2000

UniversityIE: Information Extraction From University Web Pages

Angel Janevski
University of Kentucky, ajane0@sac.uky.edu

ABSTRACT OF THESIS

UNIVERSITYIE: INFORMATION EXTRACTION FROM UNIVERSITY WEB PAGES

The amount of information available on the web is growing constantly. As a result, the problem of retrieving any desired information is getting more difficult by the day. To alleviate this problem, several techniques are currently being used, both for locating pages of interest and for extracting meaningful information from the retrieved pages. Information extraction (IE) is one such technology that is used for summarizing unrestricted natural language text into a structured set of facts. IE is already being applied within several domains such as news transcripts, insurance information, and weather reports. Various approaches to IE have been taken and a number of significant results have been reported.

In this thesis, we describe the application of IE techniques to the domain of university web pages. This domain is broader than previously evaluated domains and has a variety of idiosyncratic problems to address. We present an analysis of the domain of university web pages and the consequences of having them input to IE systems. We then present UniversityIE, a system that can search a web site, extract relevant pages, and process them for information such as admission requirements or general information. The UniversityIE system, developed as part of this research, contributes three IE methods and a web-crawling heuristic that worked relatively well and predictably over a test set of university web sites.

We designed UniversityIE as a generic framework for plugging in and executing IE methods over pages acquired from the web. We also integrated in the system a generic web crawler (built at the University of Kentucky) and ported to Java and integrated an external word lexicon (WordNet) and a syntax parser (Link Grammar Parser).

Angel Janevski

Date

UNIVERSITYIE: INFORMATION EXTRACTION FROM UNIVERSITY WEB PAGES

By

Angel Janevski

Director of Thesis

Director of Graduate Studies

Date

RULES FOR THE USE OF THESES

Unpublished theses submitted for the Master's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgments.

Extensive copying or publication of the theses in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

A library that borrows this thesis for use by its patrons is expected to secure the signature of each user.

Name	Date

THESIS

Angel Janevski

The Graduate School

University of Kentucky

2000

UNIVERSITYIE: INFORMATION EXTRACTION FROM UNIVERSITY WEB PAGES

THESIS

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at the University of Kentucky

By

Angel Janevski

Lexington, Kentucky

Director: Dr. Victor Marek, Professor of Computer Science

Lexington, Kentucky

2000

MASTER’S THESIS RELEASE

I authorize the University of Kentucky
Libraries to reproduce this thesis in whole or
in part for purposes of research.

Signed: _____

Date: _____

ACKNOWLEDGEMENTS

I want to begin by thanking my Thesis Chair, Dr. Victor Marek for his continuous guidance and support. This includes not only the period of the creation of this Thesis, but also throughout my Master's Studies at the University of Kentucky. Then, I would like to thank Dr. Mirek Truszczynski who provided me with lot of insight and help during my Independent Studies projects. Dr. Marek, Dr. Truszczynski and Dr. Raphael Finkel, the members of my Thesis Committee, guided me through my research work and added many valuable comments integrated into this work. Tony Borchers and Joe Oldham, as part of the same research group, were valuable sources of ideas and comments and also helped the integration with their work.

I would also like to thank my colleagues at Philips Research for the support and understanding during my work on the Thesis. Special thanks to Nevenka Dimitrova and Guru Banavar who magically managed to find energy and time to provide me with numerous valuable suggestions for the Thesis text.

Many other have made my studies at UKY one of the brightest experience in my life. I would like to thank everyone at the CS Department with an emphasis to Dr. Grzegorz Wasilkowski for his trust in me and his continuous help from the first day at UKY.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE: BACKGROUND

Information Extraction (IE) deals with data extraction from unrestricted texts. IE is can best be classified as Natural Language Processing (NLP). In this chapter we briefly introduce the reader to NLP and IE and set the stage for the following chapters. Here we present a generic IE system and the two different NLP approaches also applicable to IE. Finally, we introduce the input domain – Web documents.

NATURAL LANGUAGE PROCESSING

From the earliest days of the history of Artificial Intelligence (AI), Natural Language Processing (NLP) has been one of the primary interests. NLP is investigates mechanisms for communication through natural language – both natural language generation and understanding. A system that contains either one will is considered a NLP system.

The initial results were highly encouraging for the emerging NLP community, but the work also pointed out many challenges of the problem domain. NLP not only requires good lexical and grammatical processing abilities, but also semantic, pragmatic and general knowledge (or at least domain) understanding.

The work in NLP can be divided in two major directions: rationalist (rule-based) and empirical (corpus-based). The rationalist approach takes the route of encoding all domain knowledge and later using it to process natural-language texts. The empirical approach is a different paradigm where the domain knowledge encoding is (at least partially) automated. The automation is achieved by training the NLP system with annotated examples of the extraction results. After a number of iterations, the NLP system builds some domain knowledge.

The NLP research community attacked many problems: Machine Translation, Syntax Analysis, Semantic Analysis, Speech Recognition, Discourse Analysis and more recently Information Extraction (IE). The remaining part of this section will focus on IE. For an overview of the present state of research in NLP, [1] provides a good starting point.

INFORMATION EXTRACTION

There are many NLP systems that can process arbitrary text. Several general-purpose linguistic capabilities are characteristic for this type of systems: part-of-speech tagging, parsing, word-sense disambiguation, higher level (semantic) understanding, dialog systems, natural language interfaces and queries, etc. We are interested in NLP systems that are built with a pre-specified task over a well-defined domain of interest. These systems are Information Extraction Systems. Unlike the "in-depth" NLP systems, IE systems effectively skim the text from the domain, find relevant sections and then focus only on those sections in the subsequent processing.

In other words, IE systems:

- take as input an unrestricted text and "summarize" the text with respect to a pre-specified topic or domain of interest
- find useful information about the domain from the summarized text
- encode the information in a structured form that is suitable for populating databases

An example IE system is given in Figure 1 where news articles are taken as input and information is extracted on certain events related to a specific company. We could think of it as a filter for articles, but also as a system that produces results that are easy to manipulate. For example, we may be interested in new product announcements that IBM has released in the news. We could specify few attributes that we want to extract, such as product group, product name, release date and estimated price.

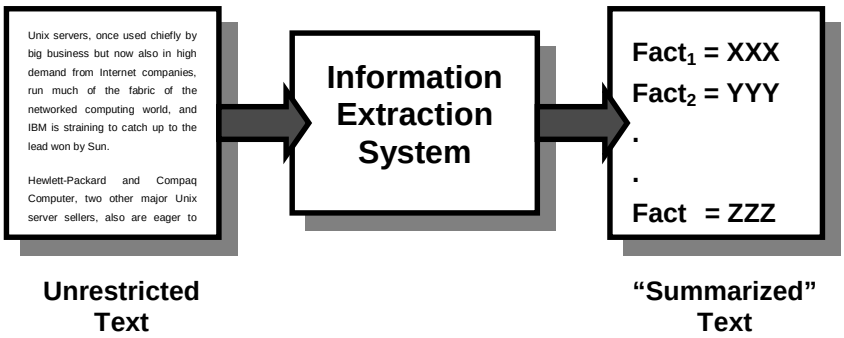

Figure 1: Example IE system for extraction of news events

In general, IE systems are capable of (at least partially) understanding the content in the input. This capability is realized by cutting out the uninteresting portions of the text and extracting information from the remainder. This information is then structured and provided as output. IE takes more than some trivial pattern matching or document structure analysis. In the IBM example, it would not be sufficient to look for 'IBM', 'product' and an amount in US dollars mentioned in the same sentence in an article. Product announcements can be structured differently from, but can also have many similarities with, for example, product reviews. In addition, different sources (e.g. magazines, newspapers) may have significantly different styles of presenting product announcements.

From the above brief discussion, we can make several observations about IE systems:

- They work over unrestricted natural language text as input – IE is a NLP problem.
- IE systems are domain-specific – they need a domain model.
- Only a portion of the input is of interest – real in-depth natural language understanding may not be necessary.
- The extracted facts are a well-defined set.
- Substantial computational resources are required.

In the early days of IE, there were many different approaches to text processing. Some ran full-scale syntax analysis, while some applied additional semantic and discourse analysis of the input. At present, IE systems generally follow the architecture presented in Figure 2. The architecture description is extracted from [5].

GENERIC IE SYSTEM ARCHITECTURE

In the generic IE system in Figure 2, there are no assumptions on the input format. The Tokenization and Tagging phase tokenizes the text – divides it into sentences with words. Some systems can eventually disambiguate or tag for parts of speech (POS) or semantic class. The Sentence Analysis stage parses the sentence for simple constructs (verb, noun, prepositional, and other phrases). This stage could also involve parsing for higher-level constructs and even label semantic entities in the text and transform them to normalized form. Up to this point, the two stages of the system are not necessary domain-specific.

Extraction is the first stage where the processing is tied to the domain. Here, relevant text parts are extracted, but it is important to point out that this stage is not just extraction of text fragments. In addition, this stage annotates relations between parts of the text, and additional information that describes the triggers of the extraction. The Merging stage compares the entities extracted in the previous stage and deduces whether they refer to the same information. This comparison filters the extracted information and is the place where separate extraction results are combined into one when they refer to the same piece of information. The final, Template Generation stage produces the output. This stage considers only extracted information relevant to the output format.

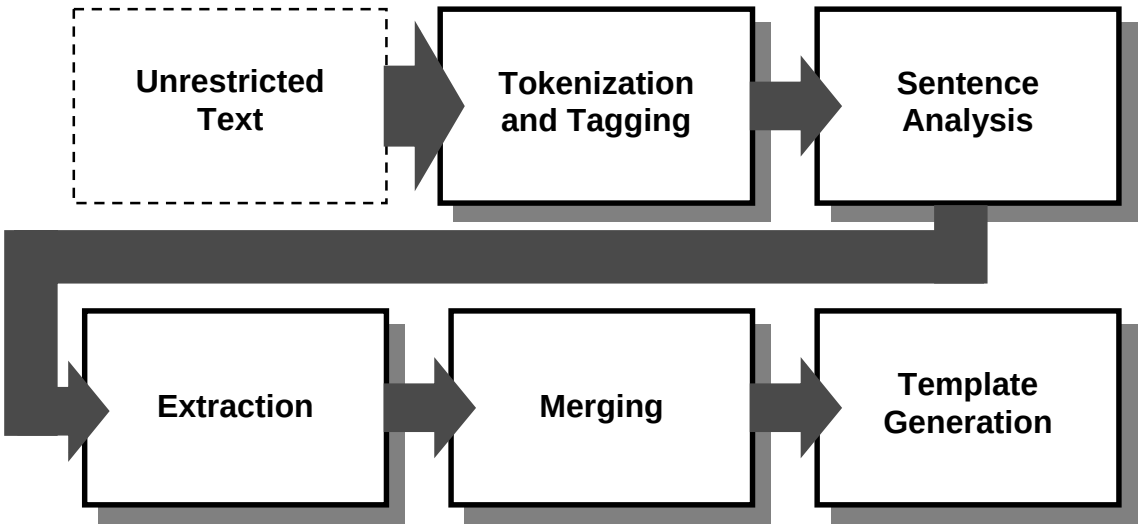

Figure 2: Generic IE System Architecture

To get a better picture of how the stages work, let us look at several examples.

We'll apply the Tokenization and Tagging stage to the following two sentences: *International students are required to present scores from the Test Of English as a Foreign Language test. A minimum of 550 is required for the Graduate, and 500 for Undergraduate School applicants.* Two sentences are recognized and words are tagged with data types. For example, 550 and 500 in the second sentence are assigned the type Integer.

The Sentence Analysis stage analyzes the sentence and extracts constructs. Consider another example: *Northern's main campus is situated on 325 acres of rolling countryside*

along U.S. Highway 27, seven miles southeast of Cincinnati, Ohio. Say we are interested in the campus location. The Sentence Analysis stage will extract the verb phrase: *situated on 325 acres … southeast of Cincinnati, Ohio.* This stage could also normalize parts of the text. In the case of the first example sentences, *Test Of English as a Foreign Language* is replaced with *TOEFL*. This way, any further processing will expect only the normalized form and will look only for TOEFL in the text.

The Extraction stage extracts relevant data. It extracts 550 or 500 from the TOEFL sentences, or the location description from the campus-location sentence. The Merging stage interpolates various pieces of data extracted earlier. An example of merging is when one extraction match produces part of the tuition, e.g. fall and spring semesters, and another the summer semester tuition. This stage merges the two pieces of extracted data. The final, Template Generation stage, formats the output from the extraction and merging.

RATIONALIST (RULE-BASED) vs. EMPIRICAL (CORPUS-BASED) APPROACH

IE has been considered from both rational and empirical perspectives. The fact that IE is always tied to a domain means that there must be some (presumably rich) knowledge of it. Encoding the rules for IE is certainly not an easy task and is constrained by the domain characteristics.

An IE system designed using the rationalist approach incorporates a precise domain definition and can give very good results. Developing such a system with a domain model is time consuming and implies duplication of the effort of when the same system is applied to a new domain. On the other hand, one of the motives for building corpus-based IE systems using the empirical approach is the availability of annotated text. If a training corpus is available, it could be easier to configure an IE system for the domain with no, or very little, human interaction. Table 1 summarizes the differences between the two approaches.

There are several widely accepted training corpora such as Penn Tree Bank and WordNet that are used to support automated learning systems. Other corpora are available through

the Linguistic Data Consortium [5]. A longer discussion and comparison of the two approaches can be found in [3].

EXTRACTING INFORMATION FROM THE WEB

Among the rapidly changing characteristics of the Web is the trend of increasing amount of online content. Most of it is, of course, natural language text. Aside from being presented as HTML, these texts have no common structure or access standards. Although there is an ongoing process of incorporating meta-information in it, most of Web content can currently be considered as noisy input to NLP systems. These systems expect well-formed, grammatical sentences and are rarely efficient on noisy input.

	Rationalist	**Empirical**
Training Set	At least a small set of annotated examples	Larger training corpus
Vulnerability to Imperfect Input	The expert can filter out inconsistencies	Could introduce bigger errors due to the automation
Domain Description	Ontology and expert's knowledge	Ontology and annotations in the training set
Learning Algorithm	Not required	Major component (resource consuming)
Training	Only content observed by the designer of the rules	Depends on the learning algorithm, but in general, a larger set of examples is required
Performance	Very good	Very good, close to the rationalist-based system
Portability	Very hard (new domains defined from scratch)	Relatively easy, to domains with existing training corpus

Table 1: Rationalist vs. Empirical IE

Looking at web content from the perspective of IE brings out two additional considerations: text pre-processing and building IE rules.

In the discussion so far, we assumed the IE system input to be *documents* relevant to the extraction task. When thinking about the Web, on the other hand, we usually consider *sites* as input. Each site has a starting point – a home page, and each page has, in addition to the content, an HTML description of the title, as well as links to other pages (each of them with a link description). There are other constructs in HTML that can be of interest – paragraph tags, tables, frames etc. A site can easily contain thousands of pages, but not

all pages are of interest to the extraction task. In fact, most pages are usually not relevant for a domain-specific extraction. As a result, every IE from a Web site involves *crawling*, or acquiring a sequence of pages, and visiting numerous pages. Numerous documents in the input introduce an additional element to a Web-based IE system – the need for a *page-selection heuristic*. In the case of processing a web site with potentially numerous pages, applying selection to the pages can considerably decrease the number of processed pages.

A BRIEF LOOK AT WEB TEXT CONTENT

Although the Web is very well defined on the level of communication protocols and content representation languages, the content itself is characterized by a variety of writing and presentation styles. Here are some characteristics of Web sites as sources of information:

- Information is physically distributed over multiple domains and documents.
- There are numerous documents both per site and site's subsets.
- Similar/related documents and sites lack standardized structure.
- Syntactically incorrect and difficult to understand texts are frequently present.

If we look at domain-specific texts from an IE perspective, one of the first notable attributes is the domain "lingua" or terminology. As one can find in the descriptions of related IE systems, the design relies on the text structure and style. While the same can't be said for all of Web content, the characteristics listed above do hold for a large portion.

Finally, from the perspective of IE from university web pages, here are some initial observations that illustrate the challenges and potential problems:

- The information is contained in a variety of data types:
  text, number, date, name, phone, address, e-mail, URL.
- The information can be found in words, lists, sentence phrases, parts of document, tables, with multiple versions of same/similar facts.
- Information is located at department/school/university pages.
- The desired (required) search depth is not known in advance.

Due to the lack of pre-processed training sets, and more importantly, due to the presence of unstructured input, we found it necessary to use a rule-based IE system. In addition, the characterization of NLP approaches in Table 1 indicates that our also domain belongs in the Rationalist, or rule-based category.

THESIS OVERVIEW

UniversityIE is an IE system built to achieve several objectives:

- Use the domain of Web pages of universities as a test field.
- Explore NLP techniques to extract information from Web university sites.
- Incorporate methods that can process the content and extract relevant facts from the texts regarding admission and general information and:
    - use an existing Web crawler, SCOUT [6], to obtain Web pages.
    - provide input for DEXTER [7] – a system for presenting data contained in database records of known structure.

We present the system architecture and describe why and how are the external components incorporated. This thesis describes the domain with its specifics and introduces a rule-based approach to perform IE. We describe the performance of the current system implementation. In the end, we point out future directions and possible improvements. We also present several IE systems related to UniversityIE.

UniversityIE performs a specific task – IE from university Web pages. However, we did every system design step with consideration to application to other Web domains. We believe that we can easily generalize UniversityIE and apply it to other IE systems. The two major contributions of this project to IE, specifically from natural-language texts on the Web are: (1) extended crawling, i.e. *navigating intelligently* through Web sites, and (2) *extracting information* from extremely unstructured Web content.

If the Web-specific parts of UniversityIE are removed, the remaining part is still a full-fledged IE system that fits in the scheme given in Figure 2. However, this “stripped-down” system will be easily outperformed by most of the dedicated IE systems described in Chapter Six.

CHAPTER TWO: EXTERNAL COMPONENTS

In the process of design and implementation of UniversityIE, we integrated several existing components widely used in the NLP community. One of the objectives of this work was to build the system on the top of SCOUT. We also used three other systems – WordNet, Link Grammar Parser and DEXTER as UniversityIE components. This chapter will give a brief overview of these four systems and their integration.

WORDNET

Like any other IE system, UniversityIE needs an ontology. Having in mind the breadth of the domain, we looked at general knowledge for ontology rather than domain-specific models. The intended application of the ontology in UniversityIE is a component that can be "consulted" for semantic information. Our choice for this task is WordNet - a lexical database that has all the attributes of such a system. In fact, we only use a portion of the functionality of WordNet. The main usage of WordNet in UniversityIE is querying for overview of word meanings and synonyms.

WordNet source code is available in C. We ported the source code entirely to Java and provided API to access WordNet functions from UniversityIE. For details about WordNet, see [8 – 12].

WordNet Overview

Once we have databases that contain dictionaries and we can search through them, it is natural to introduce some links between terms and take advantage of computer-based data retrieval. WordNet is based on psycholinguistics[1] and is an attempt to encode a database that would mimic the human mental lexicon of English words. WordNet presently contains approximately 95,600 different word forms (51,500 simple words and 44,100 collocations) organized into 70,100 word meanings, or sets of synonyms. WordNet divides the lexicon into five categories: nouns, verbs, adjectives, adverbs, and function

[1] Psycholinguistics is an interdisciplinary field of research concerned with the cognitive bases of linguistic competence. In other words, psycholinguistics models human lexical memory.

words. WordNet attempts to organize lexical information in terms of word meanings, rather than word forms. In that respect, WordNet resembles a thesaurus more than a dictionary.

WordNet is best represented with a Lexical Matrix such as the one in Figure 3. In this model, presence of entry $E_{i,j}$ symbolizes that word form F_j can be used to express the meaning M_i. Also, if there are two entries in a row, then the word forms are synonymous, if there are two entries in a column, the word forms are polysemous. Several relations are used to build WordNet. Some of these morphological relations are synonymy (*board* and *plank*), antonymy (*rich* and *poor*), hyponymy (*tree* is a hyponym of *plant*), meronymy (*has a* relationship).

Word Meanings	Word Forms			
	F_1	F_2	…	F_n
M_1	$E_{1,1}$	$E_{1,2}$		
M_2		$E_{2,1}$		
.			…	
M_m				$E_{m,n}$

Figure 3: Lexical Matrix

WordNet Implementation

WordNet's source files are a product of a detailed relational analysis of lexical semantics. A variety of lexical and semantic relations are used to represent the organization of the lexical knowledge. Two kinds of building blocks, word forms and word meanings, are distinguished in the source files. Word forms are represented in their familiar orthography; word meanings are represented by synonym sets - lists of synonymous word forms that are interchangeable in some syntax. There are two kinds of relations: lexical and semantic. Lexical relations hold between word forms; semantic relations hold between word meanings.

WordNet organizes nouns, verbs, adjectives and adverbs into synonym sets *(synset*s), which are further arranged into a set of lexicographers' source files by syntactic category and other organizational criteria. Adverbs are maintained in one file, while nouns and verbs are grouped according to semantic fields. Adjectives are divided between two files: one for descriptive adjectives and one for relational adjectives.

Parts of speech are described in separate files with lists of synsets. The synsets consist of synonymous word forms, relational pointers, and other information. Descriptive adjectives are organized into clusters that represent the range of values of some attribute. Each adjective cluster has two or three parts. Each part initially contains an antonymous pair of word forms called head synset. Most head synsets are followed by one or more satellite synsets, each representing a concept that is similar in meaning to the concept represented by the head synset. One way to think of the cluster organization is to visualize a wheel, with each head synset as a hub and its satellite synsets as the spokes. Two or more wheels are logically connected via antonymy, which can be thought of as an axis between the wheels.

The Grinder is a utility used to compile the lexicographers' files. It verifies the syntax of the files, resolves the relational pointers, then generates the WordNet database that is used with the retrieval software and other research tools.

Word Form Representation

In WordNet, a word form is represented as the orthographic representation of an individual word or a string of individual words joined with underscore characters. A string of words so joined is referred to as a collocation and represents a single concept, such as *fountain_pen*.

In the lexicographers' files a word form may be augmented with additional information, necessary for the correct processing and interpretation of the data. An integer sense number is added for sense disambiguation if the same word form appears more than once in a lexicographer file. A syntactic marker, enclosed in parentheses, is added to any adjective word form whose use is limited to a specific syntactic position in relation to the noun that it modifies. Each word form in WordNet is known by its orthographic representation, syntactic category, semantic field, and sense number. Together, these data make a "key" that uniquely identifies each word form in the database.

Relational Pointers

Relational pointers represent the relations between the word forms in a synset and other synsets and are either lexical or semantic. Lexical relations exist between relational

adjectives and the nouns that they relate to, and between adverbs and the adjectives from which they are derived. The semantic relation between adjectives and the nouns for which they express values are encoded as attributes. The semantic relation between noun attributes and the adjectives expressing their values are also encoded. At the present stage of implementation, these are the only pointers that cross from one syntactic category to another.

Many pointers are reflexive, meaning that if a synset contains a pointer to another synset, the other synset should contain a corresponding reflexive pointer back to the original synset. The Grinder automatically generates the relations for missing reflexive pointers.

Interface to WordNet

Interfaces enable end users to retrieve the lexical data and display it via a window-based tool or the command line. When considering the role of the interface, it is important to recognize the difference between a printed dictionary and the lexical database. WordNet's interface software creates its responses to a user's requests on the fly. We ported the console interface and added API for direct access from Java code. A sample output is shown in Figure 4.

```
> wn tree -over

Overview of noun tree

The noun tree has 2 senses (first 1 from tagged texts)

1. tree -- (a tall perennial woody plant having a main trunk and branches
forming a distinct elevated crown; includes both gymnosperms and angiosperms)
2. tree, tree diagram -- (a figure that branches from a single root;
"genealogical tree")

Overview of verb tree

The verb tree has 1 sense (no senses from tagged texts)

1. tree -- (chase a bear up a tree with dogs and kill it)
```

Figure 4: Sample output from the WordNet console interface

Modifications of WordNet

Porting WordNet was more than mere translation from C in Java. The resulting Java implementation is object-oriented and has modifications and adaptations in the data

structures and resource handling (memory and garbage collection). Since WordNet's interface is generic and command-line/GUI based, we added an additional method that has the role of an interface. WordNet's API consists of one method that returns all queried word meanings.

LINK GRAMMAR PARSER

UniversityIE needs a syntax parser. In the case of IE, the need for full-scale syntax analysis is not big, but if we take into account the domain and the characteristics of the text, having a powerful parser is useful. In addition, the parser implementation allows control over the parsing process in terms of resources (time and space). In terms of space, we work under the assumption that memory is not constrained, but time was critical, especially with our domain. Whenever the parser encounters grammatically "ill behaved" (e.g. very long) or incorrect sentences, the parsing process will try to map the sentence to a syntactically correct structure. While doing that, the parser will go over a large space of possibilities and their combinations. If this happens frequently on numerous documents, the performance of UniversityIE can significantly drop down. The Link Grammar Parser has a mechanism that allows the user to set time and space boundaries which, once crossed, puts the parser into a panic mode when a "light" variant of parsing is executed and the best result is delivered in a short time. For details on panic mode, the parser source code and more Link Grammar documentation, consult [13].

We ported the Link Grammar Parser code from C to Java, added some extra custom code and wrapped the parser in a Java server. We used IP sockets for communication and data exchange with the server.

Link Grammar Overview

The underlying concept in Link Grammar is to think of words as blocks with connectors coming out. There are different types of connectors that point to the right or to the left. A left-pointing connector connects with a right-pointing connector of the same type on another word. The two connectors together form a "link" – hence the name Link Grammar.

Right-pointing connectors are labeled "+", left-pointing connectors are labeled "-". Words have rules about how their connectors can be connected, that is, rules about what would constitute a valid use of that word. A valid sentence is one in which all the words present are used in a way that is valid according to their rules, and satisfies certain global rules.

Of course, the links for a word are likely to be complicated. The words we use in our everyday language can have different roles and meanings depending on the context. Without going into the details of Link Grammar, the examples in Figure 5 gives an idea how link requirements are encoded and different link types are combined.

```
word: A+;
word: A+ & B+;
word: A+ or B-;
word: A+ or (B- & C+);
word: (A+ or B-) & ((C- & A+ & (D- or E-)) or F+);
word: A+ & {B+};
word: (A+ or B+) & {C- & (D+ or E-)};
word: (A+ or B+) & {C- & (D+ or E-)} & {@F+};
```

Figure 5: Example Link Grammar linking requirements

All connections must obey two rules: links cannot cross (planarity rule) and all words in a sentence must directly or indirectly be connected with each other (connectivity rule). Figure 6 shows an example output of the parser when presented with the sentence “The brown dog has gone”.

```
> The brown dog has gone.

+-----Ds-----+
|     +---A--+-Ss-+-PP-+
|     |      |    |    |
the brown.a dog.n has gone
```

Figure 6: The parsing result from the input sentence “The brown dog has gone”

Link names are not always related to the meaning. Still, many times they are. For example: ‘S’ in ‘Ss’ stands for subject ‘PP’ stands for past-participle, ‘D’ in ‘Ds’ stands for determiner.

Parts of speech, syntactic functions, and constituents may be recovered from the link structure. For example, whatever word is on the left end of an "S" link is the subject of a clause (or the head word of the subject phrase); whatever is on the right end is the finite verb; whatever is on the left-end of a D link is a determiner; etc. Moreover, all nouns,

verbs, and adjectives in the dictionary are subscripted (as “.n”, “.v”, or “.a”). Word subscripts are used to distinguish different entries of the same word in the dictionary. The main word subscripts used are ".n" for nouns, ".v" for verbs, and ".a" for adjectives. In these cases the syntactic category of the word is made explicit. For example, brown is an adjective and dog is a noun as presented in Figure 6.

One of the features of the parser used in UniversityIE is identifying constituents from linkages. Link Grammar provides precise rules that enable us to extract verb, prepositional, noun and adjective phrases as well as clauses.

The Parser

There are many implementation details that help deal with different constructs and multiple parse trees – organizing the words in the dictionary, the subscripts mentioned earlier, dealing with capitalized words, hyphenated expressions, number expressions, unknown words, punctuation, and idioms. The parser also effectively deals with conjunctions.

Additional parameters add robustness and speed up the parsing. The ability to avoid long parsing was particularly useful for the unusually long sentences we frequently encountered in the input.

Modifications of Ling Grammar Parser

This port was far more complex than WordNet – primarily due to the size of the system, but also due to the complexity of the processing in the parser. Unlike WordNet, the Link Grammar Parser does a lot of input processing and produces large structures of intermediate results. The parsing also uses a dictionary, which, although not as big as WordNet in number of terms, has entries divided into many more categories that have complex relations between themselves.

The resulting Java code is object-oriented. We replaced the memory management model in the original code with a Java version that has memory management additions and garbage collection-enforcing triggers. Finally, we wrapped the Java as a server that listens to parsing and other requests related to the parsed sentence. We added a proxy in UniversityIE to handle the communication with the server.

SCOUT

SCOUT is a Web crawler that provides a framework where “rule” Java code is plugged in and shapes the behavior of the application. The implementation in Java and the design of SCOUT rules influenced many design decisions during the work on UniversityIE. More information about SCOUT can be found in [6].

Scout Overview

SCOUT is a multithreaded Java application built after the reader-writer model. The main thread – *Scout* (the writer), removes URLs from a search queue and requests the associated documents from the Web servers on which they reside using the HyperText Transfer Protocol (HTTP). Successfully collected documents and their HTTP headers are stored in a shared buffer where rules executing as concurrent threads may access them. Each rule is required to access (and release) the document once only to maintain synchronization. *Scout* and the rule threads also synchronize on the URL queue so *Scout* can differentiate an empty queue from one that is waiting on a rule to produce a URL.

The *Scout* thread avoids collecting redundant documents that could lead to cycling and caches documents to minimize network traffic. It implements the Robots Exclusion Protocol and can be configured to stall for a specified interval between successive accesses to the same server to reduce remote server load.

Since HTML is the most common format of Web pages, Scout attempts to parse each document as HTML before buffering it, and stores the tags and text separately if the parse is successful. This preprocessing step permits rules to specialize in markup or text processing. As necessary, tags may be mapped back into their positions in the text or tags and text recombined into a normalized HTML document.

Throughout a SCOUT session, both the *Scout* thread and the rules write detailed activity records to a log file. The last few lines of the log can optionally be monitored in a graphical window.

Rules

Rules are implemented by extending the base *Rule* class that provides standard interactions with *Scout*, including thread synchronization and result storage. To perform useful work, the rules must override an entry point for document processing. This method is called once by the *Rule* parent class for each document buffered by the *Scout* thread.

When a rule attempts to process a document, one the following is true:

1. A previously unseen document is buffered.
2. The document is assigned a unique identifier that all threads will consistently use to refer to the buffered document.
3. The results from the document processed (if any) are accumulated.
4. All results previously generated by step 3 are kept in a shared repository of results.

Modifications of SCOUT

We modified SCOUT in order to use the system as a framework for UniversityIE and, in general, to be able to support all stages from the generic IE system given in Figure 2. Most of the changes are related to passing some information in different parts of the code. The changes and/or additions are the following:

- Logging: added debug levels and minor modifications;
- HTML pages acquisition and caching: storing of extra information for each document (heuristic flags, title and description of the referring link);
- URL queue: change from FIFO to a sorted list;
- Heuristic: Assigning value to each Web page based on the link description, document title and path;
- Initialization and activation of external components: WordNet and Link Grammar;
- Rules: incorporated heuristic, added department awareness[2], extended rule dependency mechanism to more than one level

[2] department awareness will be discussed later in the text

- Debug modes: Collect (only fill the cache – no processing) and Finalize (only merge the results);
- Wrap-up: added additional activities (result merging) after rules are finished;
- Force garbage collection;

DEXTER

If the data is structured and stored in a Database, it is logical to use standard reporting tools to present views of the data. Still, those presentations are static (in terms of formatting) and are usually not data-aware to a deeper level. The goal of Data EXpression Through Edited Registers (DEXTER) is to develop general tools to display data requested by the user subject to two broad goals: express the facts in a manner tailored and familiar to the user, and vary the expression to suit the data. DEXTER is implemented in Java and provides a set of tools that handle register building.

For example, a database of medical records may be used by several different groups of people. Physicians and nurses probably need different views on the data, which follows directly from the nature of work and relationship with the patients. But even for the same group of users, or the same user, it can be helpful if the presentation of data varies in special cases such as value boundaries or absence of the data. A combination of the two points, in short, should provide intelligent presentations similar to a human-created equivalent.

The basic concept of DEXTER is that of a register. The term is borrowed from descriptive linguistics. A register consists of field rules, mode rules and tenor rules. Field rules are responsible for data transformation. Mode rules describe the structure and specific content of the target presentation. Finally, tenor processing constructs the target document taking into account linguistic correctness and variability.

For more information on DEXTER, please refer to [7].

CHAPTER THREE: UNIVERSITYIE DESCRIPTION

This chapter describes the approach and the considerations at each IE stage from Figure 2 with examples that illustrate the IE approach in UniversityIE. We then present the architecture of UniversityIE and discuss the implementation and necessary modifications and enhancements of SCOUT.

IE APPROACH

Document Acquisition

We assume that the UniversityIE works over an entire Web site. The input processing works over hyperlinked texts – the text in a document contains links to other documents. Each document is assumed to contain at least the body text, but may or may not have a title and description of the link that led to it.

We use the quantified heuristic for each document to place it in the document queue. In other words, we use it as the index of the URL queue. Using a heuristic to determine the direction of document acquisition is one of the contributions of UniversityIE. As discussed earlier, IE systems are built under the assumption that all input documents are of interest. We have two levels of input documents filtering in UniversityIE. The first one is reducing the number of documents. Acquiring all documents that can be reached from a single URL brings in thousands of pages in a short time. There are several independent IE tasks running in parallel at UniversityIE. A drawback of having multiple IE tasks is, if each task process every document input in the system, there will be cases when a task will be looking at a document it is not "interested" in. In other words, we introduce unnecessary resource-consuming computations in the middle of the IE chain. Therefore, the second level of filtering is selective rule application where each rule is applied only over documents that were acquired due to the rule's heuristic settings.

The acquisition method extracts links from the visited pages and stores them in a sorted list. The criterion for sorting the list is the quantification of the heuristic flags of the page. Once acquired, the document is ready to be processed. We access the documents as plain texts – there are no remainders of the original (HTML) page. Although we could use the

HTML tags in the IE process, at this point (and as in the initial design of SCOUT), we leave this task for a future functional extension of the system.

In order to group the acquired pages, we have an additional process at acquisition time that tracks whether the page is under the *department* subset of pages. We chose *department* as the name for the subset because of the intended application of UniversityIE. We define *department* as a subset of pages that can be detected primarily by the URL paths and document titles.

The tasks are not related, therefore, each IE task has a separate *heuristic* configuration. Once we acquire the document, we check it against the IE task heuristic requirements. The heuristic vector of each document consists of six flags given in Table 2. Each flag, when set, means that the condition it stands for is fulfilled. The order of the items is increasing with respect to significance. The quantification of the heuristic is realized through a sum of weighted flags. The weights are also given in Table 2.

Flag Symbol	**Flag Weight**	**Flag Description**
IE_NOHEURISTIC	0	No heuristic match
IE_FORCEDHEURISTIC	0	Forced acquisition
IE_PATHHEURISTIC	1	URL path
IE_DESCHEURISTIC	2	Description of the link to the document
IE_TITLEHEURISTIC	4	Document title
IE_DEPARTMENTHEURISTIC	8	Under the department pages

Table 2: Heuristics flags descriptions and weights

Tokenization & Tagging

We demonstrated in the domain analysis that texts in web pages are highly unstructured and tend to present information with a lack of structure or style. Hence, it is very important to structure the content before the system performs further tasks. This phase performs three tasks: type tagging, normalization and pre-parsing of the text.

Type tagging is the first step. It does not modify the text, but rather introduces the type by attaching a tag to each word. The types are defined using regular expressions. At present time, the following types are used:

- Zip Code

- Date
- e-mail address
- URL
- Integer
- Currency
- Float
- Middle Initial
- String

There are two major benefits of type tagging. The first one is that we use assigned types in the Normalization step and in IE task descriptions. The other benefit is that it helps accurately divide the documents into sentences. The significance of type tagging is obvious once we think just how many periods are there in Internet domain names or e-mail addresses. For example, if this process recognizes “www.cs.uky.edu” as one word and tags with type “URL” – any consecutive processing will not consider “.” from this word as possible sentence separators.

Normalization replaces a group of words with another, "normalized" group of words. We use normalization to standardize key portions of the text throughout the IE process. The list of normalized phrases is domain-specific and depends on the IE tasks. Some examples are to standardize date format to MM/DD, to replace expanded phrases with their abbreviations, to replace numbers in words with digits. Examples of normalization are replacing ‘Test Of English As a Foreign Language’ with ‘TOEFL’, ’10 percent’ with ‘10%’, and ‘January 15’ with ‘01/15’.

During the *Pre-Parsing* phase, the system tokenizes the words, activates type tagging and normalization and finally produces a structured version of the text – *sentences* with *words* tagged with type.

Sentence Analysis

This phase of the IE chain is optional. Only some of the methods need deeper syntax analysis, while other methods use only the structured content generated in the previous

step. The tasks that need to parse a sentence do so just before executing the next (Extraction) phase. The implementation of this stage is slightly different because it is not designed as a part of the IE chain, but is rather a utility component we use when necessary. Nevertheless, the concepts remain unchanged while the performance is significantly improved because we execute resource-demanding parsing over smaller portions of the text.

For the tasks that actually use syntax structure analysis, we decided to use a full-scale parser. Link Grammar Parser generates sufficiently good output and provides methods to identify basic constructs, which IE tasks use in the Extraction stage. In fact, if another parser could provide the same information, it would not be hard to integrate it in the system by simply providing the proper interface to its functions.

Extraction

The key stage of the IE process is the Extraction stage. The previous and the following stages are important, but the heart of the solution is here. IE methods are chosen according to the domain characteristics. Due to the variety of information to be extracted, one method may not be suitable for all of the cases. In this work, we evaluated a number of methods implemented as IE tasks. The initial set of IE tasks considered application data (application deadlines, tuition, international student requirements - TOEFL, financial aid) and general university information (president's name, campus location, enrollment data, demographics, contact info).

Results can be of different types: string, numeric value(s), or a table of values. We use three different methods for information extraction. IE tasks implement each of the methods and are defined with a separate set of parameters for each task. The three different IE methods are:

Positional: The extracted value is easy to identify from its type and range and is typically located in the neighborhood of a keyword. The task is described through the position of the keyword and the value. Here are two examples:

- Find the name of the university president
  keyword: String [] [president]
  value: Name [] []
  position: vkv
- Find TOEFL score minimal requirement
  keyword: String [] [TOEFL]
  value: Integer [400..660] []
  position: vksv

Both *keyword* and *value* are described with a type+range+value triplet. *Position* describes where the position of the value word 'v' is in relation to the keyword 'k'. The keyword can be in front, after the value or both. The 's' is a sentence separator annotating that the search can be extended over to the end of the adjacent sentence. These examples demonstrate only the concept. In practice, we added several attributes to improve performance: *maximum distance* - between the keyword and the value, *other keywords* – additional trigger words positioned between the value and the keyword, and *concatenate* – a flag that makes the task return not just the value, but the entire string between the value and the keyword.

Tabular: Extracts a set of values usually in the neighborhood of several keywords. The output structure of this method resembles a table, hence the name tabular. An example for this kind of information is tuition (full-time, part-time, resident, non-resident) or deadlines (spring, fall, summer semester). Those are illustrated in the following examples:

- Tuition:
  keywords: tuition
  columns: non_resident\out_of_state, resident\in_state
  rows: full_time, part_time, undergraduate, graduate, ----
  type1: Integer [50..]
  type2: Float [50..]
- Application deadlines:
  keywords: deadline
  columns: ∅
  rows: fall, spring, summer, ----
  type1: Date []
  type2: ∅

The result of this method is a table with columns and rows defined by the keywords in *columns* and *rows*. Keywords trigger the IE task, while *type1* and *type 2* specify the values in the table. For example, tuition can be of two types: integer or (float) decimal. As with the positional method, we added one more attribute *maximum distance* with the

same role as earlier – it measures the maximum allowed number of words between the first found keyword and the last found value.

An additional characteristic of this method is that it also attempts to extract the transposed table. Therefore, this method does two extractions – one with the values for rows and columns given in the definition and another with swapped values.

Figure 7 contains a document fragment that the method is likely to be applied upon.

Tuition and Fees for 1999-2000

Tuition Schedule

Per Semester	**Resident**	**Non-resident**
Full-Time Student	$1,798	$5,058
Part-Time Student	$ 188	$ 550
Summer Terms	**Resident**	**Non-resident**
Full-Time Student	$ 946	$2,756
Part-Time Student	$ 188	$ 550

Figure 7: An example text fragment for the Tabular method

Syntax: Extracts parts of sentences using their syntax structure. The description of the IE task is a query syntax tree –a sentence template. The first criterion for the match is to map the query tree with the tree generated by the parser. The next step is to map keywords with the words in the phrase. The following example demonstrates query descriptions:

- University location:

```
phrase: Noun Phrase
links: G (obligatory)
keywords: university.n
fixed: false
result: No
immediate: false

phrase: Verb Phrase
links: ∅
keywords: locate.v, lie.v, occupy.v
fixed: true
result: No
immediate: true

subphrase: Prepositional Phrase
links: ND (optional), Yd (optional)
keywords: null
fixed: false
result: Yes
immediate: false
```

The *phrase* attribute describes the expected phrase type. The *subphrase* section describes a subphrase – phrase contained within another phrase. Link Grammar links specified under the *links* attribute help narrow down the search and obtain reliable results. These links can be obligatory or optional. In the case of optional links, the presence of the link adds more weight to the result. A phrase is also characterized with keywords under the *keywords* attribute, suffixed by '.v' for a verb or '.n' for a noun. The *result* flag marks the phrase that is the actual result of the query. In the case when the phrases need to be adjacent, the *immediate* flag is set to true.

To clarify the links entries from the example, 'G' link stands for proper noun words connected together. This covers phrases such as 'University of Kentucky', 'Northern Kentucky University'. The link 'ND' connects numbers with expressions that will help locate phrases like '…three miles from', while the 'Yd' link is used in distance expressions like '… miles from'.

This rule extracts location from sentences like 'Eastern Kentucky University lies on the south edge of Richmond, a community of 25,000…' and 'Northern University is situated on 325 acres of rolling countryside along U.S. Highway 27, seven miles southeast of Cincinnati, Ohio.'

To improve the method performance, we use WordNet to refine the matching of the words from the document with the keywords. WordNet expands the word to a list of synonyms and tries to match them with the word from the document. For example, the verb 'situated' in 'The XXX University Main Campus is situated in …' would not be matched, but since 'locate' is a synonym of 'situate', it will be identified as a match.

Merging

Each IE task may extract multiple results from different documents or even from a single document. The output of each IE task is one data item at most. Each IE task (IE method instance) applies different post-processing methods to the results. From an implementation point of view, everything discussed in this section involved changing and, in most cases, adding functionality to SCOUT.

To merge all results from one IE task, the task performs several actions after processing all documents. As for the Extraction stage, we will discuss each method separately. Each method's merge routine returns a list of results. This simplifies the next stage because Merging stores all results in the same format, regardless of the IE method.

Positional: For this method, we use *minmax* – a flag for whether to use the shortest or the longest result and *department* – an indicator whether the result should be under the department pages. Possible values for the department attribute are YES, NO, MAYBE and DONTCARE. A corresponding match between the value and the page placement provides more weight to the result rating. For example, if the department attribute is MAYBE, a page that is within the department is rated higher than a page that isn't.

Tabular: This method has only the *department* attribute with same usage as before. In addition, this rule orders the results by the count of elements in the result tables.

Syntax: Since this rule always extracts a string, the only ordering criterion is the extracted phrase length – longer phrases are favored simply because it is safer to assume that the longer phrase is more likely to contain useful information.

To summarize, part of the data characterization comes from the data itself. It can be its length, the count of elements in the table or some other feature. The merging stage uses additional inputs to make sure that it chooses the correct instance of the extracted information: repetitions per document, repetition in the entire set of results and the number of documents where the instance was found.

Template Generation

This stage has well-structured data as input and is a filter and formatter of the output. The template used to generate the input for DEXTER, is a set of SQL statements over previously agreed relational tables. We implemented this stage as a plug-in (rule) for SCOUT. Due to the separation from the previous stage, it is very easy to replace the SQL statement generator with a different template generator. In UniversityIE, this stage does not alter the data.

SYSTEM ARCHITECTURE

UniversityIE architecture supports document processing with all stages described in the previous section. Even more, the design approach for the system was to analyze the domain, propose solution for IE and then integrate it with SCOUT. The relationships between the components of UniversityIE are presented in Figure 8.

The components are:

Documents: Any sequence of documents. The only assumption is that the documents contain natural language text and have a tree-like organization. In principle, it is appropriate to think of these as a series of documents systematically acquired from a site.

Input Interface: The documents coming at the input need not be ready for processing. The module deals with the format of Documents and separates the processing from the data format. It also provides reusability and generality of the IE code. This component also serves as a filter of documents - it restricts the input to documents of interest and this way increases system throughput.

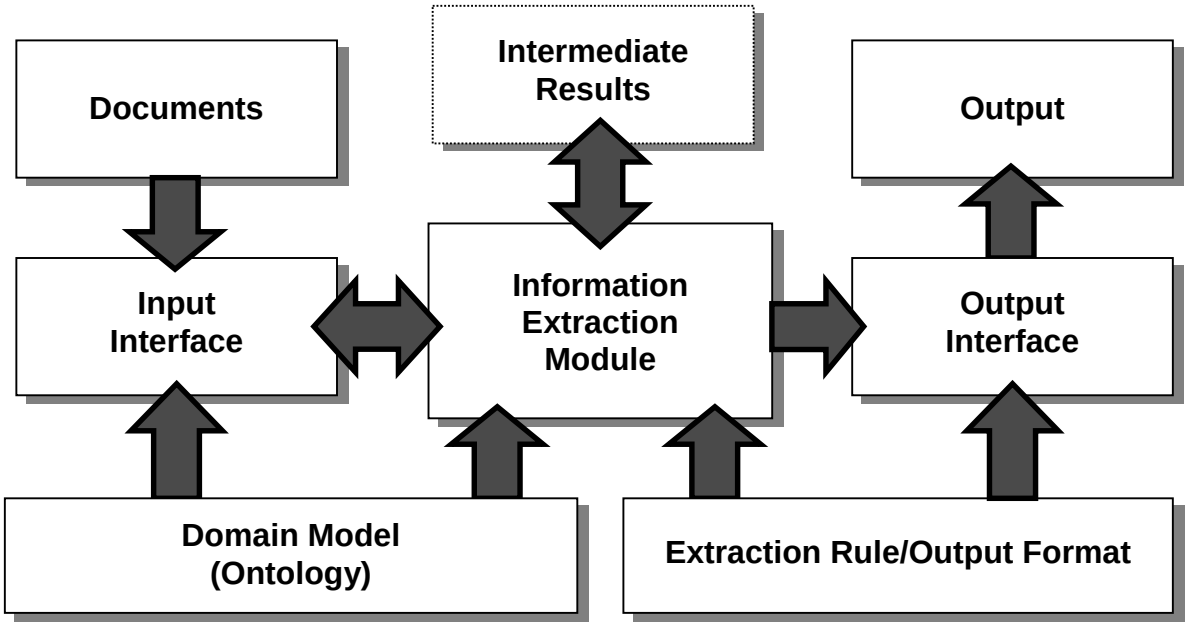

Figure 8: UniversityIE System Architecture

Domain Model: It is possible to extract information by feeding the input text to the IE methods, but this would do only for the simpler tasks. Once the tasks are more complex,

the methods need some knowledge as well. In the case of IE, this is the domain ontology. UniversityIE uses WordNet for this purpose.

Extraction Rule/Output Format: The system implements various IE methods and each task is described with a set of parameters and format of the results. It is possible to have several methods running in parallel over the same input.

Intermediate Results: Most of the data flow, as per the generic IE architecture from Figure 2 is serial. It is still necessary to account for a manager of intermediate results. The main reason for this is the assumption that there are several IE tasks running in parallel all accessing the same repository of results. The content of the repository can be: unverified or uncertain conclusions, partial information on an ongoing search or additional data that is not needed for the final output – but necessary in the intermediate stages.

Output Interface: A module that generates the output based on the information retrieved and the format demanded by the user. As in the case of Input Interface, this module separates the main parts of the program from the output constraints.

Output: Once the IE tasks produce the structured output, this component exports the data in a format suitable for the next user of the extracted information. In UniversityIE, this module produces a collection of SQL statements as input for DEXTER.

Information Extraction Module: The “heart” of the system, through which the appropriate IE tasks are activated upon the input documents. It consists of a text tokenizer and tagger, syntax parser, results merger and most importantly, a collection of IE methods.

IMPLEMENTATION ISSUES

SCOUT

SCOUT is built as a crawler that maintains a queue of URLs to visit and a set of rules that can be plugged in. The information exchange mechanism is a result structure where rules store and retrieve results. Although SCOUT could be configured with rules that execute most of the IE stages, it is not best suited to provide all necessary mechanisms.

The features described here are those that exist in SCOUT and are not related to the UniversityIE implementation.

SCOUT's main class *Scout* removes a URL from the search queue and requests the corresponding document. Once the document is acquired and is available, all rules access it in a synchronized fashion. The rules are the mechanism through which SCOUT's functionality can be arbitrarily extended. This is done through the "plug-in" interface for Java classes that inherit the base class *Rule* each running as a separate thread. SCOUT, among the many tasks, also maintains a structure for storing results, implemented by the *Results* class.

SCOUT uses several third-party packages. These are the HTTP client, Regular Expression Package and *Hashlookup* class. All are described in [6]. The Regular Expressions package is further reused in the UniversityIE type-tagging module.

UniversityIE over SCOUT

To better illustrate the structure of UniversityIE and its relation to SCOUT, this section describes the UniversityIE implementation in two stages. These do not correspond to the actual stages of the implementation, but are rather a conceptual model. Through the design of this model, the reader will get a better idea of the architecture and the logic behind parts of UniversityIE. The first stage introduces the components that could (more or less) be directly incorporated with SCOUT without much adaptation. The second stage presents the rest of the components and the additional functions we add to those from the first stage.

Initially, SCOUT provides the following IE system functions:

- serialized access to enqueued documents;
- synchronized document access of the IE tasks;
- exchange of data between IE tasks;

In this stage, we write several IE tasks, by extending the *Rule* class, and hence extending SCOUT to a system that implements the IE stages of the generic architecture. For this purpose, we plug in several classes into SCOUT. The reader is referred to Figure 9 for a graphical representation of the following discussion. An instance of *IEBFS* class does the

Document Acquisition and a *IEParserRule* class performs the Tokenization and Tagging. The IE tasks are instantiations of *IEKeywordValueRule1*, *IETableValuesRule* or *IEFindPhraseRule* classes. We describe these classes in the remainder of this section.

All classes mentioned above extend the class *Rule*. Several methods were overwritten or implemented in order to have a proper extension of *Rule*. The most important one for us at this time is *processDoc()* which is invoked once for each document.

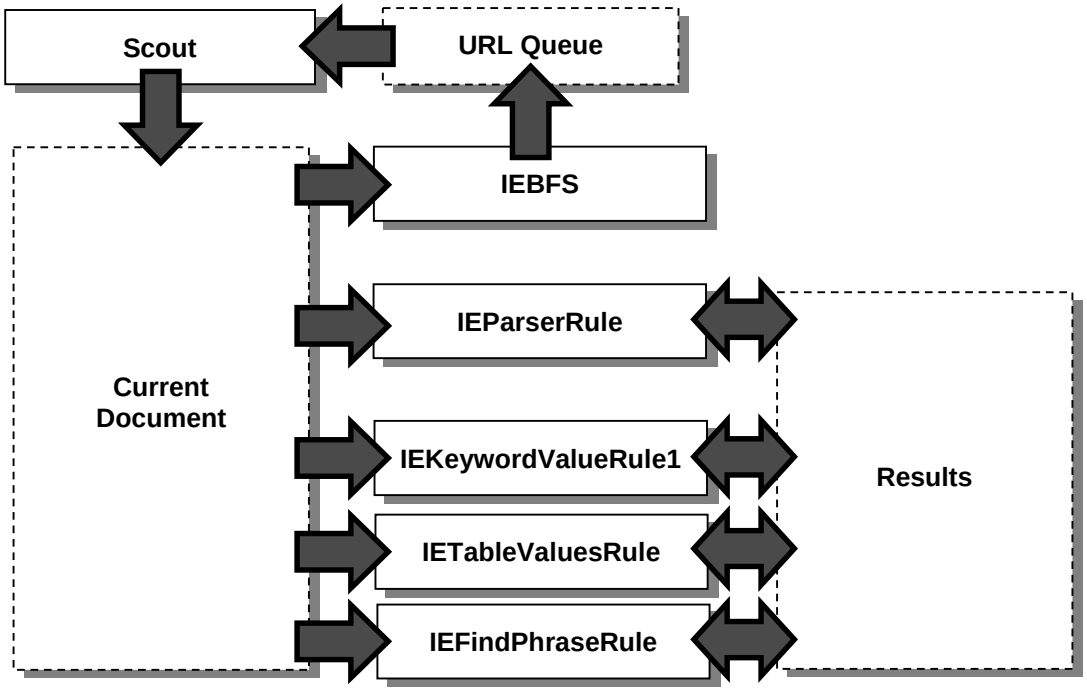

Figure 9: Classes and data flow in UniversityIE

IEBFS – extracts all URLs from the current document and enqueues them for acquisition.

IEParserRule – eliminates content that is not needed (scripts and other extra text), applies type tagging and normalization and returns the document tagged and tokenized in sentences with words.

IEKeywordValueRule1 – implements the Positional IE method.

IETableValuesRule – implements the Tabular IE method.

IEFindPhraseRule – implements the Syntax IE method.

SCOUT result managing mechanism still handles the serialization of the document access. *Results* hold the results for every rule for each document. Each result read from

the *Results* object is identified by two parameters: rule name and document number. If the rule has not inserted a result yet, *Scout* places the requesting rule in a wait state and resumes it when the results are ready. If all rules use standardized result format, this solution will implement all IE stages up to the Merge stage.

The Results access is not immune to deadlocks. In SCOUT, due to the serialization of the requests for results, a deadlock may occur if a rule crashes or simply never finishes processing particular document. This is not addressed in this implementation, but certainly is a consideration in the rule implementation.

One additional remark on the Sentence Analysis Stage. The implementation includes an interface to Link Grammar Parser, which we don't use as a separate stage (rule). In our implementation, it is the *IEFindPhraseRule* class that executes the Sentence Analysis stage just before applying the Extraction stage. We actually wrote a rule that implements it as a stage, which was used during development. The final implementation executes the parser API directly from the extraction rule.

Enhancing SCOUT

Two functions have not been discussed so far. It is the selective document acquisition (heuristic) and post-processing (merge) of the results. These are the functions in the second stage of our conceptual model.

Although we incorporated the above heuristic mainly by adding code to *IEBFS*, there are also a few additions to the *Scout* and *Rule* classes. This way, together with the selective document acquisition and ordering of the URL queue, we implement selective rule application. For example, let us suppose that the Campus Location rule is "interested" in pages that have 'prospective students' and 'campus guide' in the links or the title. The President name rule can have 'prospective students' and 'president' as pages of interest. In this case, if a page were acquired because it had 'prospective students' in the description/title, it would be processed by both rules. A page containing 'president' will only be processed by the President rule.

After the URL queue is exhausted and all rules have finished processing the acquired documents, the system executes post-processing activities. Having the initial version of

SCOUT, it would still be possible to build a rule that can do the merging stage. We could use the results to pass various flags and that way activate rules at certain times, but it would be very hard to implement specific behavior for each instantiation of the IE rules. In other words, an IE system built using only the mechanisms provided by SCOUT would be complicated to configure and difficult to maintain. Because of that, we modified some parts of SCOUT to provide a framework more suitable for the Merge and Template Generation stages of UniversityIE. We implement the Merge stage with two augmentations of the existing framework – an addition to *Scout* and added methods to the *Rule* class. These methods are part of the *Rule* class implementation and are parameterized. By overriding the methods, the Merge stage can be adjusted to better fit the IE methods. For the Template Generation stage, the new version of *Scout* has a "hook" where a rule of class type *IEExportResults* can be plugged in. This rule takes the structure generated by the Merge stage and formats it to fit the output specification.

CHAPTER FOUR: EXPLORING THE DOMAIN

We used a set of IE tasks as a guideline in the development of UniversityIE and a set of test sites to evaluate the system. We later introduced one more task and several additional sites to test the system's behavior on new input.

TASKS AND SITES

Here are the IE tasks grouped under the three methods:

Positional:

- PRESIDENT - Extract the name of the President of the University
- TOEFL - Extract the TOEFL requirements (points required)
- GRSTUDENTS - Extract the number of graduate students at the University
- UGRSTUDENTS - Extract the number of undergraduate students at the University
- ADDRESS - Find the addresses of CS/CIS/EECS/CSE/MCS departments in various Universities and Colleges (CS/CIS/EECS/CSE/MCS: Computer Science, Computer and Information Science(s), Electrical Engineering and Computer Science, Computer Science and Engineering, Mathematics and Computer Science).
  This is the task that was introduced after UniversityIE was implemented.

Tabular:

- TUITION - Extract tuition cost
- DEADLINES - Extract application deadlines

Syntax:

- LOCATION - Extract sentence phrase containing the location of the University
- CAMPUSES - Extract sentence phrase containing the location of the campuses

We describe the domain and the experiences with the content with examples from the test site set that incorporated sites of various size and structure complexity. During the development of UniversityIE, we used the following sites to analyze the domain and do the testing:

- Eastern Kentucky University (www.eku.edu)
- Morehead State University (www.morehead.edu)
- Northern Kentucky University (www.nku.edu)
- University of Kentucky (www.uky.edu)
- University of Louisville (www.louisville.edu)
- Western Kentucky University (www.wku.edu)

We later added additional sites to evaluate the performance of the tasks on content that was previously not analyzed:

- Ohio University (www.ohiou.edu)
- University of Miami (www.miami.edu)
- South Dakota State University (web.sdstate.edu)
- Smith College (www.smith.edu)
- University of Michigan (www.umich.edu)

INITIAL WEB SITE ANALYSIS

We discuss the case when information exists somewhere on the site from the perspective of page distribution, page structure and forms of information containment.

Page Distribution: The starting URL usually is the site's *home page*. If this is the case, it makes sense to assume that the organization of the home page provides relatively convenient access to the entire site. In other words, we assume that the site design enables quick and convenient access for a human browser. It is reasonable to consider three to four layers deep in the site to be appropriately few. That is, most of pages can be reached by following up to four hyperlinks. This assumption is also part of the heuristic. We acquire the first 2 layers regardless of the pages' attributes and apply the heuristic to all pages acquired later.

Larger sites usually spread over several domain names. A typical university domain name is *www.university_name.edu* (where *university_name* is a string specific to the university). The university may include several college sites (we will call them sub-sites),

possibly with their own domains, named *www.sub_site_name.university_name.edu*. There is a mechanism in SCOUT that can restrict the crawling to a group of domains by following only links with domain name ending with, say, *university_name.edu*.

Most similarities between sites disappear beyond domain naming. Even a small test sample of few sites demonstrated that, within any sub domain, the content and its placement is entirely proprietary to the university or the college. For example, most sites have a page dedicated to the 'Message from the President'. Still, some university sites are organized in such a way that makes it hard to track the President's page. The President's page is clearly present and referred to at the Louisville and Northern Kentucky University sites, while at the University of Kentucky and Morehead University the President information can be found, but at a very non-intuitive place.

Page Structure: The primary interest of the analysis is to locate the pages at different sites that contain similar information. It is also of interest to analyze pages from a same site or sub-site, a college for example. In most of the cases there is no similarity in style even on "neighboring" pages. Some of the pages are short and contain only few facts and many pages usually contain highly unstructured lengthy texts. A reason for this can be that the authors are usually not highly skilled writers and are frequently not aware of the information put on other parts of the site. Human browsers build a false impression of order from sites with unified page background, common banners and links style. Unfortunately, uniform-appearing pages often don't imply order in the actual content.

The domain analysis reinforced the expectations that having several annotated examples is of little use. In fact, our conclusions are in line with [15]. Having in mind how hard it is to generalize from the examples, we designed UniversityIE as a rule-based IE system where we manually define the rules on the grounds of the domain analysis, on the definition of the page-selection heuristic and finally, "common sense" assumptions.

Information Presentation: Once we navigate through the site and acquire the page that contains some information and even locate the portion of the content where the information resides, we still need to extract the information. Depending on the method and the data, these steps can range from very simple to very complicated. The Positional method is an example of simple processing. The data is just a word and once located

there's not much more to do about it. On the other hand, the Tabular method requires more complicated processing. It is not trivial to place every extracted piece of information in the corresponding row and column of the table we are retrieving. In the case of the Syntax method, there are fewer problems because, like the positional method, it deals with one chunk of localized content.

Information containment: After we have located the page and the correct piece of the content, the information may still not be explicitly extractable. Information is sometimes implied from the text. For example, take a table describing enrollment figures for each college where the last line gives the total number of students. To extract the total number of students, we have to build a task that takes into consideration the presentation style and extract the number from the last line. Another example is when the TOEFL requirements are given in the new (computer-based) rather than the older (paper-based) point scale. In this case, the task has to extract which scale is used and even apply the conversion.

EXAMPLES

This section is intended to serve as an illustration of the domain and the logic behind the design approach. We give several extraction examples with the URL, the content out of which information was extracted and a comment that explains the example. We present only few positive extraction examples from "well behaved" pages. We present complete results with false positives and false negatives and actual extracted information in the next chapter.

President

- University of Kentucky
  URL: http://www.uky.edu/PR/News/fordappt.htm
  Content: The announcement was made today by UK President Charles T. Wethington Jr. during a meeting of the Martin School's Advisory Board.
  Comment: This page is part of the news that was currently on the site and was referenced. The page was selected because there is School on the title, which is a keyword for the task. Still, this page was selected by accident. There is no dedicated President page at the UKY Site.

- University of Louisville

  URL: http://www.louisville.edu/president/president.html

  Content: Dr. John W. Shumaker

  Comment: This page is referenced from the home page and has “President” in the URL, link description and document title.

- Northern Kentucky University

  URL: http://www.nku.edu/www/about_nku.html

  Content: Dr. James C. Votruba President of Northern Kentucky University

  Comment: This is an “about” page, but references the president.

- Morehead State University

  URL: http://www.morehead-st.edu/intranet/update/09241999-01.html

  Content: "President Emeritus Doran and his wife are great Kentuckians who have given much to this institution, to Eastern Kentucky and to the entire Commonwealth," said MSU President Ronald G. Eaglin. "We invite their many friends to join us in saluting Dr. Doran on reaching this milestone in his illustrious life."

  Comment: As can be seen from the URL and the content, the president’s name is extracted by accident. This is a similar example as UKY.

TOEFL

- University of Kentucky

  URL: http://www.rgs.uky.edu/gs/intapp/english.html

  Content: The University of Kentucky requires a score of at least 550 on the Test of English as a Foreign Language (TOEFL) or 213 on the computer-based test for all applicants whose native language is not English.

  Comment: This information and the page are located in an intuitively correct place.

- Northern Kentucky University

  URL: http://www.nku.edu/www/catalog/admissions.html

  Content: All foreign-born applicants whose primary spoken language is not English must score at least 550 on the Test of English as a Foreign Language (TOFEL) before being accepted.

  Comment: Same as above

- Eastern Kentucky University

  URL: http://www.admissions.eku.edu/internationaldefault.htm

  Content: T.O.E.F.L. (Test of English as a Foreign Language) score of at least 500 or completion of the IEP (Intensive English Program) of Eastern, EELI (EKU English Language Instruction Program);

  Comment: Same as above

- Western Kentucky University

  URL: http://www.wku.edu/Info/Pubs/Catalogs/grad/97-99/admission.html

  Content: Applicants who are not U.S. citizens must submit … (4) evidence of ability to communicate in English (a minimum of 525 on the TOEFL), and ...

  Comment: Same as above

Deadlines

- University of Kentucky

  URL: http://www.rgs.uky.edu/gs/intapp/apdeadin.html

  Content: International Application Deadlines

  Applications for admission should reach the Graduate School at least six months before the opening of the term in which you intend to begin graduate work.

  - Fall Semester Deadline: February 1
  - Spring Semester Deadline: July 15
  - Summer Sessions Deadline: October 1

  Comment: Both location and content structure are appropriate for extraction.

- Western Kentucky University

  URL: http://www.wku.edu/Info/Admissions/8.html

  Content: APPLICATION DEADLINES

  The Deadline for submitting applications are as follows:

  FALL: August 1

  SPRING: January 1

  SUMMER: May 1

  Comment: Same as UKY

- University of Louisville

  URL: http://athena.louisville.edu/student/services/admissions/gapp/procedures.html

  Content: The following majors have application and credential deadlines:

  Clinical Psychology ..... January 10

  Experimental Psychology ..... January 15

  Expressive Therapy ..... January 15

  Kent School ..... January 10

  Microbiology:

  Priority Deadline ..... December 15

  Final Deadline ..... February 1

  Nursing

  Fall ..... May 1

  Spring ..... October 1

  Summer ..... March 1

  Comment: The task uses the last three lines to extract the deadlines correctly. There is no awareness that these deadlines are not general but are for the Nursing School.

- Ohio University

  URL: http://www.ohiou.edu/nursing/academic/rn2bsn.htm

  Content: Deadlines for application for admission for transfer students are as follows:

  Applications Due

Fall admit	June 1
Winter admit	November 15
Spring admit	March 1
Summer admit	May 1

  Comment: Coincidentally, this is also a Nursing School page. Again, there is no mechanism to verify whether the page contains general admission deadlines.

Location

- University of Kentucky

  URL: http://www.uky.edu/Agriculture/AnimalSciences/Research/gradinfo.html

  Content: The University is located in the city of Lexington, in the central Kentucky

Bluegrass Region.

Comment: Although not at the main information page, the sentence does reveal the correct university location.

- Western Kentucky University

  URL: http://www.wku.edu/Dept/Academic/Education/CFS/

  Content: Western Kentucky University is located in historic Bowling Green, one of the fastest growing communities in Kentucky with a population of approximately 85,000 that features such attractions as the Kentucky Museum, Corvette Museum, Hobson House, and nearby Mammoth Cave National Park. Located on Interstate 65 about 120 miles south of Louisville, Kentucky, and 65 miles north of Nashville, Tennessee, Bowling Green gives you that college town flavor" while still offering conveniences of larger cities in the region.

  Comment: This is a very good example of noisy input. Even a human interpreter would have problems deciding just where exactly is WKU located.

- University of Louisville

  URL: http: http://www.louisville.edu/student/services/admissions/about/

  Content: The University of Louisville is a state-supported metropolitan research university located in Kentucky’s largest urban area.

  Comment: Another example with not much information.

CHAPTER FIVE: TESTING AND RESULTS

THE RESULTS

We present the data in tables with identical layout. Each table presents one IE task. Rows list the universities, where the first seven are the initial, and the remaining four are the ones added to test the concepts. The columns describe:

- if the task identified a positive instance of the information (*Positives*),
- whether an instance exists at all (*Instance Existed*),
- if the task identified a negative instance of the information (*False Positives*), or, in other words, information was extracted, but is from a wrong place in the domain.
- summarization column (*Performance*) for the previous three columns: *OK* – there was information and was extracted or there was no information and no negative instances were generated; *NOK* – there was a false positive or there was an instance, but no extraction results; *PARTIAL* –performance similar to OK, but with portion of the data missing.

TOEFL

	Positives	Instance Existed	False Positives	Performance
Eastern Kentucky University	✓	✓		OK
Morehead State University		✓		NOK
Northern Kentucky University	✓	✓		OK
University of Kentucky	✓	✓		OK
University of Louisville	✓	✓		OK
Western Kentucky University	✓	✓		OK
Ohio University				OK
University of Miami	✓	✓		OK
South Dakota State University	✓	✓		OK
Smith College				OK
University of Michigan	✓	✓		OK

Table 3: TOEFL task results

PRESIDENT

	Positives	Instance Existed	False Positives	Performance
Eastern Kentucky University		✓	✓	NOK[3]
Morehead State University	✓	✓		OK
Northern Kentucky University	✓	✓		OK
University of Kentucky	✓	✓		OK
University of Louisville				OK
Western Kentucky University	✓	✓		OK
Ohio University		✓	✓	NOK
University of Miami	✓	✓		OK
South Dakota State University	✓	✓		OK
Smith College	✓	✓		OK
University of Michigan	✓	✓		OK

Table 4: PRESIDENT task results

GRSTUDENTS

	Positives	Instance Existed	False Positives	Performance
Eastern Kentucky University	✓	✓		OK [4]
Morehead State University				OK
Northern Kentucky University		✓	✓	NOK
University of Kentucky		✓	✓	NOK
University of Louisville	✓	✓		OK [4]
Western Kentucky University				OK
Ohio University	✓	✓		OK [4]
University of Miami		✓		NOK
South Dakota State University				OK
Smith College		✓		NOK
University of Michigan			✓	NOK

Table 5: GRSTUDENTS task results

[3] Found a former president.

[4] These are extractions where the method worked properly but the page was picked by accident.

UGRSTUDENTS

	Positives	Instance Existed	False Positives	Performance
Eastern Kentucky University		✓		NOK
Morehead State University				OK
Northern Kentucky University		✓		NOK
University of Kentucky				OK
University of Louisville				OK
Western Kentucky University				OK
Ohio University		✓		NOK
University of Miami		✓		NOK
South Dakota State University				OK
Smith College		✓		NOK
University of Michigan				OK

Table 6: UGRSTUDENTS task results

ADDRESS

	Positives	Instance Existed	False Positives	Performance
Eastern Kentucky University		✓	✓	NOK [5]
Morehead State University		✓		NOK[5]
Northern Kentucky University		✓		NOK[6]
University of Kentucky		✓	✓	NOK
University of Louisville				OK
Western Kentucky University	✓	✓		PARTIAL
Ohio University				OK
University of Miami				OK
South Dakota State University		✓		NOK[6]
Smith College		✓	✓	NOK[6]
University of Michigan	✓	✓		OK

Table 7: ADDRESS task results

[5] Heuristic brought up the page, but not appropriate title on the page

[6] Heuristic brought up the page, but the addresses in the text contained no building number

LOCATION

	Positives	Instance Existed	False Positives	Performance
Eastern Kentucky University		✓		NOK
Morehead State University		✓		NOK
Northern Kentucky University		✓	✓	NOK
University of Kentucky	✓	✓		OK
University of Louisville	✓	✓		PARTIAL
Western Kentucky University	✓	✓		OK
Ohio University	✓	✓		PARTIAL
University of Miami	✓	✓		OK
South Dakota State University	✓	✓		PARTIAL
Smith College				OK
University of Michigan	✓	✓		PARTIAL

Table 8: LOCATION task results

CAMPUSES

	Positives	Instance Existed	False Positives	Performance
Eastern Kentucky University				OK
Morehead State University				OK
Northern Kentucky University	✓	✓		PARTIAL
University of Kentucky	✓	✓		PARTIAL
University of Louisville		✓	✓	NOK
Western Kentucky University				OK
Ohio University	✓	✓		PARTIAL
University of Miami	✓	✓		OK
South Dakota State University				OK
Smith College	✓	✓		OK
University of Michigan			✓	NOK

Table 9: CAMPUSES task results

DEADLINES

	Positives	Instance Existed	False Positives	Performance
Eastern Kentucky University			✓	NOK
Morehead State University				OK
Northern Kentucky University		✓	✓	NOK
University of Kentucky	✓	✓		OK
University of Louisville	✓	✓		PARTIAL
Western Kentucky University	✓	✓		OK
Ohio University	✓	✓		OK
University of Miami	✓	✓		OK
South Dakota State University			✓	NOK
Smith College	✓	✓		OK
University of Michigan	✓	✓		OK

Table 10: DEADLINES task results

TUITION

	Positives	Instance Existed	False Positives	Performance
Eastern Kentucky University		✓	✓	NOK
Morehead State University				OK
Northern Kentucky University		✓	✓	NOK [7]
University of Kentucky	✓	✓		OK
University of Louisville		✓	✓	NOK
Western Kentucky University	✓	✓		PARTIAL
Ohio University		✓	✓	NOK
University of Miami		✓	✓	NOK
South Dakota State University	✓	✓		OK
Smith College		✓		NOK
University of Michigan		✓	✓	NOK

Table 11: TUITION task results

[7] The page is too far in the queue for processing. The IE task extracts the information when presented with the page content, but in this case, it gets to the document after a long time. The negative instance is favored because the actual page was not processed.

A CLOSER LOOK AT THE RESULTS

Presenting the results to gain overall idea of the systems' performance is not an easy task. Each method is designed to extract data when presented with a text fragment with the information of interest. If we were to present the methods' performance over such fragments – the performance tables from this chapter would be filled with successful results. UniversityIE is built to work on entire web sites and the following discussion stays focused on that performance.

Characterizing UniversityIE performance numerically is hard. The number of test sites is small and there are also aspects of the system that are hard to quantify. For example, heuristic and merging are quite challenging tasks even for a human. Some people are simply better than other in site-navigation. The amount of domain knowledge is usually proportional to the human's speed and skills. In addition, there always is the factor of general knowledge, experience and visual perception of the web content, which is not incorporated in UniversityIE. The remainder of this chapter to characterizes the results in a less formal way – we believe this is the best way to illustrate the performance of the system.

Positives

UniversityIE successfully extracted prominent data that was easy to describe with the provided mechanisms. Following are the tasks where this was the case (50% or more OK and PARTIAL performance):

- TOEFL, where the keyword (TOEFL) with an integer in a specific range was a good sign of presence of the information.
- PRESIDENT, where the type *Name* is in the keyword's neighborhood (President). In this case, additional help was the frequent existence of a web page dedicated to contain the message of the president to future students.
- LOCATION/CAMPUSES, where sentence constructs and keywords triggered correct information extraction. In this case, it is worth mentioning that the extracted text was not always meaningful or informative due to the text style.

- DEADLINES, where the data type is *Date*, again in the neighborhood of several keywords.

We added the ADDRESS task later for the purpose of testing UniversityIE. We built the task definition and executed it over the set of sites. A notable flaw in the performance was due to the fact that some addresses listed on the web pages did not contain building numbers. The heuristic selected the correct pages, but did not extract the information – the addresses didn't match the definition in the corresponding IE task. If the task has definitions for addresses with and without building numbers, the performance would be positive on all but one site.

False Negatives

Two tasks that didn't perform well are GRSTUDENTS and UGRSTUDENTS. Both tasks are positional. It is interesting to note that most of the best-performing tasks were also positional. The reason these two failed is mainly because of the information characteristics. An integer in the neighborhood of the keywords (*undergraduate students* or *graduate students*) is likely to be found in many places throughout the site. Most such instances will not even have the correct semantic meaning. A human crawler would also have difficulties with a similar task. Some universities have pages with numbers summarized in tables. A tabular rule that would recognize such tables will probably perform much batter simply because the entire set of constraints describes more prominent information.

TUITION task didn't perform as well as we expected from a tabular task. The main reason for this is the highly non-uniform presentation of the tuition information. The heuristic didn't navigate efficiently and the structure of the pages containing the tuition data was, in most cases, misguiding.

ADDRESS task performed well if we ignore the fact that it didn't extract information due to a description of address. This is an example of failure where the information is prominent, but the task is poorly configured. We incorporate the results with the flaw to emphasize the problems designers encounter when adding new tasks or when adopting IE systems.

False Positives

Tabular rules produced several false positives. There clearly is a relation between the data type of the extracted information and the false positives. The rule with most false positives is TUITION. Here, the data is integer or decimal which is frequently present throughout the sites. LOCATION and CAMPUSES extracted phrases from sentences that fit the definition, but were not correct. In the case of PRESIDENT, the negative instances actually extracted a president, just not a president of a university or not the present one.

In conclusion, the reason for false positives is the system design. The Merging stage is built so that if results were extracted, one is always selected. In the case when correct information is not extracted, it is clear that there will be a false positive.

THE ROLE OF THE HEURISTIC

Table 12 contains sequence numbers of the pages out of which UniversityIE extracted the information. The last column contains the total number of pages processed in the session. We configured all extraction sessions so that UniversityIE will terminate when the URL queue is exhausted. This means that in most of the cases, the numbers in Table 12 refer to pages selected from the entire site.

The first noticeable conclusion is the relatively small number of processed pages (in the Total Acquired column). Here, the range of under 3000 pages is considered against thousands of pages in most of the web sites. This is a manifestation of the page filtering effect of the heuristic. The sequence number of each extraction is an illustration of the effectiveness of the heuristic. Most of the entries in the table are smaller the 200 and many even smaller then 100. If we take in consideration that UniversityIE acquires pages for several tasks is parallel, we can definitely draw the conclusion that the heuristic is effective in the URL acquisition and sorting.

The pattern that emerges from Table 12 is that each session retrieves relatively small number of pages (in principle less than 1500) and the actual data was within the first 150 processed pages. Three sites depart from this pattern. Smith College session processed almost 3000 pages but the results are extracted from the first 20 pages. Ohio University

and Northern Kentucky University have similarly many processed pages and even more extractions.

	TOEFL	PRESIDENT	GRSTUDENTS	UGRSTUDENTS	ADDRESS	LOCATION	CAMPUSES	DEADLINES	TUITION	Total Acquired
Eastern Kentucky University	82	60	414		87	73		69	10	**793**
University of Louisville	112		145		112		70	112	116	**1312**
University of Michigan	137	111	176		105	124	47	104	70	**1558**
University of Kentucky	61	308	450		213	291		55	170	**1608**
Smith College		1			0		20	11		**2945**
South Dakota State University	12				11	176		72	148	**1565**
Ohio University		156	184		139	184	184	56	82	**2477**
Northern Kentucky University	25	93	28		3	92	45	44		**2265**
Morehead State University		24			9					**212**
University of Miami	50	87			50	8	87		87	**1027**
Western Kentucky University	20	183			98	182		214	4	**1430**

Table 12: Sequence numbers of pages containing results

CHAPTER SIX: RELATED WORK

We present several IE systems in this chapter. All of them relate to UniversityIE in some way. Most of them are distinct in features but all share the common IE approach. Although UniversityIE also be seen as part of the IE mainstream, there is not an example presented here that can be classified as a very similar system in terms of approach or architecture.

MITA

MetLife's Intelligent Text Analyzer (MITA) is an IE system that MetLife uses to process insurance applications. The automation of the process was not trivial due to the presence of many free-form text fields on the forms. MITA uses IE techniques to structure the extensive text fields. The goal of MITA is to "become more efficient and effective by allowing the underwriters to concentrate on the unusual and difficult aspects of a case and automate the more mundane and mechanical aspects". For more information on MITA, a good source is [4].

Domain

The free-form text fields in the applications are:

- *Physician Reason* field; the reason for the last visit to a personal physician,
- *Family History* field; the family medical history,
- *Major Treatments and Exams* field; major medical event within the last five years,
- *Not Revealed* field; important medical information not provided in other questions,
- *Occupation Title and Duty* field; employment information.

The text in the fields is relatively short and from a narrow domain. Not only that the entire set of fields deals with a similar language and terms, but each field also defines a sub-domain. These characteristics definitely help the analysis. The performance of the system is improved by activating specific IE tasks for specific fields, which also lowers the possibility of triggering wrong results. This has similarities to the selective rule application to the acquired pages in UniversityIE.

MITA is not expected to replace the human expert, but rather automate the more obvious and standard observations. This is interesting because it eases out the expectations from the very beginning. It is in the definition that the system doesn't deal much with the unusual but to point it out and eventually prepare it for further analysis.

An example describing the reasons why the client has last seen a physician borrowed from [4] illustrates the domain and why IE works for MetLife. This field frequently describes a visit for a checkup with additional modifiers for a specific disease, a chronic condition, or the fulfillment of an occupation or athletic need. Simple keyword search would not provide sufficient information. On the other hand, full in-depth NLP is not required due to the limited scope of the domain. The Physician Reason texts are analyzed in terms of four concepts: reason (regular visit, school checkup, or postpartum checkup), procedure- treatment (prescribed antibiotics, earwax removed, or HIV test) result (nothing found, all ok, or no treatment) and condition (high blood pressure, ear infection, or broken leg). The analysis looks for specific phrases or keywords that correspond to the concepts above.

Overview

MetLife processed 20,000 applications per month in 1998. The analysis of the domain was done from a sample of 20,000 applications. Initially, the domain (the free text fields of the application) proved to be bad-behaved – cryptic sentences, omission of subjects and verbs, incorrect or missing punctuation. In other words, the text was simply a sequence of noun phrases. Fortunately for the creators of MITA, they were able to suggest changes of the forms which they used to structure them better. Many questions were standardized using checkboxes or choice lists. For the remaining free text questions, they created the intelligent text analyzer.

The overview of MITA's process is presented in Figure 10. The Parser processes text first and tags each word with parts of speech and semantic categories, then combines these words into larger structures called *phrases* (such as noun phrase and verb phrase) and *constituents* (such as subject and verb). Lexicon Tables contain the parts of speech. Ontology Tables hold the semantic classification of words and phrases as well as the hierarchy of classes of which they are members. The Extractor compares the parsed and

tagged text to known extraction patterns. When a pattern applies, the extractor extracts important words and their classifications and categorizes them into what are called the text's concepts.

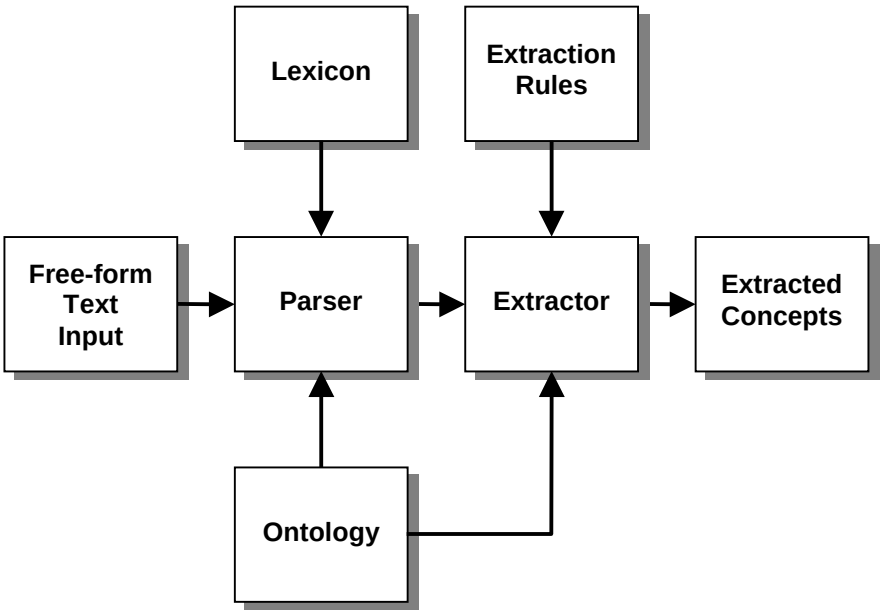

Figure 10: Overview of MITA's process

Being a commercial product, MITA has a fairly complex implementation. The parsing consists of six stages: part-of-speech tagging, part-of-speech disambiguation, bracketing (grouping in phrases), grouping (of phrases), buffering (assign subject/object to noun phrases), and segmentation. Although narrowed for the task, the parser is well suited for the task. There is no case, number or tense checking. Extraction rules are applied to check presence of concepts in the text. Each rule is an if-then rule that uses a pattern-matching mechanism. If the constraints on the left side of the rule are matched, the facts from the right-hand side fill in specific roles.

The impressive 75 megabytes of dictionary data are probably the best indicator of the implementation depth of a commercial system. MITA uses three dictionaries: Lexical, Medical and Occupational. All dictionaries were customized for MITA. A set of additional modules increases the effect of the rest of the system:

- Recognizers: Find and normalize nonstandard text contained in the input to make it easier to handle during formal parsing.
- Spell-Correction Module: A dictionary of common misspellings to correct misspelled words in the input.
- Multiword Lookup Module: A set of rules that look at typical patterns of text where multiword phrases occur. Individual words are replaced with a structure for the entire phrase.
- Composite Classifier: Multiword Lookup handles phrases at the word level, the composite classifier handles them at the class level. The Composite Classifier allows multiclass relationships to be defined at the most abstract level possible, allowing the widest number of variations to be captured in the ontology.
- Spell-Correction Module: Because many class names that are literally string concatenations of the names of other classes that compose these classes, MITA is able to cut short the complexity of the composite classifier by simply trying class-name string-concatenation combinations.
- Lexical Coder: Each class within MITA's ontology has a code. These codes are used by downstream analyzers (analyzers that use MITA's output) to automate parts of the underwriting process. These analyzers do not have access to MITA's ontology and do not have abstraction capabilities, so they use this module for that purpose. The module lists the hierarchical path of standard codes starting from the code of the extracted concept itself.

There are also many accompanying tools mostly intended for system maintenance. Using these, system support and underwriters themselves can update and manipulate the databases and maintain system integrity and performance.

CRYSTAL

This is a system that learns text analysis rules automatically from examples with an efficient learning algorithm. For more information on the system, consult [3] and [14].

Overview

CRYSTAL automatically induces a dictionary of *concept node* (CN) definitions. CNs are the information representations from a sentence analyzer also developed at the University of Massachusetts - BADGER. This system performs selective concept extraction. The role of CRYSTAL is to provide the CN dictionary that BADGER will later use for the extraction. An example of a CN from the medical domain is given in Figure 11.

```
CN-type: Sign or Symptom
Subtype: Absent
Extract from Direct Object
Active voice verb
Subject constraints:
       words include "PATIENT"
       head class: <Patient or Disabled Group>
Verb constraints:
       words include "DENIES"
Direct Object constraints:
       head class <Sign or Symptom>
```

Figure 11: A CN definition to identify "sign or symptom, absent"

The CN definition extracts "any episodes of nausea" from the sentence "The patient denies any episodes of nausea". The same CN will fail when applied to the sentence "Patient denies a history of asthma", since asthma is of semantic class <Disease or Syndrome>, which is not a subclass of <Sign or Symptom>.

From the example given above, it is clear that CN definitions are domain-specific and would not be simple to apply them to a different domain. Conceptually, the design of CRYSTAL assumes that a new conceptual dictionary is to be built for each IE application. Authors describe CRYSTAL as a tool to easily port BADGER to new domains. They achieve this by enabling automatic learning from a training corpus.

CN definitions are rules that apply a combination of lexical, semantic and syntactic constraints to an input instance. If all the constraints are satisfied, the system creates a case frame with the extracted information. To connect this to the concepts discussed in UniversityIE, CN definitions correspond to any of the IE task descriptions, and the case frames are the Templates. Another advantage of CRYSTAL is that it is not tied to a specific syntactic analysis. It uses whatever syntactic labels are found in the training set.

The CRYSTAL Algorithm

The algorithm is supervised, so it needs a human-annotated training set. The set consists of positive and negative instances of the concept being learned. The dictionary is initialized with a CN definition. Later, as instances of the training set are introduced, the constraints are gradually relaxed and similar definitions are merged. The generalization is as big as possible without producing errors on the training corpus. The algorithm is presented in Figure 12.

```
Initialize Dictionary and Training Instances Database
Do until no more initial CN definitions in Dictionary
    D = an initial CN definition removed from the Dictionary
    Loop
        D' = the most similar CN definition to D
        If D' = NULL, exit loop
        U = the unification of D and D'
        Test the coverage of U in Training Instances
        If the error rate of U ? Tolerance
            exit loop
        Delete all CN definitions covered by U
        Set D = U
    Add D to the Dictionary
Return the Dictionary
```

Figure 12: The CRYSTAL Algorithm

As an example of unifying two CN definitions, suppose that one definition has the class constraint <Sign or Symptom> for the subject buffer and the other has the class constraint <Laboratory or Test Result>. These unify to <Finding>, their common parent in the semantic hierarchy. If the direct object of one definition has a class constraint requiring both <Disease or Syndrome> and <Acquired Abnormality> and the other only requires <Disease or Syndrome>, then the unified class constraint on direct object will have only <Disease or Syndrome>.

Training and Performance

As any other NLP system, CRYSTAL faces noisy input. The syntax analysis may fail to correctly annotate the text. Mistaking verb for a noun or attaching the wrong sense to a word may also bring in noise. Finally, human annotators are not perfect and can also produce inconsistent data.

Even with perfect input, a set of training instances will never contain all possible ways to refer to a target concept. Or, with a relatively complete lexicon, some texts may simply

not contain enough distinguishable information. For example, the sentence “He succeeds Mr. Adams” is not clearly about management succession or political appointment.

For the application to the management succession domain, a training set of 359 annotated training texts and 240 blind test sets were used. CRYSTAL achieves 90% of the performance of the hand-coded equivalent system. Similarly, in the domain of hospital discharges, two sets of 251 documents were used and achieved 93% of the hand-coded rules performance.

WEBFOOT

Webfoot is even more interesting since it deals with World Wide Web documents as input. Although Webfoot has a very narrow domain, it brings up some of the issues discussed earlier in this document. Most of the discussion on Webfoot comes from [15]. The system presented here is a preprocessor for web pages. It formats web pages into logically coherent segments based on page layout cues.

Overview

The application of an IE system over a domain of web pages is a problem in itself with various levels of complexity. The experience described in the UniversityIE domain discussion is probably closer to the maximum end of the complexity axis. UniversityIE faces a domain where web pages are of different authors, standards and even have spatial distribution. Webfoot, on the other hand, uses a very narrow domain where the natural language text is usually not even grammatically correct. The content is later structured as input to CRYSTAL, a system discussed in the previous section of the chapter.

The domain described is web pages of weather forecast. Webfoot looks into extracting weather conditions associated with a day and location. The output is case frames with slots for Day, Conditions, High Temperature, Low Temperature and Location. The analysis of the domain showed that the web pages are usually filled with natural language segments that only cause difficulty to classical syntax parsers. An example is the sentence “Chance of rain 80 percent”. Also, the temporal character of the domain (weather forecast is good only for specific day or time of day) would introduce extra confusion. “Today” in a sentence will not have the same meaning on Tuesday and on Friday.

Parsing Web Pages

Webfoot takes the web page HTML source as input, applies the rules based on page layout cues, and divides the text into logically coherent segments. Webfoot handles a wide range of web page styles, including pages whose layout is indicated by HTML tags or by blank lines and white space, and pages with information in tabular or narrative format.

Ideally, the division of the web page groups logically related facts. In the weather forecast domain, a group of logically related facts should be a set of conditions related to the forecast for the day, or even more precisely the time of the day. A sample HTML page is given in Figure 13. Later, in Figure 14, sample segmentation by Webfoot divides the same text into segments.

```
<HEAD>
<TITLE>Forecast for NY072</TITLE>
</HEAD>
<BODY>
<pre>
BRONX-KINGS (BROOKLYN)-NASSAU-NEW YORK (MANHATTAN)-QUEENS-
RICHMOND (STATEN IS.)-
300 PM EST WED FEB 26 1997

.TONIGHT...CLOUDY WITH OCCASIONAL LIGHT RAIN. LOW IN THE MID 40S.
WIND SOUTHWEST 10 TO 15 MPH. CHANCE OF RAIN 80 PERCENT.
.THURSDAY...MOSTLY CLOUDY...WINDY AND MILD WITH A 30 PERCENT CHANCE
OF SHOWERS. HIGH 60 TO 65. WIND SOUTHWEST INCREASING TO 20 TO 30 MPH
WITH HIGHER GUSTS DURING THE AFTERNOON.
.THURSDAY NIGHT...PARTLY CLOUDY. LOW 40 TO 45.
.FRIDAY...MOSTLY SUNNY. HIGH 50 TO 55.
```

Figure 13: Sample HTML source from the National Weather Forecast web site

As an intermediate stage of extracting the page layout, Webfoot extracts the tags from segments with fewer that 20 words. The result is a sequence of delimiters (tags). In addition, Webfoot divides the tags in layers that reflect the structure of segments with more than 20 words. This itself is enough representation of the structure of the pages. On the top of these domain independent rules for page segmentation, additional delimiters can be added characteristic for the domain. For example, three delimiters were added for the weather domain to force a new segment on the mentioning of a new day of the week and for a conventional bulleted items from the National Weather Service reports.

```
<segment>
Field(1): <HEAD> <TITLE> Forecast for NY072 </TITLE>
Field(2): </HEAD> <BODY>
</segment>

<segment>
Field(1): <pre>
Field(2): BRONX - KINGS ( BROOKLYN )- NASSAU -
NEW YORK ( MANHATTAN )- QUEENS -
RICHMOND ( STATEN IS.)-
Field(3): 300 PM EST WED FEB 26 1997
</segment>

<segment>
Field(1): . TONIGHT ...
Field(2): CLOUDY WITH OCCASIONAL LIGHT RAIN .
Field(3): LOW IN THE MID 40S .
Field(4): WIND SOUTHWEST 10 TO 15 MPH .
Field(5): CHANCE OF RAIN 80 PERCENT .
</segment>
...
<segment>
Field(1): . FRIDAY ...
Field(2): MOSTLY SUNNY .
Field(3): HIGH 50 TO 55 .
</segment>
```

Figure 14: The sample text segmented by Webfoot

The sole task of segmenting the web pages would be purposeless if it is not further used to extract information from similar pages. The way that is done with Webfoot is by using the segmented pages as a training set for CRYSTAL, the system presented earlier in this chapter. In the presentation of CRYSTAL, the input was annotated text, but the segments take the role of that from Webfoot. In this application, CRYSTAL learns the rules from the segments and is later able to extract information from pages with a similar structure.

The system was applied to three sites: CNN Weather Service (CNN), National Weather Service (NWS) and the Australian Bureau of Meteorology (AUS). CNN contains automatically generated pages with intense usage of HTML. NWS uses sequences of sentence fragments, but has different style for different centers. AUS has no HTML or layout structure. The system was tested with a corpus of 20 annotated pages for each site. The results of the testing not only describe the performance of Webfoot, but also describe what can we expect from web pages with different degrees of structure.

A set of several CNN pages were enough to achieve high performance, a bit more for NWS and many pages from various AUS centers to train CRYSTAL to cope with their

pages. If we were to set UniversityIE for the task of Webfoot, we could use the Positional and Tabular rules in the more structured text segments, and the Syntax rules for the less-structured texts. In UniversityIE, there is no usage of HTML tags, except for the title and link description text. The human rule builder takes the role of Webfoot. There are two reasons for that. First, UniversityIE works over a wider domain (covered by three different IE methods) and second, it extracts information from pages with much less structure.

Webfoot performs well, but only where there is structure to be “learned”. As the results demonstrate, less structured sites are harder to learn. As a matter of fact, the author points out that he had to use not only more examples in number, but also examples from all AUS centers. The conclusion is that for sites with almost no structure, any pre-processing of this kind would give no results or require a large training set.

TIPSTER

The TIPSTER Text Program was a Defense Advanced Research Projects Agency (DARPA) led government effort to advance the state of the art in text processing technologies through the cooperation of researchers and developers in Government, industry and academia. Due to lack of funding, this program formally ended in the fall of 1998.

In its efforts to improve document processing efficiency and cost effectiveness TIPSTER focused on three underlying technologies.

- Document Detection: the capability to locate documents containing the type of information the user wants from either a text stream or a store of documents.
- Information Extraction: the capability to locate specified information within a text.
- Summarization: the capability to condense the size of a document or collection while retaining the key ideas in the material.

The work of TIPSTER can be seen through three phases. Phase I achieved major advances in creating the algorithms for document detection and information extraction and in improving the techniques for measuring those advances, through activities such as the Message Understanding Conferences (MUC) and the Text Retrieval Conferences

(TREC). Dramatic gains were made in the ability to automatically identify a wide range of items such as names (both personal and organizational), dates, locations, times, phone numbers, etc. Creation of software architecture was the main focus during the second phase. In order to standardize the technology components, enable "plug and play" capabilities among the various tools being developed, and permit the sharing of software. In Phase III, DARPA and other TIPSTER members sponsored 17 research and architecture development contracts with academic institutions and commercial companies in their effort to continue a balanced overall program, consisting of four basic parts: Advanced Research, Metrics-based Evaluations, a Structured Software Architecture, and Demonstration and Implementation Projects. For more information, the reader can consult the TIPSTER web site at [16].

TREC

The Text REtrieval Conference (TREC), co-sponsored by the National Institute of Standards and Technology (NIST) and the DARPA, started as part of the TIPSTER Text program. Its purpose was to provide the infrastructure necessary for large-scale evaluation of text retrieval methodologies. Many projects were executed in the domains of cross-lingual document processing, document filtering, high precision retrieval, interactive retrieval, querying, spoken document retrieval, and retrieving documents from very large collections. Many of the techniques are applicable and related to IE. For more information and TREC-related documents, please consult [17].

There are many TREC-related systems that describe processes whose stages use IE or similar techniques. All these systems are NLP system at the first place, and they deal with issues close to IE. Therefore, it is not surprising that we found many TREC papers useful in our work. As an example and also an insight into TREC, the system described in [18] is a good example. Although the paper concentrates more on retrieval, the text processing methods also influenced our initial approach in the design of UniversityIE.

MUC

MUC performs evaluations of information extraction system according to pre-established tasks. These tasks with appropriate examples borrowed from [20] are:

- Entities - Named Entities (NE): Proper names and quantities of interest: person, organization and location names, dates, times, percentages, and monetary amounts. The following example is annotated with SGML within the text stream:
  The <ENAMEX TYPE="LOCATION">U.K.</ENAMEX> satellite television broadcaster said its subscriber base grew <NUMEX TYPE="PERCENT">17.5 percent</NUMEX> during <TIMEX TYPE="DATE">the past year</TIMEX> to 5.35 million
- Equivalence Classes - Coreference (CO): coreference of type identity was marked and scored. Here's an example illustrates identity coreference between "its" and "The U.K. satellite television broadcaster" as well as that between the function "its subscriber base" and the value "5.35 million.":
  The U.K. satellite television broadcaster* said **its* subscriber base* grew 17.5 percent during the past year to *5.35 million
- Attributes - Template Elements (TE): Consist of name, type, descriptor, and category slots. The attributes in the Template Element serve to further identify the entity beyond the name level. NAME slot. TYPE contains persons, organizations, artifacts, and locations. DESCRIPTOR contains substantial descriptors used in the text. CATEGORY contains categories dependent on the element involved: persons (civilian, military, or other), organizations (government, company, or other), artifacts (vehicles - for traveling on land, water or in air), locations (city, province, country, region, water, airport, or unknown). Here's an example:
  <ENTITY-9602040136-11> :=
  ENT_NAME: "Dennis Gillespie"
  ENT_TYPE: PERSON
  ENT_DESCRIPTOR: "Capt."
  / "the commander of Carrier Air Wing 11"
  ENT_CATEGORY: PER_MIL
- Facts - Template Relations (TR): Captures relationships between template elements and can be thought of as a task in which well-defined facts are extracted from newswire text. MUC-7 limited TR to relationships with organizations: employee_of, product_of, location_of. The following is an example of TR:

<EMPLOYEE_OF-9602040136-5> :=

PERSON: <ENTITY-9602040136-11>

ORGANIZATION: <ENTITY-9602040136-1>

<ENTITY-9602040136-11> :=

ENT_NAME: "Dennis Gillespie"

ENT_TYPE: PERSON

ENT_DESCRIPTOR: "Capt."

/ "the commander of Carrier Air Wing 11"

ENT_CATEGORY: PER_MIL

<ENTITY-9602040136-1> :=

ENT_NAME: "NAVY"

ENT_TYPE: ORGANIZATION

ENT_CATEGORY: ORG_GOVT

- Events - Scenario Template (ST): Built around an event in which entities participated. The domain of the dataset is provided by the scenario and allowed for relevancy judgments. The task definition for ST required relevancy and fill rules. The structure of the template and the task definition depends on the author of the task, but the richness of the templates also illustrates the usefulness of information extraction to users most effectively.

CHAPTER SEVEN: CONCLUSION

SUMMARY AND CONCLUDING REMARKS

We developed UniversityIE – a rule-based NLP system that executes IE tasks over web content. We implemented it entirely in Java on top of SCOUT, a general-purpose web crawler. UniversityIE extracts information with a set of tasks that are instances of three basic IE methods we designed and implemented – Positional, Tabular and Syntax.

We also overtook extensive analysis of the domain of university web pages. The domain characteristics, as determined by our analysis, are in line with the experience of related IE systems and in fact, helped to anticipate the performance of UniversityIE. Furthermore, this analysis led to a heuristic for selective page retrieval, which significantly improved the performance of the system. The system applies heuristic to each web page and uses it to order the URLs for retrieval. We also use heuristic flags in the IE methods as part of the retrieval.

We used a set of university web sites to develop the methods and define the IE tasks. We executed a configuration of eight tasks over the initial set of sites. We further tested UniversityIE over five additional sites and one more task. The performance was in line with the performance over the initial set, so it is appropriate to say that the system has predictable performance.

This project tightly integrates two existing systems: WordNet and the Link Grammar Parser. The integration included porting and restructuring of the complete C code to Java and wrapping the systems with interfaces (APIs) used by the rest of the system. The Link grammar Parser is wrapped as a standalone server.

CONTRIBUTIONS

The main contributions of this work are:

- Domain analysis of university web pages from an IE perspective.
- Designed and implemented three IE methods that can be plugged into a web-crawling engine.

- Introduced and developed IE-based web-crawling heuristic.
- Effective text tagging and normalization.
- Ported, restructured and integrated WordNet and Ling Grammar Parser.

Each of these contributions is described in more details below.

Domain analysis of university web pages from an IE perspective. IE techniques have previously not been applied to the domain of university web pages. The main characteristics of this domain we pointed out from the domain analysis, are the variety of data types, different forms of containment of information, and the spatial distribution. These specific features are moderate or more uniform in cases where the domain is narrower and/or more structured.

Designed and implemented three IE methods that can be plugged into a web-crawling engine. We implemented each IE algorithm – the Positional, Tabular and Syntax methods as a Java class. The clear and uniform interface is due to the plug-in architecture of SCOUT. Each algorithm implements a relatively generic method. The methods are highly parameterized and can be configured for a variety of different extraction tasks. This enables easy reuse of the system for new domains where most of the effort would involve analyzing the domain and configuring the tasks.

Introduced and developed IE-based web-crawling heuristic. One of the main issues that the domain analysis brought up is the high number of pages that a multi-task IE system would extract in relatively few steps. This can also cause placement of the page of interest too deep in the queue. Shortening the access time to the information-containing page is not just a performance issue. It is also an assurance that the page will be processed at all. A simple Breadth First Search may easily accumulate a large number of URLs only few layers below the home page. For example, the TUITION task for Northern Kentucky University didn’t make it to the page containing the desired information, even with the heuristic incorporated into the crawling. A major contribution of this project is the analysis that brought up the need for a web-crawling heuristic and its implementation with noticeable effect on the performance.

Effective text tagging and normalization. Every aspect of Tagging and Normalization used in UniversityIE is in one form or another present in other IE systems. The tagging process in UniversityIE, although fairly simple, manages to provide sufficient information for the succeeding IE stages. Most of the existing IE systems require a collection of complex dictionaries, lexicons or collections of annotated sentences. The main reason why the Tagging process in UniversityIE is simplified is because the domain is not supported by such domain-descriptive sets. Normalization in UniversityIE has additional role – it is used as a preparatory stage for the actual extraction. For example, TOEFL is sure to be present only as an abbreviation and the names are tagged with the type Name, which is then used in to extract, for example, the name of the President.

Ported, restructured and integrated WordNet and Ling Grammar Parser. We invested a significant effort in the Java port of these systems. Having all of the code in Java assures that our implementation of UniversityIE is platform independent, and also provides two reusable components for a variety of NLP systems that may require Java-written lexicon or parser.

FUTURE RESEARCH

One of the immediate next steps is a detailed study of UniversityIE's performance over a different domain. A similar set of sites and number of IE tasks can help build a better understanding of the performance of the system. One candidate is the domain of online news sites such as CNN, ABC, and BBC. Although this domain has the same level of complexity, these sites are far more structured and better organized. News from the continents or regions, or specific topics of interest can correspond to the departments in the university domain. We group the remaining points of future research under the IE stages.

Document Acquisition. This stage can be improved both in terms of performance and flexibility of the acquisition process. The direction of research is mainly in developing superior heuristics and a framework that would allow different heuristics to be used per IE task.

Tokenization & Tagging. The current mechanisms and support data structures for this stage are fairly simple. With an application to a new domain, or by incorporating existing annotated text sets into the system, we can significantly improve this stage. The Normalization stage can also be improved by enhancing it with both spatial (normalization of larger portions of the text) and semantic intelligence.

Sentence Analysis. The syntax analysis capability in the system is currently used only if the IE task is specifically defined as a Syntax task. An obvious usage of the parser interface is in the other two methods (Positional and Tabular). These tasks would then rely on more than just keywords, but would have more information about the sentence. Enhancements like this would be helpful in both the extraction and merging stage.

Extraction. At present, there is no interaction between IE tasks. Existing mechanisms in the system allow setting up a configuration where for example, the results from a Syntax task are used by a Positional task. This would be a simple use of the Results structure and will most likely not improve the system. An interesting research topic is development of more complex IE methods that utilize several methods at the same time.

Merging. Our experience demonstrated that this stage is non-trivial. In a domain as broad as ours, it is likely to retrieve many potential matches. In most cases, only one result is required and only one piece of extracted information is correct. It is important to be able to distinguish the desired extracted data. Looking into enhancements of the Merge stage in the present system and different domain solutions is certainly one of the key future activities.

Template Generation. This stage needs additional capabilities for result processing. Even when the correct information is extracted, it may often be necessary to further interpret the data. An example for such a case is tuition information – sometimes it provided as cost per credit, sometimes as cost per semester/year, or a part of the total expenses.

Additional future work is necessary for building support tools to configure parameters in all stages of IE and system’s environment.

APPENDICES

APPENDIX A: ENVIRONMENT SETUP

Java

UniversityIE code is completely Java. The JDK version we used for development is 1.1.8, but most of the code will probably work with any 1.1 Java release with very little or no modification.

The packages that constituted SCOUT are:

- *ADT* Basic abstract data types, e.g. queues and stacks
- *HTTPClient* Ronald Tschalär's HTTP library
- *JavaDoc* The JavaDoc application
- *Logger* Package for disk and screen logging
- *pat* Steven R. Brandt's regular expression package
- *Scout* Core Scout classes
- *SGMLKit* A rudimentary SGML parser

Most of the modifications and additions to SCOUT are classes that are also in the *Scout* package. In addition, UniversityIE introduces two more packages:

- *link* Java port of the Link Grammar Parser
- *wordnet* Java port of the WordNet Lexical Database

Please refer to [6] for more details on SCOUT-related packages.

To use *link* and *wordnet*, it is necessary to initialize the packages from the code. We initialize WordNet with a call to *wnutil.wninit()*. The Ling Grammar Parser is a little bit more complicated since we implemented it as a separate task (on the same or different machine on the network) and we initialize only the interface to the task by instantiating the appropriate stub class: *linkParserRPC*. The actual sequence is given in Figure 15.

```
lpRPC = new linkParserRPC(confFile, logger);
if (wnutil.wninit()!=0) {         // open database
    wnrtl.display_message("wn: Fatal error - cannot open WordNet database\n");
    return;
}
else {
    wnrtl.dflag = wnrtl.fileinfoflag = wnrtl.offsetflag = wnrtl.wnsnsflag = 0;
}
```

Figure 15: Initialization code for the link and wordnet packages

The API for *wordnet*, beside the initialization call, consists of one other function - *wordnet_search* that returns a vector of synonymous words to the one given in the parameters.

In the case of *link*, the API consists of the methods in the class *linkParserRPC*. Two of these (open and close) are just synchronization calls to serialize the access to the parser task. The other four methods are:

- *parse*: parse a sentence and return just a flag for correct parsing,
- *findPhrase*: uses the generated parse tree(s) to find specific phrase,
- *getLinkage*: retrieves all linkages from the parse structure,
- *getSentenceWords*: retrieves all sentence words from the parse structure.

IE Tasks Setup

UniversityIE uses the setup mechanisms from Scout with minor extensions. Each search is defined with two configuration files.

The “crawling” setup file, usually with extension *ini* contains the parameters that describe the URL, site- and domain-access parameters, as well as cache management and logging. We added two debug-related parameters for UniversityIE. One is a flag that puts the system into a collect-only mode where the cache is filled with the collected pages but no IE tasks are actually executed. The other flag re-runs the final, merging phase that simply takes all the results and extract the best candidates for each task.

The template file is the place where we describe all IE task instantiations. Not only it is a list of tasks, but also is the place where we can set additional parameters. For discussion on the syntax of the tags used in this file (usually with extension *html*) please consult [6].

We also added one parameter for the purposes of UniversityIE – PARAMS, which points to a section of the corresponding IE, task description file described next.

Each IE method comes with a description file (usually with extension *dat*) where we describe all variants (instances) of the method. Each IE task described in this document has a corresponding description in one of these files:

- *KVRule.dat* – positional method instances
- *TVRule.dat* – tabular method instances
- *PhraseRule.dat* – syntax method instances
- *SubsetRule.dat* – subset task description

The sections in these files correspond to the parameters discussed earlier in the text.

Additional Files

allnames.dat – a list of names (first and last) used to tag words with the name type.

IERegWords.dat – regular expressions describing the types used in tagging

IENrm.dat – normalization rules

export.dat – description of the SQL generation in the merge phase

APPENDIX B: CONFIGURATION FILES

This section lists all configuration files necessary to execute an IE session. These files are the working environment of SCOUT and UniversityIE. The University of Kentucky task is used as an example and hence uky.ini is the configuration file presented in this section.

uky.ini

```
[SCOUT]
StartURL=http://www.cs.uky.edu
UseGUI=true
RuleBase=Rules
LogFile=UKY10.log
PersistFile=UKY.dat
ScoutDelay=30
NetDelay=30
TemplateBase=Templates
RestrictHost=null
RestrictDomain=uky.edu
CacheDir=UKY1Cache
MaxCacheFiles=6400
MaxURLs=0
RequestRobotsFile=false
SearchCache=true
SearchWeb=false
Collect=false
Finalize=true
[EXTRACTOR]
EntityFile=../SGMLKit/entities.txt
GlobRegExps=false
[LINK]
Dictionary=tiny.dict
```

allnames.dat

This file contains more than 20,000 first and last names.

IERegWords.dat

The list of regular expressions describing different type tags assigned to words.

```
P[A-Z][A-Z][0-9][0-9][0-9][0-9][0-9]([-][0-9][0-9][0-9][0-9])?;;;ZipCode
P[0-9][0-9][0-9][0-9][0-9]([-][0-9][0-9][0-9][0-9])?;;;ZipCode
D[0-1][0-9]/[0-2][0-9];;;Date
D[0-1][0-9]/[0-2][0-9]/[0-9][0-9];;;Date
E.+@.+\..+;;;e-mail
U[w][w][w]\..+;;;URL
I[1-9]+[0-9,]*;;;Integer
M[$][0-9]+[0-9,]*[\.]*[0-9][0-9]+;;;Money
F[0-9]+[0-9,]*\.[0-9]+;;;Float
i[A-Z]\.;;;MidInitial
SU.S.;;;String
```

IENrm.dat

The list of normalization rules.

```
Sin Sstate > Sin_state
Snon Sresident > Snon_resident
Snon-resident > Snon_resident
Sout Sof Sstate > Sout_of_state
Sfull Stime > Sfull_time
Spart Stime > Spart_time
Sfull-time > Sfull_time
Spart-time > Spart_time
f[] i[] S[] > n<S1><><S2><><S3>
f[] i[] f[] > n<S1><><S2><><S3>
f[] f[] > n<S1><><S2>
Stest Sof Senglish Sas Sa Sforeign Slanguage > STOEFL
Stest Sof Senglish Sas Sforeign Slanguage    > STOEFL
Stest Sof fenglish Sas Sa Sforeign Slanguage > STOEFL
Stest Sof fenglish Sas Sforeign Slanguage    > STOEFL
Sgraduate Smanagement Sadmission Stest       > SGMAT
I[] Spercent > S<I1>%
F[] Spercent > S<F1>%
S$ F[] > M<F1>
S$ I[] > M<I1>
Sjanuary   I[1..31] > D01/<I1>
Sfebruary  I[1..29] > D02/<I1>
Smarch     I[1..31] > D03/<I1>
Sapril     I[1..30] > D04/<I1>
Smay       I[1..31] > D05/<I1>
Sjune      I[1..30] > D06/<I1>
Sjuly      I[1..31] > D07/<I1>
Saugust    I[1..31] > D08/<I1>
Sseptember I[1..30] > D09/<I1>
Soctober   I[1..31] > D10/<I1>
Snovember  I[1..30] > D11/<I1>
Sdecember  I[1..31] > D12/<I1>
Szero  > I0
Sone   > I1
Stwo   > I2
Sthree > I3
Sfour  > I4
Sfive  > I5
Ssix   > I6
Sseven > I7
Seight > I8
Snine  > I9
Sten   > I10
```

KVRule.dat

The list of Positional IE tasks.

```
\\PRESIDENT
\HEURISTICS
0president
0president's
0future+students
0prospective
0fact

\KEYWORD
Spresident

\OTHERKEYWORDS
null

\RANGE
n[]

\POSITION
vkv

\MAXDISTANCE
7

\COLUMN
```

```
PRESIDENT

\MINMAX
MAX

\DEPARTMENT
NO

\\TOEFL
\HEURISTICS
1requirement
0requirements
0international
0graduate
0undergraduate
0education
0academic
0admission
0admissions
0department
0school
0master
0engineering
0computer+science
0fact

\KEYWORD
STOEFL

\OTHERKEYWORDS
null

\RANGE
I400..660

\POSITION
vksv

\MAXDISTANCE
10

\COLUMN
TOEFL

\MINMAX
MAX

\DEPARTMENT
MAYBE

\CONCATENATE
false

\\GRSTUDENTS
\HEURISTICS
0future+students
0prospective
0fact

\KEYWORD
Sgraduate

\OTHERKEYWORDS
student

\RANGE
I5..5000

\POSITION
vk

\MAXDISTANCE
2

\COLUMN
GRSTUDENTS

\MINMAX
MAX

\DEPARTMENT
```

```
MAYBE

\\UGRSTUDENTS
\HEURISTICS
0future+students
0prospective
0fact

\KEYWORD
Sundergraduate

\OTHERKEYWORDS
student

\RANGE
I5..5000

\POSITION
vk

\MAXDISTANCE
2
\COLUMN
UGRSTUDENTS

\MINMAX
MAX

\DEPARTMENT
MAYBE

\\ADDRESS
\HEURISTICS
1requirement
0requirements
0international
0graduate
0undergraduate
0education
0academic
0admission
0admissions
0department
0school
0master
0engineering
0fact
0contact
0computer+science
0computer+information+science
0electrical+engineering+computer+science
0computer+science+engineering
0mathematics+computer+science
\KEYWORD
I[1..9999]

\OTHERKEYWORDS
null

\RANGE
P[]

\POSITION
ksv

\MAXDISTANCE
10

\COLUMN
ADDRESS

\MINMAX
MAX

\DEPARTMENT
YES

\CONCATENATE
true
```

TVRule.dat

The list of Tabular IE tasks.

```
\\TUITION
\HEURISTICS
0finance
0tuition
0fees
0fact

\KEYWORDS
tuition

\COLUMNS
resident\in_state
non_resident\out_of_state

\ROWS
full_time
part_time
undergraduate
graduate
----

\TYPE
I[50..]
\TYPE_F
F[50..]
\COLUMN_DESCRIPTIONS
IFULL_TIME\OFULL_TIME
IPART_TIME\OPART_TIME
IUNDERGRADUATE\OUNDERGRADUATE
IGRADUATE\OGRADUATE
IUNKNOWN\OUNKNOWN

\MAXDISTANCE
30

\DEPARTMENT
MAYBE

\\DEADLINES
\HEURISTICS
1application
1admission
0date
0procedure
0fact

\KEYWORDS
deadline

\COLUMNS
%%%%%%%%%%

\ROWS
fall
spring
summer
----

\TYPE
D[]
\TYPE_F
D[]

\COLUMN_DESCRIPTIONS
DEADLINE_FALL
DEADLINE_SPRING
DEADLINE_SUMMER
UNKNOWN

\MAXDISTANCE
50

\DEPARTMENT
MAYBE
```

PhraseRule.dat

The list of Syntax IE tasks.

```
\\LOCATION
\HEURISTICS
1visitor
1visiting
0tour
0about
0profile
1visitning+campus
1prospective+student
1information+campus
0fact

\QPHRASE
NP

\LINKS
1G

\KEYWORDS
university.n

\FIXED
false

\RESULT
No

\IMMEDIATE
false

\QPHRASE
VP

\KEYWORDS
locate.v
situate.v
lie.v
occupy.v

\FIXED
true

\RESULT
No

\IMMEDIATE
true

\SUBPHRASE
PP

\LINKS
0ND
0Yd

\KEYWORDS
null

\FIXED
false

\RESULT
Yes

\IMMEDIATE
false

\DEPARTMENT
MAYBE

\\CAMPUSES
\HEURISTICS
1visitor
1visiting
```

```
0tour
0about
0profile
1visitning+campus
1prospective+student
1information+campus
0fact

\QPHRASE
NP

\LINKS
1G

\KEYWORDS
campus.n

\FIXED
false

\RESULT
No

\IMMEDIATE
false

\QPHRASE
VP

\KEYWORDS
locate.v
lie.v
occupy.v

\FIXED
true

\RESULT
No

\IMMEDIATE
true

\SUBPHRASE
PP

\LINKS
0ND
0Yd

\KEYWORDS
null

\FIXED
false

\RESULT
Yes

\IMMEDIATE
false

\DEPARTMENT
MAYBE
```

SubsetRule.dat

The subset (department) rule settings.

```
\\DEPARTMENTS
\KEYWORDS
university
college
school
department
office
```

EduTemplate.html

The configuration file that describes the SCOUT settings and IE tasks.

```
<SCOUT TSTART NAME="School">

<SCOUT TYPE=I NAME=RULEBASE VALUE=Templates/>
<SCOUT TYPE=I NAME=TEMPLATEBASE VALUE=Rules/>
<SCOUT TYPE=C NAME=SETVAR VALUE="DEPARTMENT=computer+science">

<SCOUT
TYPE=S
NAME=DEPARTMENTS
VALUE=null
RULE=Scout.IESubsetRule
VALIDATE=true>

<SCOUT
TYPE=E
NAME=SQL
VALUE=null
RULE=Scout.IEExportResults>

<SCOUT
TYPE=D
NAME=BFS
VALUE=null
PARSE=void
RULE=Scout.IEBFS
VALIDATE=true>

<SCOUT
TYPE=D
NAME=PARSE
VALUE=null
LOGOPT=NOLOG
RULE=Scout.IEParserRule
VALIDATE=true>

<SCOUT
TYPE=D
NAME=KV
PARAMS=PRESIDENT
VALUE=null
REQUIRE=PARSE
RULE=Scout.IEKeywordValueRule1
VALIDATE=true>

<SCOUT
TYPE=D
NAME=TV
PARAMS=TUITION
VALUE=null
REQUIRE=PARSE
RULE=Scout.IETableValuesRule
VALIDATE=true>

<SCOUT
TYPE=D
NAME=TV
PARAMS=DEADLINES
VALUE=null
REQUIRE=PARSE
RULE=Scout.IETableValuesRule
VALIDATE=true>


<SCOUT
TYPE=D
NAME=KV
PARAMS=TOEFL
VALUE=null
REQUIRE=PARSE
RULE=Scout.IEKeywordValueRule1
VALIDATE=true>

<SCOUT
TYPE=D
```

```
NAME=KV
PARAMS=ADDRESS
VALUE=null
REQUIRE=PARSE
RULE=Scout.IEKeywordValueRule1
VALIDATE=true>

<SCOUT
TYPE=D
NAME=KV
PARAMS=GRSTUDENTS
VALUE=null
REQUIRE=PARSE
RULE=Scout.IEKeywordValueRule1
VALIDATE=true>

<SCOUT
TYPE=D
NAME=KV
PARAMS=UGRSTUDENTS
VALUE=null
REQUIRE=PARSE
RULE=Scout.IEKeywordValueRule1
VALIDATE=true>

<SCOUT
TYPE=D
NAME=PHRASE
PARAMS=LOCATION
VALUE=null
REQUIRE=PARSE
RULE=Scout.IEFindPhraseRule
VALIDATE=true>

<SCOUT
TYPE=D
NAME=PHRASE
PARAMS=CAMPUSES
VALUE=null
REQUIRE=PARSE
RULE=Scout.IEFindPhraseRule
VALIDATE=true>

<SCOUT TEND>
```

EduTemplate1.html

The configuration file used for testing the ADDRESS task.

```
<SCOUT TSTART NAME="School">
<SCOUT TYPE=I NAME=RULEBASE VALUE=Templates/>
<SCOUT TYPE=I NAME=TEMPLATEBASE VALUE=Rules/>
<SCOUT TYPE=C NAME=SETVAR
VALUE="DEPARTMENT=CS#CIS#EECS#CSE#MCS#computer+science#computer+information+scie
nce#electrical+engineering+computer+science#computer+science+engineering#mathema
tics+computer+science">

<SCOUT
TYPE=S
NAME=DEPARTMENTS
VALUE=null
RULE=Scout.IESubsetRule
VALIDATE=true>

<SCOUT
TYPE=E
NAME=SQL
VALUE=null
RULE=Scout.IEExportResults>

<SCOUT
TYPE=D
NAME=BFS
VALUE=null
PARSE=void
RULE=Scout.IEBFS
VALIDATE=true>

<SCOUT
TYPE=D
NAME=PARSE
VALUE=null
LOGOPT=NOLOG
RULE=Scout.IEParserRule
VALIDATE=true>

<SCOUT
TYPE=D
NAME=KV
PARAMS=ADDRESS
VALUE=null
REQUIRE=PARSE
RULE=Scout.IEKeywordValueRule1
VALIDATE=true>

<SCOUT TEND>
```

export.dat

```
joe.res

\TUITION
PHRASE/CAMPUSES,CAMPUSES
TV/TUITION,IPART_TIME,IFULL_TIME,OPART_TIME,OFULL_TIME,IUNDERGRADUATE,OUNDERGRAD
UATE,IGRADUATE,OGRADUATE,IUNKNOWN,OUNKNOWN
KV/PRESIDENT,PRESIDENT

\CAMPUS
PHRASE/CAMPUSES,CAMPUSES
PHRASE/LOCATION,LOCATION
TV/DEADLINES,DEADLINE_FALL,DEADLINE_SPRING,DEADLINE_SUMMER,UNKNOWN
```

APPENDIX C: UNIVERSITY INFORMATION EXTRACTED

EKU

```
KV/GRSTUDENTS: [0,0]: GRSTUDENTS: 414
KV/PRESIDENT: [0,0]: PRESIDENT: Robert R. Martin
PHRASE/LOCATION: [0,0]: S: (0)  through the EKU Laboratory School
TV/TUITION: [0,0]: IFULL_TIME:50
TV/TUITION: [0,1]: IUNDERGRADUATE:245
TV/TUITION: [0,2]: IGRADUATE:463
TV/TUITION: [0,3]: IPART_TIME:200
TV/TUITION: [0,4]: ???:
TV/TUITION: [0,5]: ???:
TV/TUITION: [0,6]: ???:
TV/TUITION: [0,7]: ???:
TV/TUITION: [0,8]: ???:
TV/TUITION: [0,9]: ???:
PHRASE/CAMPUSES:
KV/ADDRESS: [0,0]: 368 Eastern Kentucky University Richmond KY 40475
TV/DEADLINES: [0,0]: DEADLINE_SUMMER: 12/17
TV/DEADLINES: [0,1]: ???:
TV/DEADLINES: [0,2]: ???:
TV/DEADLINES: [0,3]: ???:
KV/TOEFL: [0,0]: TOEFL: 500
KV/UGRSTUDENTS:
```

Louisville

```
KV/GRSTUDENTS: [0,0]: GRSTUDENTS:100
KV/PRESIDENT:
PHRASE/LOCATION: [0,0]: S:(0)  in Kentucky ’s largest urban area
TV/TUITION: [0,0]: IUNDERGRADUATE:2000
TV/TUITION: [0,1]: ???:
TV/TUITION: [0,2]: ???:
TV/TUITION: [0,3]: ???:
TV/TUITION: [0,4]: ???:
TV/TUITION: [0,5]: ???:
TV/TUITION: [0,6]: ???:
TV/TUITION: [0,7]: ???:
TV/TUITION: [0,8]: ???:
TV/TUITION: [0,9]: ???:
PHRASE/CAMPUSES: [0,0]: S:(0)  on Belknap Campus at 1800 Arthur St . The
Radiation Safety office is located on the Health Sciences Center Campus in the
Medical Dental Research Building Room 728 at 511 South Floyd St. History of DEHS
KV/ADDRESS: [0,0]: ADDRESS: 555 Academic Court San Antonio TX 78204
TV/DEADLINES: [0,0]: DEADLINE_FALL:05/1
TV/DEADLINES: [0,1]: DEADLINE_SPRING:10/1
TV/DEADLINES: [0,2]: DEADLINE_SUMMER:03/1
TV/DEADLINES: [0,3]: ???:
KV/TOEFL: [0,0]: TOEFL: 550
```

UMICH

```
KV/GRSTUDENTS: [0,0]: GRSTUDENTS: 1999
KV/PRESIDENT: [0,0]: PRESIDENT: Lee C. Bollinger
PHRASE/LOCATION: [0,0]: S:(0)  quadrangle the Executive Education Center is
located near the center of campus which gives visitors the option of exploring
the University and Ann Arbor at their leisure without the need for cars or
public transportation
TV/TUITION: [0,0]: IUNDERGRADUATE: 1999
TV/TUITION: [0,1]: ???:
TV/TUITION: [0,2]: ???:
TV/TUITION: [0,3]: ???:
TV/TUITION: [0,4]: ???:
TV/TUITION: [0,5]: ???:
TV/TUITION: [0,6]: ???:
TV/TUITION: [0,7]: ???:
TV/TUITION: [0,8]: ???:
TV/TUITION: [0,9]: ???:
PHRASE/CAMPUSES: [0,0]: S:(0)  on the first floor of the Michigan Union and in
the lobby of the North Campus Commons
KV/ADDRESS: [0,0]: ADDRESS: 1108 Ann Arbor MI 48109
TV/DEADLINES: [0,0]: DEADLINE_FALL: 02/1
TV/DEADLINES: [0,1]: DEADLINE_SPRING: 11/1
TV/DEADLINES: [0,2]: DEADLINE_SUMMER: 02/1
TV/DEADLINES: [0,3]: ???:
KV/TOEFL: [0,0]: TOEFL: 560
KV/UGRSTUDENTS:
```

UKY

```
KV/GRSTUDENTS: 75
KV/PRESIDENT: [0,0]: PRESIDENT: Charles T. Wethington
PHRASE/LOCATION: [0,0]: S:(0)  in the city of Lexington in the central Kentucky
Bluegrass Region RIGHT-WALL
TV/TUITION: [0,0]: IGRADUATE: 1,798
TV/TUITION: [0,1]: OGRADUATE: 5,058
TV/TUITION: [0,2]: IFULL_TIME: 188
TV/TUITION: [0,3]: OFULL_TIME: 550
TV/TUITION: [0,4]: IPART_TIME: 946
TV/TUITION: [0,5]: OPART_TIME: 2,756
TV/TUITION: [0,6]: ???:
TV/TUITION: [0,7]: ???:
TV/TUITION: [0,8]: ???:
TV/TUITION: [0,9]: ???:
PHRASE/CAMPUSES: [0,0]: S:(1)  to the north of campus
KV/ADDRESS: [0,0]: ADDRESS: 3147 Custer Drive Suite E Lexington  KY 40517
TV/DEADLINES: [0,0]: DEADLINE_FALL: 02/1
TV/DEADLINES: [0,1]: DEADLINE_SPRING: 07/15
TV/DEADLINES: [0,2]: DEADLINE_SUMMER: 10/1
TV/DEADLINES: [0,3]: ???:
KV/TOEFL: [0,0]: TOEFL: 550
KV/UGRSTUDENTS:
```

SMITH

```
KV/GRSTUDENTS:
KV/PRESIDENT: [0,0]: PRESIDENT: Ruth J. Simmons
PHRASE/LOCATION:
TV/TUITION:
PHRASE/CAMPUSES:
PHRASE/CAMPUSES: [0,0]: S:(0)  of New England
KV/ADDRESS: [0,0]: ADDRESS: 9700 9800 9900 do lab9demo -Dim 10001
TV/DEADLINES: [0,0]: DEADL INE_FALL:11/15
TV/DEADLINES: [0,1]: DEADLINE_SPRING:01/1
TV/DEADLINES: [0,2]: ???:
TV/DEADLINES: [0,3]: ???:
KV/TOEFL:
KV/UGRSTUDENTS:
```

SDSTATE

```
KV/GRSTUDENTS:
KV/PRESIDENT: Peggy Gordon
PHRASE/LOCATION: [0,0]: S:(0)  in Brookings
TV/TUITION: [0,0]: IUNDERGRADUATE: 104.94
TV/TUITION: [0,1]: OUNDERGRADUATE: 111.93
TV/TUITION: [0,2]: ???:
TV/TUITION: [0,3]: ???:
TV/TUITION: [0,4]: ???:
TV/TUITION: [0,5]: ???:
TV/TUITION: [0,6]: ???:
TV/TUITION: [0,7]: ???:
TV/TUITION: [0,8]: ???:
TV/TUITION: [0,9]: ???:
PHRASE/CAMPUSES:
KV/ADDRESS: [0,0]: ADDRESS:511  Brookings  SD 57007
TV/DEADLINES: [0,0]: DEADLINE_SPRING: 01/19
TV/DEADLINES: [0,1]: DEADLINE_SUMMER: 05/17
TV/DEADLINES: [0,2]: DEADLINE_FALL: 09/13
TV/DEADLINES: [0,3]: ???:
KV/TOEFL: [0,0]: TOEFL: 500
KV/UGRSTUDENTS:
```

OHIOU

```
KV/GRSTUDENTS: [0,0]: GRSTUDENTS:30
KV/PRESIDENT: [0,0]: PRESIDENT: Daniel S. Williams
PHRASE/LOCATION: [0,0]: S:(1)  about 75 miles southeast of Columbus
TV/TUITION: [0,0]: IGRADUATE: 75
TV/TUITION: [0,1]: ???:
TV/TUITION: [0,2]: IUNDERGRADUATE: 2000
TV/TUITION: [0,3]: ???:
TV/TUITION: [0,4]: ???:
TV/TUITION: [0,5]: ???:
TV/TUITION: [0,6]: ???:
TV/TUITION: [0,7]: ???:
TV/TUITION: [0,8]: ???:
TV/TUITION: [0,9]: ???:
PHRASE/CAMPUSES: [0,0]: S:(0)  in a small town surrounded by numerous state
parks RIGHT-WALL
KV/ADDRESS: [0,0]: ADDRESS:312 Athens  Ohio 45701
TV/DEADLINES: [0,0]: DEADLINE_FALL: 06/1
TV/DEADLINES: [0,1]: DEADLINE_SPRING: 03/1
TV/DEADLINES: [0,2]: DEADLINE_SUMMER: 05/1
TV/DEADLINES: [0,3]: ???:
KV/TOEFL:
KV/UGRSTUDENTS:
```

NKU

```
KV/GRSTUDENTS: [0,0]: GRSTUDENTS:7
KV/PRESIDENT: [0,0]: PRESIDENT: James L. Alford
PHRASE/LOCATION: [0,0]: S:(0)  in 1971 The name Norse selected in a campus-wide
contest that year is a fitting nickname for NKU 's students and athletic teams
since the University is the northern-most university in Kentucky and is located
at the highest spot in northern Kentucky
TV/TUITION: [0,0]: IGRADUATE: 1999
TV/TUITION: [0,1]: ???:
TV/TUITION: [0,2]: IUNDERGRADUATE:
TV/TUITION: [0,3]: ???:
TV/TUITION: [0,4]: IFULL_TIME:
TV/TUITION: [0,5]: ???:
TV/TUITION: [0,6]: ???:
TV/TUITION: [0,7]: ???:
TV/TUITION: [0,8]: ???:
TV/TUITION: [0,9]: ???:
PHRASE/CAMPUSES: [0,0]: S:(0)  in the US RIGHT-WALL
KV/ADDRESS: [0,0]: ADDRESS:812 Northern Kentucky University Highland Heights  KY
41099
TV/DEADLINES: [0,0]: DEADLINE_SPRING: 04/16
TV/DEADLINES: [0,1]: DEADLINE_SUMMER: 11/21
TV/DEADLINES: [0,2]: ???:
TV/DEADLINES: [0,3]: ???:
KV/TOEFL: [0,0]: TOEFL: 550
KV/UGRSTUDENTS:
```

MOREHEAD

```
KV/GRSTUDENTS:
KV/PRESIDENT:
KV/PRESIDENT: [0,0]: PRESIDENT: Ronald G. Eaglin
PHRASE/LOCATION:
TV/TUITION:
PHRASE/CAMPUSES:
KV/ADDRESS: [0,0]: ADDRESS:220 UPO 1228 Morehead  KY 40351
TV/DEADLINES:
KV/TOEFL:
KV/UGRSTUDENTS:
```

MIAMI

```
KV/GRSTUDENTS:
KV/PRESIDENT: [0,0]: PRESIDENT:Edward T. Foote
PHRASE/LOCATION: [0,0]: S:(0)  in Coral Gables Florida attracts students from
all 50 states and over 100 countries
TV/TUITION: [0,0]: IFULL_TIME: 4,424
TV/TUITION: [0,1]: IPART_TIME: 3,358
TV/TUITION: [0,2]: IUNDERGRADUATE: 2,928
TV/TUITION: [0,3]: IGRADUATE: 81.0
TV/TUITION: [0,4]: ???:
TV/TUITION: [0,5]: ???:
TV/TUITION: [0,6]: ???:
TV/TUITION: [0,7]: ???:
TV/TUITION: [0,8]: ???:
TV/TUITION: [0,9]: ???:
PHRASE/CAMPUSES: [0,0]: S:(1)  southwest of Coral Gables opened in 1986 on a
106-acre site for the purpose of conducting research and development projects
KV/ADDRESS: [0,0]: ADDRESS: 800 Welch Road Palo Alto California 94304
TV/DEADLINES: [0,0]: DEADLINE_FALL: 05/1
TV/DEADLINES: [0,1]: DEADLINE_SPRING: 11/1
TV/DEADLINES: [0,2]: DEADLINE_SUMMER: 03/1
TV/DEADLINES: [0,3]: ???:
KV/TOEFL: [0,0]: TOEFL: 550
KV/UGRSTUDENTS:
```

WKU

```
KV/GRSTUDENTS:
KV/PRESIDENT:
KV/PRESIDENT: [0,0]: PRESIDENT: Gary A. Ransdell
PHRASE/LOCATION: [0,0]: S:(0)  in Kentucky with a population of approximately
85,000 that features such attractions as the Kentucky Museum Corvette Museum
Hobson House and nearby Mammoth Cave National Park
TV/TUITION: [0,0]: IFULL_TIME: 1,195.00
TV/TUITION: [0,1]: IPART_TIME: 97.00
TV/TUITION: [0,2]: IUNDERGRADUATE: 94.00
TV/TUITION: [0,3]: ???:
TV/TUITION: [0,4]: ???:
TV/TUITION: [0,5]: ???:
TV/TUITION: [0,6]: ???:
TV/TUITION: [0,7]: ???:
TV/TUITION: [0,8]: ???:
TV/TUITION: [0,9]: ???:
PHRASE/CAMPUSES:
KV/ADDRESS: [0,0]: ADDRESS: 1 Big Red Way Bowling Green Kentucky 42101
TV/DEADLINES: [0,0]: DEADLINE_SUMMER: 04/01
TV/DEADLINES: [0,1]: DEADLINE_FALL: 11/1
TV/DEADLINES: [0,2]: DEADLINE_SPRING: 04/1
TV/DEADLINES: [0,3]: ???:
KV/TOEFL: [0,0]: TOEFL: 525
KV/UGRSTUDENTS:
```

REFERENCES

[1] Eric Brill and Raymond J. Moone; *An Overview of Empirical Natural Language Processing;* AI Magazine Winter 1997;

[2] Claire Cardie; *Empirical Methods in Information Extraction;* AI Magazine Winter 1997;

[3] Stephen Soderland, David Fisher, Wendy Lehnert; *Automatically Learned vs. Hand-crafted Text Analysis Rules;*

[4] Barry Glasgow, Alan Mandell, Dan Binney, Lila Ghemri, and David Fisher; *MITA - An Information-Extraction Approach to the Analysis of Free-Form Text in Life Insurance Applications;* AI Magazine Winter 1997;

[5] Linguistic Data Consortium; www.ldc.upenn.edu;

[6] http://www.cs.engr.uky.edu/~raphael/studentWork/scout.tar.gz

[7] Joseph D. Oldham; *Generating Documents by means of Computational Registers;* Ph.D. Thesis, University of Kentucky, 2000

[8] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller; *Introduction to WordNet: An On-line Lexical Database*; 1993

[9] George A. Miller; *Nouns in WordNet: A Lexical Inheritance System;*1993

[10] Christiane Fellbaum, Derek Gross, and Katherine Miller; *Adjectives in WordNet*; 1993

[11] Christiane Fellbaum; *English Verbs as a Semantic Net*; 1993

[12] Richard Beckwith, George A. Miller, Randee Tengi; *Design and Implementation of the WordNet Lexical Database and Searching Software*;1993

[13] Link Grammar Parser; http://www.link.cs.cmu.edu/link

[14] Stephen Soderland, David Fisher, Jonathan Aseltine, Wendy Lahnert; CRYSTAL: Inducing a Conceptual Dictionary; Proceedings of the IJCAI '95; 1995

[15] Stephen Soderland; Learning to Extract Text-based Information from the World Wide Web; Proceedings of the KDD-97; 1997

[16] TIPSTER

[17] Text REtrieval Conference (TREC) Home Page; http://trec.nist.gov/

[18] Tomek Strzalkowski, Fang Lin, Jose Perez-Carballo; Natural Language Information Retrieval (TREC-6 Report); Proceedings of the TREC-6; 1997

[19] SAIC Information Extraction; http://www.muc.saic.com/

[20] Nancy A. Chinchor; Overview of MUC-7/MET-2; Message Understanding Conference Proceedings; 1998

VITA

Janevski Angel was born on 09/25/1968 in Veles, Republic of Macedonia. He received his BS in Electrical Engineering at the University "Sv. Kiril i Metodij" in Skopje, Macedonia. Past professional engagements include Publishing House ‘M’ and ‘Pexim Computers’ in Skopje, Macedonia.

He is presently Member of Research Staff at ‘Philips Research’ in Briarcliff Manor, NY.

Angel Janevski