INTELLIGENT INTERFACE DESIGN FOR A QUESTION ANSWERING SYSTEM

By

NICHOLAS ANTONIO

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2001

To the innocent victims of terrorism.

ACKNOWLEDGMENTS

I wish to thank my advisor, Dr. Douglas Dankel, for his extensive assistance and support on this thesis and all aspects of my college life at the University of Florida. Despite his hectic schedule, he never failed to respond to my many requests. I also wish to thank Dr. Joseph Wilson and Dr. Paul Fishwick for serving on my supervisory committee. Finally, I wish to thank my mother, Elpida.

TABLE OF CONTENTS

page

LIST OF FIGURES

Figure Page

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

INTELLIGENT INTERFACE DESIGN
FOR A QUESTION ANSWERING SYSTEM

By

Nicholas Antonio

December 2001

Chairman: Douglas D. Dankel II
Major Department: Computer and Information Science and Engineering

This thesis describes the design and implementation of an intelligent interface for a question answering system. The system accepts natural language questions and provides natural language answers within the domain of the graduate Web pages. The system has four components, the intelligent interface, the parser, the query generator and the XML knowledge base.

The system sends the user's question to the parser, which in turn passes its results on to the query generator. The latter retrieves a part of the knowledge base and writes an XML file that the interface then processes. It also presents the generated answer to the user. Processing the XML file includes natural language generation and information filtering.

The interface, implemented using Macromedia Flash, runs on any Flash-enabled web browser. The system works by reading and parsing the XML file created by the query generator. It then generates natural language content using the template-based

realization approach. It filters out information by creating links from high-level concepts to specific details, as well as external links to the Internet. It finally presents the answer to the user who in turn can follow the aforementioned links or ask another question.

This interface has been designed for question answering and can be used to view answers generated by the query generator, but can also be used to view any XML file of similar specification. Thus, the intelligent interface is a solution to the generic problem of presenting filtered information to users, whether it is part of a question answering system or not.

CHAPTER 1
THE PROBLEM

The last decade of the previous millennium has seen a revolution unlike any other witnessed by mankind. The information revolution has started a transformation of the world in a speed unrivaled in the three thousand years of recorded history.

The explosion of the Internet has brought access to vast amounts of knowledge, which were previously only present in some of the world's finest libraries, to everybody's home. As with most prior technologies, people have developed a love-hate relationship with the Internet. They love access to the information, but they hate the painstaking process of searching through huge quantities of irrelevant material with the hope of finding the specific information for which they are looking.

Historical Attempts to Solve the Problem

In the infancy of the Internet and more specifically of the World Wide Web, people could only reach information by knowing its web address a priori. The solution to this problem came in the form of search engines (e.g., Altavista, Lycos, Yahoo, etc.).

Search engines attempted to solve the problem of finding relevant information using keywords entered by the user. These engines use the provided keywords to search through indexes they have built to return a list of web pages containing those keywords. At this point, the agonizing procedure begins for the user—manually searching through the returned pages in the hope of finding the information he desires.

It became evident that people wanted a better solution. More specifically, the need for direct question answering arose. Users wanted, and still want, not only to be able to ask questions in natural language but to have their questions precisely answered as well.

A second wave of attempts (i.e., Ask Jeeves! [1]) resulted in search engines that have a sizeable amount of prebuilt natural language questions. If the user is lucky, the question he has asked is present in the system and he is redirected to a web page providing an answer. However, if the user is looking for something other than the local weather forecast, chances are that the system just returns a list of web pages which must be manually searched, placing the user back to square one!

An Ideal Solution to the Problem

What is an ideal solution to this problem? Is there only one ideal solution, or are multiple solutions available? Does a solution satisfying the majority of Internet users exist?

The only fact that is beyond doubt is that people desire different solutions to this problem. A solution satisfying a power user will probably leave the novice stranded, while the novice user's solution will leave the power user unsatisfied. People have been using natural language for communication purposes for thousands of years. Among human societies, fairly universal rules have evolved for the question answering procedure. When you ask someone for the time, you expect a direct response. Something along the lines of "it's four o' clock" or "sorry, I don't have the time." And with the exception of running into someone having a bad day, that is what you usually get as a response. What you definitely do not want and do not get is an answer like "the time is probably available at one of the following places."

Therefore, it is safe to argue that the main guidelines for an ideal solution satisfying the majority of the population is:

To be able to take a question from a user on any subject and use the Web as a knowledge base to construct and provide a "good" direct answer.

This solution can be explained in more detail. First, it refers to the ability to take natural language questions as input. Second, it considers the lack of restrictions on what the system can answer. Ideally the system should be able to answer any question as long as the information needed to provide the answer resides somewhere on the Internet. This raises the issue of using the Web as a knowledge base. Most of the information on the Web is in Hyper Text Markup Language (HTML) format with quite a few documents also being in simple text, Adobe Acrobat format, Postscript format, and recently Extended Markup Language (XML). This means that the system should be able to handle all these file types. Finally, the most vague part of the description is the construction and provision of a *good* direct answer. The term *good* refers to an answer that is *intelligent* and in a presentable format where intelligent is judged content-wise and presentable is judged appearance-wise.

A user friendly and aesthetically satisfying web page that does not answer the question is probably better than one that does, but leaves the user helpless in trying to locate the answer among the *garbage* (i.e., the results returned from the current search engines). In addition, presentable format refers to a page that fits on the user's screen without the need to scroll.

Description of Selected Problem Sub-Area

The ideal solution to the question-answering problem presents two major obstacles that at this time are too complicated to tackle. The first is the lack of domain restrictions or, more simply, the ability to answer questions on any subject. The second is the use of the Internet as a huge knowledge base.

No Domain Restrictions

In human-to-human conversations whenever a question is asked, the domain is usually implied by the surrounding circumstances. When a student is asking his advisor about his thesis *defense*, when a basketball player is asking his coach about *defense*, and when two politicians are discussing defense there is no confusion about the terms: *thesis defense*, *basketball defense*, and *military defense*, respectively. People can usually determine the domain and, hence, the meaning of any ambiguous words.

However, when a user fires a question at a computer system with the word defense the latter has no *contextual knowledge* to determine the domain the user is discussing. Natural language and English words, specifically, can have a multitude of different meanings rendering the task of disambiguation almost impossible at the present time. Therefore, any realistic attempt at a question answering system must be domain specific.

Using the Internet as a Knowledge Base

Knowledge bases are usually in some hierarchical or relational form. This allows easy and efficient querying. When dealing with text, numerous problems arise including the fact that all efficiency is lost since the document must be searched in a serial manner. With the unprecedented amount of information available on the Internet, inefficiency is

not an option. Most importantly, it becomes extremely difficult for queries to return good results, since the most they can do is return a paragraph of text within which the selected keywords are found. This severely limits the ways in which an answer can be formulated for the user.

Ideally, the pages holding the necessary information can be transformed into a knowledge base from which the answer can then be formulated requiring a module capable of turning text into knowledge. This is a major issue in computer science, one that the field of "*textual knowledge acquisition*" [2] hopes to solve. Until this solution is developed, we must find intermediate solutions like manually encoding knowledge bases for small domains.

Description of an Intermediate Solution

Since an ideal solution is not achievable at this time, we must look for intermediate solutions that can be used as stepping-stones towards the ideal one. A description of an intermediate solution to question answering, on which our research group has been working, is:

> *To take a question from a user about the information in the Computer and Information Science and Engineering (CISE) department's graduate Web pages and use a specially formatted version of these pages as a knowledge base to construct and provide a "good" direct answer to this question.*

There are two key differences between this system and the ideal one. The domain is now restricted to the CISE graduate Web pages and the Web pages are not used as is, but have instead been manually converted into a hierarchical XML version.

Chapter 2 provides a quick overview of our question answering system and how the various modules of the system should operate. The system consists of four modules: the *parser*, the *query generator*, the *XML knowledge base*, and the *intelligent interface*, which is the subject of this thesis. Chapter 3 covers the background material that is required for a more in-depth description of the intelligent interface module covered in Chapter 4. Chapter 5 ends the thesis with conclusions and suggested extensions for future research with the ideal solution always being the ultimate goal.

CHAPTER 2
SYSTEM OVERVIEW

The question answering system described in this thesis consists of four modules. Three of these are active while one is inactive. The active modules are the *parser*, the *query generator*, and the *intelligent interface*. All three of these take some input and generate some output, hence the term active. The fourth module is the XML knowledge base (XMLKB), which neither takes input nor generates output, hence the term inactive. This chapter provides a high level description of this question answering system.

Flow of Operation

The only module visible to the user is the interface. As far as he is concerned, he types in questions and receives answers from this interface. Behind the scenes, the interface starts the parser and sends the question asked by the user to it. The parser then generates a specifically modified XML parse tree of the question. The interface then starts the query generator, which reads the parse tree, creates a query, executes it on the XMLKB, and retrieves a part of the XMLKB as the answer. The interface reads this XML structure and in turn generates a page as the answer to the user's question. This process can be seen more clearly in Figure 2.1.

The XML Knowledge Base

The heart and soul of the system is the knowledge base. This is the repository of all the information that the user desires. As shown in Figure 2.1, a manual transformation

of the current graduate web pages from HTML to XML has been performed. This comes in contrast with what the ideal system should do, which is to perform this operation automatically.

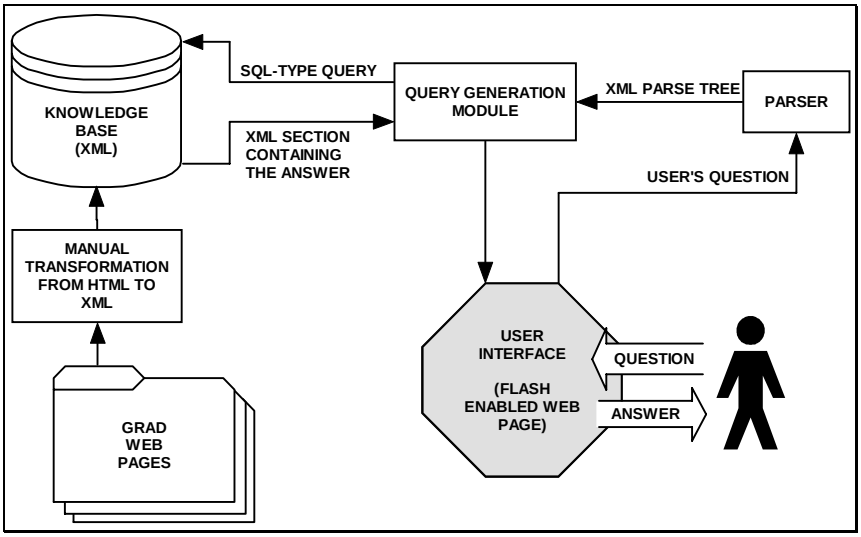

Figure 2.1: How the system works

The manual transformation of the web pages to an agreed upon hierarchical structure, enables us to use this structure as a knowledge base that can be queried. It also serves as an indication of what the desired output of an automatic transformation module should look like.

The knowledge base consists of a set of *sub-area files*, *the directory (meta) file*, and the *DTD file*. The sub-area files contain all the information present on the graduate web pages, while the other two files serve as internal meta-knowledge for the system. Figure 2.2 shows a sample XML fragment detailing how sub-area files are structured, Figure 2.3 shows a sample fragment of the XMLKB DTD and Figure 2.4 shows a sample fragment of the XMLKB directory file.

```
<?xml version='1.0' encoding="UTF-8" standalone="no"?>
<GRAD_PAGES lastRevised="09/25/01">
  <OVERVIEW>
    <CW>overview</CW>
    <CONTENT>Overview of the information in the Graduate Brochure</CONTENT>
    <TEXT>This document describes the degree requirements for students entering the
Graduate Program in Computer and Information Science and Engineering (CISE) with
the intention of receiving the Master's, Engineer, or Ph.D. Degree. It is intended to be
used in conjunction with the University of Florida's Graduate Catalog. While this guide is
intended to be self-contained and accurate, the CISE Department reserves the right to
correct errors when found, without further notice to students. It is the student's
responsibility to ensure that they are in compliance with both Departmental and
University requirements.</TEXT>
    <ROOT_TEXT>DOCUMENT DESCRIBE DEGREE REQUIREMENT STUDENT
ENTERING GRADUATE PROGRAM COMPUTER INFORMATION SCIENCE
ENGINEERING CISE INTENTION RECEIVING MASTER ENGINEER PHD
INTEND CONJUNCTION UNIVERSITY FLORIDA CATALOG GUIDE SELF-
CONTAINED ACCURATE DEPARTMENT RESERVES RIGHT CORRECT ERROR
FOUND WITHOUT FURTHER NOTICE 'S RESPONSIBILITY ENSURE
COMPLIANCE DEPARTMENTAL</ROOT_TEXT>
    <LINK>
      <TEXT>CISE</TEXT>
      <TARGET>http://www.cise.ufl.edu</TARGET>
    </LINK>
    <LINK>
      <TEXT>Master's Degree</TEXT>
      <TARGET>http://www.cise.ufl.edu/~ddd/grad/ms.html</TARGET>
    </LINK>
  </OVERVIEW>
</GRAD_PAGES>
```

Figure 2.2: Sample sub-area XML file fragment from the knowledge base

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- ********** CORE_COURSES element ********** -->
<!ELEMENT CORE_COURSES
(CW,CONTENT,TEXT?,ROOT_TEXT?,MASTERS_CORE,PHD_CORE)>
<!ELEMENT MASTERS_CORE
(CW,CONTENT,TEXT?,ROOT_TEXT?,LINK?,COURSE*)>
<!ELEMENT PHD_CORE (CW,CONTENT,TEXT?,ROOT_TEXT?,COURSE*)>
<!ELEMENT COURSE
(CW,CONTENT,TEXT,ROOT_TEXT?,LINK?,NUMBER?,DESCRIPTION?,PREREQ?
)>
<!ELEMENT NUMBER (CW,CONTENT,TEXT,ROOT_TEXT?)>
<!ELEMENT DESCRIPTION (CW,CONTENT,TEXT,ROOT_TEXT)>
<!ELEMENT PREREQ (CW,CONTENT,TEXT,ROOT_TEXT,LINK*)>

<!-- ********** OVERVIEW element ********** -->
<!ELEMENT OVERVIEW (CW,CONTENT,TEXT,ROOT_TEXT,LINK*)>

<!-- ********** GEN_INFO element ********** -->
<!ELEMENT GEN_INFO
(CW,CONTENT,TEXT?,ROOT_TEXT?,DEGREES_OFFERED,STUDY_AREAS,CO
MPUTING_RESOURCES)>
<!ELEMENT DEGREES_OFFERED
(CW,CONTENT,TEXT?,ROOT_TEXT?,LINK*,DEGREE*)>
<!ELEMENT DEGREE (CW,CONTENT,TEXT,ROOT_TEXT?)>
<!ELEMENT STUDY_AREAS
(CW,CONTENT,TEXT?,ROOT_TEXT?,STUDY_AREA*)>
<!ELEMENT STUDY_AREA
(CW,CONTENT,TEXT,ROOT_TEXT?,DESCRIPTION)>
<!ELEMENT COMPUTING_RESOURCES
(CW,CONTENT,TEXT?,ROOT_TEXT?,RESOURCE*)>
<!ELEMENT RESOURCE (CW,CONTENT,TEXT,ROOT_TEXT)>
```

Figure 2.3: Sample fragment of the XMLKB DTD

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<!DOCTYPE GRAD_PAGES SYSTEM "mainDTD.dtd">

<GRAD_PAGES lastRevised="08/29/01">
  <DIRECTORY domain="www.cise.ufl.edu/~ddd/grad">
    <LISTING file="core_courses.xml">
      <CW>core course master master's degree ph.d. doctor philosophy phd ms m.s.
      </CW>
      <CONTENT>CISE Graduate Program core courses</CONTENT>
    </LISTING>
    <LISTING file="overview.xml">
      <CW>overview summary</CW>
      <CONTENT>Overview of the information in the Graduate Brochure</CONTENT>
    </LISTING>
    <LISTING file="gen_info.xml">
      <CW>general information graduate degree offer study area specialization compute
computing resource</CW>
      <CONTENT>General information about the CISE graduate program</CONTENT>
    </LISTING>
    <LISTING file="admission.xml">
      <CW>application apply admission information material mail office cise computer
science department submit submission process</CW>
      <CONTENT>Information on admission to the CISE graduate
program</CONTENT>
    </LISTING>
    <LISTING file="financial.xml">
      <CW>financial assistance option assistantship fellowship tuition payment fee
responsibility certification</CW>
      <CONTENT>Information on available financial assistance</CONTENT>
    </LISTING>
```

Figure 2.4: Sample fragment of XMKB directory file

The Parser

After the user submits a question, the interface sends that question to the parser. Parsers are an integral part of every natural language processing system. Their job is to identify and tag every word with an appropriate part of speech and generate a tree structure that correctly represents the syntax of the sentence.

Many parsers exist, each of which operates slightly differently, but all follow the same principles. For our system, a parser that can generate XML parse trees is needed. Since many very good parsers already exist, we decided to modify an existing one to suit our needs rather than to write one from scratch.

After a review of various parsers, the “Link Grammar Parser” from Carnegie Mellon University was chosen. In the developers’ own words, “the Link Grammar Parser is a syntactic parser of English, based on link grammar, an original theory of English syntax.” [3] Given a sentence, the system assigns a syntactic structure consisting of a set of labeled links connecting pairs of words.

The parser has a dictionary of approximately 60000 word forms. It covers a wide variety of syntactic constructions, including many that are rare and idiomatic. The parser is robust; skipping over portions of the sentence that it cannot understand to assign some structure to the rest of the sentence. It is able to handle unknown vocabulary making intelligent guesses about the syntactic categories of unknown words from context and spelling. It has knowledge of capitalization, numerical expressions, and a variety of punctuation symbols.

```
<SENTENCE>
<NOUNPHRASE>
   <PRONOUN string="what">
      <ROOT>what</ROOT>
      <NUMBER>indeterminate</NUMBER>
   </PRONOUN>
</NOUNPHRASE>
   <SENTENCE string="are the core classes?">
      <VERBPHRASE>
         <VERB string="are">
            <ROOT>be</ROOT>
            <TENSE>present</TENSE>
           <NUMBER>plural</NUMBER>
                  </VERB>
         <NOUNPHRASE>
            <NOUNPHRASE>
               <ARTICLE string="the">
                  <ROOT>the</ROOT>
                  <type>definite</type>
                                 </ARTICLE>
               <ADJECTIVE string="core">
                   <ROOT>core</ROOT>
                                    </ADJECTIVE>
                <NOUN string="classes">
                   <ROOT>class</ROOT>
                   <NUMBER>plural</NUMBER>
                                    </NOUN>
                               </NOUNPHRASE>
                        </NOUNPHRASE>
                  </VERBPHRASE>
   </SENTENCE>
</SENTENCE>
```

Figure 2.5: Desired modified output of parser on input “what are the core classes?”

Figure 2.5 shows a sample of the desired modified output of this parser on input "what are the core classes?" Modifications include an XML parse tree as output and extra tags defining a word's root, tense, and number.

```
<?xml version='1.0'?>
<RESULT number='1'>
<QUERY string="How can I get the Degree of Engineer?">
</QUERY>
<ANSWER type="E">
   <ENGINEER>
      <CONTENT>Information on the Degree of Engineer</CONTENT>
         <TEXT>To be admitted to the Degree of Engineer Program students must have
completed a Master's Degree in Engineering. To earn the Degree of Engineer, the
students must obtain at least a 3.0 GPA in at least 30 graduate credit hours beyond the
Master's Degree, within five calendar years of enrollment. These credit hours may
include CIS 6972, Research for Engineer's Thesis, hours. The supervisory committee for
this degree consists of at least three members of the graduate faculty. Two members are
from CISE and at least one from a supporting department. The options for the Degree of
Engineer regarding program completion are the same as for the Master's Degree: there
are both thesis and non- thesis options. NOTE: Credits counted toward the Degree of
Engineer are not credited toward any other degree - including the Ph.D. if a Ph.D. is
subsequently pursued.</TEXT>
      <LINK>
         <TEXT>CIS 6972</TEXT>
            <TARGET>http://www.cise.ufl.edu/~ddd/grad/grad_courses.html#CIS6972
            </TARGET>
      </LINK>
   </ENGINEER>
</ANSWER>
</RESULT>
```

Figure 2.6: Sample XML file written by the query generator

The Query Generator

Once the parser finishes its work, the query generator looks at the parse tree and creates an SQL-type query, which in turn is applied to the knowledge base. If the query is successful, it returns a section of the XMLKB containing the answer. The query generator removes unnecessary tags, used in the querying process, and writes the result to a file. It also writes some extra information, such as how many answers were retrieved and whether there was a partial or exact match.

An exact match means that the query was successful in returning results containing all the keywords the user used in his initial question to the system. A partial match means that the query was unsuccessful in finding an answer containing all of the keywords, but was successful in retrieving an answer containing some of them.

Figure 2.6 shows the contents of a sample file written by the query generator. The user's initial question was "How can I get the degree of Engineer?" The system retrieves one answer, as shown by <RESULT number="1"> and has an exact match, as shown by <ANSWER type="E">.

The Intelligent Interface

The importance of graphical user interfaces has grown steadily during the last years and will probably continue to do so. The days of command-line prompts are gone since the latter are being replaced by "windowed" environments. Great applications are ones that the users enjoy looking at and interacting with. No matter how many features or how efficient an application is, if the user does not like what he sees or how he interacts with the system, it all goes to waste, since he is probably not going to use the software.

In this spirit, a great deal of emphasis has been placed on how the retrieved information is displayed to the user. The primary goal of a question answering system

and our system in particular, is the elimination, as much as possible, of the time spent searching through web pages for the sought after data. Thus, a system presenting a very long answer defeats the system's purpose, which is to provide a quick, clearly presented answer to a user's question. At the same time, we have no appreciation of a system that provides a very terse yes or no answer. The system should instead try to generate a page of useful information for the user.

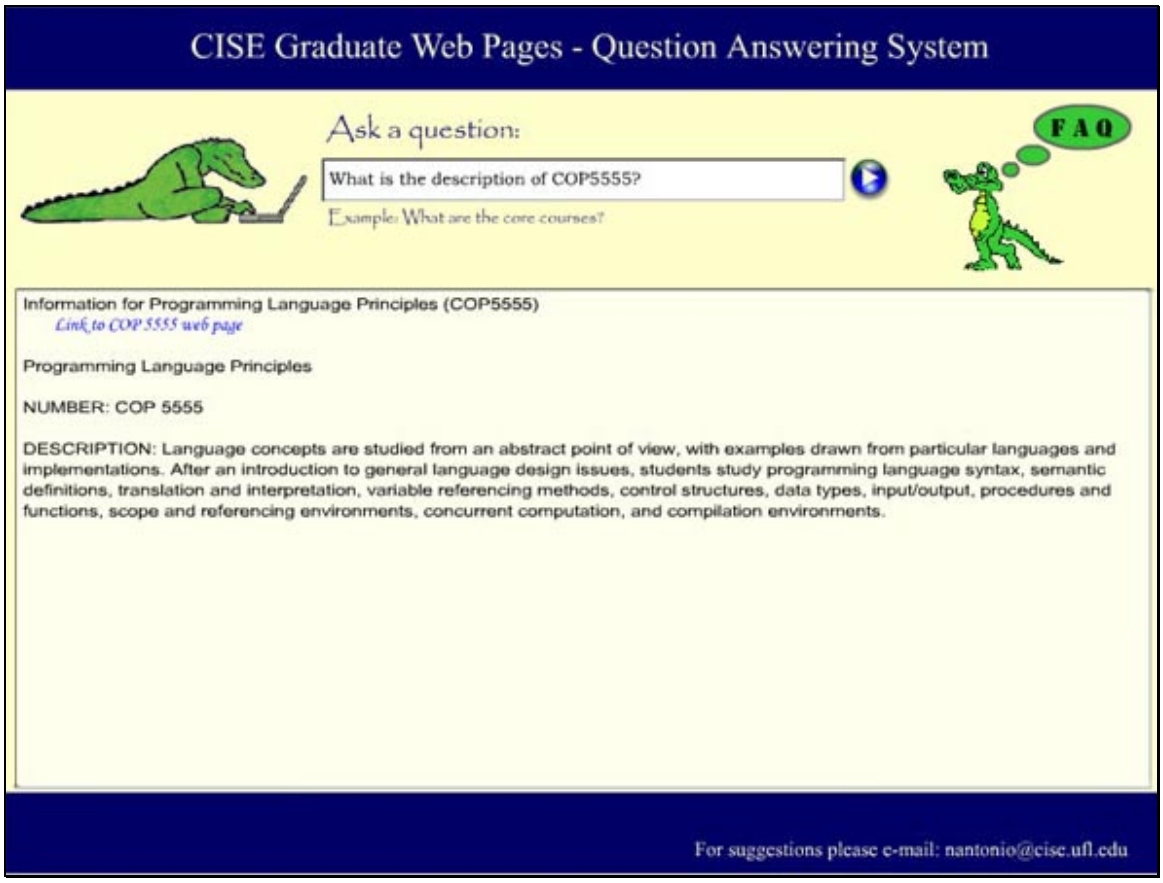

Figure 2.7: System's response to question, "what is the description of COP5555?"

The emphasis of this thesis is the interface; therefore, Chapter 4 is devoted to a detailed description of this module. This section constitutes a quick overview of the interface module.

When the query generator writes the retrieved data, the interface reads the file and parses the XML structure. It then determines which information will be displayed, as well as the best possible way to present that data to the user. Once these two factors are determined, it dynamically creates a page as the answer. The page usually contains links to sub-domains of the answer along with external links whenever these are available. The whole module was created using Macromedia Flash 5 and runs embedded in an HTML web page. Figure 2.7 shows a screenshot of the system's response to question, "What is the description of COP5555?"

Summary

Natural language question answering is a very complex problem; so complex that an ideal solution is not feasible at present. What developers are doing instead, is focusing on intermediate solutions that create increasingly better systems. In this spirit, the system created by our research team is an intermediate solution to this problem. It has a restricted domain (the CISE graduate web pages) where the knowledge base was encoded manually. The system consists of three active modules plus the knowledge base. These are: the interface, the parser, and the query generator.

CHAPTER 3
BACKGROUND MATERIAL

Before delving deeper into the details of the intelligent interface module, which is the subject of the next chapter, certain background knowledge is necessary. The interface can be summarized as a natural language generator of answers from XML data, developed using Macromedia's Flash. This chapter serves as a basic overview of the aforementioned three topics, natural language generation, XML, and Macromedia Flash and can be skipped by the reader who is already familiar with these topics.

Natural Language Generation

Introduction

Language is the primary means of human communication and is a crucial aspect of our civilization. Without language, there would not be a way of passing knowledge from generation to generation, be it in spoken or written form. Since the arrival of computers, it has been a researcher's dream to be able to communicate with computers using the same natural language.

Language is studied in several different disciplines, each defining its own set of problems and solutions. These are summarized in Table 3.1 [4]. This thesis is concerned with the computational linguistics aspect of the problem, dubbed natural language processing.

Table 3.1: Problems and tools for natural language according to the different disciplines

Discipline	Typical Problems	Tools
Linguists	How do words form phrases and sentences? What constrains the possible meaning for a sentence?	Intuitions about well-formedness and meaning; mathematical models of structure (for example, formal language theory, model theoretic semantics)
Psycholinguists	How do people identify the structure of sentences? How are word meanings identified? When does understanding take place?	Experimental techniques based on measuring human performance; statistical analysis of observations
Philosophers	What is meaning, and how do words and sentences acquire it? How do words identify objects in the world?	Natural language argumentation using intuition about counter-examples; mathematical models (for example, logic and model theory)
Computational Linguists	How is the structure of sentences identified? How can knowledge and reasoning be modeled? How can language be used to accomplish specific tasks?	Algorithms, data structures; formal models of representation and reasoning; AI techniques (search and representation methods)

There are also seven different levels of language analysis [4]:

1. Phonetic and Phonological knowledge – how words are related to the sounds that realize them.

2. Morphological knowledge – how words are constructed from more basic meaning units called morphemes.

3. Syntactic knowledge – how words can be put together to form correct sentences.

4. Semantic knowledge – what words and sentences mean.

5. Pragmatic knowledge – how sentences are used and how this affects their interpretation.

6. Discourse knowledge – how the immediately preceding sentences affect the interpretation of the next sentence.

7. World knowledge – general knowledge about the structure of the world that language users must have, for example, to maintain a conversation.

Natural language generation (NLG) is one of the two main areas concerning the field of natural language processing. The other is natural language understanding (NLU). Natural language understanding examines the mapping from some surface representation of linguistic material – expressed as speech or text – to an underlying representation of the meaning carried by that surface representation [5]. On the other hand, natural language generation is concerned with the mapping from some underlying representation of meaning into text or speech. Both problems are equally large and important, but the literature contains much less work on NLG than it does on NLU.

Historical Perspective of Natural Language Generation

The first significant pieces of work in NLG appeared in 1974 with the paper of Goldman on the problem of lexicalizing underlying conceptual material [6] and in 1979 with Davey's work on the generation of paragraph-long descriptions of tic-tac-toe games [7]. These were followed by the Ph.D. theses of McDonald [8], Appelt [9], and McKeown [10].

The field never became significant and since those major pieces of work were published, not much has happened. Journals like *Computational Linguistics* publish

papers on NLG, but most new material appears at the two biennial series of workshops that take place, one in Europe and one International.

Natural Language Generation Perspectives

According to McDonald [8], "NLG is the process of deliberately constructing a natural language text in order to meet specified communicative goals."

Reiter and Dale [11] give a more recent and lengthy definition:

> "NLG is the subfield of artificial intelligence and computational linguistics that is concerned with the construction of computer systems that can produce understandable texts in…human languages from some underlying non-linguistic representation of information."

There are two important points in these definitions. From the first definition we note the generation of *text* rather than sentences and from the second we note the *underlying non-linguistic representation of information*. Both of these issues had to be dealt with in our question answering system and more specifically in the intelligent interface module.

Kathleen McKeown [12] states that there are two major aspects of computer-based text generation:

1. determining the content and textual shape of what is to be said, and
2. transforming that message into natural language.

Determining the content, that is, what the hearer should be told is clearly the most difficult of these tasks. The following questions arise [13]:

- When does the hearer need to know the details of the topic?
- What is the effect of telling him or her only interpretations?
- What is the effect of telling both details and interpretations?

If the hearer can make the high-level interpretations, then all you need to provide are the details. In contrast, if the hearer has only minimal knowledge about or interest in the topic then you need to relieve him of the burden of details and the task of interpretation. In summary, you must be as specific as the hearer's knowledge of the topic allows: if you are too specific the hearer will not understand, and if you are too general you run the risk of seeming to hide things or of being uncooperative. In the first case, you violate the default speaker goal to be intelligible, and, in the second, you violate Grice's (1975) maxim of quantity to say neither more nor less than is required.

Natural Language Generator Tasks

All generators have to perform three principal functions: inclusion, ordering, and casting. These functions arise from the nature of their task, which is the production of chunked linear output from complex, large, non-linear input. Eduard Hovy [13] describes these in more detail:

1. Inclusion: select which of the input elements, and perhaps which portions of them, to consider. Only those selected will eventually appear.

2. Ordering: select the order in which to consider the input elements. They will appear in the text in this order.

3. Casting: select, for each element, a syntactic class or type. This class determines the form of the eventual realization – that is, the syntactic category in which the element will eventually appear in the text.

There is also a more generic way of defining the standard tasks of a natural language generator as: *text planning* (also called "discourse" planning), *sentence planning*, and *text realization*. Text planning determines the content and discourse

structure, sentence planning improves the fluency and understandability of text, and text realization maps sentence plans into sequences of words.

Traditional Approaches to Text Realization

There are four basic approaches to text realization: canned text, template, phrase-based, and feature-based realization [14]:

1. Canned Text: is a predefined string that was written by the system designer before run time to achieve a fixed communicative goal (such as a warning, suggestion, or hint) whenever its trigger occurs.

2. Template-Based Realization: a template is a predefined form that is filled by information specified by either the user or the application at run time.

3. Phrase-Based Realization: these systems define a set of generalized templates that represent various types of phrases found in a natural language, such as noun phrases or verb phrases. These phrases are related to each other by a set of rules (grammar) that specifies how different words or phrases may be combined into legitimate sequences.

4. Feature-Based Realization: in these systems, every feature of a grammar is represented by a single feature (for example, tense, number, and person) in the text specification. The generation process involves either traversing a feature selection network or by unification with a grammar of feature structures until a complete set of feature value pairs is collected to realize each part of the input.

XML

XML is an acronym for Extensible Markup Language and is a subset of Standard Generalized Markup Language (SGML) defined in ISO standard 8879:1986. XML is a way of putting structured data in text files and strives to become the universal format for structured documents and data on the Web. The initial XML draft was compiled in November 1996 at an SGML conference in Boston and the base specifications were recommended by the World Wide Web Consortium (W3C) on February 10 1998, under the name of XML 1.0 [15].

XML is a markup language that gives the user the ability to use his own tags and attributes, something that HTML does not allow. XML files always clearly mark the start and end of each of the elements of an interchanged document. XML restricts the use of SGML constructs to ensure that fallback options are available when access to certain components of the document is not currently possible over the Internet. It also defines how Internet Uniform Resource Locators can be used to identify component parts of XML data streams [16].

An XML file must meet two criteria: it has to be well formed and valid.

Well-Formed XML

XML files normally consist of three types of markup, the first two of which are optional. These are:

1. An XML processing instruction identifying the version of XML being used, the way in which it is encoded, and whether it references other files or not, e.g., <?xml version="1.0" encoding="UTF-8" standalone="no" ?>

2. A document type declaration that either contains the formal markup declarations in its internal subset or references a file containing the relevant markup declarations, e.g.: <!DOCTYPE grad_pages SYSTEM "http://www.cise.ufl.edu/~nantonio/dtds/grad_pages.dtd">

3. A fully-tagged document instance which consists of a root element, whose element type name must match that assigned as the document type name in the document type declaration, within which all other markup is nested.

If each element is properly nested within its parent elements and each attribute is specified as an attribute name followed by a value indicator (=) and a quoted string, the document is said to be *well-formed*. Figure 3.2 shows an example of a well-formed XML document.

```
<?xml version="1.0" ?>
<!—sample XML file -->
<GRAD_PAGES lastRevised="09/24/01">
    <STUDENT ssnumber = “999-99-9999”>
       <NAME>Nicholas Antonio</NAME>
       <DEPARTMENT>CISE</DEPARTMENT>
       <WEB_PAGE>http://www.cise.ufl.edu/~nantonio</WEB_PAGE>
       <EMAIL>nantonio@ufl.edu</EMAIL>
   </ STUDENT >
</GRAD_PAGES>
```

Figure 3.2: Sample XML file

Valid XML

XML users can define the role of each element of text in a formal model, known as a Document Type Definition (DTD). This allows them to check that each component of the document occurs in a valid place within the interchanged data stream. An XML DTD allows computers to check, for example, that users do not accidentally enter a third-level heading without first having entered a second-level heading, something that cannot be checked using HTML. The presence of a DTD is not required.

Elements are defined in a DTD by using the <!ELEMENT> tag and attributes by using the <!ATTLIST> tag. Declarations must begin with a <!DOCTYPE> tag followed by the name of the root element of the document. Commonly used text can be declared within the DTD as a text entity. A typical text entity definition can take the form: <!ENTITY department "Computer Information Science and Engineering">. Formats other than XML require a <!NOTATION> tag to tell the program what to do with the unparsed data contained in the referenced file.

Figure 3.3 gives an example of a DTD file. This model tells the computer that a "student" consists of a sequence of header elements, <name>, <department>, optionally <web_page>, and <email>. (#PCDATA) stands for parsed character data. The cardinality of elements can be specified with the following operators: ? for "zero or one," * for "zero-or-more," and + for "one-or-more." The default cardinality is one.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE student [
<!ELEMENT student    (name, department, web_page?, email) >
<!ELEMENT name    (#PCDATA) >
<!ELEMENT department    (#PCDATA) >
<!ELEMENT web_page    (#PCDATA) >
<!ELEMENT email    (#PCDATA) >
]>
```

Figure 3.3: Sample DTD

If all three types of markup mentioned above are present and the document instance conforms to the rules defined in the document type definition, the document is said to be *valid*. If only the last component is present and no formal model is present, all the XML processor can do is to check that the document instance is well-formed.

A more recent addition to the XML family is XML Schema. The word *schema* is Greek and means "a diagrammatic representation; an outline or model" [17]. The XML-Schema definition is an XML language for describing and constraining the contents of XML documents [18].

Although DTDs have served SGML and HTML developers well for 20 years as a mechanism of describing structured information, DTDs have severe restrictions compared to XML Schema. An XML-Schema is very much like a DTD, however there are some critical differences, the most notable being that an XML-Schema document is, itself, an XML document, while a DTD is not. XML-Schema is also capable of mixing namespaces. An XML namespace is a collection of names, identified by a URI reference [RFC2396], which are used in XML documents as element types and attribute names. XML namespaces differ from the *namespaces* conventionally used in computing disciplines in that the XML version has internal structure and is not, mathematically speaking, a set [19]. Figure 3.4 gives an example of an XML-Schema file. The line xmlns:xsd = "http://www.w3.org/2001/XMLSchema" indicates that all XML-Schema elements are to be prefixed with an <xsd:> tag. This namespace is hard-wired, and will always be picked up. If an element has attributes and non-text children we consider it as a *complexType* since the other datatype, *simpleType* is reserved for datatypes holding only values and no element or attribute sub-nodes. The sequence is a *compositor* that defines an ordered sequence of sub-elements. Elements must have a name and a type, can contain simple, predefined data-types (xsd:string), can reference some other element definition rather than contain their own name and type, and can have complex types that are defined directly within the element definition. The cardinality of an element can be precisely defined by using the *minOccurs* (minimum number of occurrences) and *maxOccurs* (maximum number of occurrences). In addition to having reference elements and locally defined complexTypes, a complexType can be defined as an entity in it's own right. Attributes of the document elements must always be declared last.

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="student">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="department" type="xsd:string"/>
        <xsd:element name="web-page" minOccurs="0" maxOccurs="unbounded">
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="ssnumber" type="xs:string"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Figure 3.4: Sample XML-Schema document

Macromedia Flash

Now in its fifth version, Macromedia Flash is a timeline-based Web authoring tool that uses vector graphics. It can be used to design and deliver low-bandwidth animations, presentations, and Web sites. It offers scripting capabilities and server-side connectivity for creating engaging applications, Web interfaces, and training courses. Once the desired content is created, online audience can view it using the Macromedia Flash Player. According to Macromedia, the Flash Player is the most widely distributed software on the web. Macromedia's partnership with key players in the Web-browsing business, allows 96% of all Internet users to view Macromedia Flash content without downloading the Player.

Vector-based files are compact and efficient and scale cleanly to any size compared to their bitmap equivalents. This is a key attribute for Web interfaces because of slow dial-up connection speeds and the great diversity of screen resolutions on user's monitors. Thus, developers do not have to worry about creating multiple versions of applications for each connection speed or monitor resolution.

Macromedia Flash uses Actionscript as its scripting language, which has a syntax similar to Javascript, making it easy for designers to develop. Flash also provides connectivity directly to middleware or databases via standard XML protocols.

Vector-graphics, Actionscript, XML support, and its practically ubiquitous nature, were all key reasons why Macromedia Flash was chosen for the design of the intelligent interface module.

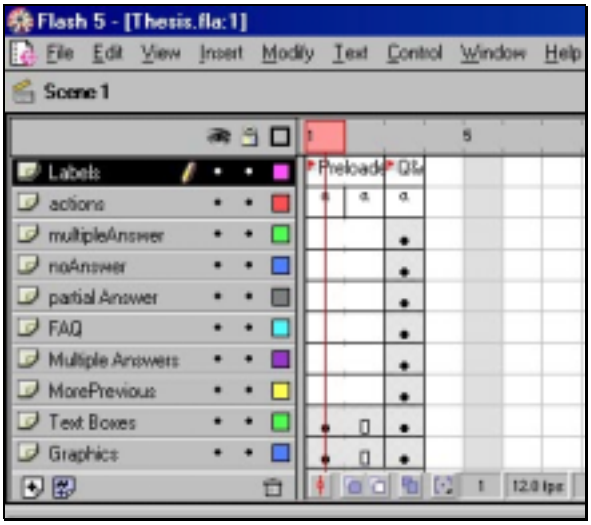

Figure 3.5: Frames and layers in Flash

Flash is quite different from other programming environments. It uses a certain terminology that needs to be described. Every Flash program is called a "movie." Every

movie consists of other smaller movies, the equivalent of sub-routines, procedures, or methods in object-oriented programming languages. There are also *buttons*, *graphics*, and *text*.

Movies consist of an unlimited number of *layers* and *frames*, each of which can contain other movies, buttons, graphics, or code. Every button consists of four frames. These are called *up*, *over*, *down*, and *hit*, which represent the equivalent mouse events. For example, *down* is activated when the mouse is pressed and resides in the down position on top of the button. Each of these frames can contain code. On the other hand, graphics and text contain no code. Figure 3.5 shows the frames and layers of a Flash movie.

When movies are placed on screen, which in essence means that they are placed in some frame of some layer, we have an *instance* of a movie that can have a name. There can be any number of instances of the same movie or button, like a sub-routine used repeatedly. Every instance can contain its own code, which can differ from that of another instance of the same object. However, only instances of movies can be manipulated through code, since only these can have names.

Each movie has certain associated attributes, like *alpha*, *visibility*, *transformation*, and *x and y coordinates*. These can be given values through code. For example, movie1._alpha = 70 changes the movie's alpha value to 70%. Movies within movies can be addressed by using the dot notation, like object oriented programming languages. For example, movie1.movie2.movie3._visible = 0 makes movie3 that resides within movie2, which in turn resides within movie1 invisible.

There are also three types of text; *static*, *dynamic*, and *input*. Static text is created at programming-time and remains unchanged throughout the movie's existence. Dynamic text has a variable associated with it and can be manipulated through code during the movie's run-time. Input text also has a variable associated with it, which cannot be controlled by the programmer, but by the user every time he inputs text in that object again during run-time.

Graphics are simple shapes or pictures that have no other function than being present in the frame in which they have been placed. If they are created within Flash, they are vector graphics and, if they are imported in the library, they can be bitmaps, gifs, etc.

All the objects mentioned above reside in the *library*. If a movie is exported in the library, it can be dynamically attached to another movie during run-time, without having been instantiated during programming-time. In the same fashion, it can be *killed*, which means that it can be removed from wherever it was attached.

Summary

The intelligent interface for the question answering system was created using Macromedia Flash, a web-based authoring tool supporting vector graphics. At the beginning of the process of building an answer for the user, the interface receives an XML structure that contains the answer. XML is a markup language that can be used to describe the semantics of data structures. The last action of the system is the provision of a natural language answer to the user's question. Natural language generation is the process of deliberately constructing a natural language text to meet specified communicative goals.

CHAPTER 4
THE INTELLIGENT INTERFACE

What is an Intelligent Interface?

Introduction

The term Intelligent Interface is very wide, and, in practice, people tend to avoid using it. Instead, articles have been written on intelligent tutoring, adaptive interfaces, explanations, or multi-modal dialogue. This is a very heterogeneous research subject and all of the aforementioned areas can claim to develop intelligent interfaces, but none of them addresses this aspect specifically.

The notion of intelligent interfaces captures a set of problems and ideas common in all these specific research areas without being artificially constrained by an application area or a specific technical solution.

Intelligent Interface Issues

It is important to note that an *intelligent system* does not necessarily have an intelligent interface. The intelligence of such a system does not necessarily manifest itself in a user interface. Good examples of this are expert systems, which are constructed to reason about and act upon some limited field of application. These systems do not necessarily produce visual output for the user, but can instead perform some kind of action like solving difficult diagnostic problems.

It is equally important to note that well-designed interfaces are not by default intelligent. There are many approaches to the development of easy-to-use and effective interfaces, documented by guidelines or interface standards [20 - 21]. These restrictions usually refer to how easy to learn interfaces must be and do not always lead to optional behavior.

However, not all users are the same. Advanced users have different needs than novices. Conventions and guidelines shift the entire burden of adaptation to the user, and the restrictions they impose are geared towards easing this task for the user.

Intelligent interfaces must not necessarily mimic human dialogue or adapt to the user's behavior. Even though systems maintaining human dialogue can be considered intelligent, many intelligent interfaces do not look human at all. The PUSH interface [22] [23] presents hypertext in a manner that is adapted to the user's current task. The output consists of a text where certain pieces of information are hidden from view to give a comprehensive overview of the most relevant information. This system does not mimic human behavior, but can be considered intelligent because of its adaptive nature.

Computer interfaces must be transparent and predictable to allow users to understand and use them. The inherent ability of computers to store vast amounts of information can be used by interfaces to maintain previous interactions and to present information in multiple modalities to enhance the presentation.

An ideal interface is one that always gives the optimal response. Obviously, the ideal interface does not exist. How can we define then an intelligent interface?

> *An intelligent interface has limited capabilities, but gives optimal responses within its architectural and interactive limitations.*

The main issue of this definition is the optimality of the responses of the intelligent interface taking into consideration its limitations. An optimal response for a system without speech capability differs greatly from an optimal response for a system that supports speech.

Application Areas

Intelligent interfaces are useful in application areas where the knowledge about how to solve a task resides primarily in the computer system. In these cases, users need to ask the system to do something for them. Typical application areas include *intelligent tutoring*, *intelligent help*, and *information filtering*.

- Intelligent Tutoring: A "tutor" is a program that aims to give a personalized "education" to a user in a specific domain of knowledge [24].
- Intelligent Help: A "help" system aids a user in performing a specific task [25].
- Information Filtering: The task of "filtering" is to extract the information that is relevant to the user from a large mass of available information, like the Internet.

Description of the Intelligent Interface Module

Our question answering system needed an intelligent way of providing the retrieved results to the user. This was realized in the form of an intelligent interface that generates natural language responses.

As seen in Figure 4.1, the interface first parses the XML file written by the "query generator" that contains a portion of the knowledge base, which is believed to contain the answer to the user's question. After the file is parsed it is converted to plain natural language text, hypertext, and external links. If the answer is considered too long, it is filtered and finally presented to the user.

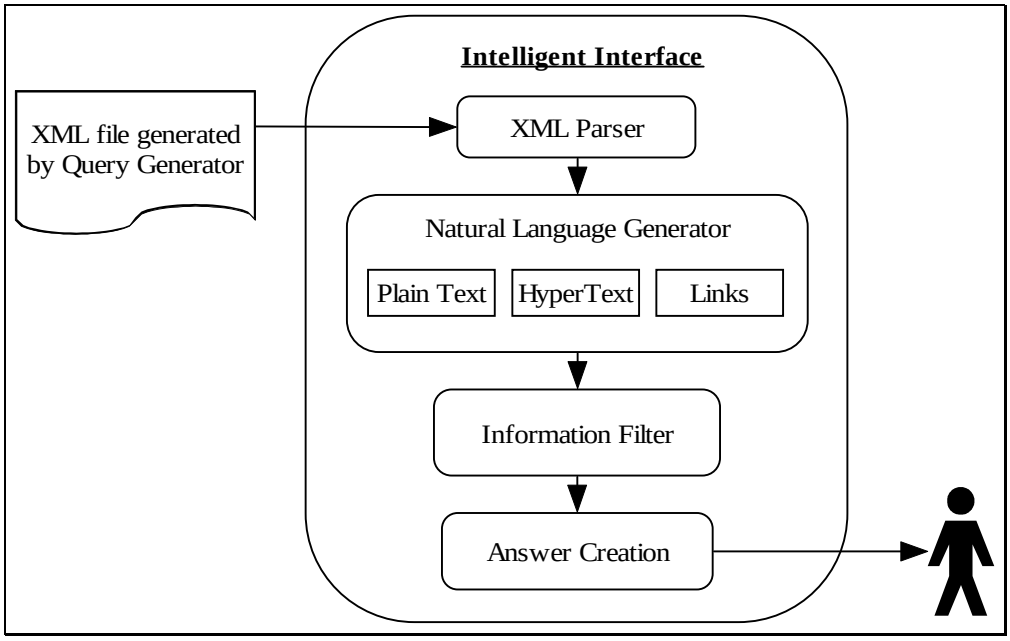

Figure 4.1: How the intelligent interface works

The entire interface was created using Macromedia Flash. Because of the nature of authoring tools like Flash, the code does not reside in a single location, nor do the sub-modules exist in the form of sub-routines that can be easily provided. Instead, the code resides in the frames, movies, and buttons.

Appearance of the Interface

The interface has two main text boxes. The question input box, where the user can type in his question, and the answer output box, where the system provides the generated response. The size of the latter serves as the upper bound of information that can be shown to the user at any one time. No scrolling is allowed. These two boxes are shown in Figure 4.2. In addition, there is an FAQ button, which brings up a window with a list of frequently asked questions. The FAQ window can be seen in Figure 4.3.

XML Parser

Flash has a built-in XML parser. Once given a file name, it parses the file's contents into XML objects, which are essentially tree structures. The input files have to be well-formed but not valid, since Flash does not make use of DTDs. The section of code in Figure 4.4 is a sub-set of a function called "askQuestion," which is called every time a new question is asked. The comments in the code explain the functionality of each line of code.

After the XML file is loaded into memory, it can be referenced by a set of special XML functions provided by Flash, like *firstChild*, *parentNode*, *nodeName*, *nodeValue*, etc. Their function is self-explanatory. For example, *firstChild* refers to the first child of a particular node in a tree. Thus, questionTree.firstChild.nodeName returns the node name of the first child of the question tree.

Figure 4.2: Appearance of the question answering system's interface

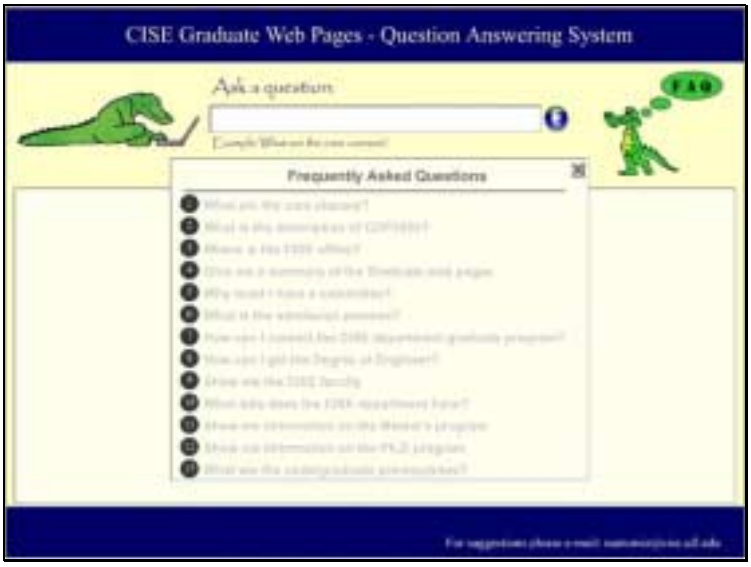

Figure 4.3: The FAQ window

```
function askQuestion() {
        // create a new xml object
        urlXML = new XML();
        // if the xml data is successfully loaded, XML Object will run the convertXML
function
        urlXML.onLoad = convertXML;
        // Write to the text area that the data is loading.
        Answer = "Loading data...";
        // now load up the url
        urlXML.load(_root.xmlFileName);
}
```

Figure 4.4: Part of the code of function *askQuestion*

```
<?xml version="1.0" ?>
<RESULT number="1">
<QUERY string="What is the description of COP5555?"> </QUERY>
<ANSWER type="E">
   <COURSE>
      <CONTENT>Information for Programming Language Principles (COP5555)
      </CONTENT>
         <TEXT>Programming Language Principles</TEXT>
            <LINK>
               <TEXT>COP 5555</TEXT>
                 <TARGET>
                      http://www.cise.ufl.edu/~ddd/grad/grad_courses.html#COP5555
                  </TARGET>
            </LINK>
            <NUMBER>
              <CONTENT>The course number of Programming Language Principles
(COP5555)</CONTENT>
               <TEXT>COP 5555</TEXT>
         </NUMBER>
         <DESCRIPTION>
            <CONTENT>The description of Programming Language Principles
            </CONTENT>
            <TEXT>Language concepts are studied from an abstract point of view,
with examples drawn from particular languages and implementations…
            </TEXT>
       </DESCRIPTION>
    </COURSE>
</ANSWER>
</RESULT>
```

Figure 4.5: Key features of XML files created by “query generator”

```
function convertXML () {
        partialAnswer._visible = 0;
        noAnswer._visible = 0;
        if (this.loaded) {Answer = "Data loaded.";}          // Get the number of Results
        numberOfResults = ParseInt(this.firstChild.nextSibling.attributes.number);
        // set the question text field to the question asked
        Question = this.firstChild.nextSibling.childNodes[1].attributes.string;
        // set the dynamic text field to blank so we can write new data to it
        Answer = "";
        // we get the child node array beneath the question
        questionTree = this.firstChild.nextSibling.childNodes[1];
        // we get the answer node array
        answerTree = this.firstChild.nextSibling.childNodes[3];
        //Get the type of answer(E - exact, P - Partial, N - No answer)
        answerType = answerTree.attributes.type;
        if (numberOfResults == 1) {
                //Start by showing the first level children of each answer
                showNodeContents(1);
                if (answerType == "E") {                    //exact answer
                } else if (answerType == "P"){
                        partialAnswer._visible = 1;         //partial answer
                }
        } else if (numberOfResults > 1) {
                multipleAnswerFunction();                   //multiple answers
        } else {
                noAnswer._visible = 1;                      //no answer
        }
}
```

Figure 4.6: Part of function *convertXML*

Natural Language Generator

Before examining the details of this system, we need to first look at the structure of the XML files returned by the *Query Generator*. Figure 4.5 shows some of the key features of these files. There are two tags that convey information as attributes. These are <RESULT>, which shows the number of answers retrieved by the system, and <ANSWER>, which identifies if the answer found is an exact (E) or a partial (P) match.

When the XML file loads successfully, a function called convertXML is called. See Figure 4.6. This function checks to see how many answers were generated by the system and if the system found an *exact* or a *partial* match to the user's question. It also retrieves the tree representing the answer and stores it in a new XML tree for further processing. If there is only one answer, then the function *showNodeContents* is called, which proceeds with further processing of the answer. If more than one answer exists, then *multipleAnswerFunction* is called. When the system is unable to retrieve an exact match, the system displays a note to the user informing him that the generated answer may not be the desired one if there is a partial match, or that the system was unable to retrieve an answer if the number of results in <RESULT> is zero.

The function *showNodeContents* processes the entire answer tree and transforms the nodes and their contents into natural language text, hypertext, or external links. It does so by checking the node names and, if it finds a match in its internal logic, it follows the instructions there; otherwise, it follows a generic set of rules. For example, if the node name is <COURSE> then from the DTD the system knows its specification, which is:

<!ELEMENT COURSE (CW, CONTENT, TEXT, ROOT_TEXT?, LINK?, NUMBER?, DESCRIPTION?, PREREQ?)>

If the specification is known, then it can be handled intelligently. In the case of <COURSE>, the system knows what to expect and can construct an appropriate answer. Figure 4.7 shows how the system handles the <COURSE> element and Figure 4.8 shows how the system handles the elements defined in the specification of <COURSE>.

For a better understanding of the system, lets look into the code fragments of Figures 4.7 and 4.8. First, in Figure 4.7 when the system encounters a <COURSE> tag, it either displays the relevant information (described later) if there is no space limitation, or it creates a hypertext link with the name of the particular course. For example, if the course happens to be “Artificial Intelligence Concepts” and it is one of many courses to be displayed, a space limitation exists. Thus, a hypertext link is created with that name. If the user clicks on the link then the sub-elements of <COURSE> are processed and displayed (Figure 4.8). The title, the number, and the description of the class appear as plain text, an external link is created to the course’s description in the graduate catalog, and the prerequisites appear as hypertext links.

```
if (answerTree.childNodes[i].nodeName.toUpperCase() == "COURSE") {
        eval("answer" + Math.floor(i/2))._visible = 1; //Make the answer button visible
        eval("textBox" + Math.floor(i/2))._visible = 1; //Make the text Box visible
        // Give the value of the <TEXT> node to the text Box
        // (second child, after <content>)
        eval("textBox" + Math.floor(i/2)).Box =
answerTree.childNodes[i].childNodes[3].firstChild;
}
```

Figure 4.7: How the system handles the <COURSE> element

```
if (answerTree.childNodes[i].nodeName.toUpperCase() == "TEXT") {

        Answer = Answer + "<p></p><p></p>" + answerTree.childNodes[i].firstChild;
}
if (answerTree.childNodes[i].nodeName.toUpperCase() == "LINK") {
        linkCounter++;
        linkTextName = "linkText" + linkCounter;
        eval(linkTextName).Box = "<a href=\"" +
answerTree.childNodes[i].childNodes[3].firstChild + "\"> Link to " +
answerTree.childNodes[i].childNodes[1].firstChild + " web page </a>";
}
if (answerTree.childNodes[i].nodeName.toUpperCase() == "NAME") {
        Answer = Answer + "<p></p><p></p>NAME: " +
answerTree.childNodes[i].childNodes[3].firstChild;
}
if (answerTree.childNodes[i].nodeName.toUpperCase() == "NUMBER") {
Answer = Answer + "<p></p><p></p>NUMBER: " +
answerTree.childNodes[i].childNodes[3].firstChild;
}
if (answerTree.childNodes[i].nodeName.toUpperCase() == "DESCRIPTION") {
        Answer = Answer + "<p></p><p></p>DESCRIPTION: " +
answerTree.childNodes[i].childNodes[3].firstChild;
}
if (answerTree.childNodes[i].nodeName.toUpperCase() == "PREREQ") {
        eval("answer" + Math.floor(i/2))._visible = 1; //Make the answer button visible
        eval("textBox" + Math.floor(i/2))._visible = 1; //Make the text Box visible
                        //Give the value of the <TEXT> Node to the text Box (second
child, after <content>)
        eval("textBox" + Math.floor(i/2)).Box = "Prerequisite: " +
answerTree.childNodes[i].childNodes[3].firstChild;
}
```

Figure 4.8: How the system handles the sub-elements of <COURSE>

Every element in the DTD is treated similarly to the process described above for the element <COURSE>. If an element does not have a specification, it is treated by the system as plain text and appended to the end of the current answer output box. Therefore, if a file with unknown elements is presented to the system it will just display intelligently its contents as plain text.

The natural language generation approach used by the system is called Template-Based Realization [14]. According to Hovy, “a template is a predefined form that is filled by information specified by either the user or the application at run-time.” The approach used by the intelligent interface falls within this category by determining at run-time how the various predefined forms are filled.

Information Filter

In the previous paragraphs, it was briefly mentioned that the size of the answer output box represents the upper bound of information that can be shown to the user at any one time. A typical file written by the Query Generator can consist of three or four pages of information. Therefore, a filtering method is required to reduce the information displayed.

Every XML file is a tree structure and every element in the tree has a parent node and probably some children nodes as well. The way the system filters out information is by presenting to the user the high level concepts as links and omitting the children of these nodes. For example, if the user’s question is “what are the master’s core courses?” the answer will consist of the titles of the master’s core courses, each of which will be a link to the course specific details.

Sometimes, the high level concepts might be too many to show at once. The output box can handle up to twenty concepts at a time. If more than twenty are found by the system, a *MORE* button appears on screen. Clicking this button updates the output box with the next set of concept links. There is also a *PREVIOUS* button, which returns the user to the previous set of options. Figure 4.9 presents the algorithm that the information filter uses in the general case. Some elements use a slightly different logic to match their DTD specification better.

```
Display <CONTENT>
IF (<TEXT> present) THEN
        Display <TEXT>
IF (<EMAIL_LINK> or <LINK> present as a sub-element) THEN
        Create appropriate links
IF (<TEXT> of sub-elements fits on screen) THEN
        Display each one as a paragraph
ELSE IF (sub-elements fit on screen [<=20]) THEN
        Create hyperlinks for every one
ELSE IF (sub-elements don't fit on screen [>20]) THEN
        Create hyperlinks for first 20
        Display a "more" button that gives access to the next 20
```

Figure 4.9: Algorithm used by the information filter

Examples

The system's behavior can be better observed through an illustrated example. After the user asks "What are the core classes? ," the system sends the question to the parser, which in turn sends its output to the query generator. The latter writes an XML

file containing the answer. The interface reads this answer (a six page long file) and processes it.

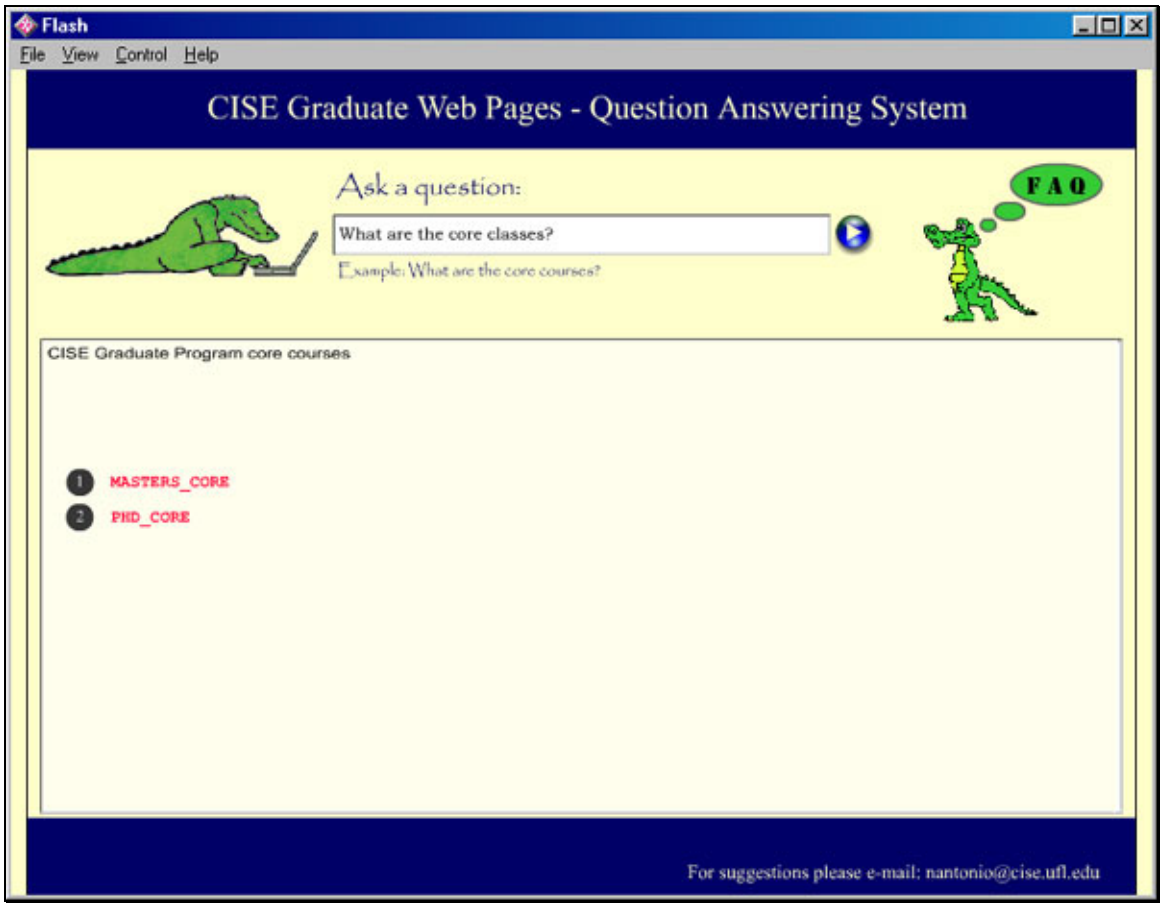

Figure 4.10: Initial system response to “what are the core classes?”

The system finds an exact match to the user’s question and therefore proceeds normally. There are two major sections to the answer, since the Master’s and Ph.D. core classes differ slightly. Thus the system first generates two hypertext links named *MASTERS_CORE* and *PHD_CORE*, as can be seen in Figure 4.10.

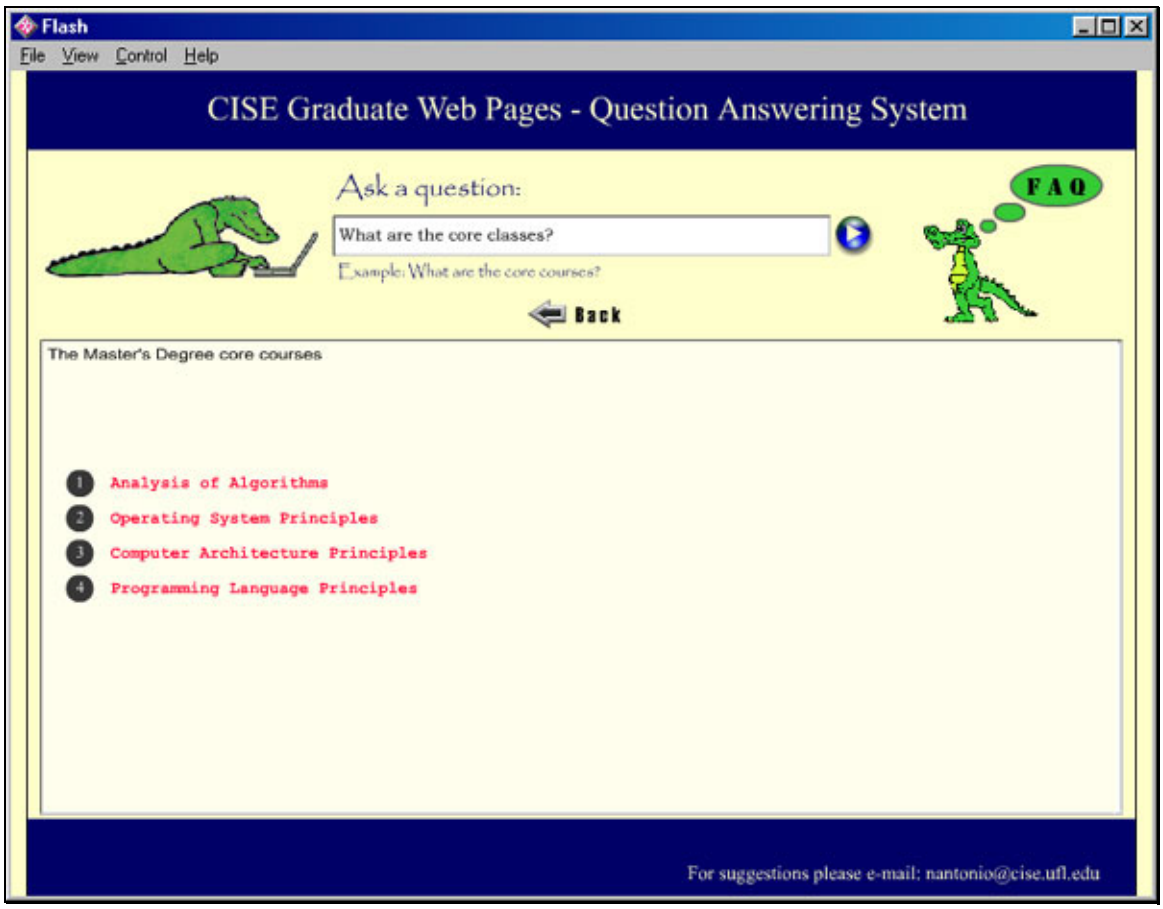

Figure 4.11: The Master's core courses

By clicking on one of these links, the user can see either the Master's or the Ph.D. core classes respectively. Figure 4.11 shows the system's response when the user clicks on *MASTERS_CORE*. The four titles of the Master's core classes appear, each of which is a hypertext link to the respective description of the class. A *BACK* button also appears, allowing the user to return to the previous menu.

Figure 4.12 shows the system's response when "Analysis of Algorithms" is clicked. The class name, number, and description are given as plain formatted text and an external link is created to the course's description on the graduate catalog.

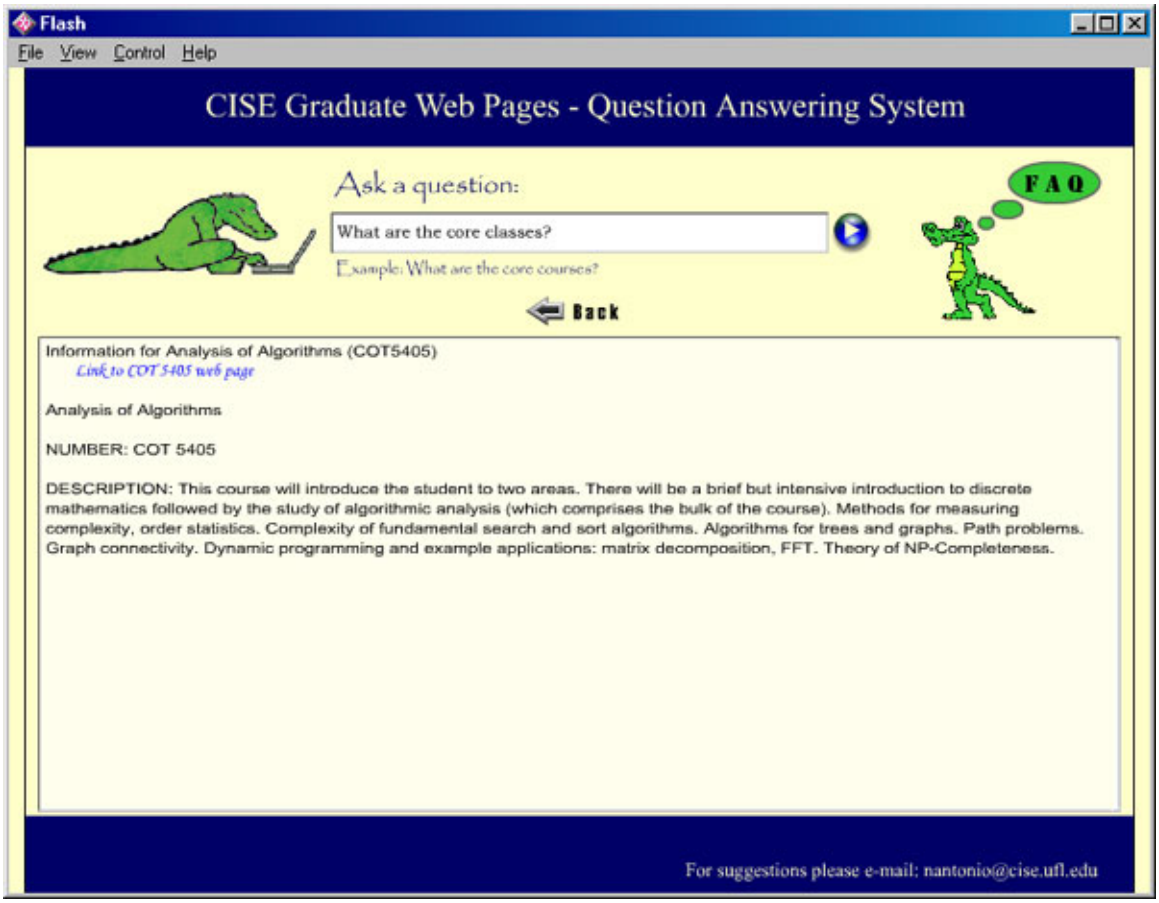

Figure 4.12: Description of analysis of algorithms

If the user wishes to view the Ph.D. core classes, he can do so by clicking twice on the *BACK* button, which returns him to the first menu, shown in Figure 4.10 and clicking on the *PHD_CORE* hypertext link. The system responds by presenting the five Ph.D. core classes as hypertext links, see Figure 4.13. A message is also displayed explaining to the user that the Ph.D. core courses consist of all the Master's core courses plus COT6315, which is the number of the course titled "Formal Languages and Computation Theory." Just like the Master's core classes, clicking on the course titles displays a description of each.

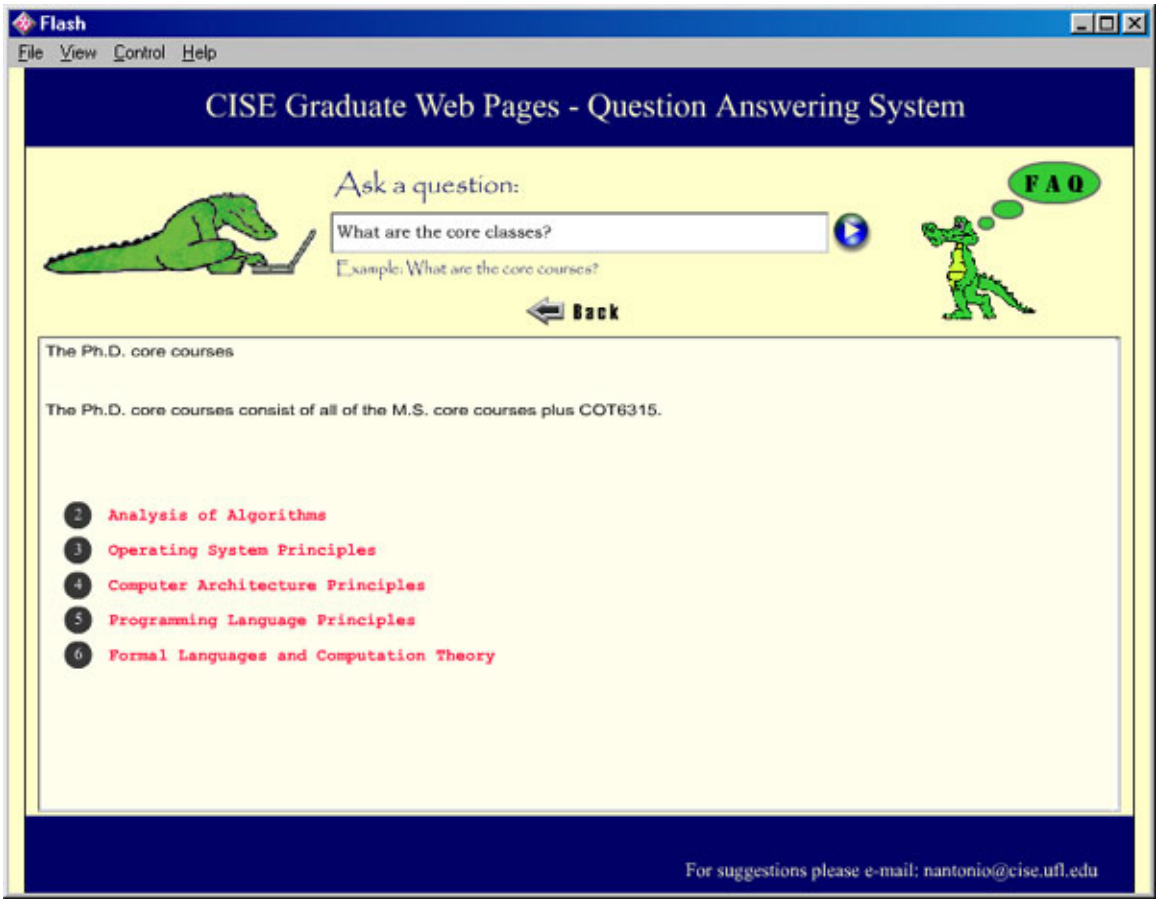

Figure 4.13: The Ph.D. core courses

In this example, the interface takes a six page long file retrieved as the answer by the query generator and displays it intelligently in a presentable fashion. It decomposes the answer hierarchically and presents only a limited amount of information at a time, without denying or making it difficult to access the details. The question is “What are the core classes?” and it is answered both generally, by providing the titles of the classes, and specifically, by allowing easy access to the details concerning each class.

Every one of the intermediate screens is retrieved directly by the system if the question asked is made more specific. For example, if the user asks, “What are the Master’s core classes?” then the system’s response will be that of Figure 4.11. In the

same manner, if the question asked is “Show me a description of Analysis of Algorithms,” then the system’s response will be that of Figure 4.12.

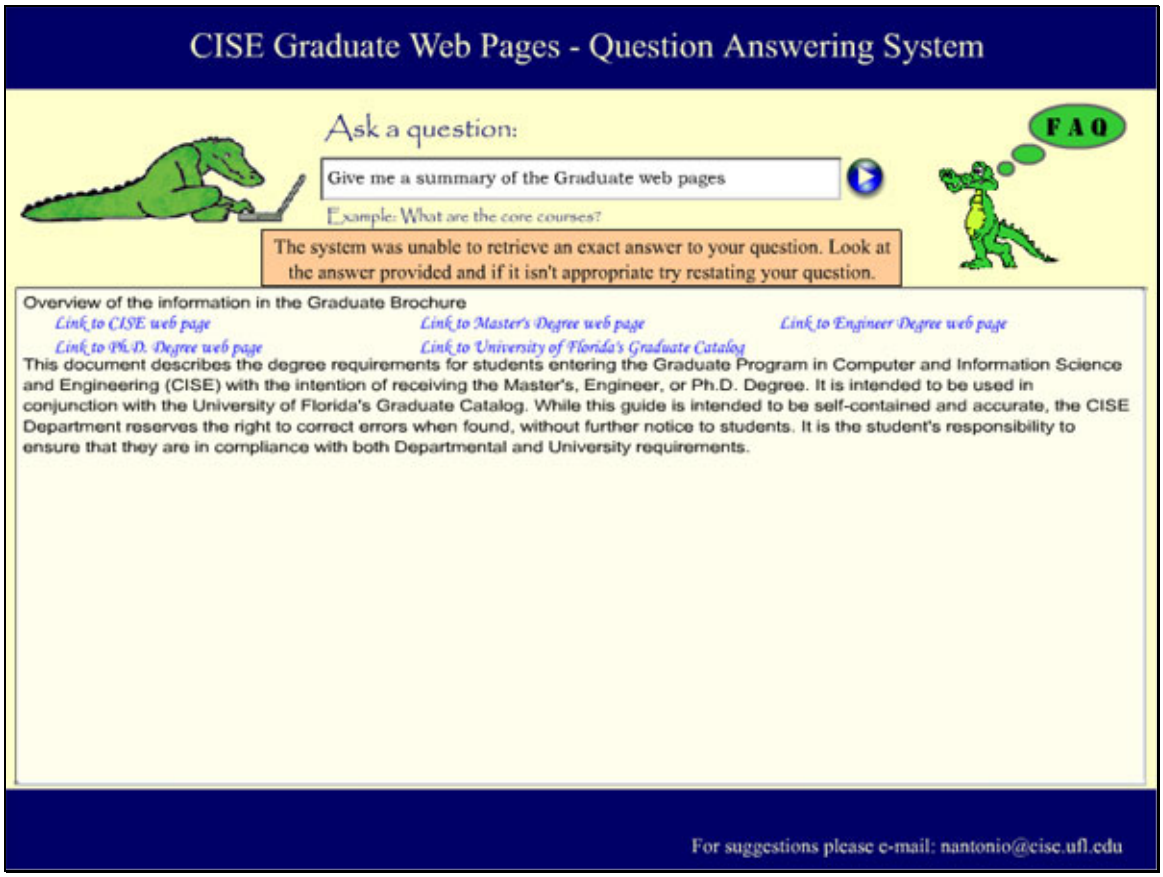

Figure 4.14: Summary of the graduate web pages

Figure 4.14 shows the system’s response to the request “Give me a summary of the Graduate Web Pages.” The system is unable to find an exact match to the user’s request and the interface displays a message warning the user about this and encouraging him to restate his question if the answer provided is not appropriate. It then proceeds to display the answer, which in this case is an overview of the graduate catalog contents. It also provides five external links.

Summary

An intelligent interface is a system that attempts to optimize the responses given to the user within its architectural and interactive limitations. These interfaces display some form of intelligent behavior, whether it is presenting information in natural language, adapting to the user's actions, or filtering information.

Even though the interface described in this thesis has a standard behavior that does not adapt to the user's actions, it can be considered intelligent because it provides answers in natural language, it filters out information to avoid overwhelming the user, and it creates hypertext and external links.

The system works by reading and parsing an XML file created by the Query Generator. It then proceeds to generate natural language content using a template-based realization approach. It filters out information by creating links from high-level concepts to specific details, as well as external links to the Internet. It finally presents the answer to the user who in turn can follow the aforementioned links or ask another question.

There are of course limitations to the system, as well as room for improvement. The next chapter describes conclusions made about this intelligent interface, its strengths, its weaknesses, and recommendations for further research.

CHAPTER 5
CONCLUSIONS

The problem of effective and efficient question answering is gaining increased interest as is the problem of information presentation and information filtering. This thesis presents, in the form of an intelligent interface, a solution to the latter as a sub-problem of the general question answering problem. This section reviews the strengths and limitations of the implemented system. Limitations of the system are then discussed and finally some possibilities for future research are presented.

Intelligent Interface Evaluation

The intelligent interface system described in this thesis is a proposed solution to the problem of information presentation and information filtering. Specifically, the thesis examines the area of answering questions regarding the domain of the CISE graduate Web pages. It generates natural language responses in the form of enriched text—enriched with hypertext and external links.

Methodology

The system uses a predefined set of rules, custom built according to the DTD definitions of the XML knowledge base. The interface can be used to view answers generated by the query generator, but can also be used to view any XML file of similar specification. Thus, the intelligent interface is a solution to the generic problem of

presenting filtered information to users, whether it is part of a question answering system or not.

Usability and Aesthetics

Users demand programs that are easy to use and aesthetically pleasing. It is true that beauty is in the eye of the beholder, but certain guidelines concerning usability and aesthetics can be considered universal. The interface is very simple in its use. The user types in a question and receives an answer. If the system is unable to find an exact match, it informs the user. Navigating is made easy with hyperlinks and navigation buttons whenever these are appropriate. The appearance of the system is simple—consisting of a question input field and an answer output box. There is also a “frequently asked questions” button for fast access to common questions.

Information Filtering

The system can take very long XML files as input and produce filtered output. Every time the information to be displayed is considered overwhelming to the user, a hierarchical answer is displayed, with each parent node of the input XML file being a link to its children. Therefore, users can navigate through files of great length very effectively, without ever being overwhelmed by the information presented to them. At the same time, the process is efficient and does not require an unreasonable effort to reach the most specific detail of each file. For example, the most specific details (leaf nodes) of a ten page long file, on average can be reached with three or four mouse clicks. If it takes the user one second for each mouse click, he can reach the information he desires in three or four seconds, in contrast to the huge amount of time it would have taken him to search through all ten pages manually.

Limitations of the Implemented System

The system does have a severe limitation. It is not generic. It is custom built according to the DTD of the knowledge base. The idea behind it, though, is very generic. Ideally, an intelligent interface of this kind can be used to filter and present information in any domain, as long as the input is in XML. The system described in this thesis was built to serve as an intelligent interface to a specific question answering system and therefore did not have to be generic. How the system can be transformed into a generic module is described next.

Future Research

Intelligent Interface

The solution proposed in this thesis can be expanded to work on every XML file that has a DTD specification, along with some extra Meta information. An ideal system would read the DTD and Meta files, which would include all the information needed on how to process all the data and use this knowledge to build its internal logic automatically. This would allow the interface to work in every case, independent of domain, since all the information needed would be provided.

This idea poses a problem though; that of assuming the presence of a DTD and a Meta file for every XML file. Requiring a DTD is not outrageous, since it is required to validate an XML file and a solution can be built without requiring the extra Meta file. Implementation-wise, unfortunately, the current version (5.0) of Macromedia Flash does not support DTDs, resulting in the added burden of coding everything from scratch.

At the time of development of the system, XML-Schema was not approved technology and therefore our research team chose to use a DTD. However, XML-Schema

has been approved on May 2, 2001 and can be used to replace the DTD. This would alleviate the problem of Flash not supporting DTDs, since XML-Schema documents are XML files.

Question Answering System

The problem of question answering is far greater than that of presenting the information to the user and future research must be treated with more caution. The first logical task is expand the domain. Now, the domain is very specific and only refers to the CISE graduate web pages. As a start, the domain can be expanded to include all the CISE department's web pages. This would immediately indicate the inherent limitations of building a domain-specific solution and should enlighten the process of what to do next.

Another addition to the system can be the automatic conversion of HTML pages into XML knowledge bases. The automatic conversion of text into knowledge is a problem that has plagued many a researcher, but I believe the transformation required for our solution is not far fetched. Automatic converters from HTML to XML already exist. The two markup languages are very similar and this was one of the primary reasons XML was chosen, rather than some first predicate logic, like prolog. The problem to be solved is the transformation of the information into a hierarchical structure that conveys its semantics and can be queried intelligently.

Afterword

Natural language question answering and intelligent interfaces are the inevitable future of human-computer interaction. Search engines are improving and so are interfaces. Many solutions to the problem of human-computer interaction have been

proposed, with intelligent agents being the current buzzword in this field. As we strive for more human-like interaction with our computers, natural language gains significance. At the same time, the Internet explosion mandates better information filtering and information presentation.

XML has been introduced as a step in the right direction of transforming the World Wide Web into a huge knowledge base. The W3C, recognizing the growing significance of natural language processing and XML's future role, has proposed a natural language specification for XML.

Better human-computer interaction also implies the use of multimedia. The Internet with its low bandwidth, demands files of small size, a direct opposite of multimedia requirements. Macromedia Flash manages to bridge this chasm and has become almost ubiquitous on computers today.

Recognizing all these factors, our proposed solution to question answering utilizes these tools that represent the future of Internet programming and appear to have a life span that allows future research to be continued from where this attempt has left off.

LIST OF REFERENCES

[1] Ask Jeeves! Available at http://www.ask.com, last accessed on 11/10/2001.

[2] Regoczei, Stephen and Hirst, Graeme 1989. *On 'extracting knowledge from text': Modeling the architecture of language users*. Technical Report CSRI-225.

[3] Temperley Davy, Sleator, Daniel and Lafferty, John. *The Link Grammar Parser*. Carnegie Mellon University. Available at http://www.link.cs.cmu.edu/link/, last accessed on 11/10/2001.

[4] Allen, James 1995. *Natural Language Understanding*. The Benjamin/ Cummings Publishing Company, Inc. Redwood City, CA, pp. 2-10.

[5] Dale, Robert, Eugenio, D. Barbara and Scott, Donia 1998. Introduction to the special Issue on Natural Language Generation. *Computational Linguistics*, Vol 24, Number 3. MIT Press for the Association for Computational Linguistics.

[6] Goldman, Neil M. 1974. *Computer Generation of Natural Language from a Deep Conceptual Base*. Ph.D. thesis, Stanford University, Stanford, CA.

[7] Davey, Anthony C. 1979. *Discourse Production*. Edinburgh University Press, Edinburgh.

[8] McDonald, David D. 1987. *Natural language generation*. Encyclopedia of Artificial Intelligence. John Wiley and Sons, New York, NY, pp. 642-655.

[9] Appelt, Douglas E. 1981. *Planning Natural Language Utterances to Satisfy Multiple Goals*. Ph.D. thesis, Stanford University, Stanford, CA. Available as SRI Technical Note 259.

[10] McKeown, Kathleen R. 1982. *Generating Natural Language Text in Response to Questions About Database Structure*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA. Available as Technical Report MS-CIS-82-05.

[11] Reiter, Ehud and Dale, Robert 1997. Building Applied Natural Language Generation Systems. *Natural Language Engineering*, Vol 3, Number 1, pp. 57-87.

[12] McKeown, Kathleen R. 1985. *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*. Cambridge University Press, Cambridge.

[13] Hovy, Eduard H. 1988. *Generating Natural Language Under Pragmatic Constraints*. Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, pp. 42-106.

[14] Hovy, Eduard H. 1997. *Language Generation: Overview* (Chapter 4.1). *Survey of The State of the Art in Human Language Technology*, Studies in Natural Language Processing. Cambridge University Press, Cambridge, pp. 139-146.

[15] World Wide Web Consortium, 1998. *Extensible Markup Language (XML) 1.0.* Available at http://www.w3c.org/TR/2000/REC-xml-20001006, last accessed on 11/10/2001.

[16] World Wide Web Consortium, 1999. *XML in ten points*. Available at http://www.w3c.org/XML/1999/XML-in-10-points, last accessed on 11/10/2001.

[17] The American Heritage Dictionary of the English Language, Fourth Edition, Copyright © 2000 by Houghton Mifflin Company, Boston, MA.

[18] Using W3C XML Schema, Eric van der Vlist, 2001. Available at http://www.xml.com/pub/a/2000/11/29/schemas/, last accessed on 11/10/2001.

[19] http://www.w3.org/TR/REC-xml-names/, last accessed on 11/10/2001.

[20] Smith, S.L. and Mosier J.N. 1986. *Guidelines for Designing User Interface Software*. MTR-10090, The Mitre corporation, Bedford, MA.

[21] Nielsen, Jakob 1989. Usability Engineering at a Discount. *Designing and Using Human-Computer Interfaces and Knowledge Based Systems*, Elsevier Science Publishers, Amsterdam, pp. 394-401.

[22] Höök, Kristina, Karlgren, Jussi, Woern, Annika, Dahlback, Nils, Jansson Carl-Gustav, Karlgren, Klas and Lemaire, Benoit 1996. A Glass-Box Approach to Adaptive Hypermedia. *User Modeling and User Adapted Interaction*, Vol 6, Number 2-3, pp. 157-184.

[23] Höök, Kristina 1997. *Evaluating the Utility and Usability of an Adaptive Hypermedia System*. In Proceedings of the International Conference on Intelligent User Interfaces, Orlando, Florida, January. ACM.

[24] Shute, Valerie J. and Psotka, Joseph 1994. Intelligent Tutoring Systems: past, present and future. *Handbook of Research on Educational Communications and Technology*, Scholastic Publications, New York, NY.

[25] Breuker, Joost 1990. *EUROHELP: Developing Intelligent Help Systems*. Report on the P280 ESPRIT Project EUROHELP, University of Leeds, Leeds.

BIOGRAPHICAL SKETCH

Nicholas Antonio was born on October 17, 1974, in London, England. He moved to Cyprus in 1981 where he remained until the completion of his military service in 1994. He attended the University of Macedonia in Thessalonica, Greece, as an undergraduate, where he received a Bachelor of Science in applied informatics. He was the valedictorian of his class. Upon graduation, he returned to Cyprus where he worked in the IT department of a local bank for a year. In 1999, he was awarded a Fulbright scholarship and he continued his studies in the Department of Computer and Information Science and Engineering at the University of Florida. He is also working for Healthy Outcomes Technology developing a decision support system for dentistry. His primary area of interest is intelligent interfaces. He plans to stay at the University of Florida and pursue a Ph.D.