Vassilka Deltcheva
EARS – An Automatic E-mail Responding System
(Under the direction of Michael A. Covington)

EARS is an intelligent e-mail responding system, which allows its users to receive timely replies to their computer problems and questions. This project targets the most common computer support requests currently handled by the computer center at The University of Georgia (UGA). The users of the system are UGA students, faculty and staff and for their benefit the system is designed to communicate with them in English. The paper describes the natural language processing component of the system, designed to work in conjunction with e-mail handling tools implemented by Michael Covington. When fully developed, EARS will benefit the computer center staff, as well as its users in numerous ways. Most importantly, it will increase the quality of technical support service offered to the UGA community.

The main goal of this project was to show that simple keyword and template techniques, used in the implementation of the natural language component of EARS, can perform satisfactorily in the limited technical domain of computer related questions handled by the computer center. Although nobody claims that keywords and templates are realistic models of the way a human mind processes language, they offer a quick way to extract useful information from natural language input and are adequate for many practical purposes.

Index words: Text retrieval, Text categorization, E-mail responding, Automated helpdesk, Natural Language Understanding

EARS – An Automatic E-mail Responding System

by

Vassilka Deltcheva

B.A., University of Plovdiv Paisij Hilendarski, 1990

A Thesis Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Master of Science

Athens, Georgia

2001

EARS – An Automatic E-mail Responding System

by

Vassilka Deltcheva

Acknowledgments

Wow, this was fun. Now that I have completed this project, I look back and see numerous people that have contributed to it in various ways. Let me begin with my family.

Family is everything. I gained a new respect for them for the last two years while I was in school. My husband, Penio, deserves the most credit for supporting me in each and every possible way. I am thankful for my wonderful children, Alexander and Lily, who have always been a great source of inspiration for me. I owe a great deal to my parents for their unending support and devotion. Thank you, family. I couldn't have accomplished anything without your guidance and support.

Next, I would like to thank the AI faculty for sharing their knowledge and ideas about successful AI. In particular, my appreciation goes to Dr. Michael Covington for his supervisory role in my thesis work, Dr. Don Potter and Dr. Donald Nute for their useful advice and encouragement, and Dr. Elizabeth Preston and Dr. Bruce Britton for participating in my advisory committee.

I appreciate my friends and colleagues Angie Paul, Paul Brunk, Scott Bitterman, Anil Bahuman, Fred Maier, and Cartic Ramakrishnan for being there when I most needed them. In particular, I'd like to thank Brenda Keen, Paul Brunk and Fred Maier for helping me edit this and other important papers.

Last but not least, my thanks go to the University Computing and Network Services (UCNS) Help Desk for providing the assistantship, thus financing my study and giving me the opportunity to gain considerable knowledge and useful experience with computer support in general.

Table of Contents

Chapter 1

Introduction

Computers and computing are steadily becoming part of our daily life just as the radio and the TV set are. Unlike the radio and the TV set, computers are more interactive and in order to use them extensively people need prolonged and systematic training. The hardware and, even more noticeably, the software present a constant challenge for computer users. This is why users constantly need help.

Technical support comes in many forms and obviously includes communication between the user and the tech-support person. A client might contact the technical support team in person, via the telephone, via the World Wide Web (WWW), or by e-mail. University Computing and Networking Services (UCNS), the technical support institution at The University of Georgia (UGA), makes use of all of the above methods of communication. This paper specifically addresses e-mail interaction between the UCNS Help Desk and its clients.

E-mail popularity is constantly growing. This is because e-mail has proven to be a time efficient and reliable type of communication. Currently, members of the UGA community submit questions about any computer problem they have by e-mailing the UCNS Help Desk at helpdesk@uga.edu. The e-mail is captured and stored by a fairly sophisticated system called Remedy. Remedy is a case management system, used to keep records of each case referred to UCNS. It was created by the Remedy Corporation and adapted by UCNS in 1998. Not all e-mail questions can be answered by Help Desk personnel. If the Help Desk workers are not able to answer a question, either because of lack of knowledge or inaccessibility of a resource, they use Remedy to assign the request to one of the groups of secondary level UCNS consultants.

The UCNS Help Desk answers a variety of questions concerning Arches, the primary e-mail server. The questions are usually related to Arches space usage; connections to Arches using Webmail, Netscape, Simeon, Microsoft Outlook and Pine; connections to Oasis, WebCT, and Calendar; web page launching; network configurations; and printing. EARS, the prototype automatic e-mail responding system I have implemented, deals with the most frequently asked questions currently handled by the Help Desk staff.

1.1 Overview of the project

EARS is an intelligent e-mail responding system, allowing its users to receive timely responses via e-mail to many computer problems or questions normally answered by the UCNS Help Desk staff. EARS is built as an agent who is meant to replace a human consultant and not, like many other systems before, to replace the command structured interface with a natural language interface. Unlike an interface, which is a conduit through which information flows, an agent is a participant in a situation. In particular, EARS has explicit goals of its own, and is able to make obvious inferences and display judgment in making decisions. The users of the system are UGA students, faculty, and staff. It communicates with them in English.

The consultation task performed by EARS addresses fundamental problems in natural-language processing like planning, problem solving, and knowledge representation. The domain of the computer applications supported by the UCNS Help Desk is quite large and complex, and sending a specific response requires the system to understand the user's language, hypothesize the user's intentions, reason about the user's problem, access knowledge about the topic in question, and choose a reasonable response. In sum, virtually all the problems of language processing and reasoning arise in some fashion. The problems that EARS faces are highly challenging.

The UCNS Help Desk receives anywhere from 10 to 30 e-mail submissions per day. Each one of them has to be read, analyzed, and responded to by one of the Help Desk staff members. Considering that this process will likely be interrupted by a few phone calls, the time spent on a response averages 10 minutes. A large number of the questions answered by Help Desk staff are common ones. To cut down on response time, the Help Desk has developed a collection of canned responses (see samples in Appendix B), which are used to answer these high-volume, frequently asked questions. Before sending, each canned message is double-checked for accuracy and personalized by addressing the user (and optionally adding the name of the case worker). Next, a copy of the response is sent to the end-of-the-line user via Arches. The response is also stored in Remedy by copying and pasting.

EARS, which is a slightly permuted acronym for **A**utomatic **E**-mail **R**esponding **S**ystem, is designed to deal exactly with this high volume of common e-mail inquiries. The input to the system consists of informal e-messages, which are more or less spontaneously created and often contain large amounts of jargon, misspellings, and grammatical inaccuracy. The system processes these incoming messages using a combination of keyword and template matching techniques, classifies the message in one of the predefined categories, matches it to an appropriate response found on the server, and sends the response to the inquirer. If the system does not recognize a problem in a message, it classifies a message as "unrecognized." For testing purposes, the system is currently set to reply with an appropriate response to each message, whether the problem is recognized or not. The output is a response that contains a solution to the question or problem contained in the original message.

An automated responding system is deemed artificially intelligent for several reasons. It acts the way a knowledgeable human consultant would act when asked to respond correctly to an e-mail request. It understands and responds using natural language processing, which falls under the rubric of AI. Furthermore, it was implemented in Prolog, a declarative language, closely associated with the area of AI.

1.2 Goal of the project

The main goal of this project is to show that simple keyword and template matching techniques, used in the implementation of the natural language component of EARS, can perform satisfactorily in the limited technical domain of computer related questions coming to the UCNS Help Desk. A well performing e-mail response system will benefit the UCNS Help Desk staff, as well as the end-of-the-line users, in numerous ways:

- It will save UCNS worker-hours and decrease expenses allowing UCNS to hire fewer employees.
- It will increase the quality of service by reducing mistakes caused by tiredness or lack of concentration.
- It will relieve Help Desk employees from the burden of answering trivial questions.
- It will lessen the pressure of pending requests, assuring Help Desk personnel time for self-education and research.
- It will provide timely responses to clients. This will prove especially valuable in resolving time-dependent problems and providing immediate information, as illustrated by the example below.

Example 1:

```
CLIENT: I am trying to connect to OASIS but I can't. Please help!

RESPONSE: Currently there is no detected technical problem with OASIS.
          Due to a high volume of users at this time, the system might
          be overloaded. Please try again later.
```

1.3 Definition of terms

Natural language processing (NLP) is divided roughly into three subfields: information retrieval, information extraction, and text understanding. Even though NLP covers language understanding in general, this section concentrates on text processing.

1.3.1 Information Retrieval

Information retrieval (IR) research is concerned with developing algorithms and models for retrieving information from document repositories. IR is regarded as a natural subfield of NLP because it deals with a particular application of NLP. The classical problem in IR is the ad-hoc retrieval problem. In ad-hoc retrieval, the user enters a query describing the desired information and the system returns a list of documents. There are two main models. Exact match systems return documents that exactly satisfy some structured query expression (e.g., Boolean queries). This technique is widely used in commercial information systems expression handling. However, for large and heterogeneous document collections, the results either are empty or contain large sets of documents. This is why most recent work in IR has concentrated on systems which rank documents according to their estimated relevance to the query [14].

Some subfields of IR rely on a training corpus of documents that have been classified either as relevant or non-relevant to a particular query. In this way, IR is like an isolated case of what is called text categorization. In text categorization, one attempts to classify the topic or theme of a document. The purpose is to provide relevant information to a group of people with similar interests or to prepare for subsequent human or automatic processing.

Filtering and *routing* are two special instances of text categorization which attempt to group text in only two categories: relevant and non-relevant to a particular query or information need. The methods used depend upon each technique [14]. In routing, the desired output is a ranking of documents according to estimated relevance, similar to the ranking performed in an ad-hoc retrieval problem. The difference between routing and the ad-hoc method is that the training information, in the form of relevance labels, is available in routing but not in ad-hoc retrieval. In filtering, an estimation of relevance has to be made for each document, typically in the form of a probability estimate. Filtering is harder than routing because an absolute estimate is required, for instance, "document D1 is relevant," instead of a relative assessment, "document D1 is more relevant than D2." In many practical applications, including EARS, an absolute assessment of relevance for each document is required.[1]

1.3.2 Information extraction

Consider the information needs of an analyst who tracks changes in companies' management. A system is required to take input from news articles and extract information about any management succession events – the post, the company or companies concerned, the current and incoming managers, the reason the post is or

[1]While traditional IR research deals with text, retrieval of speech, images, and video is becoming increasingly common.

will be vacant, etc. This task might normally be undertaken by a news clipping service, where information retrieval techniques might be used to fetch relevant articles which would then be laboriously and expensively scanned by workers. Information extraction (IE) systems perform this task automatically. An information extraction system is a hierarchy of transducers or modules that at each step add structure and often lose information, hopefully irrelevant, by applying rules that are acquired manually and/or automatically [13] [9].

An example of an information extraction task is given below. It includes a sample text from the Wall Street Journal together with a template [10]. The task is to fill the template with information extracted from the text about succession events for that company. It is shown that there are six events in total, although complete information is not available for all of them.

Sample Text:

```
<HL> Marketing Brief:
<SO> WALL STREET JOURNAL (J), PAGE B5 </SO>
<CO> NYTA </CO>
<IN> MEDIA (MED), PUBLISHING (PUB) </IN>
<p>
New York Times Co. named Russell T. Lewis, 45, president and
general manager of its flagship New York Times newspaper,
responsible for all business-side activities. He was executive
vice president and deputy general manager. He succeeds Lance
R. Primis, who in September was named president and chief
operating officer of the parent.
</p>
```

This is how the given template is filled with information about successive events extracted from this sample text:

Sample Templates:

```
<ORGANIZATION-1>
NAME : "New York Times Co."
ORGANIZATION-2>
NAME : "New York Times"
<PERSON-1>
NAME : "Russell T. Lewis"
PERSON-2>
NAME : "Lance R. Primis"

<SUCCESSION-1>
ORGANIZATION : <ORGANIZATION-2>
POST : "president"
WHO_IS_IN : <PERSON-1>
WHO_IS_OUT : <PERSON-2>

<SUCCESSION-2>
ORGANIZATION : <ORGANIZATION-2>
POST : "general manager"
WHO_IS_IN : <PERSON-1>
WHO_IS_OUT : <PERSON-2>

<SUCCESSION-3>
ORGANIZATION : <ORGANIZATION-2>
POST : "executive vice president"
WHO_IS_IN :
WHO_IS_OUT : <PERSON-1>

<SUCCESSION-4>
ORGANIZATION : <ORGANIZATION-2>
POST : "deputy general manager"
WHO_IS_IN :
WHO_IS_OUT : <PERSON-1>

<SUCCESSION-5>
ORGANIZATION : <ORGANIZATION-1>
POST : "president"
WHO_IS_IN : <PERSON-2>
WHO_IS_OUT :

<SUCCESSION-6>
ORGANIZATION : <ORGANIZATION-1>
POST : "chief operating officer"
WHO_IS_IN : <PERSON-2>
WHO_IS_OUT :
```

1.3.3 Text understanding

While much research on the hard problem of in-depth story understanding by computer was performed starting in the 1970s, interest shifted in the 1990s to information extraction and word sense disambiguation. Given the heavy ambiguity of natural language, word sense disambiguation is essential to understanding.

One reason for the departure from text understanding is that, until recently, the conditions for studying the process of language understanding did not exist. Psychologists needed to develop and refine their research methods; furthermore, the computers and computational methods needed simply were not available [11]. Another reason could be explained as "the human factor." Some researchers abandoned story understanding because they got tired of the topic or ran out of ideas; some didn't spend enough time coding and debugging and thus were never able to realize their ideas; and some wished to develop immediate business applications. Funding pressures and changing fashions played a big role, too [15].

To study a process like reading comprehension, cognitive scientists typically begin by breaking it down into constituent parts, such as lexical access, syntactic processing, semantic analysis, retrieval from long term memory, inference generation, and knowledge acquisition. The key to success with this approach is that the several homunculi are "dumber" than the one they replaced. Each dumb homunculus can then be replaced, in turn, by even dumber homunculi, until the total set of homunculi is sufficiently simple that it can be replaced by a set of data, structures, and algorithms that can be executed on a computer [4].

Attempts to use computer models of human understanding to solve practical problems have been limited by at least two factors. First, far more attention has been paid to decomposing comprehension into its constituent parts and then implementing those parts than to reassembling them into a complete and useful model.

Attempts to move toward more complicated situations in reading and understanding by combining models of syntax and semantics, comprehension, and retrieval from long term memory are described in detail in Large and Wharton [12] and Cox and Ram [8]. A second factor that has limited the practical usefulness of computer models of human reading and understanding is the difficulty of knowledge representation. Most models depend on enormous amounts of manual knowledge engineering. A model that develops its own representations from experience would soon have a much larger knowledge base to draw on, making it much more useful. Learning models is the direction in which the field is headed [16].

Chapter 2

Related Projects

In this chapter, I review text processing projects similar to EARS in goals or strategies. Some of the systems described are working in commercial settings; some are just prototypes illustrating an ability to perform a particular task to some extent.

2.1 HelpRef: an e-mail referring system

The Computer Help Desk Referral System (HelpRef), developed by Xiaochang Yu at the Artificial Intelligence Center at UGA, is an information retrieval system with a menu-based interface and a database maintenance component [17]. What relates it to EARS is the fact that HelpRef was written for the UCNS Help Desk. The project applies logic programming as well as some information retrieval techniques. The task of the system is to assign a client's question to an appropriate UCNS consultant, once that question is categorized by one of the Help Desk workers. Remedy offers three criteria for categorizing a question. The first one specifies the group of consultants, the second one specifies the software affected, and the third criterion points to the problematic task within this product. HelpRef contains a database which captures the area of expertise of each one of the UCNS consultants. To reflect the fast changing and fast growing area of computing and networking services, the system updates and expands its database automatically.

The users of HelpRef are the Help Desk personnel, not the general public as is the case with EARS. Maybe this is one of the reasons HelpRef does not use natural language interface; instead, HelpRef uses a combination of fill-in-the-blanks, form-based, and menu-based interface. This works well in minimizing errors on the user site.

Initially, it seemed that HelpRef was a good idea that could save UCNS consultants time by eliminating irrelevant messages. However, it has never been implemented. A significant reason for this is the way UCNS consultant groups are organized. Each group consists of five or six members with overlapping areas of expertise and works as a self organizing unit. The set of rules within each group are simple: in there is a case assigned to a group, the first available consultant in the group reads the message and resolves the problem if he can. If he can't completely resolve the problem he may or may not write down a remark and leave the case for the next available consultant. Thus, a message assigned to a group and not to a specific person tends to be processed in a more time efficient manner.

2.2 Skip: a web based e-mail responding system

Skip is a WWW-based prototype of a system meant to handle the most common questions coming daily to the UCNS Help Desk [3]. It was developed by a team of graduate students at the University of Georgia. It is implemented partially in Perl and partially in GNU Prolog. The system allows the client to go to a web page and enter a query, expressed in English. The most interesting part of Skip albeit the most limited one, is the natural language processing. The text interpretation involves some text extraction techniques as well as some information retrieval techniques.

Text extraction is involved in the task of building a database of keywords and key phrases, which are the means by which the system analyzes each incoming message. The search for keywords has been done in two traditional ways. The authors acknowledge the "common sense" approach as a " superb way of finding starting points in the search for keywords, phrases, tags, and types of ellipses" [3]. In addition, an implemented word count program performs a single tokenization procedure on a list of ASCII characters obtained from the input and inserts the tokens into a dynamic knowledge base, which keeps a tally of the number of times each token occurs.

The authors have used several techniques in analyzing the text message itself. For the purposes of this project all ASCII characters delineated by white-space (all ASCII codes with values less then 33), end-of-line or end-of-file codes are considered tokens. *Simplification* is done in two steps. The first one involves simplifying single words or phrases. For example, `[what,',s]` is simplified to `[what,is]`. The next level of simplification is tagging. When a word is to be tagged, it has reached the most basic level of simplification, i.e., it has been further simplified to its most basic form. *Tagging* assigns to each of these recognized and simplified keywords additional information. The *the-problem-tag* should supply the information whether a message simply requests information or it reports a problem. *The-regard-tag* is assigned to the object of attention, the one that is causing the problem or the one the users requests information about. The third tag, the *user-input-tag* is gives the system more details about the addressed problem or request. For example, in the sentence "I can't get into WebCT or Arches, because I've forgotten my password." Skip assigns "get into" an access-problem-tag. It assigns "WebCT" and "Arches" regard-tags, and "password" a user-input-tag. The system will fail to respond to a message if the obligatory tags are not found.

The idea of assigning tags, seems quite efficient if there were effective rules by which tags were assigned. Unfortunately, there are not such rules described in the paper. The code itself, is not enough to determine those rules if there were any. The shortcoming, I see, is that only single words, and not a combination of words qualify for tags assignments. Keeping in mind the ambiguity of natural languages, one has to be aware that one word may be used in a different content carrying different information but at the same time qualifying for the same tag. For example `e-mail` may be used to address the problem itself or to provide details about the actual problem. To avoid this problem EARS uses templates and group of keywords. Another problem I foresee with SKIP comes from the fact that the subject of the messages are not tokenized. The subject line often contains important information which is not repeated in the body of the message.

Nevertheless, SKIP remains an ambitious project that has shown that very shallow text processing, when done in a clever way, may be useful in analyzing and understanding written text in an isolated domain.

2.3 ICC-Mail: Help Desk assistance system

ICC-Mail is an assistance system currently undergoing extensive tests at the Help Desk of AOL Bertelsmann Online GmbH & Co. KG [5]. ICC-Mail answers requests about the German version of AOL software. A Help Desk worker reads a request in the form of an e-mail and then he activates the ICC-Mail. The system displays a suggested answer to the request just read, and also shows alternative solutions. If the consultant finds the appropriate answer within these proposals, he inserts the associated text into the correct position of the e-mail response. If no proposal is found to be adequate, the ICC-Mail allows the consultant to manually select any text block from its database. The consultant can also alter or even create a new text block. In each case, the classified text together with the selected category is stored in the ICC-Mail database for use in future learning steps.

Currently the ICC-Mail database contains 47 categories covering 94% of all e-mails. Nevertheless, the system cannot classify documents containing multiple requests. Such messages are still treated manually. The ICC-Mail implementation is aiming to reduce the average time for a consultant to answer an e-mail message. Trials show that ICC-Mail reduces the time to complete the cycle of reading the mail, checking the proposed solution(s), choosing the appropriate solution(s), inserting additional text fragments, and sending the answer back to a bit more than a minute, which is half the time used before.

Because of the nature of the task, which is processing free text about arbitrary topics which are not stable, the developers have chosen to use a combination of shallow text processing (STP) and statistics-based machine learning techniques (SML) [5].

Shallow text processing gathers partial information about text, such as part of speech, word stems, negations, or sentence type. These types of information can be used to identify the linguistics properties of a large training set of categorized e-mails. Usually human experts define one or several template structures to be filled automatically by extracting information from the documents. Afterwards, the partially filled templates are classified by hand-made rules. STP tools yield high recall/precision (see the section on Criteria for Evaluation in Chapter 3). Nevertheless, the cost of analyzing and modeling the application domain, especially if it is to take into account the problem of changing categories, is very high. This, unfortunately, is also true for EARS.

On the other hand, SML promises low cost both in analyzing and modeling the application but at the expense of lower accuracy. This doesn't work for for EARS, since its goal is superior accuracy. Furthermore, SML is independent of the domain; thus it does not consider any domain specific knowledge. This is why I have chosen not to use any form of SML in EARS.

In EARS shallow parsing techniques were used to heuristically identify sentences containing relevant information. For example, in a manual analysis, linguistic constructions which are frequently used to express a problem were identified, and only significant words (keywords) were extracted and processed in MorphAna, the morphological analyzer of the system. All other words were ignored. If no results were found that way, MorphAna was applied again to the original text. The frequent constructions found in the manual analysis included negation of the sentence at the phrasal level, yes-no and wh-questions, and declaratives immediately preceding questions.

Would more linguistic preprocessing help? This is one question the authors have investigated further. They performed some tests which involved more general and sophisticated STP, such as chunk parsing (proposed by Steven Abney [1] in 1991). The results were very discouraging, leading to a significant decrease in performance compared to MorphAna. Most likely the noncompliance of e-mail texts to grammatical standards was the reason for the bad performance. The conclusion from those experiments is that task-specific linguistic processing is encouraging for this domain while general STP is not.

Chapter 3

Development and implementation of EARS

3.1 Automatic Processing of E-mail as Performed by EARS

The three main building blocks of EARS are the *E-receptionist*, the *E-technician*, and the *NLP-expert*. E-receptionist and E-technician were originally coded by Dr. Michael Covington at the AI Center at UGA and modified for this project as needed. The modifications consist in using the already developed tokenization tools not only for tokenizing the body of the message but also for tokenizing the subject of the message. Consequently, the two lists of tokens were appended and fed to the NLP-expert. Furthermore, before a response is sent, UNIX is commanded to append two additional files to the message. These files are the same for all e-mail responses. An introductory file is appended at the beginning of the message. This file acknowledges the automated response produced by EARS. Another file, called `signature`, is appended at the end of the response supplying contact information for the UCNS Help Desk. Nevertheless, the NLP-expert is the core of this project and I shall explain in details the ideas and techniques behind its implementation.

Let's follow the path of a sample e-mail sent to the Help Desk.

Example 1: Sample e-mail

```
Date: Wed, 24 May 2000 18:14:10 - 0700
From: Lily Penny <lpenny@bellsouth.net>
To: helpdesk@uga.edu
Subject: Need Summer Job!

Hi!
My name is Lily Penny. I am a student at UGA. I just finished my
freshman year. My major at this time is Computer Science. Please
email me if you have any summer job vacancies. My email address is
lpenny@bellsouth.net.

Thanks. Lily Penny
```

Given a piece of incoming e-mail on standard input, E-receptionist, a C++ program, creates a new directory and saves the incoming message in a file called `incoming`. Next, E-receptionist calls E-technician. E-technician is a SICStus Prolog program providing the tools for input, output, and tokenization of an e-mail message. It works in the following way: E-technician separates the header of the message from its body, using the fact that the body of a single e-mail begins where the header ends, precisely on the second new line after the header. Once the header is separated from the body the system proceeds with extracting the sender's e-mail address.

Determining the sender of an e-mail starts with a conservative search for the "Reply-To:" token. If, as is true in most cases, the "Reply-To:" token is not found, the search begins again, this time for the "From:" token. There is always a "From:" token in the header of an e-mail.

After the sender's e-mail address is extracted and the subject of the message is found, E-technician writes the header data back out as a header for the responding e-mail (see Example 2). The content of this header will change as the characteristics of the message, such as date, sender, and subject, change. Currently all parts of the system are located in my experimental e-mail account on the UNIX server at the

AI Center. In the example of the new header below, this is reflected in the “From:” field.

Example 2: New header for sample e-mail

```
Date: Sat, 22 Sep 2001 16:24:13 -0400 (EDT)
From: "Vassi Deltcheva (experimental) [MSAI]" <vassid2@aisun0.ai.uga.edu>
To: <lpenny@bellsouth.net>
Subject: UCNS Re: Need Summer Job!
```

Discovering the subject of the e-mail is fairly straightforward. The header is searched for the “Subject:” token. It is assumed that all tokens between the “Subject:” token and the end-of-line character constitute the subject. Once the subject is extracted, it is saved and used for creating the new header, as in Example 2 above, and also is tokenized and appended to the list of tokens obtained from the body of the message. For the purposes of this project, only entities like words and punctuation signs were counted as tokens. The tokenization process includes isolating groups of characters separated by spaces, tabs or end-of-line symbols and replacing capitals with lower case letters. The list of tokens created by E-technician from the Sample message looks like this:

Example 3: List of tokens obtained from the sample e-mail

```
[need,summer,job,!,,hi,!,my,name,is,lily,penny,.,i,am,a,student,at,
uga,.,i,just,finished,my,freshman,year,.,my,major,at,this,time,is,
computer,science,.,please,email,me,if,you,have,any,summer,job,vacancies,
.,my,email,address,is,lpenny,@,bellsouth,.,net,.,thanks,.,lily,penny,]
```

The NLP-expert is implemented in SICStus Prolog (see Appendix A). The input to the NLP-expert is a list of tokens, as in Example 3, derived from the subject and the body of the incoming e-message. The NLP-expert limits the length of the list of tokens to 150. The number 150 was chosen after carefully examining the maximum length of an e-mail received by the Help Desk and by considering the fact that the address of a forwarded message is considered a part of the body of the message, and as such becomes a subject of tokenization.

The NLP-expert scans the list of tokens for a combination of predefined keywords and templates of keywords trying to identify the category of request(s) the original message contain(s). In its current stage of development the NLP-expert has the knowledge and abilities to classify questions pertaining to: WebCT password, WebCT creation and maintaining, employment with UCNS, Arches disk quota, forwarding e-mail from Arches, information and issues on access to Arches and OASIS, user-id change requests, and conflicts between Webmail and particular Internet Service Providers.

As soon as the NLP-expert classifies a request, it sets a flag and appends the appropriate response to a file named `outgoing`. The usage of flags allows capturing and responding to multiple requests in a single e-message. Once the flag is set, the message is read again, this time scanned for a different problem. The process very much resembles the idea of using flags in the UGA registration system OASIS. A student may be able to register for classes before a flag appears in his records or after the flag is removed. However, the mere existence of a flag prevents registering. The flag resembles a new constraint, which is dynamically added to the existing ones. Unlike OASIS, once a flag has been added for a particular message it cannot be removed. The flag indicates that a particular request within that message has been responded to.

Prolog does the flagging by inserting a new fact in its database. This is done by using the command `assert(flag(Type_of_flag))` where `Type_of_flag` denotes a variable, which is replaced with a different category name depending on the segment in which it appears. In the example below, `Type_of_flag` is replaced by `employment` because this particular segment classifies questions in the `employment` category.

Example 4: The list of tokens obtained from the sample e-mail satisfies the second occurance of `identify_as`.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% discussing employment opportunities
% works great even with forwarded messages

identify_as(employment,Body):-
         m([employment],Body),

\+flag(employment),       % makes sure there is no 'employment' flag
assert(flag(employment)). % inserts 'employment' flag

identify_as(employment,Body):-
(m([job],Body);                                         %
 m([jobs],Body);                                        % FIRST GROUP
 m([position],Body);                                    % OF KEYWORDS
 m([positions],Body)),                                  %
(m([available],Body);                                   %
 m([inquire],Body); m([inquiring],Body);                % SECOND GROUP
 m([seek],Body); m([seeking],Body);                     % OF KEYWORDS
 m([vacancy],Body); m([vacancies],Body);                % AND TEMPLATES
 m([possibility],Body); m([possibilities],Body);        %
 m([part,time],Body); m([full,time],Body);              %
 m([opportunity],Body); m([opportunities],Body)),       %
\+flag(employment),        % makes sure there is no 'employment' flag
assert(flag(employment)).  % inserts 'employment' flag
```

As we can see, the above selection contains two `identify_as` clauses. A message will be classified in the ‘employment questions’ category if either one of these clauses is satisfied. A necessary condition for either one of the clauses to succeed is the nonexistence of a flag. Furthermore, to satisfy the first clause the word *employment* must be found as a token in the input list. The second clause will be satisfied if there is a match to at least one keyword from the first group of keywords and at least one keyword or template from the second group.

After a request is classified to a category, it is assigned this category’s name. The category name is used as a link to a predefined response. Below is a sample of a response, which is the matching response to the sample e-mail in Example 1:

Example 5: Response to the sample e-mail

```
This is an automated response sent to you by EARS, the Help Desk email
handler. If this message does not answer your question or you have more
questions to ask, please use the contact information below to call us.

                            ********

Thank you for your inquiry about employment with us! If you are
interested in full time employment with Help Desk or any other group
in UCNS, you will have to fill in an application with UGA Human
Resources. Information about their location and office hours can
be found at:

               http://www.busfin.uga.edu/employment/

If you are a student and are interested in a part-time employment with
the Help Desk or any of the groups working for UCNS, please send your
resume (no attachments, please) and your academic schedule for the
semester to:

                          helpdesk@uga.edu

If you are interested in part-time employment with any of the UCNS computer
labs you must contact the supervisor of the particular lab. Please find
more information about the UCNS labs at:

                     http://www.uga.edu/ucns/sites/

Let us know if we can be of further assistance.

*************************************************************************
* University Computing and Networking Services, University of Georgia  *
*       Computer Services Annex, Athens, Georgia 30602-1911            *
*                            (706) 542-3106                            *
*                        Internet: helpdesk@uga.edu                    *
*************************************************************************
```

After the file `outgoing` is created, the control is given to E-receptionist. E-receptionist sends it via Sendmail to the sender and cleans up the directory. When tested on the server which hosts EARS, the above described process takes less then 30 seconds.

3.2 NLP-expert methodology

Three years ago when my parents came to the United States, they were already in their fifties. Never before had they been exposed to the English language. Although both were experienced drivers, they found it extremely difficult to pass the test for a Georgia driving license. After all, the test was available only in English. By learning a few English words, common to the limited domain of the traffic rules (as well as important negation words), they were able to pass the test without actually knowing the language. Even now, after three years of studying English, they wonder how they passed the language barrier then. Although they did not know the language, they did understand the questions, if "understanding" means responding appropriately to the information content. A simple strategy consisting mainly of the use of keywords and templates in combination had worked for them.

Nobody claims that keyword and template systems are a realistic model of the way the human mind processes language. They are, however, a quick way to extract useful information from natural language input, adequate for many practical user applications. Since one of the problems in NLP comes from the ambiguity of natural languages, limiting the domain in which a language is used will surely lead to better performance. This is what I have counted on when choosing to work on the collection of e-mails in the limited technical domain of questions coming daily to the UCNS Help Desk.

3.2.1 Template techniques

Along with the wording, template techniques preserve the order in which words appear in a sentence. Thus, templates capture semantic composition, the combining of the meanings of words to form the meanings of phrases or sentences.

Why not then use only templates to analyze e-mail messages? Unfortunately (or may be not) language is arbitrary. Everyone uses his own language style and vocabulary. In natural languages, there is an incredible number of ways to say the same thing. People express themselves in their own unique way. Syntax, or sentence construction, is the lowest level at which human language is constantly creative. Templates are easily thrown off by slight variations of wording; consequently they can be applied with success only rarely. In an attempt to compensate for that, I have used templates that contain 'wild card' slots. For example, the template `[am,_,_,instructor]` will allow the system to capture phrases like "I am a WebCT instructor" or "I am a new instructor."

Most template systems contain simplification and translation rules. Simplification rules are rules that discard unnecessary words and make equivalent words look alike. Translation rules map a template to a desirable match. The purpose of the simplification rules is to reduce the numbers of the translation rules. However, there is a trade-off; if one puts in too many simplification rules, he may loose a semantic distinction that he might want to preserve later on when more rules are added. This is the reason why the NLP-expert consists only of translation rules. Since it is common for more simplification rules to get dropped as the system becomes more complicated, I have decided not to use them in EARS at all. My decision has also been influenced by the fact that English, compared to other natural languages, has a fairly simple morphology. Thus, simplification rules will not be effective; they will require as much work as they might save.

3.2.2 Keyword tecninque

Keyword analysis has been reinvented several times, and keyword systems have had a long and successful history. Unlike template systems, they are not thrown off by slight variations in wording; unrecognized words are simply skipped. Keyword systems are useful in any situation where the input is known to contain certain kinds of information and other information can simply be ignored.

3.2.3 Building groups of keywords and templates

In an attempt to extend the capabilities of the above mentioned techniques, the NLP-expert in EARS uses groups containing both keywords and templates. This means that some of the keywords are just single words while others are constructed as templates. The advantage of templates over single words is that, although more specific, they preserve the order of the words and thus are able to distinguish similar information.

The NLP-expert relies on three main groups of keywords and templates. In order to explain the ideas and principles behind building these groups, I will introduce the reader to the Propositional Representation theory.

There are various notations that represent the meaningful structure of sentences. One of them is known as 'propositional representation'. Propositional representation preserves the meaning of a text which remains after the perceptual details have been abstracted away. Propositional analysis represents meaning of complex sentences in terms of simple, abstract propositional units [2]. In this way, regardless of the form in which a request is stated, capturing its meaning will be equally successful.

The concept of a 'proposition', borrowed from logic and linguistics, is central to such analysis. A 'proposition' is the smallest unit of knowledge that can stand as a separate assertion; that is, the smallest unit which can be judged true or false. A sentence may be constructed of several propositions. Here is an example: "For the last three days I have been unable to read my Arches e-mail on the Web." The information conveyed in this sentence can be communicated by the following simple sentences:

A. I can't read my Arches e-mail.

B. I am attempting to read my e-mail through the Web.

C. The problem persisted for the last three days.

Each simple sentence expresses a primitive unit of meaning. They all closely correspond to the propositions that underly the meaning of the complex sentence. Notice that the complex sentence above may be, in fact, the whole email message. It is also possible that the above information may be presented as two less complicated complex sentences.

Let's imagine our complex sentence as an unit of four chunks: subject, object, action, and details. If one of them is not true, the complex sentence cannot be true either. For example, in the sentence discussed above, the subject is "I"; the objects, "Arches" and "e-mail"; the action, "can't read"; and the details, "for three days" and "Web". The NLP-expert tries to match each complex sentence using three (out of four) groups of keywords and templates. Apparently, there is no need to define a group for the subject since it is assumed that the subject is always the sender of the e-mail.

The first group of keywords and templates addresses the object; what is it that concerns the user? Is it his Arches e-mail, or disk quota on the Arches server, or what? The first group consists of multiple entries, keywords and templates, commonly used to address the specific topic of the category. Thus, the object group for different categories of messages has different content. For example, "Arches" and "e-mail" are used many times interchangeably, and either one or the other might appear in the message if the user is concerned with his Arches e-mail. It is also possible for both of them to appear at the same time. Here, the Prolog implementation makes extensive use of the logic operator 'OR' which allows a clause to succeed if either one or both of the specified keywords are matched.

The second group of keywords and templates includes possible entries for the action. More often then not, action is represented by a collection of words ordered in templates. This is dictated by the fact that the inability to perform an action is usually described by using negative verbs and the presence of other action verbs may mislead the NLP-expert. For example, a message might read: “I am able to connect to Arches just fine and I can view the list of messages in my mailbox. However, I can’t see the content of a single message.” Here E-technician will tokenize the last part of the sentence as `[...can,',t,see,the,content,of,a,single,message]`. So the existence of the template `[t,see]` as one of the choices in the action group will ensure the correct match in this category.

The third group of keywords and templates makes use of the details in a non-traditional way. Since sending an e-mail is considered an informal mode of communication, the messages do not follow a formal or predefined structure. Many times the provided information is incomplete, and there are numerous reasons for that. For example, the user might not specify the mail client he is using, simply because he may not be aware of the diversity of available mail clients. In other instances, the user might not know that a specific fact could provide important information to the UCNS consultant, or he simply may not have the time or patience to provide this information. Consequently, the system should not necessarily try to look for a specific details in order to classify a request. However, the system should take advantage of such details if they were found. Keywords and templates comprising the group of details are called ‘negative.’

To explain the nature and justify the function of negative keywords and templates, I am going to introduce the reader to two categories of messages found in the Help Desk e-mail collection. ‘General info messages’ are messages that request info on whether something is possible and how it can be done. The question: “How do I connect to Arches from home?” will be classified as a general info message. On the other hand, the ‘problem reporting messages’ will generally address a specific issue or difficulty, as in the following example: “I tried to connect to Arches from home . . . but I could not.”

Apparently, the two messages belong to similar categories and contain the same keywords: “Arches” and “connect.” This information alone, however, will not allow the NLP-expert to classify the messages correctly. This necessitates the use of negative keywords. Negative keywords generally describe common problems that would be mentioned in a problem reporting message. For example, the user may report not being able to connect to Arches because he has forgotten his password or user-id. Thus, in the example above, the negative keywords could be “password,” “quota,” or “user-id.” A match to one or more of them will classify a message as an ‘Arches problem connection’ message, and no match will classify a message as an ‘Arches info connection’ message.

Chapter 4

Evaluation of EARS

4.1 Model evaluation

The success of any natural-language driven software depends on whether the software can do what the user wants, not just on how well it understands English or any other natural language [6].

4.1.1 Tool Selection

The NLP-expert was implemented in Prolog. Why choose Prolog? Prolog stands for PROgramming in LOGic and is used widely in AI research. The declarative nature of Prolog, its pattern-matching abilities, and the fact that it can modify itself make it very useful in building and modifying large complex data structures [7]. Natural language, with its enormous richness, is a quite suitable field for employing Prolog.

Isn't Perl better suited to handle pattern matching? At first sight it seems that Perl, with its capabilities to match up patterns easily, is more suited for this project than Prolog. However, if we go deeper and consider the internal representation of the data structures in Prolog, we will find that an atom is not represented as a character string; it is represented only as a location in a symbol table. That is, the computer stores only one copy of each atom no matter how many times it occurs in the program. All occurrences of the atom in the program are then replaced by pointers to its location in the symbol table. In the same manner, a function is represented as a linked tree made from pointers to its substructures and to entries in the symbol table. A very useful kind of structure in Prolog is the list. Lists have a big advantage over multi-argument structures. The advantage is that you can process a list without knowing how many elements it has. This is because each list is represented in Prolog as an element plus another list or [ ], the empty list. The internal representation of lists, even though similar to the representation of structures, is further simplified. Thus, a Prolog program is time and memory efficient.

4.1.2 Criteria for Evaluation

The principal valve measures for information retrieval tasks are *Recall* and *Precision*. Recall is the number of documents the system retrieved correctly divided by the number of all available documents. It measures how complete the system is in retrieving of relevant documents. Precision is the number of documents the system got right divided by the number of all documents the system retrieved. It measures the system's correctness or accuracy. For example, if there are 100 available documents and the system retrieves 80 of them but only 60 are correctly retrieved, its recall is $\frac{60}{100} = 60\%$ and its precision is $\frac{60}{80} = 75\%$.

The main idea behind the development of EARS was increasing the quality of service provided by the UCNS Help Desk. EARS performs the closed cycle of receiving the initial messages, recognizing the problems, and providing the solutions to the original inquirers with no human control involved at any point of the process. Based upon the purposes for which EARS was designed, the system was evaluated mainly for precision. Since the main goal was customer satisfaction, less than 100% accuracy would not be acceptable. Thus, EARS was developed with more than a superior human performance in mind.

4.2 Data collection and recording

The initial set of documents used for development of the system contained more than 300 actual messages sent to the Help Desk in the period between 1997 (when Remedy was adopted by UCNS) and February 2001. A necessary criterion a message had to satisfy, in order to be chosen for the collection, was that it had to address frequently asked questions. The selected e-mails were manually divided into 39 custom categories, depending on the type of request/question they contained. Fifteen of those categories contained a satisfactory number of documents (8-12) to perform an in-depth analysis. Twelve out of the fifteen currently have been implemented and tested.

In designing the custom-made categories, I have considered two criteria. The first one naturally assigned different questions to different categories. This natural tendency, however, would overwhelm the system in distinguishing similar messages, since similar questions use similar keywords. The second criterion involved a consideration of how detailed and diverse a response should be. A lengthy response covering multiple areas is, in general, not appealing to users. Even employing tricks like including links to web pages did not completely solve this problem. Thus, I had to define categories of messages, which were truly distinguishable and at the same time were not too broad to be addressed by a single e-mail.

With these different categories in mind, I proceeded with designing the response messages. I began with the canned messages created by Help Desk full-time staff. These messages contained detailed guidance on what can be done if a particular problem is encountered. These messages, however, are modified by the staff every time they are sent, depending on the particular problem they are addressing. Thus, I had to redesign these messages and make them suitable to answer more than one question within a category.

4.2.1 Test results

The system was tested using all relevant messages retained in Remedy between February 2001 and August 2001. In addition, I have asked my Help Desk coworkers and my family to send "made up" e-mails concerning one or more specified problems to EARS and forward me the results. As the testing results show, the system's average precision is 88%, with best performance achieved in responding to "Arches forward" and "WebCT" questions.

Test results yielded 45% recall. This is not surprising. The template technique, which proved so valuable and contributed significantly to the high percentage of precision, affected the recall of the system in a negative way, making it less effective. One way to improve this aspect of performance is to add more templates in order to capture more phrases or parts of phrases humans use to express themselves. Limiting the number of templates by substituting them with keywords would also increase the recall, but it would harm the precision, which is not desirable.

4.2.2 Limitations and weaknesses

One fact that limits the system's effectiveness is that no technique for capturing spelling mistakes is yet employed. Misspelled words are simply skipped as no match for them is found. E-mail requests are written most of the time under stress and in a hurry. This is why developing a dictionary of commonly misspelled words will significantly improve the recall and precision of EARS.

One serious weakness of the system will show up in the following situation. Consider the case of an e-mail containing two requests, only one of which is recognized by the system. In this case, EARS will send a response addressing only the request it recognizes. The other request will be discarded and the case will be considered resolved. This will force the user to send another e-mail asking for help, thus significantly hurting the quality of service offered by the UCNS Help Desk. One way such a problem can be resolved is to give a human consultant a chance to look at the responses suggested by the system and add or modify them accordingly, much in the way the message classification system in the German AOL Call Center does.

Another problem concerns users who provide their Arches e-mail address as contact information while reporting a problem with their Arches account. Even though the user may give different contact information in the body of the message, the system will send a response to the e-mail account that cannot be accessed at the time. This is caused by the way EARS is set to retrieve the sender's address. The system does not possess the capability to conclude that, since the problem reported concerns Arches e-mail, it should not send information to the Arches account of the user. One way to fix that problem is to drop any knowledge the system has about fixing inaccessibility issues and leave these messages for human intervention.

4.3 Conclusion and future directions

EARS is an automatic e-mail responding system, designed to handle the most common requests arriving to the UCNS Help Desk. The system performs the closed cycle of receiving initial messages, recognizing the problems addressed in them, and providing solutions to the users with no human control involved at any point in the process. This is an attempt to build a system that substitutes for a human consultant in the limited technical domain of commonly asked questions handled daily by the Help Desk staff.

The final goal behind building EARS was to increase the quality of service offered by the UCNS Help Desk. This could only happen if EARS, in its full development, offered 100% accuracy. As the test results show, the prototype of the system currently yields 88% accuracy, which is not enough to fulfill the initial purpose of EARS. Nevertheless, its performance shouldn't be overlooked. I have shown that a system that uses simple keyword and template techniques can perform satisfactorily. Furthermore, new ideas have arisen on how to use EARS more effectively.

The system and human can complement each other and become a powerful team, accomplishing the task better and faster. As a result, after each collaboration human and machine will acquire improved knowledge, because they will share knowledge. I believe that the main purpose of EARS should not be automatically responding to clients but tutoring an inexperienced Help Desk worker while helping him do his job.

In my experience, I have found that the two main factors behind successful tutoring are a comprehensive explanation on the tutor's side and motivation to learn on the student's side. In considering the team of human consultant and EARS, we should not worry about the explanation part. The response messages which the system will use as a tutorial were written for a computer-illiterate user. As such, they should be comprehensive enough for the new Help Desk worker. On the other hand, an incoming message containing a problem will motivate the Help Desk worker enough to look for its solution.

By the time the Help Desk worker reads a message, EARS will classify the message in a category and offer a suggested e-mail response. Using common sense, the consultant can judge whether the system has functioned correctly. If EARS was correct, the consultant may proceed in one of two possible ways. If he is not familiar with the answer to this particular question or he wishes to verify the information offered, he may read the response and then send it to the user. This is how the human will benefit from the system's knowledge. In a case where the consultant is already familiar with the problem, he can send the response immediately. In both cases, however, EARS will find and conveniently place a response in the body of the message, saving time and effort otherwise spent manually retrieving, cutting, and pasting.

What will happen if EARS fails to categorize the message correctly, thus suggesting a wrong response? As I mentioned above, along with the suggested response, EARS will indicate the category under which the original e-mail was classified. This will allow the human consultant to judge the system's performance. If EARS fails, the consultants can choose to manually browse through the responses and find the most suitable one, then edit and send it to the user. The original message can be scanned again for specific keywords and typical templates can be extracted. This may be done manually or implemented automatically. Thus adding a learning component to the system will increase its knowledge base and its capabilities.

I believe that such a tutorial will constantly increase and update the knowledge of the Help Desk workers, ensuring consistency of their knowledge in an environment with high personnel turnover and a fast changing range of technical problems.

Bibliography

[1] Abney, Steven. "Parsing by Chunks." In *Principle-Based Parsing*, edited by Robert Berwick, Steven Abney, and Carol Tenny, 257-278. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1991.

[2] Anderson, John. "Language Comprehension." Chapter 12 in *Cognitive Psychology and its Implementations*, 388-421. New York, New York: Worth Publishers, 2000.

[3] Bitterman, Scott, and Aaron Dallas. *Web-Based Automatic Reply System: A Perl and Prolog Implementation.* Term paper for CSCI 8570 class, The University of Georgia, Athens, 2000.

[4] Brooks, Rodney. "Intelligence without Representation." *Artificial Intelligence* 47 (1991): 139-151.

[5] Busemann, Stephan, Sven Schmeier, and Roman G. Arens. *Message Classification in the Call Center.* The German Research Center for Artificial Intelligence GmbH, Saarbrücken, 1999. Available at http://www.dfki.de/ busemann/papers.html.

[6] Covington, Michael A. *Natural Language Processing for Prolog Programmers.* Engelwood Cliffs, New Jersey: Prentice Hall, 1994.

[7] Covington, Michael A., Donald Nute, and André Vellino. *Prolog Programming in Depth.* Upper Saddle River, New Jersey: Prentice Hall, 1997.

[8] Cox, Michael T., and Ashley Ram. "On the Intersection of Story Understanding and Learning." In *Understanding Language Understanding: Computational Models of Reading*, edited by A. Ram and K. Moorman, 397-434. Cambridge, Massachusetts: The MIT Press, 1999.

[9] Cunningham, Hamish, and Saliha Azzam. *Information Extraction.* University of Sheffield, Western Bank, UK, 1995. Available at http://www.dcs.shef.ac.uk/research/groups/nlp/groups/ie.html.

[10] Hobbs, Jerry. *Generic Information Extraction System.* Artificial Intelligence Center SRI International, Menlo Park, California, 2001. Available at http://www-nlpir.nist.gov/related_projects/tipster/gen_ie.htm.

[11] Kintsch, Walter. Foreword to *Understanding Language Understanding: Computational Models of Reading*, edited by Ashwin Ram and Kenneth Moorman. Cambridge, Massachusetts: The MIT Press, 1999.

[12] Lange, Trent E. and Charles M. Wharton. "Retrieval from Episodic Memory by Inferencing and Disambiguation." In *Understanding Language Understanding: Computational Models of Reading*, edited by Ashwin Ram and Kenneth Moorman, 107-181. Cambridge, Massachusetts: The MIT Press, 1999.

[13] Lehnert, Wendy. *Information Extraction.* University of Massachusetts, Amherst, 1999. Available at http://www-nlp.cs.umass.edu/nlpie.html

[14] Manning, Christopher D. and Heinrich Schütze. *Foundations of Statistical Natural Language Processing.* Cambridge, Massachusetts: The MIT Press, 1999.

[15] Mueller, Erik T. *Prospects for In-depth Story Understanding by Computer.* IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 1999. Available at http://cogprints.soton.ac.uk/documents/disk0/00/00/05/54/.

[16] SRI International. *Text Understanding.* Artificial Intelligence Center SRI International, Menlo Park, California, 1998. Available at www.arpa.mil/ito/psum/1998/B363-0.html.

[17] Yu, Xiaochang. *A Computer Help Desk Referral System.* Master's thesis, Artificial Intelligence Center, The University of Georgia, Athens, 1992.

Appendix A

Prolog Code for the NLP-expert

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%****************************%%%
%%%*******responses.pl*********%%%
%%%***  by Vassi Deltcheva****%%%%
%%%****************************%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


:- use_module(library(system)).
:- dynamic(flag/1).

start(Body):-
        clear_message_flag,
        respond_all(Body).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% clears all dynamically inserted facts: \verb"flag(_)"
%
clear_message_flag:-retract(flag(_)),
                    fail.
clear_message_flag.



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% respond_all(+Body).
% makes sure all possible questions in a message are analyzed
%
respond_all(Body):-
        identify_as(Subj,Body),
        send_respond(Subj),
        fail.

respond_all(_). % succeeds without further action
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% succeeds if the elements of the first list (in the same order) are
% found in the second one
% m([a,b],[a,c,a,b]). -- succeeds
% m([a,b],[a,c,b]). -- fails
%
m(L1,L2):-the_same(L1,L2).
m(L1,[_|L2]):-m(L1,L2).

the_same([F|L1],[F|L2]):-the_same(L1,L2).
the_same([],_).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% identify_as(-Subject,+TokenizedMessage).
% recognizes the empty message and pastes a canned answer to outgoing
% this case will be hardly used because most of the senders have
% signatures at the bottom of their email messages, which are read as
% a part of the message
%
identify_as(empty,Body):-
        length(Body,L),
        L<3,
        !,                      % do not check the rest of identify_as/2
        assert(flag(empty)).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% criteria for recognizing WebCT questions
% recognizes well ‘student question’ from an ‘instructor question’
%
identify_as(webct_student_idpass,Body):-
        (m([webct],Body); m([web-ct],Body); m([web,ct],Body)),
        (m([password],Body); m([how,do,i,log,in],Body);
         m([not,log,in],Body); m([t,log,in],Body)),
        \+ m([my,student],Body),
        \+ m([am,teaching],Body),
        \+ m([am,instructor],Body),
        \+ m([am,_,instructor],Body),
        \+ m([am,_,teacher],Body),
        \+flag(webct_student_idpass),
        assert(flag(webct_student_idpass)).
```

```
%%%%%%55%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% discussing employment opportunities
% works great even with forwarded messages
%
identify_as(employment,Body):-
         m([employment],Body),
         \+flag(employment),
         assert(flag(employment)).

identify_as(employment,Body):-
        (m([job],Body);
         m([jobs],Body);
         m([position],Body);
         m([positions],Body)),
        (m([available],Body);
         m([inquire],Body); m([inquiring],Body);
         m([seek],Body); m([seeking],Body);
         m([vacancy],Body); m([vacancies],Body);
         m([possibility],Body); m([possibilities],Body);
         m([part,time],Body); m([full,time],Body);
         m([opportunity],Body); m([opportunities],Body)),
         \+flag(employment),
         assert(flag(employment)).


&&&&&&%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% discussing why still getting the Arches disk quota warning
%
identify_as(arches_quota,Body):-
        m([disk],Body),
        m([low],Body),
        m([space],Body),
        exclude_quota(Body), % make sure no unwanted words are in the message
        \+flag(arches_quota),
        assert(flag(arches_quota)).

identify_as(arches_quota,Body):-
       (m([taking,up],Body);  m([take,up],Body);  m([taken,up],Body);
        m([filling,up],Body); m([fills,up],Body); m([filled,up],Body);
        m([filling],Body);    m([fills],Body);    m([filled],Body);
        m([occupying],Body);  m([occupies],Body); m([occupied],Body);
        m([absorbing],Body);  m([absorbs],Body);  m([absorbed],Body);
        m([are,consuming],Body);  m([consumes],Body); m([consumed],Body);
        m([reaching],Body);   m([reaches],Body);  m([reached],Body)),
       (m([space],Body);
        m([quota],Body);
        m([_,consuming,all,the,space],Body)),
        exclude_quota(Body),
        \+flag(arches_quota),
        assert(flag(arches_quota)).

identify_as(arches_quota,Body):-
       (m([in,response],Body);
        m([responded,by],Body);
        m([responding,by],Body);
        m([answering,by],Body);
        m([answered,by],Body)),
       (m([deleted],Body);
        m([deleting],Body);
        m([erased],Body);
        m([erasing],Body);
        m([remove],Body);
        m([removed],Body)),
       (m([message],Body);
        m([messages],Body);
        m([mail],Body);
        m([email],Body);
        m([e-mail],Body);
        m([item],Body);
        m([items],Body);
        m([note],Body);
        m([notes],Body);
        m([letter],Body);
        m([letters],Body);
        m([entry],Body);
        m([entries],Body);
        m([thing],Body);
        m([things],Body)),
        exclude_quota(Body),
       \+flag(arches_quota),
       assert(flag(arches_quota)).

exclude_quota(Body):-
       \+m([increase],Body),
       \+m([larger,space],Body),
       \+m([request],Body).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% discussing Arches account durability
%
```

```
identify_as(arch_durability,Body):-
       (m([arches],Body);
        m([email],Body);
        m([e-mail],Body);
        m([mail],Body);
        m([account],Body)),
       (m([no,longer,work],Body);
        m([no,longer,exists],Body);
        m([clean,out],Body);
        m([cleaned,out],Body);
        m([clear,out],Body);
        m([cleared,out],Body);
        m([removed],Body);
        m([remove],Body);
        m([erased],Body);
        m([erase],Body);
        m([shut,off],Body);
        m([how,long],Body);
        m([close],Body);
        m([closed],Body);
        m([keep],Body);
        m([kept],Body)),
       \+m([low,space],Body),
       \+m([space,low],Body),
       \+flag(arch_durability),
       assert(flag(arch_durability)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Arches mail forwarding
%
identify_as(arch_forwarding,Body):-
        (m([arches],Body);
         m([email],Body);
         m([e-mail],Body);
         m([mail],Body);
         m([account],Body);
         m([my,files],Body)),
        (m([forward],Body);
         m([forwarded],Body);
         m([forwarding],Body);
         m([collect],Body);
         m([collected],Body);
         m([collecting],Body);
         m([transfer],Body);
         m([transferred],Body);
         m([transferring],Body);
         m([sent,to],Body);
         m([sent,to],Body)),
        \+m([encountered,problem],Body),
        \+m([have,difficulties],Body),
        \+m([having,difficulties],Body),
        \+m([have,difficulty],Body),
        \+m([having,difficulty],Body),
        \+m([have,trouble],Body),
        \+m([having,trouble],Body),
        \+m([error],Body),
        \+m([have,a,problem],Body),
        \+m([problem,persists],Body),
        \+flag(arch_forwarding),
        assert(flag(arch_forwarding)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Arches user-id changes requests
%
identify_as(arch_userid_change,Body):-
        (m([email,address],Body);
         m([e-mail,address],Body);
         m([user-id],Body); m([userid],Body);
         m([my,address],Body);
         m([login,name],Body)),
         m([change],Body),
        \+flag(arch_userid_change),
        assert(flag(arch_userid_change)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Access to OASIS
%
identify_as(oasis_logon,Body):-
       (m([oasis,connection],Body);
        m([dial,into,oasis],Body);
        m([log,on,in,oasis],Body);
        m([logon,in,oasis],Body);
        m([get,on,oasis],Body);
        m([connect,to,oasis],Body);
        m([check,my,grades],Body);
        m([checking,grades],Body);
        m([check,my,final,grades],Body);
        m([schedule,of,classes],Body);
        m([grades,on,oasis],Body);
        m([grades,on,arches],Body);
        m([register],Body);
```

```
         m([registering],Body)),
        \+m([webct],Body),
        \+m([web-ct],Body),
        \+m([ethernet,card],Body),
        \+m([register,_,computer],Body),   % avoids ‘Restech’ questions
        \+m([register,_,_,computer],Body), % avoids ‘Restech’ questions
        \+flag(oasis_logon),
        assert(flag(oasis_logon)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Catches the bug in Arches Web-Mail. Returns "arch.nowebmail"
%
identify_as(arch_nowebmail,Body):-
         (m([arches,mail],Body);
          m([arches,account],Body);
          m([arches,acct],Body);
          m([email],Body);
          m([e-mail],Body);
          m([mail],Body);
          m([my,account],Body);
          m([account,on,the,server],Body);
          m([webmail],Body);
          m([mail,folders,on,the,server],Body)),
         (m([difficulty],Body);
          m([difficulties],Body);
          m([not,been,able,to],Body);
          m([t,been,able,to],Body);
          m([not,able,to],Body);
          m([be,able,to,access],Body);
          m([unable,to,check],Body);
          m([cannot,check],Body);
          m([not,check],Body);
          m([t,check],Body);
          m([not,opening],Body);
          m([t,open],Body);
          m([not,open],Body);
          m([cannot,open],Body);
          m([have,been,trying],Body);
          m([try,to,logon],Body);
          m([t,logon],Body);
          m([cannot,logon],Body);
          m([not,logon],Body);
          m([not,let,me,access],Body);
          m([can,not,access],Body);
          m([cannot,access],Body);
          m([t,access],Body);
          m([through,arches,website],Body);
          m([no,response],Body);
          m([t,get,a,response],Body);
          m([never,connects],Body);
          m([cannot,connect],Body);
          m([t,connect],Body);
          m([can,not,connect],Body);
          m([cannot,come,up],Body);
          m([can,not,come,up],Body);
          m([t,come,up],Body);
          m([cannot,go],Body);
          m([can,not,go],Body);
          m([t,go],Body);
          m([screen,hangs,up],Body);
          m([have,a,problem],Body)),
         \+m([forward],Body),
         \+m([password],Body),
         \+m([subject],Body),
         \+flag(arch_nowebmail),
         assert(flag(arch_nowebmail)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% one can click on the subject but can't view the message using Web-Mail
%
identify_as(webmail_gray_line,Body):-
        (m([message],Body);
         m([messages],Body)),
        (m([not,been,able,to,read],Body);
         m([not,been,able,to,Var,read],Body);
         m([not,been,able,to,see],Body);
         m([not,been,able,to,Var,see],Body);
         m([not,been,able,to,view],Body);
         m([not,been,able,to,Var,view],Body);
         m([t,been,able,to,read],Body);
         m([t,been,able,to,Var,read],Body);
         m([t,been,able,to,see],Body);
         m([t,been,able,to,Var,see],Body);
         m([t,been,able,to,view],Body);
         m([t,been,able,to,Var,view],Body);
         m([not,able,to,read],Body);
         m([not,able,to,Var,read],Body);
         m([not,able,to,see],Body);
         m([not,able,to,Var,see],Body);
         m([not,able,to,view],Body);
         m([not,able,to,Var,view],Body);
         m([unable,to,check],Body);
         m([cannot,check],Body);
         m([not,check],Body);
         m([t,check],Body);
         m([not,opening],Body);
         m([t,open],Body);
         m([not,open],Body);
         m([cannot,open],Body);
         m([have,been,trying],Body);
         m([not,let,me,access],Body);
         m([can,not,access],Body);
         m([cannot,access],Body);
         m([t,access],Body);
         m([through,arches,website],Body);
         m([cannot,come,up],Body);
         m([can,not,come,up],Body);
         m([t,come,up],Body);
         m([cannot,go],Body);
         m([t,appear],Body);
         m([cannot,appear],Body);
         m([not,appear],Body);
         m([t,read],Body);
         m([t,Var,read],Body);
         m([cannot,read],Body);
         m([not,read],Body);
         m([t,see],Body);
         m([t,Var,see],Body);
         m([cannot,see],Body);
         m([not,see],Body);
         m([t,view],Body);
         m([cannot,view],Body);
         m([not,view],Body);
         m([never,displayed],Body);
         m([not,taken,to,a,page,on,which,i,can,see],Body)),
        (m([click,on,the,message],Body);
         m([click,on,a,message],Body);
         m([click,on,a,mail,message],Body);
         m([clicking,on,the,message],Body);
         m([can,see,what,has,been,sent],Body);
         m([select,a,message],Body);
         m([selects,a,message],Body);
         m([click,on,the,subject],Body)),
        \+flag(webmail_gray_line),
        assert(flag(webmail_gray_line)).
% to handle messages from a third person the code will have to
% include the same phrases with verbs in 3 person singular
% identify_as(S,[click,on,the,subject,of,the,message,doesn,',t,appear]).
% identify_as(S,[i,can,see,what,has,been,sent,not,able,to,actually,
% read,or,open,messages]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% connect to Arches (3 possible ways)
%
identify_as(arch_connect,Body):-
        m([arches],Body),
       (m([check],Body);
        m([checking],Body)),
       (m([mail],Body);
        m([email],Body);
        m([e-mail],Body)),
       \+flag(arch_connect),
       assert(flag(arch_connect)).


identify_as(arch_connect,Body):-
         m([arches],Body),
        (m([t,know,how,to],Body);
         m([do,not,know,how,to],Body);
         m([not,able,to],Body);
         m([need,to],Body);
         m([needs,to],Body)),
        (m([logging],Body);
         m([logon],Body);
         m([sending],Body);
         m([send],Body);
         m([accessing],Body);
         m([access],Body);
         m([checking],Body);
         m([check],Body);
         m([connect],Body);
         m([connecting],Body)),
        \+m([password],Body),
        \+m([quota],Body),
        \+m([disk,space],Body),
        \+flag(arch_connect),
        assert(flag(arch_connect)).

identify_as(unrecognized,_):-
        \+flag(_).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% send_respond(+Subj) is called with the \verb"Subj" found
% it identifies the right respond and sends it to outgoing
%
send_respond(empty):-
        % this clause is for easy testing within emacs
        write('**This message contains insufficient information.**'),
        system('cat </home/student/vassid2/can/empty>> outgoing').

send_respond(webct_student_idpass):-
       write('**This is a webct question.**'),
       system('cat </home/student/vassid2/can/webct.student.idpass>>outgoing').

send_respond(employment):-
         write('**This message discusses employment.**'),
         system('cat </home/student/vassid2/can/employment>> outgoing').

send_respond(arches_quota):-
        write('**Arches quota message**'),
        system('cat </home/student/vassid2/can/arch.quota>> outgoing').

send_respond(arch_durability):-
       write('**Arches durability message**'),
       system('cat </home/student/vassid2/can/arch.durability>> outgoing').

send_respond(arch_forwarding):-
        write('**sent:How to set Arches to forward messages**'),
        system('cat </home/student/vassid2/can/arch.forwarding>> outgoing').

send_respond(arch_userid_change):-
        write('**arch.userid.change message**'),
        system('cat </home/student/vassid2/can/arch.userid.change>> outgoing').

send_respond(oasis_logon):-
        write('**oasis.logon message**'),
        system('cat </home/student/vassid2/can/oasis.logon>> outgoing').

send_respond(arch_nowebmail):-
        write('**arch.nowebmail message chosen**'),
        system('cat </home/student/vassid2/can/arch.nowebmail>> outgoing').

send_respond(webmail_gray_line):-
        write('**webmail.gray.line selected**'),
        system('cat </home/student/vassid2/can/webmail.gray.line>> outgoing').

send_respond(arch_connect):-
```

```
        write('**arch.connect message**'),
        system('cat </home/student/vassid2/can/arch.connect>> outgoing').

send_respond(unrecognized):-
        write('**unrecognized**'),
        system('cat </home/student/vassid2/can/unrecognized>> outgoing').
```

Appendix B

Sample Responses

Sample 1: This sample response below is sent to users who wish to know how to check their Arches e-mail.

```
Dear Arches user,

To access Arches you must have Internet access. Currently there are
three different ways you can connect to Arches:

1. You can access Arches on the Web at http://www.arches.uga.edu/.
To be able to check your mail all you have to do is provide your user
name and password.

2. To access your e-mail on Arches you can also configure Netscape
Messenger or Outlook Express. For information about downloading and
configuring an IMAP client, please go to:
                http://www.arches.uga.edu/imap.html

3. You may also manage your Arches e-mail through Pine. If you have
"telnet" already installed on your computer, simply go to the Start
menu and click on "Run". Type "telnet arches.uga.edu" (without the
quotes) and press "OK". You will be prompted to enter your Arches
user name and password.

If you are using a Macintosh you can obtain "telnet" software at:
http://www.ncsa.uiuc.edu/SDG/Software/MacTelnet/MacTelnet.Home.html

Users without Internet access cannot currently access Arches.
```

Sample 2: The sample message below will be sent to an user who wants to know how to forward his Arches e-mail to a different account.

```
Dear Arches user,

Arches provides an easy-to-use feature that allows you to forward
your e-mail to a different e-mail account. In order to forward
your Arches e-mail, you need to go to www.arches.uga.edu. Do not
login. Instead, choose the option "Manage Your Account" and then
"Mail Forwarding". Next, you need to login and then, in the
specified field, add the forwarding address.

If, however, you prefer to telnet to Arches, you may forward your
e-mail by using the Arches Main Menu. Please choose the following
options in each consecutive screen:

   [4] Account Management (Password, Printing, Disk Space, ...)

   [4] Mail forwarding

Type in your forwarding address and press "Enter".

Important:

** The mail previously accumulated on Arches will not be forwarded
to the newly specified e-mail address. Only mail that comes after
the configuration takes place will be forwarded to the new address.

** The process of forwarding an address book can be very different
depending on the software you are using to read your mail. Please
call for assistance.
```

Sample 3: This is a sample of a response EARS will send to users who want to know how long they may keep their Arches accounts.

```
Dear Arches user,

All faculty, staff, and students are eligible for an Arches account
free of charge while their affiliation with UGA lasts, except for
retirees. Retired faculty and staff can keep their accounts as long
as they wish. Students may keep their accounts up to one year after
they graduate/drop school. Ex-employees’ accounts are closed by a
request from the department in which they used to work.

You don’t need to undertake any action if you have changed your
status with UGA. The update will be done automatically.
```