Structural and Semantic Information Extraction

by

Ralph Harik

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2003

Author .....
Department of Electrical Engineering and Computer Science
February 5, 2003

Certified by.....
Peter Szolovits
Professor
Thesis Supervisor

Accepted by.....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Structural and Semantic Information Extraction

by

Ralph Harik

Submitted to the Department of Electrical Engineering and Computer Science
on February 5, 2003, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Information Extraction, the procedure where only relevant segments of text are extracted from a document, has been applied to several real world problems including stock quote extraction and comparison shopping. This thesis presents a method of Information Extraction where HTML pages are analyzed first at the structural level and secondly at a semantic level. Furthermore, this thesis presents a software agent that crawls the web in search of ongoing MIT lectures, seminars, and events. The software agent individually parses events and provides users with a web interface to a database of heterogeneous events.

Thesis Supervisor: Peter Szolovits
Title: Professor

Acknowledgments

I am grateful to Professor Peter Szolovits for offering me the opportunity to work on such an interesting project, for his enthusiasm concerning the project, for his excellent advice, and for his patience.

I would also like to extend my sincere gratitude to several members of the Clinical Decision Making Group. I would like to thank Doctor Meghan Dierks for her advice, for her effort in securing funding, and for her humor. I would like to thank Doctor Mojdeh Mohtashemi for conceiving the project, for her many technical discussions, and for her encouragement. Finally, I would like to thank both Lik Mui and Michael McGeachie for bringing me into the group, for countless hours of help with technical issues, and for many moments of laughter.

I would like to extend a special thank you to Anna Gallagher for her friendship, support, and encouragement.

Finally and most importantly I would like to thank my family. I would have never made it this far in life without the support, encouragement, and love of my three favorite siblings Georges, Pierre, and Nadine. Nor would I have survived without the love of my wonderful parents Raif and Hanane, who have always put their children first. Mom and Dad your endless love is priceless.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Although searching the world wide web has become easier with the advent of search engines such as Google, problems still remain unsolved. For instance, it is still not possible for an individual to ask for all computer science job postings in California and not have to wade through several pages of irrelevant search results. Ideally, the individual would be presented with a list of all the computer science job postings in California that are published on the Internet. No other irrelevant information should appear on the returned page(s).

1.2 Information Extraction

The aforementioned system requires a software agent that is able to extract relevant items of information from a heterogeneous collection of web pages. The notion of *Information Extraction* [11], the procedure where only relevant segments of text are extracted from a document, is well studied. Note that *Information Extraction* differs from the field of *Information Retrieval*, a procedure where relevant documents are retrieved from a collection of documents. Information Extraction has been used to solve real world problems including stock quote extraction, comparison shopping, restaurant listing extraction, job listing extraction, removal of web page advertise-

ments etc. [2]. Information Extraction is discussed in more detail in the Related Works chapter of this thesis (Chapter 2).

1.3 Event Discovery System

This thesis describes a new method of Information Extraction, where HTML pages are analyzed first at a structural level and secondly at a semantic level. Furthermore, this new approach is used to tackle the problem of discovering ongoing lectures, seminars, and events in the academic community.

Each day, an academic community such as MIT holds several lectures, seminars, and events ranging in topics from "Rank Conditions in Multiple View Geometry" to "Hydrodynamics of Aqueous Humor Outflow". These talks are advertised via fliers posted on bulletin boards, department web pages, and email. In spite of this advertising effort, many individuals are not informed of events that they may find interesting.

One reason for this lack of awareness is that the event information is not located in one place. Each department posts relevant events on its own department web page(s), or in the case of fliers in the vicinity of their own department buildings. Therefore, individuals in the computer science department are unlikely to be aware of events in the biology or math departments. A second reason for the lack of awareness is that there is too much information. Even if the individuals were directed to every department's event listing pages, they may not have the time or patience to read through all the events or sift through extraneous material.

One solution to this problem is to develop a software agent that crawls the web in search of ongoing lectures, seminars, and events and presents the information to the user through a web interface. This thesis describes an agent which crawls the entire MIT domain, structurally and semantically analyzes each page, extracts individual academic events from the heterogeneous pages, and presents the extracted information to the user through an interface available on the world wide web.

1.4 Thesis Road Map

The remainder of this thesis is structured as follows. Chapter 2 provides a discussion of related work. Section 3 provides an in depth discussion on the architecture of the system. Section 4 is an evaluation of the system. Section 5 is a discussion of future work. Finally, the thesis is concluded in Section 6.

Chapter 2

Related Work

This section will discuss Information Extraction, Wrappers (a technique to perform Information Extraction), Wrapper Induction (a technique of learning wrappers), and a few Information Extraction systems that have been built in the past.

2.1 Information Extraction

Information Extraction is the process of extracting relevant segments of text from a document. Although this thesis will focus on Information Extraction of semistructured text, it should be noted that a large amount of research has been done on grammatical parsing and Information Extraction of free text.

In order to automate the process of Information Extraction researchers have built software agents to carry out the extraction. Because the world wide web is so large, and because it contains a vast amount of useful information, a lot of research has been carried out on performing Information Extraction on HTML web pages. HTML pages are examples of semistructured text, structure is found in the source code of the HTML page. As mentioned earlier, researchers have built Information Extraction systems to help collect useful information from the world wide web. They have built systems to automate the retrieval of stock quotes, develop restaurant listings, comparison shopping, job advertisements, seminar announcements etc. [2].

09/12/2002	Athletic Department	02-0000893	**HEAD COACH,** WOMEN'S LIGHTWEIGHT CREW, Athletics, to manage all facets of the women's lightweight crew program, i.e., administration, budget management, fund-raising, recruiting, competition scheduling, team travel, ...
12/12/2002	Athletic Department	02-0001207	**LIFEGUARD,** Athletics (part-time, 20 hrs/wk.), to patrol premises to prevent accidents, rescue bathers in danger of drowning and administer resuscitation, enforce policies and procedures, insure cleanliness, and provide ...
12/18/2002	Center For Cancer Research	02-0000312	**TECHNICAL ASSISTANT,** Center for Cancer Research, to study cell cycle and cancer. Will participate in several research projects under the supervision of the principal investigator with the potential to conduct ...
12/18/2002	Center For Cancer Research	02-0000393	**TECHNICAL ASSISTANT,** Center for Cancer Research, to assist in studies of Huntington's disease (HD) using a range of molecular genetic techniques. Primary lab duties will be genotyping of single nucleotide polymorphisms ...

Figure 2-1: MIT job listings

2.2 Wrappers

One approach to extracting information from semistructured web pages is to use wrappers. A wrapper is a procedure for extracting desired information from a document [8, 1, 10]. In order to extract several pieces of information from a given resource one would develop several wrappers and invoke these wrappers on the page. For example given a semistructured page as shown in Figure 2-1 a user may want to extract all the job titles on the page. As you might have noticed all the job titles appear in bold font. Examination of the HTML source code shows that the job titles are all surrounded by the <B> HTML tag on the left and the </B> HTML tag on the right. Therefore, an appropriate wrapper for the job title might be to extract any phrase surrounded by bold tags. To extract other fields in the document such as the date or the department, similar wrappers would have to be developed. This approach is sometimes known as the left-right (LR) approach (left tag right tag). The nice thing about developing wrappers for Information Extraction is that once the wrappers are developed a user can easily submit queries such as "Give me all job openings posted after 1/23/2003 in the Department of Physics".

2.3 Wrapper Induction

In the past wrappers were written by hand, individuals had to examine the source code and pick out the relevant surrounding tags themselves. This work becomes tedious especially since the layout of web pages change frequently. As a result there has been extensive research on automating the process of producing wrappers. The idea of learning or inducing an appropriate wrapper is named Wrapper Induction. Many systems have been built to learn wrappers including WIEN and STALKER [4, 7]. These systems are discussed in the following section.

Typically wrapper induction has been carried out on pre-selected web sites that display large amounts of information on similarly structured pages. For instance Wrapper Induction could be carried out on a web site such as Amazon.com, which carries millions of books and displays the book information in a similar fashion. The first step that is required when carrying out Wrapper Induction is acquiring a training data set. To create a training set the user picks a few sample pages and identifies the items he is interested in. For instance, a user may identify a certain region of the page as the title/author of the book, and a different region of the page as the book summary. Many systems have developed graphical user interfaces to ease the task of identifying desired information [7, 6]. Once the training set is created, the system uses the selected examples to create extraction rules or wrappers for each marked field. The algorithm used to generate the wrapper varies from system to system; it is what makes each system unique. These wrappers are then used to actually extract the desired information.

2.4 Information Extraction Applications

2.4.1 WIEN

The Wrapper Induction ENvironment (WIEN) [4] was developed by Nicholas Kushmerick, who is accredited as the first person to use the term wrapper induction. WIEN uses the head-left-right-tail (HLRT) approach in order to construct its wrap-

pers. This approach is an enhancement of the LR approach discussed above in section 2.2. Instead of just looking at the preceding and succeeding HTML tags, HLRT also identifies the HTML tags that delimit the head and tail of the desired information. WIEN uses a set of labeled examples to learn an HLRT wrapper. The learning process or wrapper induction continues until the probability of the wrapper failing is below a specified threshold.

2.4.2 STALKER

STALKER [7] uses a supervised learning algorithm to induce its wrappers. The system requires a user to label examples of the information desired. Once this is done the system runs a sequential covering algorithm to learn the wrapper. In other words, the algorithm does not return a wrapper solution until all positive examples are accepted and all negative examples are rejected. Along with the highlighted examples of the desired information, STALKER uses an *embedded catalog* (EC) description of a document in order to generate extraction rules. The EC is a tree-like structure where the leaves represent the information to be extracted [7]. Each internal node of the EC denotes either a homogeneous list of items or a heterogeneous tuple of items. For an example of an EC tree please refer to Figure 2-2. The advantage STALKER has above a system such as WIEN is that STALKER is able to extract information independent of its ordering, and even more it is still capable of extraction if there are missing items.

2.4.3 Commercial Systems

There have also been a few commercial systems that use Information Extraction Junglee, Jango, and MySimon to name a few. Junglee was founded in June 1996 by Stanford graduates and was bought by Amazon for approximately $ 180 million dollars in August 1998 [2]. Junglee is a comparison-shopper which uses information extraction techniques to collect product information from a variety of sites. A similar

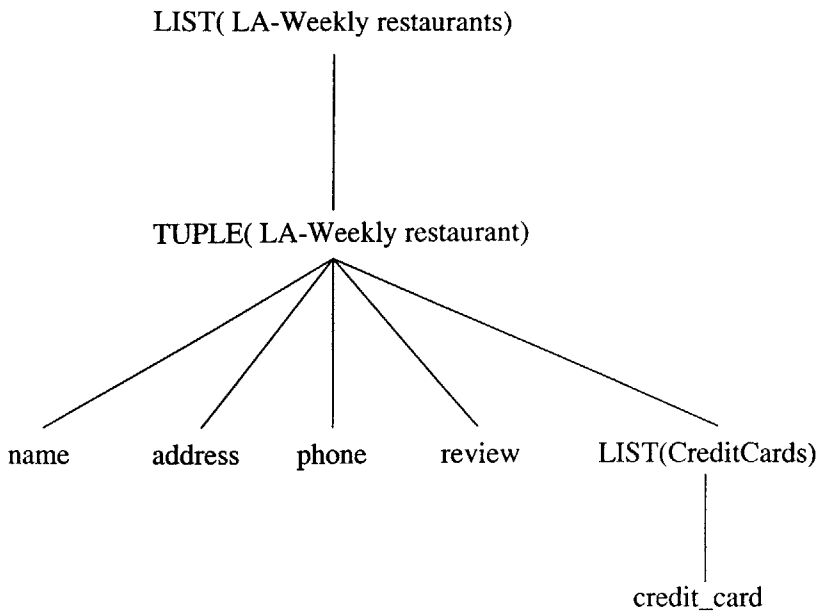

Figure 2-2: An EC description of restaurant listings

application Jango, previously named ShopBot, is a product of NetBot which was acquired by Excite for $ 35 million in October 1997 [2]. Finally, MySimon co-founded by Michael Yang and Yeogirl Yun allowed agents to be manually trained to shop at a merchant site through a GUI environment. For a more complete survey of Information Extraction please refer to [2].

Chapter 3

System Design

The Event Finder system contains three main components: a web crawler, a parse engine, and a web user-interface. The web crawler is a program that systematically visits web sites and saves the HTML source code of the sites in a database. The parse engine processes the crawled pages and extracts any event listings. Finally the web user-interface provides a user with a mechanism to search through the extracted event listings. The C# programming language and the Microsoft Visual Studio.NET environment were used to develop these components.

3.1 Web Crawler

The web crawler processes a page, P, by saving P's HTML source code to a database, and by recursively processing all the pages that P points to. The pages are visited using a breadth first search algorithm. The web crawler takes two variables as inputs, a starting page and a domain specification. The domain specification is used to limit the search space of the web crawler. For instance, if the domain specification is "mit.edu", then the web crawler will only process pages of the form "http://*.mit.edu". The web crawler adheres to the "Robot Exclusion Standard".

3.2 Parse Engine

The parse engine contains three components: an HTML parser, a repeated unit identifier, and an event identifier. The HTML parser takes HTML as input and generates a tree structure. The repeated unit identifier scans the tree and identifies repeated subtree structures in the tree. Finally, the event identifier determines if the repeated units are event listings.

3.2.1 HTML Parser

The HTML parser generates a tree from the HTML source code of a web page. To illustrate the HTML source code shown in Table 3.1 has the corresponding tree structure displayed in Figure 3-1.

With the exception of the root node, the nodes of the tree contain a sequence of HTML tags and the text encapsulated by those tags. For example, node D contains the HTML sequence <tr><td></td><td></td><td></td></tr> and the encapsulated text is "11/18/2002 Michael McGeachie MIT "RedBook Demo: an Ontology-Driven WebPortal" CDM PhD Student". Similarly, node B contains the HTML sequence and the encapsulated text is "2002-2003 MEDG Seminar Series takes place on Mondays at 4:00pm, refreshments are served at 3:45pm.".

On correctly formed HTML pages the parser works by following the simple rule that a child node is added whenever an open tag is encountered. As an example of this observe Nodes B and C, they are child nodes of the root node, Node A. Node B was created when the open tag was encountered, similarly Node C was encountered when the open tag <table> was encountered. Until the corresponding close tag </table> is encountered all new open tags will result in a child node of Node C. Notice that Node C has Nodes D, E, and F as children, because open tags <tr> were encountered before the close tag </table> was encountered.

Only a subset of all HTML tags are allowed to generate new nodes in the tree structure. These tags are listed in Table 3.2. These tags define the structure of an HTML page and are considered enclosing tags. Enclosing tags are used to generate

```
<html>
  <body>
    <p>
      The 2002-2003 MEDG Seminar Series takes place on Mondays at 4:00pm,
      refreshments are served at 3:45pm.
    </p>
    <table>
      <tr>
        <td>11/18/2002</td>
        <td> Michael McGeachie (CDM PhD Student)
          <b>MIT</b>
        </td>
        <td> "RedBook Demo: an Ontology-Driven WebPortal"
        </td>
      </tr>
      <tr>
        <td>11/25/2002</td>
        <td> Ralph Harik (CDM Meng. Student)
          <b>MIT</b>
        </td>
        <td>
          "EventFinder:A software agent that crawls the web for events
          and matches people to events of their interest."
        </td>
      </tr>
      <tr>
        <td>12/2/2002</td>
        <td> Peter Szolovits (CDM Director, MIT Professor)
          <b>MIT</b>
        </td>
        <td> "Topic to be determined.""
        </td>
      </tr>
    </table>
  </body>
</html>
```

Table 3.1: Example HTML source code

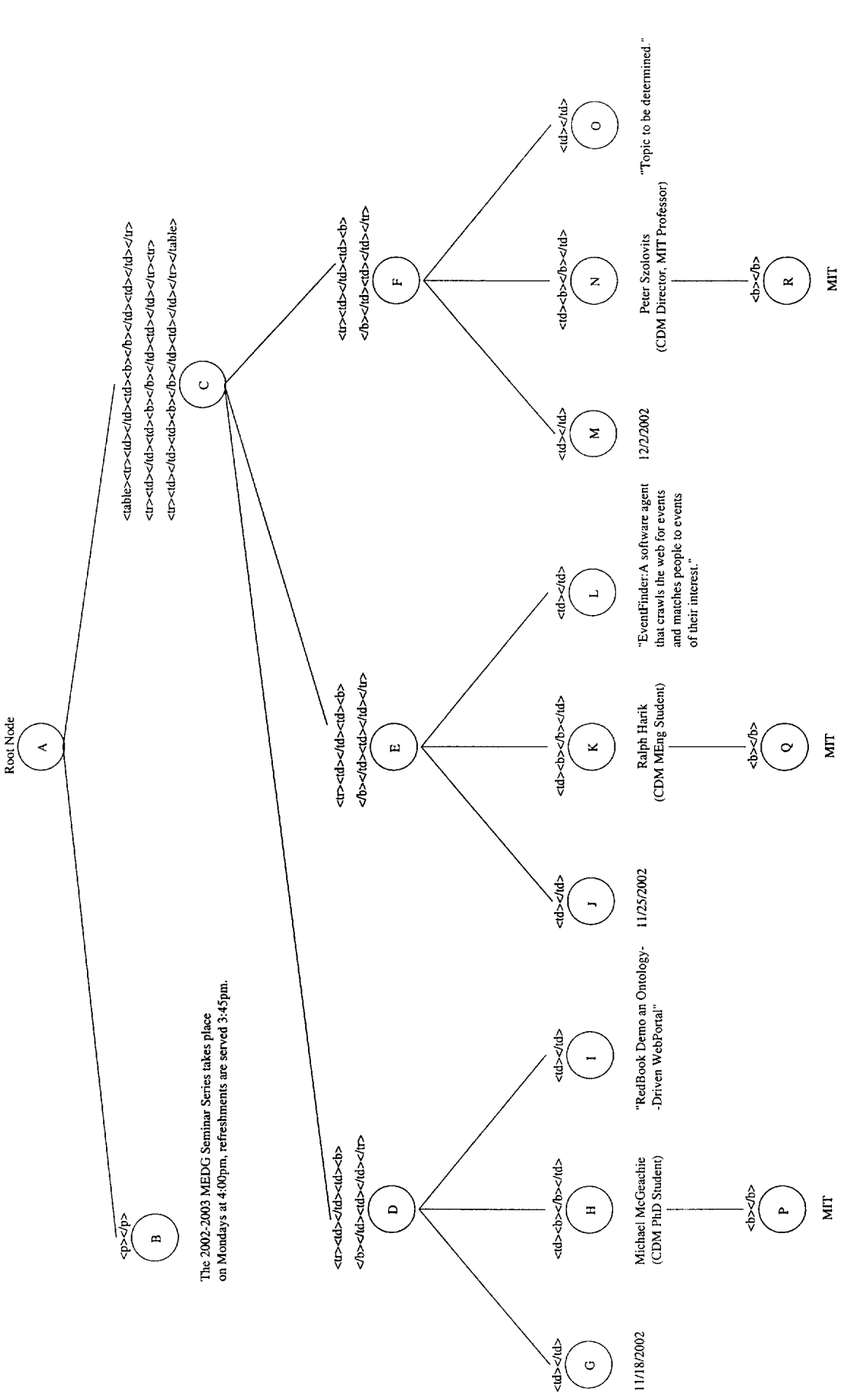

Figure 3-1: Generation of tree from HTML source code

<TABLE>
<TR>
<HR>
<UL>
<LI>
<DL>
<DT>
<DIV>
<P>
<BLOCKQUOTE>
<SPAN>

Table 3.2: Tags that generate new nodes in the parse tree

Event
Seminar
Lecture
Colloquia
Colloquium

Table 3.3: One word from this list must appear on page in order for page to be parsed.

lists (<ul>, <ol>, <li>), tables (<table>, <tr>), and paragraphs (<p>). Tags that don't have a structural effect such as <b> or <i>, tags that affect the font of the text, are not included.

Not all pages are processed by the HTML parser. The page must contain one of the words shown in Table 3.3

3.2.2 Repeated Unit Identifier

The Repeated Unit Identifier, RUI, scans the parse tree generated by the HTML parser and identifies repeated subtree structures in the tree. More specifically, for all the nodes at a given depth of the parse tree the RUI groups nodes that share similar HTML tag sequences.

It's important to note that only nodes at the same tree depth in the HTML parse tree are grouped together. So for each level of the HTML parse tree the RUI iterates over all the nodes and compares each node's HTML tag sequence to the HTML tag

sequences already observed at that level. If the HTML tag sequence of a Node X has already been observed, meaning if a previous node had the exact same sequence as Node X, then Node X will be added to the same group as that previous node. For example two nodes that both have the HTML tag sequence <TD><B></B></TD> will be grouped together. If the HTML tag sequence of Node X has never been seen, then the RUI checks if the HTML tag sequence of Node X is an approximate match with any of the HTML tag sequences that have already been observed. If an approximate match does not exist, then a new group is created with Node X as a member. If however an approximate match is made then Node X is added to the group identified by the HTML tag sequence that was approximately matched.

Two HTML tag sequences X and Y are approximate matches if the following equation holds:

$$\frac{LCS\ Length}{Max\ Length\,(X,Y)} \geq \alpha \tag{3.1}$$

The value of LCS Length is the length of the longest common subsequence of the two sequences X and Y. MaxLength(X, Y) returns the length of the longest sequence amongst X and Y. Finally, α is a value that was picked experimentally, currently the value of α is .45. The rationale behind the equation is that for sequences that are not similar the LCS Length divided by Max Length(X,Y) will be close to 0, and for those that are similar it will be close to 1. The Max Length(X,Y) is in the denominator to compensate for the case when X is a subsequence of Y and X is a much shorter sequence than Y. In this case X and Y are not really similar and the equation will return a value close to 0.

Before completing the RUI performs two more procedures. First the RUI merges groups that have approximate matching HTML tag sequences together. Secondly it filters the groups by removing any group that contains fewer than 3 nodes. Groups containing fewer than 3 nodes are removed to ensure that the system only processes lists. Some pages, pages that contain news articles for example, share the same qualities as events (name, date, institutional affiliation). Therefore, by enforcing a

minimum of 3 nodes the system is less likely to mistaken a news article for an event.

3.2.3 Event Identifier

The Event Identifier (EI) uses the result of the RUI to determine if the repeated units in the HTML parse tree represent events. The EI performs this task by analyzing the text encapsulated by the nodes in the repeated unit group. More specifically, the EI scans the text for qualities inherent in an event listing such as a date, an institutional affiliation, and a person's first name. If the text encapsulated by a node contains 2 of the 3 aforementioned qualities then that node is marked as an event node. Furthermore, if within a repeated unit group more than 50% of the nodes are marked as events then the group is marked as an event listing. The text under each node is then extracted individually and stored into a database.

The following sections describe how the three attributes of an event are identified.

Date Identifier

The Date Identifier consists of two modules a tokenizer and a grammar. The tokenizer iterates over every word in a text segment and identifies date related words. For example the module identifies the months of the year, integers between 1 and 31, and date delimiters such as '/' and '-' (12/6/1980, 12-6-1980). Once this is done, the tokenized string is processed and the tokens are reduced to a sentence in the grammar if possible. An example of a sentence in the grammar may be MONTH FORWARDSLASH DAY FORWARDSLASH YEAR.

Many pages do not repeated list the year of the date with the month and day, rather they list the year at the top of the page or at the top of the event list. In order to take this into account, if a node is identified as an event and it is missing a value for the year the event takes place, the event is given a year value equal to the most recently seen year.

Professor
University
College
Department
Institute
Laboratory

Table 3.4: Words that signal an institutional affiliation.

Affiliation Identifier

The Affiliation Identifier is rather simple; it looks for an occurrence of any of the words in the short list in Table 3.4 and their variations.

First Name Identifier

In order to write a First Name Identifier, 500,000 first names were parsed from the web site "http://www.kabalarians.com/gkh/yourbaby.html", a site that helps you select a name for your baby. Because the web site allows anyone to submit a name, the database is tainted with words such as "and", "the", "when" etc. Therefore 5,000 of the most commonly used English words were downloaded from a separate site, and they were subtracted from the 500,000 first names. A hash table of the remaining names was created. Therefore the identification of a first name only required a lookup in the hash table.

3.3 Web User-Interface

The graphical user interface (GUI) allows individuals to search the database of events through keyword matching. Furthermore, the user is allowed to limit the search results by specifying a start date and/or end date. Finally the GUI allows individuals to create a profile, a list of words that describe one's interests. When the user logs into to the system the profile is used to return any matching events. In the future the profile will be used to email users events that match their interest. Snapshots of the GUI are displayed in Figures 3-2 and 3-3.

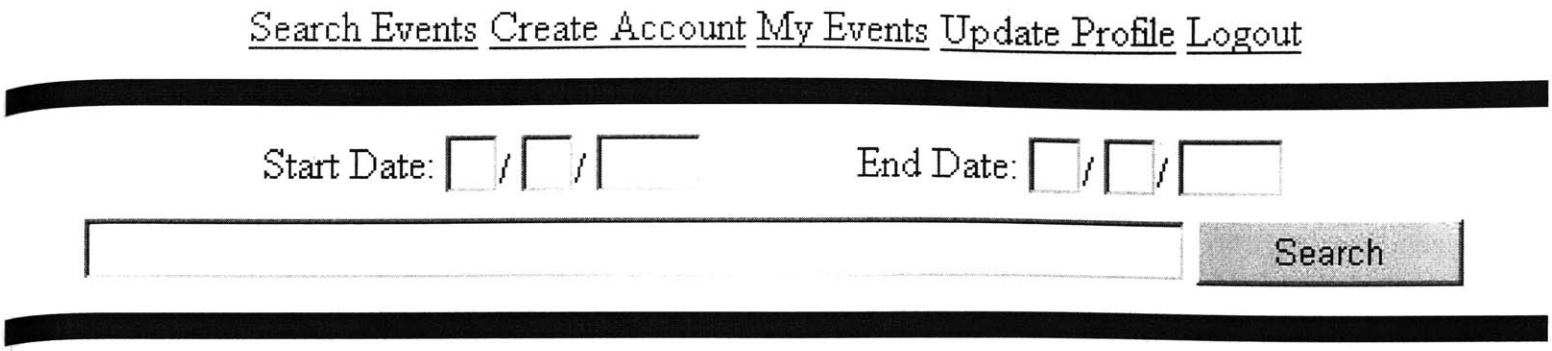

Figure 3-2: The search box presented to the user.

Date: Wednesday, 9/11/1996
Information: Date: Wednesday, September 11, 1996 Title: The Policy Crisis in Cryptography Speaker: Dr. Herb Lin, NRC Abstract: Cryptography, the work of creating and deciphering coded information using mathematical formulas, long has been the sphere of spies and the military. But in the past 10 years private-sector use of cryptography has exploded as a result of advances in electronic communications and information technologies. Decisions about national cryptography policy now have important implications not only for national security, but also for U.S. economic competitiveness, law enforcement interests, and the protection of the privacy and other rights of individuals. The Computer Science and Telecommunications Board of the National Research Council recently completed a congressionally mandated study (Cryptography's Role In Securing the Information Society) to examine the issues and conflicting interests involved in cryptography and made recommendations on national policy in this highly controverial area.
Url: http://theory.lcs.mit.edu/~cis/cis-talks.html

Date: Friday, 3/1/2002
Information: Date : Friday, March 1, 2002 Time : 10:30 a.m. Place : NE43-308, 200 Tech Square Title : Physical Security Protocols for Ubicomp Systems By : Tim Kindberg, Hewlett-Packard Labs, Palo Alto (Open to the public) Abstract : Ubiquitous systems tend to have two predominant characteristics: first, the integration of computing components and the physical world; second, spontaneous (ad hoc) interoperation between components. Each of these characteristics has ramifications for security. My talk will cover some of the implications of physical integration by considering notions of "constrained channels" and "context authentication"; it will cover protocols for authenticating a principal's location and consider the issues raised by other contextual parameters. If there is time, the talk will go on to an overview of how physical relationships can be used for validating spontaneous associations between devices. Bio: Tim Kindberg (http://champignon.net/TimKindberg) is a senior researcher at HP Labs, Palo Alto, where he works on nomadic computing systems as part of the Cooltown program. He was formerly a senior lecturer in Computer Science at the University of London. He holds a PhD in Computer Science from the University of Westminster and a BA in Mathematics from the University of Cambridge. His research interests include ubiquitous computing systems, distributed systems and human factors. He is co-author of the textbook 'Distributed Systems -- Concepts & Design'. NOTE: : No Seminar Fri, Feb 22, 2002
Url: http://theory.lcs.mit.edu/~cis/cis-talks.html

Figure 3-3: A snapshot of some events returned when the word "computer" was entered in the search box.

Chapter 4

Evaluation

This section describes experiments that assess the performance of the system. There also is a discussion of the system's strengths and weaknesses.

4.1 Event Listing Identification

In order to evaluate the performance of the system 100 pages that contained events were hand selected from the mit.edu and stanford.edu domains. The pages were selected from a variety of sources. An attempt was made to select pages that had unique event listing layouts. The goal of the experiment was to determine which types of pages the system performed well on, and which pages that system performed poorly on. The pages contained several kinds of MIT and Stanford events including social, academic, and art events.

4.1.1 Results

Of the 100 pages in the test suite, the system failed to parse a single event on 36 pages. Of the remaining 64 pages the system correctly parsed 54 pages. A page was deemed to be correctly parsed if a mistake was made on 2 events or less. The system partially parsed 10 pages. A page was deemed to be partially parsed if it made a mistake on more than 2 events.

4.1.2 Discussion

The following is a list of pages that the system performed poorly on:

- **Errors in HTML Tree Constructor** The most common reason for the system's failure, is its inability to compensate for poorly structured HTML pages. Because HTML is very flexible in its syntactical rules, writing software that creates an accurate tree is not trivial.

- **Month Listed Once:** The system is unable to extract events where the month is listed at the top of the page, and each event is then delimited only by the day of the month. The reason is that the date identifier requires the month and day of month be adjacent. Furthermore, the event identifier requires that each event have a date explicitly associated with it.

- **Flat HTML Pages:** Pages that don't have hierarchical HTML structure can't be parsed by the system. For example, pages that separate events with
 tags can't be parsed. Recognizing
 as enclosing tags is a possible solution. However, since
 tags are very prevalent in HTML source code, recognizing them as enclosing tags would increase the number of incorrectly identified event pages.

- **Event Contains Other Dates in Text:** One of the attributes the Event Identifier searches for is a single date within the text. However, as it turns out some events have references to other dates. For example, some events have RSVP dates. This problem can be solved by writing more detailed recognizers so that RSVP dates could be distinguished from other dates.

- **Date and Text Not Under Same Hierarchical Node** A few of the pages, which did have heirarchical structure, did not have the date and the event information enclosed by an HTML tag. It may be possible to recognize the repeated date and event information and group the two together (more work is required).

- **Missing Event Attribute** The system failed to extract certain events because the event lacked both a first name and an institutional affiliation. An example of such events is a list of social outings for a department lab "January 19, Movie outing to see Lord of the Rings".

- **Errors in Date Recognizer** The date recognizer failed to identify some formats.

The system tends to work best on pages whose HTML source code is hierarchically structured (events listed in an HTML table for example) and events which are uniform in presentation. Note that the since an attempt was made to select a variety of layouts, the composition of the test suite doesn't necessarily represent the true percentages of the types of pages found on the Internet. Therefore, the experiment does not give an accurate estimation of the system's success rate.

4.2 False Positives

In addition to the above experiment, a second experiment was carried out where 1000 pages were selected from a variety of sources on the the mit.edu domain. None of these pages contained event listings. These pages were then processed by the system in order to determine how many false positives existed (how many pages the system incorrectly labeled as events).

4.2.1 Results

Surprisingly the system did not incorrectly classify any of the 1000 pages.

4.2.2 Discussion

Although the experimental results were positive, working with the system has shown that there are certain pages which the system falsely identifies as event pages. Below is a listing of the type of pages that sometimes are incorrectly identified as event listings:

- **Bulletin Boards** Bulletin boards cause problems because each posting has a single date as well as the name of the person who posted the message. Furthermore, the messages are posted in a uniform and structured manner. To mitigate this problem the system recognizes the bulletin boards listings that include the email "From:", "To:", and "CC:" headers. In this case the system does not incorrectly identify the page.

- **Bibliographies** Often on university pages researchers publish long lists of their academics papers. These listings contain a date, a name, and many times an institutional affiliation.

- **Job Listings** Job listings are usually posted in a structured manner. Furthermore, each posting often contains the date the job was posted, and the name of the individual to contact.

- **News Listings** News articles also sometimes cause problems, as they are listed in a structured manner, contain a date of publication, and the name of the author.

The main reason that the system performed well on the second experiment is that the system does not process a page unless it contains one of the words listed in the table 3.3. Furthermore, the page also needs to have a structured list that contains the event attributes described in section 3.2.3.

Chapter 5

Future Work

This section will discuss possible improvements to the Event Finder System.

5.1 Crawler

A faster crawler is necessary for the system to efficiently crawl all university domains. Furthermore it is desirable that the crawler is run more frequently so that the most recent HTML pages are found in the database. A faster crawler can be attained by executing the steps listed below:

- **Implement crawler in C++:** Because the .NET environment is relatively new, it contains a few bugs. Specifically a memory leak was found in the networking library of the .NET environment. Therefore, implementing the crawler in C++ would lead to a more robust crawler.

- **Identification of mirror sites** Some sites are replicated in their entirety under different URLs. The only difference between the replicated pages and the original pages are the URLs. This causes a problem since the hash table of already seen pages is indexed by the URLs. Since the content of the pages is the same, it is inefficient to crawl both the replicated pages and the original pages. A possible solution is to take a fingerprint of the content on a page and to only crawl pages that do not match any of the existing fingerprints. Because there

may be slight variations in the HTML source code, link references for example, producing a fingerprint is not as trivial as hashing the content of the page.

- **Use of more computers** The system already makes simultaneous web requests. However, to further improve on the crawler's efficiency the crawl could be distributed over several computers.

5.2 Parser

The parser, the crux of the system, has several possibilities for improvement as well.

5.2.1 HTML Tree

Poorly written HTML source code was the main reason many of the event pages were parsed incorrectly in the experiment described in section 4.1. The consequence of missing closing HTML tags was an incorrect parse tree and a failure of the repeated event identifier module. In order to remedy this mistake the HTML tree generator needs to be more intelligent. It needs to correct sloppily written HTML code. Although the module that builds a tree out of the source code compensates for some mistakes, it is far from robust. Additionally, if more university sites are going to be crawled the HTML tree module needs to be much faster. One possible solution is to use software from the public domain implementation of Mozilla. This software is likely to be very robust.

5.2.2 Repeated Unit Identifier

The Repeated Unit Identifier can also be enhanced. A more sophisticated algorithm can be used to cluster the repeated HTML strings on the page. Many algorithms have been developed in the field of bioinformatics that look for repeated nucleotide sequences in a DNA strand [5, 3, 9]. Several of these algorithms use probabilistic models for identifying approximate tandem repeats in a string. It would be interesting

to implement some of these algorithms and compare them to the technique developed in this thesis.

5.2.3 Event Identifier

Certainly a missing aspect of the Event Identifier module is that it does not learn. As the system identifies pages that contain lists of events, it could run a learning algorithm that learned commonalities among all the events. Furthermore, a confidence score of which lists are indeed event listings could be developed. One problem is that since the system searches for repeated units, pages that contain only a single event are missed. This could perhaps be remedied with the learning module that identifies commonalities among the events.

Another improvement along the lines of learning is that the events could be classified into their most relevant fields. For instance events discussing recombinant DNA could be classified under molecular biology, and events discussing the politics of the Middle East could be classified under international affairs. This would allow a user to subscribe to all molecular biology seminars, or all international affairs seminars etc.

Currently the system only seeks repeated listings which are lectures, seminars, or events. However, it is certainly possible to customize the system to search for any other items that are contained in repeated lists. The system currently identifies repeated lists that contain a first name, date, and institutional affiliation. Although the system does allow you to substitute your own recognizers, it is not as simple as saying "find me repeated lists that contain email addresses, first names, street location, state, and price" (if you wanted to search for housing). Instead you would have to write your own recognizers in C#. Ideally a large set of recognizers would already exist and the system would allow you to simply indicate which types of recognizers you would like to use. Perhaps this could be done through a simple check-box list.

5.3 User Interface

The user interface to the system is the least developed of all the modules. Currently the only available user interface is the web interface described in section 3.3. However, many improvements can be made in order to make the system more usable. The system should be ported onto a variety of handhelds. This would allow individuals to check for new seminars and events while on the run. Furthermore, with a reliable location recognizer GPS functionality could be built into the system. With GPS a user could choose to only be notified of events that are close geographically. Another improvement is a rating system that would allow users to rate the events they attend. This would be useful for users who are deciding whether or not to attend a talk. The ratings would be useful for instance if there are repeated talks by the same person, or if a talk is part of an ongoing colloquium series. Furthermore, a module that matched people to events of their interests is also necessary.

Chapter 6

Conclusion

This thesis has presented a new method of Information Extraction where HTML pages are analyzed first at a structural level and secondly at a semantic level. Specifically, the method looks for repeated units of structured elements as an indication of a list of similar entities. It has also presented a software agent that crawls the web in search of ongoing MIT lectures, seminars, and events. Furthermore, the performance of this software agent was evaluated on a subset of the MIT and Stanford domains. The software agent was most affective on pages that had well structured HTML source code and uniform event listings. Finally, several system improvements were suggested.

Bibliography

[1] B. Chidlovskii, U. Borghoff, and P. Chevalier. Towards sophisticated wrapping of web-based information repositories. *Proc. Conf. Computer-Assisted Information Trerieval*, pages 123–35, 1997.

[2] Line Eikvil. Information extraction from world wide web - a survey. Technical Report 945, Norweigan Computing Center, 1999.

[3] Roman Kolpakov and Gregory Kucherov. Finding approximate repetitions under Hamming distance. *Lecture Notes in Computer Science*, 2161:170–??, 2001.

[4] Nickolas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper induction for information extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.

[5] G. M. Landau and J. P. Schmidt. An algorithm for approximate tandem repeats. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching*, number 684, pages 120–133, Padova, Italy, 1993. Springer-Verlag, Berlin.

[6] Robert C. Miller and Brad C. Myers. Lightweight structured text processing. In *Usenix Annual Technical Conference*, pages 131–144, 1999.

[7] I. Muslea, S. Minton, and C. Knoblock. Stalker: Learning extraction rules for semistructured, 1998.

[8] Y. Papakonstantinou, H. Garcia-Monlina, and J. Widom. Object exchange across heterogeneous information sources. *Proc. 11th Int. Conf. Data Engineering*, pages 251–60, 1995.

[9] E. Rivals, O. Delgrange, J. P. Delahaye, M. Dauchet, M. O. Delorme, A. Henaut, and E. Ollivier. Detection of significant patterns by compression algorithms: The case of approximate tandem repeats in DNA sequences. *Comput. Appl. Biosci.*, 13(2):131–136, 1997.

[10] M. Roth and P. Schwartz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. *Proc. 22nd VLDB Conf.*, pages 266–75, 1997.

[11] S. Soderland. *Learning Text Analysis Rules for Domain-specific Natural Language Processing.* PhD dissertation, University of Massachusetts, Department of Computer Science Technical Report 96-087, 1996.