
    if(/^\W*(\w+.*)].*$/)  { if($syflag == 001000)
                                 {$phase = join "$$", $phase, $1;
                                 push @capticalwords, $phase;}
                                last SWITCH;
                           }    ## [multiple]
    if(/^\W*(\w+.*)".*$/)  { if($syflag == 000100)
                                 {$phase = join "$$", $phase, $1;
                                  push @capticalwords, $phase;}
                                last SWITCH;
                           }    ## "multiple"
    if(/^\W*<(\w+.*)>.*$/)  { if($syflag == 000001)
                                 {$phase = join "$$", $phase, $1;
                                 push @capticalwords, $phase;}
                                 last SWITCH;
                             }    ## <>
    $end_nothing = 1;
  }#switch
  }#if mul_nothing

  if($end_nothing != 1){ $phase = ""; $syflag = 000000; next
NEXTLOOP;}

  if($end_nothing == 1){
    ## check syflag
    if($syflag != 000000){$phase = join "$$", $phase, $_;}
      elsif(/^\W+([A-Z]\w*\W*\?*\.*$)/ || (/^\W*([A-Z].[A-Z].).*$/))
                   ##in case B.C.
              {push @captical, $1; $mark = $flag; }
      elsif(/^[A-Z]/){
             $pre_mark = $mark;
             $mark = $flag;
             if ($pre_mark == $mark -1){
                  $get = pop @captical;

                  if($get =~ /(\w)\w+\.+\?*\W*$/){
                    #print "--->GET $get  --1\n";
                     # check if it is a title .
                    $in = &extract_term(@TITLEKEY, $1);
                    if($in =~ /$1/) {
                        $get =~ /(\w+\.)\.*\?*\W*$/; # match final word
                        #print "--->GET $get  --2\n";
                        $match = $1;
                        @titles = @{$titles{$in}};
                        $title = &extract_term(@titles, $match);
                    }
                    else{ $title = "-1";}
                    if($title =~ /$match/) {
                    $get = join "$$", $match, $_;
                    push @captical, $get;
                    }
                    else
                    {
                    $get =~ /^(\w+.*\w*)(\.+\?*\W*)$/;
                    push @captical, $1;
                    push @captical, $_;
                    }
                 }
              elsif($get =~ /\w+$/){   # no syi
                  $get = join "$$", $get, $_;
                  push @captical, $get;
              }
              elsif($get =~ /(\w+)\W*$/){  # del sy.
              push @captical, $1;
              push @captical, $_;
              }

       }#
       else { # pre_mark != mark
       push @captical, $_; }
      }#if /^[A-Z]/
  }#end_nothing ==1)
}#for

my($j) = 0;

foreach $every(@capticalwords){
   @words = split /$$/, $every;
   my($save) = "";
   foreach $_(@words){
     $i++;
     if(/^[A-Z]/){
         if($j>1){$save = join /$$/, $save, $_;}
         else{$save = $_;}
     }
   }
   if($save !~ /^$/){push @captical, $save;}
}
return @captical;
}
1;
```

Appendix D

A Perl Module for Extracting FullName and Others from News Documents

```
#!/loc/bin/perl -w

package fullname;

##########################################################
##
##     CLASS: fullname.pm
##   PURPOSE: To extract Fullname from a news document
##    AUTHOR: Hong Wan
##   CREATED: 18, Jan 1998
## MODIFIED: 20, Mar 1998
##
##########################################################

use buildrep;
use FileHandle;

sub fullname{
local($object) = pop @_;
local(@getvalue) = @_;

## Define key set of name ##
@KEYNAME = ("A","B","C","D","E","F","G","H","I","J","K","L","M",
            "N","O","P","Q","R","S","T","U","V","W","X","Y","Z");
## Defin key set of title ##
@TITLEKEY = ("A","C","D","F","G","L","M","P","R","S");

%TITLES = &buildrep::hash_sortfiles(@TITLEKEY,
          "/home/cstudent/016740w/www/cgi-
bin/dbm/sortfiles/titles_new");
%NAMES = &buildrep::hash_sortfiles(@KEYNAME,
          "/home/cstudent/016740w/www/cgi-
bin/dbm/sortfiles/names_new");
%LASTS = &buildrep::hash_sortfiles(@KEYNAME,
         "/home/cstudent/016740w/www/cgi-
bin/dbm/sortfiles/lastnames_new");

NEXTWORD: foreach $word(@getvalue){
@chars = split /$$/, $word;
$nameflag = 0;
$elementno = 0;
$t_count = 0;
$t_pre_position = 0;
$t_cur_position = 0;
@other = ();
@title = ();
@first = ();
@middle = ();
$last = "";   ## a person has only one last name

$count = scalar(@chars);
if ($count == 1) {  ## a name-phrase with a single word
   $word =~ /^\W*(\w)/;
   $key = $1;
   @NAMESET = @{$LASTS{$key}};
   $get = &buildrep::extract_term(@NAMESET, $word);
   if($get =~ /$word/) { $last = $get;  $nameflag = 1; }
```

```
                              ## get a last name
}#if

if ($count >1) {  ## a name-phrase with multiple words
   WORD :foreach $_(@chars){
      if(/^(.*?)'s$/){ $_ = $1;} ## delete 's
      $elementno ++;

      ## Identify a word as a title ##

      /^\W*(\w)/;
      $in = &buildrep::extract_term(@TITLEKEY, $1);
      if ($in =~ /$1/){
         @TITLESET = @{$TITLES{$in}};
         $get = &buildrep::extract_term(@TITLESET, $_);
         if(/$get/) {              ## get a title
            $t_cur_position = $elementno;
            if(($t_cur_position == $t_pre_position + 1)||
               ($t_count == 0)){
               $nameflag = 1;
               push @title, $_;
               $t_pre_position = $t_cur_position;
               $t_count++;
             }
         }
         elsif($get == -1) {next WORD;}
      }
      elsif($in == -1) {next WORD;}
    } #foreach WORD

   ## case: the name-phrase with title) ##
   if ($nameflag == 1){

     ## extract other part ##
     $pre_t = $t_cur_position - $t_count;
     if ( $pre_t >0) {         ## get other part of a FullName
        for ($i = 0; $i<$pre_t; $i++){
            push @other, $chars[$i];
        }#for
     }#if

     $after_t = $count - $t_cur_position;
     $last = $chars[$count-1];## get the last name of a FullName
     if($after_t >= 2){
        push @first, $chars[$t_cur_position];
                              ## get the first name of a FullName
     }
     if($after_t >2){          ## get the middle name of a FullName
        for($i = $t_cur_position+1; $i<$count-1; $i++)
           { push @middle, $chars[$i];}
      }
   }#if (has title)

   ## case: a name-phrase without title  ##

   $f_cur_position = 0;
```

```
if ($nameflag != 1){
FIRST :foreach $_(@chars){
      $f_cur_position++;
      /^\W*(\w)/;
      $key = $1;
      @NAMESET = @{$NAMES{$key}};
      $get = &buildrep::extract_term(@NAMESET, $_);
      if(/$get/) {
         if($f_cur_position != $count){
                     ## incase it isnot a final word
           $nameflag = 1;
           push @first, $_;
           last FIRST;
          }
          else {
           @LASTSET = @{$LASTS{$key}};
           $getend = &buildrep::extract_term(@LASTSET, $_);
           if(/$getend/){$nameflag = 1; $last = $_; last FIRST;}
           elsif($getend == -1)
                {$nameflag = 1; push @first, $_; last FIRST;}
          }
       }
       elsif($get == -1) {next FIRST;}
 }#for FIRST

 ## if the name-phrase has first name part
 if($nameflag == 1){
   if($f_cur_position != $count){$last = $chars[$count-1];}
                           ## get a last name
   $after_f = $count - $f_cur_position;
   if($after_f >1){
     for($i = $f_cur_position; $i<$count-2; $i++)
        { push @middle, $chars[$i];}
    }
    for($i = 0; $i<$f_cur_position-1; $i++)
        { push @other, $chars[$i];}
  }#if nameflag == 1 for first name
  else {                  ## check if it is a last name
  $elementno = 0;

  ## case one: check the final word
  $finalone = pop @chars;
  $finalone =~ /^\W*(\w)/;
  @LASTSET = @{$LASTS{$1}};
  $get = &buildrep::extract_term(@LASTSET, $_);
  if(/$get/) {            ## the final word is a last name
             push @other, @chars;
             $last = $finalone;
             $nameflag = 1;
  } # if get


  elsif($get == -1) {
        ## check all words, if get one, throw away after one
      $l_position = 0;
      $elementno = 0;
      LAST: foreach $_(@chars){
            $elementno++;
            /^\W*(\w)/;
            @LASTSET = @{$LASTS{$1}};
            $get = &buildrep::extract_term(@LASTSET, $_);
            if(/$get/) { $last = $_; $l_position = $elementno;
                         $nameflag = 1; last LAST; }
            elsif($get == -1) {next LAST;}
          }#for LAST

          if($l_position >0){
              for ($i =0; $i<= $l_position - 2; $i++){
                   push @other, $chars[$i]
              }
           }## if $l_position >0
       }# elsif case two
     }## elsif check last name
} ## if nameflag != 1
}## if count>1

## get a FullName ##
if($nameflag == 1){
   $addtitle = &get_multi_terms(@title);
   $addfirst = &get_multi_terms(@first);
   $addmiddle = &get_multi_terms(@middle);
   $addother = &get_multi_terms(@other);

   $object->addFullname("$addother","$addtitle",
            "$addfirst","$addmiddle","$last");
}## if nameflag == 1
else { push @forothers, $word ;}

$nameflag = 0;
@other = ();
@title = ();
@first = ();
@middle = ();
$last = "";
}#foreach word

$object->addOthers(@forothers);
}

sub get_multi_terms{
local(@list) = @_;
$addterm = "";

if(@list != ()) {
         $i = 0;
         foreach $_(@list){
           $i++;
           if($i == 1){ $addterm = $_;}
           else { $addterm = $addterm."$$".$_; }
         }
}
return $addterm;
}
1;
```

Appendix E

A Perl Module for Calculating the Similarity between Two News Object

```
#!/loc/bin/perl -w

package simcom;

############################################################
##
##     CLASS: simcom.pm
##   PURPOSE: To calculate the similarity between two objects
##    AUTHOR: Hong Wan
##   CREATED: 18, Nov 1997
## MODIFIED: 20, Mar 1998
##
############################################################

sub sim_newsobject{
my($item2) = pop @_;
my($item1) = @_;

my(%sim);

my(@loc1) = $item1->getLocation;
my(@loc2) = $item2->getLocation;
$loc_n1 = scalar(@loc1);
$loc_n2 = scalar(@loc2);
$loc_min = &min_num($loc_n1, $loc_n2);

if((0<=>$loc_n1)&&(0<=>$loc_n2)){ $sim_loc =
&sim_others_location(\@loc1, \@loc2); }
else {$sim_loc = 0;}  ## one of them is empty array.
$sim{'location'} = $sim_loc;
$sim{'loc_num1'} = $loc_n1;
$sim{'loc_num2'} = $loc_n2;
$sim{'loc_min'} = $loc_min;

my(@others1) = $item1 ->getOthers;
my(@others2) = $item2 ->getOthers;
$unk_n1 = scalar(@others1);
$unk_n2 = scalar(@others2);
$unk_min = &min_num($unk_n1, $unk_n2);

if(($unk_n1>0)&&($unk_n2>0))
  {$sim_others = &sim_others_location(\@others1, \@others2);}
else{$sim_others = 0;}
$sim{'others'} = $sim_others;
$sim{'unk_num1'} = $unk_n1;
$sim{'unk_num2'} = $unk_n2;
$sim{'unk_min'} = $unk_min;

my(@eventdate1) = $item1 ->getEventdate;
my(@eventdate2) = $item2 ->getEventdate;
$dat_n1 = scalar(@eventdate1);
$dat_n2 = scalar(@eventdate2);
$dat_min = &min_num($dat_n1, $dat_n2);

if(($dat_n1>0)&&($dat_n2>0))
  {$sim_date = &sim_others_location(\@eventdate1, \@eventdate2);}
else{$sim_date = 0;}
$sim{'date'} = $sim_date;
$sim{'dat_num1'} = $dat_n1;
$sim{'dat_num2'} = $dat_n2;
$sim{'dat_min'} = $dat_min;

@fullname1 = $item1 ->getFullname;
@fullname2 = $item2 ->getFullname;
$nam_n1 = scalar(@fullname1);
$nam_n2 = scalar(@fullname2);
$nam_min = &min_num($nam_n1, $nam_n2);

if(scalar(@fullname1)&&scalar(@fullname2))
  {$sim_name = &sim_fullname(\@fullname1, \@fullname2);}
else{$sim_name = 0;}
$sim{'name'} = $sim_name;
$sim{'nam_num1'} = $nam_n1;
$sim{'nam_num2'} = $nam_n2;
$sim{'nam_min'} = $nam_min;


## get alpha
$sim{'alpha'} = $loc_min + $unk_min + $dat_min + $nam_min;
return %sim;
}

sub sim_fullname{
    my($name2) = pop @_;
    my($name1) = @_;

    my(@value) = ();

    my($match_num) = 0;

    my($num1) = scalar(@$name1);
    my($num2) = scalar(@$name2);

    for($i = 0; $i<$num1; $i++){

        $max_value = 0.0;
        for($j = 0; $j<$num2; $j++){
            $match_value = &wei_fullname($$name1[$i], $$name2[$j]);

            if($max_value < $match_value){$max_value = $match_value;}
        }

        push @value, $max_value;
    }
    foreach $_(@value){
      if($_>0.8){$match_num++; }
    }
return $match_num;
}
```

```
sub wei_fullname{

local(*name2) = pop @_;
local(*name1) = @_;


if(($name1{'other'} =~ /$name2{'other'}/)||
   ($name2{'other'} =~ /$name1{'other'}/)||
   ($name1{'other'} =~ /^$/)||($name2{'other'} =~ /^$/))
   {$wei_other = 0.05;}
else {$wei_other = 0.0;}

if(($name1{'title'} =~ /$name2{'title'}/)||
   ($name2{'title'} =~ /$name1{'title'}/)||
   ($name1{'title'} =~ /^$/)||($name2{'title'} =~ /^$/))
   {$wei_title = 0.2;}
else {$wei_title = 0.0;}

if(($name1{'first'} =~ /^$/)||($name2{'first'} =~ /^$/)||
   ($name1{'first'} =~ /$name2{'first'}/))
   {$wei_first = 0.3;}
else {$wei_first = 0.0;}

if(($name1{'middle'} =~ /$name2{'middle'}/)||
   ($name2{'middle'} =~ /$name1{'middle'}/)||
   ($name1{'middle'} =~ /^$/)||($name2{'middle'} =~ /^$/))
   {$wei_middle = 0.05;}
else {$wei_middle = 0.0;}

if(($name1{'last'} =~ /$name2{'last'}/)||
  (($name1{'last'} =~ /^$/)&&($name2{'last'} =~ /^$/)))
  {$wei_last = 0.4;}
else {$wei_last = 0.0;}

$sim = $wei_other + $wei_title + $wei_first + $wei_middle + $wei_last;

return $sim;

}

sub sim_others_location{
    my($others2) = pop @_;
    my($others1) = @_;

    my($sum) = 0;
    my($sum_matchterms) = 0;
    foreach $_(@$others1){
      $sum_matchterms = &sum_matchterms(@$others2, $_);
      $sum = $sum + $sum_matchterms;
    }
    return $sum;
}
```

```
sub min_num{
    local($y) = pop @_;
    local($x) = @_;

    if($x >= $y) {return $y;}
    if($x <  $y) {return $x;}
}



sub sum_matchterms{
    my($match) = pop @_;
    my(@terms) = @_;

    my($sum) = 0;
    foreach $_(@terms){
       if(($_ =~ /$match/)||($match =~/$_/)){ return 1;}
    }
    return $sum;
}
1;
```

Appendix F

A Perl Module for Saving a News Object to a Rep File or Restoring a News Object from a Rep File

```
#!/loc/bin/perl -w

package topicdb;

#####################################################
##
##  PACKAGE: topicdb.pm
##           save/restore an newsitem or
##           create/modify the topic database
##
##   AUTHOR: Hong Wan
##     DATE: Jan. 22, 1998
##   MODIFY: Jan. 28, 1998
##
#####################################################

use newsitem;

##----------subroutine:save_newsitem()---------------##
## save an newsitem object to a file
## param1: an newsitem object
## param2: the output file name
##---------------------------------------------------##

sub save_newsitem{
local($file) = pop @_;
local($item) = @_;

@gettitle = $item ->getTitle;
@getauthor = $item ->getAuthor;
@getdate = $item ->getDate;
@getagency = $item ->getAgency;
@getothers = $item ->getOthers;
@geteventdate = $item ->getEventdate;
@getfullname = $item ->getFullname;
@geteventlocation = $item ->getLocation;

open(OBJECTFILE, ">$file") || die "Cannot create $file \n";
print OBJECTFILE "<TITLE> \n";
if(@gettitle != ()){print_list(@gettitle, "OBJECTFILE");}
print OBJECTFILE "<AUTHOR> \n";
if(@getauthor != ()){print_list(@getauthor, "OBJECTFILE");}
print OBJECTFILE "<DATE> \n";
if(@getdate != ()){print_list(@getdate, "OBJECTFILE");}
print OBJECTFILE "<AGENCY> \n";
if(@getagency != ()){print_list(@getagency, "OBJECTFILE");}
print OBJECTFILE "<EVENTLOCATION>\n";
if(scalar(@geteventlocation)){print_list(@geteventlocation,
"OBJECTFILE");}
print OBJECTFILE "<EVENTDATE> \n";
if(@geteventdate != ()){print_list(@geteventdate, "OBJECTFILE");}
print OBJECTFILE "<FULLNAME> \n";
if(@getfullname != ()){ @fullname = &save_fullname(@getfullname);
                        print_list(@fullname, "OBJECTFILE");}
print OBJECTFILE "<OTHERS> \n";
if(@getothers != ()){print_list(@getothers, "OBJECTFILE");}
close(OBJECTFILE);
```

```
}


##----------subroutine:restore_newsitem()-------------##
## restore an newsitem object from a file
## param: the input file name
##-----------------------------------------------------##

sub restore_newsitem{
my($line) = @_;

$flag = 11111111;

my($item) = new newsitem;
@title = ();
@author = ();
@date = ();
@agency = ();
@eventlocation = ();
@eventdate = ();
@fullname = ();
@others = ();

foreach $_(@$line){
  if(/<TITLE>/){ $flag = 11111110;}
  if(/<AUTHOR>/){ $flag = 11111101;}
  if(/<DATE>/){ $flag = 11111011;}
  if(/<AGENCY>/){ $flag = 11110111;}
  if(/<EVENTLOCATION>/){ $flag = 11101111;}
  if(/<EVENTDATE>/){ $flag = 11011111;}
  if(/<FULLNAME>/){ $flag = 10111111;}
  if(/<OTHERS>/){ $flag = 01111111;}
  if(($flag == 11111110) && (not /<TITLE>/)){ push @title, $_;}
  if(($flag == 11111101) && (not /<AUTHOR>/)){ push @author, $_;}
  if(($flag == 11111011) && (not /<DATE>/)){ push @date, $_;}
  if(($flag == 11110111) && (not /<AGENCY>/)){ push @agency, $_;}
  if(($flag == 11101111) && (not /<EVENTLOCATION>/)){push
@eventlocation, $_;}
  if(($flag == 11011111) && (not /<EVENTDATE>/)){push @eventdate, $_;}
  if(($flag == 10111111) && (not /<FULLNAME>/)){push @fullname, $_;}
  if(($flag == 01111111) && (not /<OTHERS>/)){push @others,$_;}
}

$item->addTitle(@title);
$item->addAuthor(@author);
$item->addDate(@date);
$item->addAgency(@agency);
$item->addLocation(@eventlocation);
$item->addEventdate(@eventdate);
$item->addOthers(@others);
```

```
foreach $_(@fullname){
     @word = split / /;
     $other = $word[0];
     $title = $word[1];
     $first = $word[2];
     $middle = $word[3];
     $last = $word[4];
     $item->addFullname($other, $title, $first, $middle, $last);
}
return $item;
}

sub save_fullname{
local(@getfullname) = @_;

@fullname = ();
$i = scalar(@getfullname);
for($j=0;$j<$i;$j++){
    $getother = $getfullname[$j]{"other"};
    $gettitle = $getfullname[$j]{"title"};
    $getfirst = $getfullname[$j]{"first"};
    $getmiddle = $getfullname[$j]{"middle"};
    $getlast = $getfullname[$j]{"last"};
    $savefullname = $getother."$$".$gettitle."$$".$getfirst.
                    "$$".$getmiddle."$$".$getlast;
    push @fullname, $savefullname;
}

return @fullname;
}

sub open_topicdb{
local($dbname) = @_;

## Open/Create the new dbm files (.dir & .pag)
dbmopen(%gethash, $dbname, "0644") || die "Cannot open $dbname DBM
files";

return %gethash;
}

sub append_topicdb{
local($record) = pop @_;
local($dbname) = @_;

%hashdb = open_topicdb($dbname);
$key = scalar(@{$hashdb});
$hashdb{($key+1)} = $record;
dbmclose %hashdb;

}

sub del_topicdb{
local($record) = pop @_;
local($dbname) = @_;

%hashdb = &open_topicdb($dbname);
```

```
foreach $_(keys(%hashdb)){
   if($hashdb{$_} =~ /$record/){
        delete $hashdb{$key};
        return 1;
   }
}
dbmclose %hashdb;
return -1;
}

sub insert_repfile{
local($insert_rep) = pop @_;
local($rep) = pop @_;
local($dbname) = @_;

## open Topic Datebase ##
%hashdb = &open_topicdb($dbname);
$findresult = &find_repfile(%hashdb, $rep);
if($findresult =~ /.*(_db)$/) {
   &append_topicdb($findresult, $insert_rep);
   return 1;
}
elsif($findresult =~ /[1-1000]/){
   $rep =~ /.*(_rep)$/;
   $db2name = $1."_db";
   %level2db = open_topicdb($db2name);
   $level2db{"1"} = $rep;
   $level2db{"2"} = $insert_rep;
   $hashdb{$findresult} = $db2name;
   dbmclose %level2db;
   return 2;
}
else {return -1;}
}

sub find_repfile{
local($rep) = pop @_;
local(%hashdb) = @_;

foreach $_(keys(%hashdb)){
   if($hashdb{$_} =~ /.*(_rep)$/) {
      if($hashdb{$_} =~ /$rep/) {return $_;}
   }
   else{
      %hashdb2 = &open_topicdb($hashdb{$_});
      foreach $a(keys(%hashdb2)){
           if($hashdb2{$a} =~ /$rep/) {return $hashdb{$_}; }
      }
      dbmclose %hashdb2;
   }
}
dbmclose %hashdb;

return -1;

}
```

```
sub print_list{
local($file) = pop @_;
local(@list) = @_;

if(@list == ()){ return -1}
else {
      foreach $_(@list){
        @word = split /$$/;
        $i = 0;
        foreach $print(@word){
           $i++;
           if($i == 1) {$get = $print;}
           else{
                $get = join " ", $get, $print;
           }
        }
        print $file $get, "\n";
        }
}
}
1
```

Appendix G

A Perl CGI Code for Creating the Main Web Page of "A News Filtering Tools"

```
#!/loc/bin/perl

use CGI qw(:standard);
use CGI::Carp qw/fatalsToBrowser/;
$main = new CGI;
&print_beginning;
&print_representation;
&print_similarity;
&print_evaluation;
&print_tail;

sub print_beginning{
    print $main->header;
    print $main->start_html
                ('Automatic News Browsing Using Vector Space  Model',
                 '016740w@dragon.acadiau.ca',
                 'true',
                 'BGCOLOR="#ffffff"');
    print "<H2><CENTER><img
src=\"http://dragon.acadiau.ca/~016740w/cgi-bin/gif/tooltitle.jpg\">
</CENTER></H2> \n";
    print "<p>\n";
}

sub print_representation{
    my($method) = "POST";
    my($action)= "http://dragon.acadiau.ca/~016740w/
                  cgi-bin/representation.cgi";
    my($encoding) = "multipart/form-data";
    print "<center>\n";
    print "<align=left><img src=\
        "http://dragon.acadiau.ca/~016740w/cgi-bin/gif/tool1.jpg\">\n";
    print $main->startform($method, $action, $encoding);
    print "<TABLE>";
    print "<tr><td><b><font color=\"959619\">News Raw Files:</b> ";
    print "<TD>";
    print $main->filefield('raw_files','',45);
    print "</tr>";
    print "</TABLE> \n";
    print "<INPUT TYPE=\"radio\" NAME=\"category\" VALUE=\"ns\"
CHECKED><font color=\"959619\"> News  \n";
    print "<INPUT TYPE=\"radio\" NAME=\"category\" VALUE=\"et\" >
Entertainment  \n";
    print "<INPUT TYPE=\"radio\" NAME=\"category\" VALUE=\"bs\" >
Business  \n";
    print "<INPUT TYPE=\"radio\" NAME=\"category\" VALUE=\"sp\"  >
Sport  \n";
    print "<INPUT TYPE=\"radio\" NAME=\"category\" VALUE=\"ed\" >
Editorial/Letters </font><BR> \n";
    print "<P ALIGN=LEFT>";
    $format = " Read";
    $build = " Build ";
    $cancel = " Cancel ";
    print $main->submit ('build',$format);
    print $main->submit ('build',$build);
    print $main->reset ('cancel',$cancel);
    print "</P>";  print "<p>\n";   print $main->endform; }
```

```
sub print_similarity{
    my($method) = "POST";
    my($action) = "http://dragon.acadiau.ca/~016740w/cgi-
bin/similarity.cgi";
    my($encoding) = "multipart/form-data";

    print "<p>\n";
    print "<align=left><img src=\
        "http://dragon.acadiau.ca/~016740w/cgi-bin/gif/tool2.jpg\">\n";
    print $main->startform($method, $action, $encoding);
    print "<table><TR>";
    print "<TD><B><font color=\"c971df\">The First News Rep File:
</B>\n";
    print "<TD>";
    print $main->filefield('sel_rep1','',25);
    print "<TD> \n";
    print $main->submit ('pick_rep1',"Open");
    print "</TR>";
    print "<TR>";
    print "<TD><B><font color=\"c971df\"> The Second News Rep File:
</B></td>\n";
    print "<TD>";
    print $main->filefield('sel_rep2','',25);
    print "<TD> \n";
    print $main->submit ('pick_rep2',"Open");
    print "</TR>";
    print "</TABLE> \n";
    print "<br>\n";
    $sim_rep = "Similarity";
    print $main->submit ('sim_rep',$sim_rep);
    print $main->reset ('cancel',$cancel);
    print "</P>";
    print "<p>\n";
    print $main->endform;
}

sub print_evaluation{
    $method = "POST";
    $action = "http://dragon.acadiau.ca/~016740w/cgi-
bin/evaluation.cgi";
    print $main->startform($method, $action);
    print "<p>\n";
    print "<align=left><img
src=\"http://dragon.acadiau.ca/~016740w/cgi-bin/gif/tool3.jpg\">\n";
    print "<P ALIGN=LEFT>";
    $query = " Query ";
    print $main->submit ('query',$query);
    print $main->reset ('cancel',$cancel);
    print "</P>";
    print $main->endform;
}

sub print_tail{

    print $main->end_html;
}
```

Appendix H

A Perl CGI Code for Building a News Object and Presenting it on the Web Page

```
#!/loc/bin/perl

#use CGI qw(:upload.pl);
#use CGI;
use newsitem;
use buildrep;
use fullname;
use topicdb;

print "Content-type: text/html", "\n\n";
do "upload.pl" || exit print "Could not load library!!\n";
#do "newprint.pl" || exit print "Could not load newprint library !!\n";

&UploadCGIParse;
$file = $input{'raw_files'};
$op = $input{'build'};
$classic = $input{'category'};
$fileContents = $filedata{'raw_files'};
@lines = split("\n",$fileContents);
$rawfile = \@lines;
$item = new newsitem;

&print_beginning;
if($op =~ /Read/){&print_rawfile;}
elsif($op =~ /Build/){&print_build;}
&print_tail;

sub print_beginning{
    print "<HTML>";
    $action = "http://dragon.acadiau.ca/~016740w/cgi-bin/saverep.cgi";
    print "<FORM ACTION= $action METHOD= \"POST\"
ENCTYPE=\"multipart/form-data\"> \n";
    print "<H2><CENTER> News Representation Object</CENTER></H2> \n";
    print "<HR>\n";
}

sub print_rawfile{
    my($sum) = 0;
    my($upcase) = 0;
    @buildbody = &buildrep::newstitle($rawfile, $item);
    foreach $_(@buildbody){
       @line = split / /, $_;
       $i = scalar(@line);
       $sum = $sum + $i;
       foreach $single(@line){
         if($single =~ /^[A-Z]/){$upcase++;}
       }
    }
    print "<CENTER> \n";
    print "<H4> The Content of $file is: </H4>";
    print "<P><P>\n";
    print "<TEXTAREA ROWS=25 COLS=100 NAME=\"showraw\"> \n";
    foreach $_(@lines){
    print $_, " \n";
    }
    print "</TEXTAREA>";
    print "<P>\n";
    print "<EM>There are $sum words and $upcase words beginning with a
captical letter. </EM>\n";
}

sub print_build{
    @buildbody = &buildrep::newstitle($rawfile, $item);
    @left = &buildrep::newsitem(@buildbody, $item);
    &fullname::fullname(@left, $item);
    print "<p>\n";
    &print_terms;
    #&table_terms($item);
    $file =~ /(\w+)(\.*\w*)$/;
    $tmpfile = $classic."_".$1.".rep";
    $repfile = "/home/cstudent/016740w/public/".$tmpfile;
    &topicdb::save_newsitem($item, $repfile);
    chmod 0777, $repfile;
    print "<CENTER><p> \n";
    print "<EM> The news object is saved as $repfile. </EM>\n";
    print "<P>\n";
    #print "<EM> Rep File Name </EM>: <INPUT TYPE=\"text\"
NAME=\"repname\" SIZE=20> \n";
    #print "<INPUT TYPE=\"radio\" NAME=\"classic\" VALUE=\"normal\"
CHECKED> Normal \n";
    #print "<INPUT TYPE=\"radio\" NAME=\"classic\" VALUE=\"centriod\">
Centriod \n";
    #print "<INPUT TYPE=\"submit\" NAME=\"save\" VALUE=\"Save\">\n";
    #print "<INPUT TYPE=\"reset\" NAME=\"cancle\" VALUE=\"Cancle\">\n";
    ## transfer $repfile
    #print "<INPUT TYPE = \"hidden\" NAME=\"tmpfile\" VALUE=$repfile>
\n";
}

sub print_terms{
    my(@gettitle) = $item ->getTitle;
    my(@getauthor) = $item ->getAuthor;
    my(@getdate) = $item ->getDate;
    my(@getagency) = $item ->getAgency;
    my(@getothers) = $item ->getOthers;
    my(@geteventdate) = $item ->getEventdate;
    my(@geteventlocation) = $item ->getLocation;
    my(@getfullname) = $item ->getFullname;
    my($i);
    print "<CENTER> \n";
    print "<TABLE border> \n";
    print "<TR> \n";
    print "<TD> \n";
    print "<TH> Title </TH>";
    &print_itemlist(@gettitle,1) if(scalar(@gettitle));
    print "<TR> \n";
    print "<TD> \n";
    print "<TH> Author </TH>";
    &print_itemlist(@getauthor,1) if(scalar(@getauthor));
    print "<TR> \n";
    print "<TD> \n";
    print "<TH> Report Date</TH>";
    &print_itemlist(@getdate,1) if(scalar(@getdate));
    print "<TR> \n";
    print "<TD> \n";
    print "<TH> Agency </TH>";
    &print_itemlist(@getagency,1) if(scalar(@getagency));
    print "<TR> \n";
    print "<TD> \n";
    print "<TH> Event Date </TH>";
    $i = scalar(@geteventdate);
    &print_itemlist(@geteventdate,$i) if(scalar(@geteventdate));
    print "<TR> \n";
    print "<TD> \n";
    print "<TH> Event Location </TH>";
    $i = scalar(@geteventlocation);
    &print_itemlist(@geteventlocation, $i)
if(scalar(@geteventlocation));
    print "<TR> \n";
    print "<TD> \n";
    print "<TH> Full Name </TH>";
    $i = scalar(@getfullname);
    if ($i != 0){
       for($j=0;$j<$i;$j++){
                $other = $getfullname[$j]{"other"};
                $title = $getfullname[$j]{"title"};
                $first = $getfullname[$j]{"first"};
                $middle = $getfullname[$j]{"middle"};
                $last = $getfullname[$j]{"last"};
                $printvalue = "[O]".$other." [T]".$title." [F]".$first.
                           " [M]".$middle." [L]".$last;
                push @namelist, $printvalue;
       }#for
       if($i > 0){
          if($i < 4){&print_itemlist(@namelist, $i);}
          else{&print_itemlist(@namelist, 4);}
       }
    }
    print "<TR> \n";
    print "<TD> \n";
    print "<TH> Unknow Terms </TH>";
    $i = scalar(@getothers);
    if($i > 0){
       if($i < 5){&print_itemlist(@getothers, $i);}
       else{&print_itemlist(@getothers, 5);}
    }
    print "</TABLE> \n";
    print "</CENTER> \n";
}

sub print_itemlist{
    my($size) = pop @_;
    my(@local_param) = @_;

    my($name) = "termlist";
    print "<TD>";
    print "<SELECT NAME=$name SIZE=$size MULTIPLE>", "\n";
    foreach $_(@local_param){
       @word = split /$$/;
       $i = 0;
       foreach $print(@word){
            $i++;
            if($i == 1) {$get = $print;}
            else{
                $get = join " ", $get, $print;
            }
        }
        #push @prnitem, $get;
        print "<OPTION>$get \n";
    } ## foreach @local_param
    print "</SELECT> \n";
    print "</TD>";
}

sub print_tail{
    print "</HTML> \n";
}
```

Appendix I

A Perl CGI Code for Calculating the Similarity between Two News Objects and Presenting the results on the Web Page

```
#!/loc/bin/perl

use newsitem;
use topicdb;
use simcom;

print "Content-type: text/html", "\n\n";
do "upload.pl" || exit print "Could not load library!!\n";
do "cgiprint.pl" || exit print "Could not load cgiprint library !! \n";
do "newprint.pl" || exit print "Could not load newprint library !!\n";

&UploadCGIParse;
$repname1 = $input{'sel_rep1'};
$fileContents1 = $filedata{'sel_rep1'};
@lines1 = split("\n",$fileContents1);

&print_beginning;

$repname2 = $input{'sel_rep2'};
$fileContents2 = $filedata{'sel_rep2'};
@lines2 = split("\n",$fileContents2);

$item1 = &topicdb::restore_newsitem(\@lines1);
$item2 = &topicdb::restore_newsitem(\@lines2);

$op1 = $input{'pick_rep1'};
$op2 = $input{'pick_rep2'};
if($op1 =~ /Open/){
   print "<H2><CENTER> News Rep File </CENTER></H2> \n";
   print "<HR>\n";
   &table_terms($item1);
#   &print_terms($item1);
}
elsif($op2 =~ /Open/){
   print "<H2><CENTER> News Rep File </CENTER></H2> \n";
   print "<HR>\n";
   &print_terms($item2);
}

$op3 = $input{'sim_rep'};

if($op3 =~ /Similarity/){
   @title1 = $item1->getTitle;
   @title2 = $item2->getTitle;
   print "<H2><CENTER> The Result of Similarity </CENTER></H2> \n";
   print "<HR>\n";
   print "<B>The header information of first rep: </B>\n";
   print "<br><li type\=\"circle\">File:$repname1 \n";
   print "<br><li type\=\"circle\">Title: @title1 \n";
   print "<P>\n";
   print "<B>The header information of second rep: </B>\n";
   print "<br><li type\=\"circle\">File:$repname2 \n";
   print "<br><li type\=\"circle\">Title: @title2 \n";
   print "<P>\n";
   print "<B>The similarity Table</B><br>\n";
   &print_simtable;
}
```

```
&print_tail;

sub print_beginning{
    print "<HTML>";
    print "<FORM ACTION= \"\" METHOD= \"POST\"> \n";
}

sub print_simtable{
    %sim = &simcom::sim_newsobject($item1, $item2);
    print "<TABLE border=0>\n";
    print "<TR bgcolor=\"e0ffe8\"><td><font color=\"green\"><b>Feature
Name</b> \n";
    print "<TD><b><font color=\"green\"># of First Rep </b> \n";
    print "<TD><b><font color=\"green\"># of Second Rep</b> \n";
    print "<TD><b><font color=\"green\"># of Common Name-phases </b>
\n";
    print "<TD><font color=\"green\"><B>Similarity</B></font> \n";
    print "<TR><TD><b><font color=\"green\">Location</b> \n";
    print "<TD align=center><b><font
color=\"green\">$sim{'loc_num1'}</b> \n";
    print "<TD align=center><b><font
color=\"green\">$sim{'loc_num2'}</b> \n";
    print "<TD align=center><b><font
color=\"green\">$sim{'location'}</b> \n";
    if($sim{'loc_min'} > 0){$def_simloc =
$sim{'location'}/$sim{'alpha'};}
    else{$def_simloc = 0;}
    if($def_simloc =~ /(\d+\.\d\d).*/){$simloc = $1;}
    else{ $simloc = $def_simloc;}
    print "<TD align=center><b><font color=\"green\">$simloc</b> \n";

    print "<TR><TD><font color=\"green\"><b>Event Date</b> \n";
    print "<TD align=center><font
color=\"green\"><b>$sim{'dat_num1'}</b> \n";
    print "<TD align=center><font
color=\"green\"><b>$sim{'dat_num2'}</b> \n";
    print "<TD align=center><font color=\"green\"><b>$sim{'date'}</b>
\n";
    if($sim{'dat_min'} > 0){$def_simdat = $sim{'date'}/$sim{'alpha'};}
    else{$def_simdat = 0;}
    ## format digits
    if($def_simdat =~ /(\d+\.\d\d).*/){$simdat = $1;}
    else{ $simdat = $def_simdat;}
    print "<TD align=center><font color=\"green\"><b>$simdat</b> \n";

    print "<TR><TD><b><font color=\"green\">Full Name</b> \n";
    print "<TD align=center><font
color=\"green\"><b>$sim{'nam_num1'}</b> \n";
    print "<TD align=center><b><font
color=\"green\">$sim{'nam_num2'}</b> \n";
    print "<TD align=center><b><font color=\"green\">$sim{'name'}</b>
\n";
    if($sim{'nam_min'}>0){$def_simnam = $sim{'name'}/$sim{'alpha'};}
    else{$def_simnam = 0;}
    if($def_simnam =~ /(\d+\.\d\d).*/){$simnam = $1;}
    else{ $simnam = $def_simnam;}
    print "<TD align=center><b><font color=\"green\">$simnam</b> \n";

    print "<TR><TD><b><font color=\"green\">Organization</b> \n";
    print "<TD align=center><b><font
color=\"green\">$sim{'unk_num1'}</b> \n";
    print "<TD align=center><b><font
color=\"green\">$sim{'unk_num2'}</b> \n";
    print "<TD align=center><b><font color=\"green\">$sim{'others'}</b>
\n";
    if($sim{'unk_min'}>0){$def_simunk = $sim{'others'}/$sim{'alpha'};}
    else{$def_simunk = 0;}
    if($def_simunk =~ /(\d+\.\d\d).*/){$simunk= $1;}
    else{ $simunk = $def_simunk;}
    print "<TD align=center><b><font color=\"green\">$simunk</b> \n";
    print "</TABLE>\n";

    print "<P>\n";
    print "<EM><b>Document-Document Similarity:  \n";
    $def_repsim = $simloc + $simdat + $simnam + $simunk;
    print "$def_repsim</EM> \n";
    print "<P>\n";
    print "</b></EM> \n";

}


sub print_tail{
    print "</FORM> \n";
    print "</HTML> \n";
}
```

Appendix J

A Perl CGI Code for Querying the Pattern-Match Dictionaries and Presenting the Results on the Web Page

```
#!/loc/bin/perl

use buildrep;

print "Content-type: text/html", "\n\n";
do "upload.pl" || exit print "Could not load library!!\n";
do "cgiprint.pl" || exit print "Could not load print library!!\n";

&UploadCGIParse;

$qy_type = $input{'qy_sortfile'};
$qy_term = $input{'name'};

&print_beginning;
## enter null
if($qy_term =~ /^$/){
    print "<P>Please enter a term in the textfield!!! </P>\n";
    exit;
}

@KEYNAME = ("A","B","C","D","E","F","G","H","I","J","K","L","M",
            "N","O","P","Q","R","S","T","U","V","W","X","Y","Z");
@TITLEKEY = ("A","C","D","F","G","L","M","P","R","S");


if($qy_type =~ /title/){
   %TITLES = &buildrep::hash_sortfiles(@TITLEKEY,
          "/home/cstudent/016740w/www/cgi-
bin/dbm/sortfiles/titles_new");
   if($qy_term =~ /^\W*(\w).*$/){
        $key = $1;
        $key =~ tr/a-z/A-Z/;    ## to upper case
        $in = &buildrep::extract_term(@TITLEKEY, $key);
        if ($in =~ /$key/){
            @TITLESET = @{$TITLES{$in}};
            $qy_term =~ /^\W*(\w+\.*).*$/;
            $get = &buildrep::extract_term(@TITLESET, $1);
            if($get =~ /$1/) {              ## get a title
               print "<TABLE border> \n";
               print "<TR> \n";
               print "<TH>Yes, it is a title term.</TH> \n";
            }
            else {
            print "<TABLE border> \n";
            print "<TR> \n";
            print "<TH>No, it is not a title term.</TH> \n";
            }
            print "<TR>\n";
            print "<TH>The term list: </TH> \n";
            print_itemlist(@TITLESET, 5);
            $i = scalar(@TITLESET);
            print "<TH>There is $i terms with $key index.</TH>\n";
            print "</TR>\n";
            print "</TABLE>\n";

        }
        else {print "<P>No, it is not a title term. </P> \n";}
    }
    else {print "<P> The enter term is not volid!</P> \n";}
}

if($qy_type =~ /first/){
    %NAME = &buildrep::hash_sortfiles(@KEYNAME,
            "/home/cstudent/016740w/www/cgi-
bin/dbm/sortfiles/names_new");
    if($qy_term =~ /^\W*(\w).*$/){
              $key = $1;
              $key =~ tr/a-z/A-Z/;       ## to upper case
              @NAMESET = @{$NAME{$key}};
              $qy_term =~ /^\W*(\w+).*$/;
              $get = &buildrep::extract_term(@NAMESET, $1);
              if($get =~ /$1/) {              ## get a title
                 print "<TABLE border> \n";
                 print "<TR> \n";
                 print "<TH>Yes, it is a first name term.</TH> \n";
              }
              else {
              print "<TABLE border> \n";
              print "<TR> \n";
              print "<TH>No, it is not a first name term.</TH> \n";
              }
              print "<TR>\n";
              print "<TH>The term list: </TH> \n";
              print_itemlist(@NAMESET, 15);
              $i = scalar(@NAMESET);
              print "<TH>There is $i terms with $key index.</TH>\n";
              print "</TR>\n";
              print "</TABLE>\n";
    }
    else {print "<P> The enter term is not volid!</P> \n";}
}

if($qy_type =~ /last/){
    %LAST = &buildrep::hash_sortfiles(@KEYNAME,
            "/home/cstudent/016740w/www/cgi-
bin/dbm/sortfiles/lastnames_new");
    if($qy_term =~ /^\W*(\w).*$/){
              $key = $1;
              $key =~ tr/a-z/A-Z/;       ## to upper case
              @LASTSET = @{$LAST{$key}};
              $qy_term =~ /^\W*(\w+).*$/;
              $get = &buildrep::extract_term(@LASTSET, $1);
              if($get =~ /$1/) {              ## get a last name
                 print "<TABLE border> \n";
                 print "<TR> \n";
                 print "<TH>Yes, it is a last name term.</TH> \n";
              }
              else {
              print "<TABLE border> \n";
              print "<TR> \n";
              print "<TH>No, it is not a last name term.</TH> \n";
              }
              print "<TR>\n";
              print "<TH>The term list: </TH> \n";
            print_itemlist(@LASTSET, 15);
            $i = scalar(@LASTSET);
            print "<TH>There is $i terms with $key index.</TH>\n";
            print "</TR>\n";
            print "</TABLE>\n";
   }
   else {print "<P> The enter term is not volid!</P> \n";}
}

if($qy_type =~ /location/){
   %LOCATION = &buildrep::hash_sortfiles(@KEYNAME,
          "/home/cstudent/016740w/www/cgi-
bin/dbm/sortfiles/world_places_new");
   if($qy_term =~ /^\W*(\w).*$/){
            $key = $1;
            $key =~ tr/a-z/A-Z/;        ## to upper case
            @LOCSET = @{$LOCATION{$key}};
            $qy_term =~ /^\W*(\w+.*)$/;
            $get = &buildrep::extract_term(@LOCSET, $1);
            if($get =~ /$1/) {               ## get a last name
               print "<TABLE border> \n";
               print "<TR> \n";
               print "<TH>Yes, it is a location term.</TH> \n";
            }
            else {
            print "<TABLE border> \n";
            print "<TR> \n";
            print "<TH>No, it is not a location term.</TH> \n";
            }
            print "<TR>\n";
            print "<TH>The term list: </TH> \n";
            print_itemlist(@LOCSET, 15);
            $i = scalar(@LOCSET);
            print "<TH>There is $i terms with $key index.</TH>\n";
            print "</TR>\n";
            print "</TABLE>\n";
   }
   else {print "<P> The enter term is not volid!</P> \n";}
}

if($qy_type =~ /stoplist/){
   ## Read stopwords into array ##
   @STOPSET = &buildrep::locate_stopwords;
   if($qy_term =~ /^\W*(\w).*$/){
            $qy_term =~ /^\W*(\w+).*$/;
            $get = &buildrep::extract_term(@STOPSET, $1);
            if($get =~ /$1/) {               ## get a stop
               print "<TABLE border> \n";
               print "<TR> \n";
               print "<TH>Yes, it is a stop word.</TH> \n";
            }
            else {
            print "<TABLE border> \n";
            print "<TR> \n";
            print "<TH>No, it is not a stop word.</TH> \n";
            }
            print "<TR>\n";
            print "<TH>The term list: </TH> \n";
            print_itemlist(@STOPSET, 25);
            $i = scalar(@STOPSET);
            print "<TH>There is $i terms with $key index.</TH>\n";
            print "</TR>\n";
            print "</TABLE>\n";
   }
   else {print "<P> The enter term is not volid!</P> \n";}
}


&print_tail;

sub print_beginning{
    print "<HTML>";
    print "<FORM ACTION= \"\" METHOD= \"POST\"> \n";
    print "<H2><CENTER> The Result of Your Query </CENTER></H2> \n";
    print "<HR>\n";
    print "<center>\n";

}

sub print_tail{
    print "</FORM> \n";
    print "</HTML> \n";
}
```

Appendix K

A Perl Module for Printing Information associated with an News Object on the Web Page

```
#!/loc/bin/perl

use newsitem;
use buildrep;
use fullname;
use topicdb;


sub table_terms{
    my($item) = @_;

    my(@gettitle) = $item ->getTitle;
    my(@getauthor) = $item ->getAuthor;
    my(@getdate) = $item ->getDate;
    my(@getagency) = $item ->getAgency;
    my(@getothers) = $item ->getOthers;
    my(@geteventdate) = $item ->getEventdate;
    my(@geteventlocation) = $item ->getLocation;
    my(@getfullname) = $item ->getFullname;
    my($i);
    my($namelist);
    my($first);
    my($middle);
    my($last);
    my($title);
    my($other);
    my($printvalue);

    print "<CENTER> \n";
    print "<TABLE border=0> \n";
    print "<TR bgcolor=\"99cccc\"> \n";
    print "<TD align=center bgcolor=\"99cccc\"> \n";
    print "<B> Title </B></TD>";
    print "<td>$gettitle[0]</td>\n";
    print "</TR>\n";
    print "<TR bgcolor=\"99cccc\"> \n";
    print "<TD align=center bgcolor=\"99cccc\"> \n";
    print "<B> Author </B></TD>";
    print "<td>$getauthor[0]</td>\n";
    print "</TR> \n";
    print "<TR bgcolor=\"99cccc\"> \n";
    print "<TD align=center bgcolor=\"99cccc\"> \n";
    print "<B> Report Date </B></TD>";
    print "<td>$getdate[0] </td>\n";;
    print "</TR>\n";
    print "</table>\n";
    print "<P><table border=2>\n";
    print "<TR bgcolor=\"99cccc\"> \n";
    print "<TD align=center bgcolor=\"99cccc\"> \n";
    print "<B> Event Date </B></TD>";
    print "<TD align=center bgcolor=\"99cccc\"> \n";
    print "<B> Event Location </B></TD>";
    print "<TD align=center bgcolor=\"99cccc\"> \n";
    print "<B> FullName </B></TD>";
    print "<TD align=center bgcolor=\"99cccc\"> \n";
    print "<B> Others </B></TD></tr>";
```

```
     $l = scalar(@geteventdate);
     $m = scalar(@geteventlocation);
     $n = scalar(@getfullname);
     $o = scalar(@getothers);

     $sem_max1 = max($l,$m);
     $sem_max2 = max($n, $o);
     $max= max($sem_max1, $sem_max2);
     for($i=0;$i<$max;$i++){
        print "<tr bgcolor=\"918f8f\">\n";
        $k=$i+1;
        if(($l == 0)||($i>=$l)){print "<td bgcolor=\"918f8f\"><font
color=\"918f8f\">$k</font></td>\n";}
        else { print "<td bgcolor=\"99cccc\">$geteventdate[$i]</td>\n";}
        if(($m == 0)||($i>$m)){print "<td bgcolor=\"918f8f\"><font
color=\"918f8f\">$k</font></td>\n";}
        else { print "<td
bgcolor=\"99cccc\">$geteventlocation[$i]</td>\n";}
        if(($n == 0)||($i>=$n)){print "<td bgcolor=\"918f8f\"><font
color=\"918f8f\">$k</font></td>\n";}
        else { print "<td bgcolor=\"99cccc\">\n";
                 $other = $getfullname[$i]{"other"};
                 $title = $getfullname[$i]{"title"};
                 $first = $getfullname[$i]{"first"};
                 $middle = $getfullname[$i]{"middle"};
                 $last = $getfullname[$i]{"last"};
                 print "<font color=\"black\">$other</font><font
color=\"red\">
                        $title</font><font color=\"blue\">$first</font>
                        <font color=\"brown\">$middle</font><font
color=\"green\">
                        $last </font></td>\n";
        }#else
        if(($o == 0)||($i>=$o)){print "<td bgcolor=\"918f8f\"><font
color=\"918f8f\">$k</font></td>\n";}
        else { print "<td bgcolor=\"99cccc\">$getothers[$i]</td>\n";}
        print "</tr>\n";
     }
    print "</TABLE> \n";
    print "</CENTER> \n";
}

sub max {
my($second) = pop @_;
my($first) = @_;

if ($first >= $second){ return $first;}
else{ return $second;}
}

sub print_tail{
    print "</HTML> \n";
}
1
```

IMAGE EVALUATION TEST TARGET (QA-3)