MyFIDO: AI Based RSS Feed Aggregator

Using Natural Language Processing and Other Artificial Intelligence Techniques To Make Suggestions Based on Observed User Interests

TERESA NICOLE BROOKS
Facilitated By: PROFESSOR PAUL BENJAMIN

Pace University
Fall 2008 – Fall 2009
# Table of Contents

## Introduction: About MyFido

### Section 1: Project Objective

1. **Basic System Functionality**
   - The “Observer” Component
   - The “Context Analyzer” Component
   - The “Fetch” Component

2. **Testing Methodology & Evaluating Effectiveness of System**
   - Testing
   - Note Regarding System Design

## Section 2: What Is Natural Language Processing?

1. **About Natural Language Processing (NLP)**
   - History of Natural Language Processing

2. **Applications of NLP**
   - Examples of NLP Applications
   - Keep The Domain Small Stupid: Scope and Knowledge Domain Management
   - What’s An Operating System?: Giving Software The Ability to Analyze The Context and “Meaning” of Text
   - Software Doesn't Have Eyes: Observing user interests and analyzing the structure of a web page
   - I Know What You Like: Making Suggestions To Users Based on Observed Interests
   - Selecting Suggested Articles

## Section 3: Technologies & Tools

1. **Natural Language Processing Tools**
   - Natural Language Processing APIs: OpenNLP
   - String Metrics Library : SimMetrics
   - JGraphT
   - WordNet
   - WordNet Interfaces

2. **Other Technologies**
   - MozSwing
   - HTMLParser
   - Rome
   - MigLayout

## Section 4: My Fido: System Architecture

1. **System Components (Overview)**
   - Brief Description of Major System Component's Responsibilities
   - Brief Description of Knowledge and Document Representation Related System Components
Section 5: My Fido In Action: System Walk Through

- Getting Started: Logging Into MyFido
- MyFido Observe Me!: Starting A New Observer Browser Session
- MyFido What Are These Articles About?: Starting the Context Analyzer
- More Regarding Context Analysis
- Knowledge Access: Matching Terms & Phrases From Observed Documents With MyFido's Category Hierarchy Graph
- MyFido Fetch!: Caching Feeds Fetched From The Web
- MyFido Fetch!: Starting a New Fetch Session
- MyFido Fetch!: Getting User Feedback

Section 6: Conclusion

6.1 Results: Testing, Statistics, Proof of System's Accuracy
- Notes Regarding The Testing Process
- Summary of Results / Accuracy Statistics
- Future Statistical Goals
- Summary of Results: Fetched Feeds Analysis

6.2 Challenges / Issues Encountered
- Challenge: Finding The “Main Idea” of an Article
- Challenge: Limited Knowledge of the World, Limits Suggestions to Users
- Challenge: Level of Human Labor Required

6.3 Proposed System Enhancements
- Improving System Accuracy: Context Analysis & User Suggestions Functionality
- Improving System Performance, Design & Architecture
- Improve the Overall User Experience

6.4 Final Remarks

Section 7: Appendix A: References

Section 8: Appendix B: Code Snippets
Introduction: About MyFido

For many of us, the World Wide Web has become our primary gateway for acquiring news and entertainment, and in some instances a tool to help manage our day to day lives. With the near infinite amount of data available at our finger tips, information overload has become more of a reality. As our dependency on the web grows, along side it's increasing amount of content, the need for personal knowledge management systems will be more apparent than ever.

The MyFido software system is a modular software system that will attempt to address this need by utilizing Natural Language Processing and other Artificial Intelligence related techniques, such as, string distance metrics and knowledge based systems, in order to aggregate customized RSS feeds based on observed user interests.

All system components, as well as the open source software libraries used, are developed in Java. In addition to Java's ease of cross platform use, it's open nature allows for cost effective development and it's JVM technology allows for developing secure applications.

About This Paper

In this paper I will address the challenges to building knowledge based systems, more specifically systems that use Natural Language Processing to analyze and find the context of documents. I will give a brief history of the field of Natural Language Processing, while identifying the challenges that past researchers faced in the field, as well as how some of these challenges are apparent in modern research projects. I will discuss the common problems addressed in NLP, projects including MyFido, that attempt to passively observe user interests, manage information overload as well as create suggestion systems.

I will also give a walk through of MyFido functionality which will include screen shots and system diagrams. I will discuss the challenges I encountered while building MyFido. Lastly, I will identify possible system enhancements that would improve the scalability and performance of the application, as well as ideas to improve MyFido's overall intelligence and capacity to learn user interests as well as grow it's domain of knowledge.
1. Project Objective

The primary objective of this project is to explore the use of an intelligence based system to suggest articles to users of IT and technology related sites, based on observed user interests. In order to keep the scope of the project small, MyFido users will be confined to reading articles from Cnet.com. I chose Cnet.com as the target site for this proof of concept as the structure / layout of it's pages is clean and simple. In addition to the news article layouts, the site has a large host of technical articles from all categories of tech / nerd news, and it is also a popular tech news website.

The suggested articles will be chosen from RSS feeds from CNET.com. These suggested documents will be presented to the user in the form of an RSS feed.

My secondary objective is to explore using natural language processing and other artificial intelligence techniques to build personal user profiles based on these observed interests, which could be potentially used for other purposes, such as market research.

1.1 Basic System Functionality

The user logs into MyFido via the desktop GUI. The user clicks the “Observe” button to activate the Observer component. By activating the Observer component, a new instance of an embedded web browser will open inside the MyFido GUI. The user proceeds to surf to their favorite site. For the purpose of developing this proof of concept application, the user will read news articles from http://www.cnet.com.

The “Observer” Component

First the user finds an article that peeks their interest. Once they read an article MyFido will attempt to parse out all the “noise” from the document. Any information that is not directly related to the article is considered noise (i.e. navigation, advertisements, reader comments etc.) The Observer component stores a “noiseless” version of the parsed article (HTML document) temporarily on the local file system.

A user’s browser session is complete once they close the MyFido GUI, they log out of the system, or they activate the Context Analyzer component.

MyFido only parses and stores articles from websites it “knows”. For this proof of concept the application can only identify articles from Cnet.com; it will not attempt to store and parse documents from any other site. For testing purposes the user will navigate to articles by copying and pasting a URL into the embedded browser. The articles read are randomly chosen from a list of 50 past and present Cnet articles.

The “Context Analyzer” Component

After their browser session is complete, the user will click the “Analyze” button, which activates both the Document and Context Analyzer components. The Document Analyzer builds a schema that is used to instruct MyFido on how to parse a document based on the website the document is from. Each supported website is
uniquely identified by a system defined site Id. All system supported websites will have schema information stored in the MyFido database. Document schema information includes the class and id attributes used in the HTML / XHTML article layout for all the non-noise document elements, such as author, date published, headline, and article body. The schema data is then passed to The Parser component which parses each cached document according to this schema. Each article is stored as a system defined document object. These document objects are then passed to the Context Analyzer.

The Context Analyzer component is responsible for analyzing the grammatical structure of the documents as well as driving the analysis of the context of “observed” articles, in an attempt to determine what the articles are about.

System-defined rules, which are defined as patterns of parts-of-speech tags, are stored in the MyFido database and are used to identify noun phrases and keywords that are important in determining the context and subject matter of observed articles. These keywords and phrases will then be matched against a system-defined category hierarchy lexicon of contextual computer and technology related terms. All terms and phrases not found in the system-defined lexicon will be added to MyFido's knowledge base as a means to facilitate a “learning” feature. Essentially, MyFido will “learn” new words. The more it's used the more words it will learn. This will be further explained later in this paper. Please note, this is functionality will not be included in the proof of concept version of MyFido.

The matched terms and phrases as well as related terms and phrases will be stored as that particular user's personal interests profile. This profile will continue to be updated through the user's use of the system. MyFido then attempts to make suggestions to users of articles that are an exact match to observed topics of interest as well as related topics of interest.

**The “Fetch” Component**

Lastly, the Fetch component is activated by the user clicking the “Fetch” button. This feature can be used anytime after the user has had at least one Observer session and the articles read during the session have been analyzed by the Context Analyzer.

Once the Fetch component is activated, MyFido sends a collection of cached system supported RSS feeds to the context analyzer. These cached RSS feeds are fetched from the web during system start up, before a user logs in. These feeds are parsed and stored in system defined RSSFeed and FeedItem objects that represent the feed and the collection of articles that are contained in that feed.

The context analyzer uses the same technique of tagging, tokenizing, chunking, and noun / noun phrase detection that it performs on articles read during a user's Observer browser sessions. The detected nouns and noun phrases are then matched against MyFido's category hierarchy. The matched and related terms are then stored as a property of the FeedItem. This property is then matched against the user's user interest profile. When the user activates the Fetch component, articles matched against the user's profile will be presented to the user as a custom RSS feed. The user's user interests for both Fetch and Observer sessions are stored as historical data in order to give MyFido a means for learning user interests over time.

After reading articles presented by the Fetch component, the user is asked to select the check box next to all the articles they enjoyed reading. This feedback is then saved at the end of the user's Fetch session as snapshot data, which is later used to build part of the user's user interest profile.
1.2 Testing Methodology & Evaluating Effectiveness of System

**Evaluating Effectiveness / Accuracy**

Users of the system will be asked to perform at least one Observer browser session, which includes running the Context Analyzer to analyze articles read during that session and multiple Fetch sessions. MyFido will passively collect statistical data regarding MyFido's user suggestion system accuracy, with regard to its ability to suggest articles based on user's observed interests over time. The system will also collect statistical data regarding the total number of fetched RSS feed articles that it is able to successfully analyze and match against it's category hierarchy knowledge base.

All statistical data will be collected during a trial tester's Fetch session. After all trial tests are completed, a summary of system accuracy will be assessed. The initial goal for this proof of concept application is to prove that in this current iteration, MyFido is able to aggregate a custom RSS feed where on average the user likes more system suggested articles over time than randomly suggested articles. The long term goal is for MyFido to be able to aggregate an RSS feed where users on average like 80% of system suggested articles.

**Testing**

MyFido's effectiveness will be determined by manual evaluation of a group of three testers. There will be one long term user, one intermediate time user and one short term user. Each tester's technical interests and knowledge will vary between novice (no prior interest) to expert level.

**Note Regarding System Design**

MyFido is a modular software system that is designed as a desktop client application that will utilize a centralized system of machines for data processing and storage. However, for the purpose of building this proof of concept application, all data processing and data storage will take place on one machine.

Also note, there are other components besides those mentioned in the previous sections that give MyFido it's functionality. Components such as the Knowledge Access component, which is a part of “MyFido's Brain,” builds the category hierarchy graph from knowledge base data stored in the MyFido database, and the User Management system, which helps MyFido not only manage its users but their personal interests profiles as well.
This diagram shows MyFido’s current system design implementation. In this proof of concept system, the application as well as the database all live on the user’s (development) machine. MyFido’s components are designed and implemented using Object Oriented development practices; hence, this system will easily port to a client/server implementation. This also makes implementing new features easy.
This diagram shows the proposed design of a client / server system implementation. In this system, the bulk of the document parsing and processing will be handled on a system of centralized database and application servers. All parsed, observed documents would be stored on the local file system of an application server, using the system directory structure currently implemented in the proof of concept version of the system.
2. What Is Natural Language Processing?

Natural Language Processing is used in a variety of applications to help determine the context of spoken as well as written text. NLP techniques are critical to helping MyFido determine the context of articles read by users during a browser session. Without the use of NLP, MyFido would have no way to determine what kind of articles it should fetch for its users.

Because of the important role Natural Language Processing plays in the MyFido system and its importance to the field of Artificial Intelligence, I felt it was necessary to give some background information regarding what Natural Language Processing is, while also discussing the evolution of the discipline as well as some examples of applications in the research and commercial space employing the use of NLP.

2.1 About Natural Language Processing (NLP)

Natural Language Processing is a sub-field of Artificial Intelligence that studies language processing and language generation of natural human languages. Linguistics, computational linguistics, computer science and cognitive psychology all contributed to shaping modern natural language processing [1]. Though NLP as a discipline is quite old, for years it was plagued with over selling of the possibilities of what could be accomplished at the time and lack of understanding of the complexities of human language [2].

History of Natural Language Processing

The earliest contributions to the “birth” of NLP were produced in the mid to late 1940s. One significant idea that sparked NLP research in the United States was introduced by Warren Weaver in 1949. Weaver was involved in code breaking during World War II. His idea was simple: given that humans of all nations are much the same (in spite of speaking a variety of languages), a document in one language could be viewed as having been written in code. Once this code was broken, it would be possible to output the document in another language. From this point of view, German was English in code [2]. Weaver, as well as other researchers in the field, proved to be unsuccessful in producing a machine translation system that yielded tangible results. The technique employed in most of these systems was using dictionary look up for appropriate words for translation and reordered the words after translation to fit the word-order rules of the target language, without taking into account the lexical ambiguity inherent in natural language [1]. The lack of linguistic theories at the time made machine translation a very difficult task.

In 1957, one language theory emerged in the field of Linguistics that made Machine Translation a realistic possibility. Noam Chomsky's publication, *Syntactic Structures*, introduced a rule based approach to studying syntax. Generative grammar of language attempts to give a set of rules that will correctly predict which combinations of words will form grammatical sentences [3].

Chomsky's work inspired other theoretical development, which in turn led to the development of prototype systems employing these principles and testing their effectiveness. "During this period syntactic analysis was the focus of many of these systems. Weizenbaum's ELIZA was built to replicate the conversation between a psychologist and a patient, simply by permuting or echoing the user input. Winograd's SHRDLU simulated a robot that manipulated blocks on a tabletop. Despite its limitations, it showed that natural language understanding was indeed possible for the computer. PARRY attempted to embody a theory of paranoia in a system. Instead of single keywords, it used groups of keywords, and used synonyms if keywords were not
found. LUNAR was developed by Woods as an interface system to a database that consisted of information about lunar rock samples using augmented transition network and procedural semantics” [1].

### 2.2 Applications of NLP

Advances in computer technology, specifically in terms of processors and memory, the advent of the Internet, as well as the ever growing amount of available electronic text, has led to rapid growth in the field of Natural Language Processing in recent years. [1]

There are four categories of approaches in Natural Language Processing: symbolic, statistical, connectionist, and hybrid. Symbolic and statistical approaches have been along since the birth of NLP. Symbolic and statistical approaches have coexisted since the early days of this field. In the 1980’s, statistical approaches regained popularity as a result of the availability of critical computational resources and the need to deal with broad, real-world contexts [1].

**Examples of NLP Applications**

Natural Language Processing has been successfully used in software systems that performed tasks such as open text processing (text mining), data extraction, ontology matching and determining context of data. Through my research, the use of NLP tools to determine the context of data seems to be the most popular use of these tools. MyFido makes use of a Natural Language Processing software library with tools for performing sentence detection, token parsing, parts of speech tagging and a host of other things. It uses these tools to help it determine the context of an article as well as fetched RSS feeds.

Below is a brief discussion of some other applications developed in the research space that demonstrate practical use of Natural Language Processing techniques, to help determine the context of data. Though there are NLP based commercial applications, since this discussion is in the context of a research project, I would like to confine this discussion to the research space.

One application that makes use of NLP tools for determining data context is by Qingyang Xu and Wanli Zuo of Jilin University in China. They developed a system that uses NLP techniques to extract the context of Hyperlink text. Link extraction is used in various information retrieval tasks on the web, but the authors propose that the link context extraction can be improved with the use of NLP tools to help filter “noisy” words that surround the link in an attempt to extract the context of the Hyperlink. [4]

Their “system uses a two pronged novel extraction model. The first analysis step, analyzes the web page structure to locate the content cohesive text region and potential relevant header or header like tags... In the second analysis step, is an English language parser. [4]

The author's extraction model hopes to mimic the browsing habit of human users that allows them (with some knowledge of the domain) to predict the context of an article that a Hyperlink leads to. [4]

Another application that makes use of NLP tools is <!metaMarker>. Developed by Solutions-United Inc, <!metaMarker> is a software application that uses Natural Language Processing and Machine Learning techniques for automatic metadata extraction. It makes use of domain-independent and domain-dependent extraction techniques to extract and process textual personalized data such as emails, discussion group postings or chat group transcriptions. By extracting this information, the system constructs user profiles automatically. It proved to be 90% accurate in extracting data, showing that NLP techniques are viable for
data extraction and generating user profiles [5].

In addition to extracting general metadata, this application attempts to extract “situational” metadata such as “dislike”, “interested”, or "not interested," for example. These elements will help determine the authors intent. [5] Essentially this <!metaMarker> not only uses NLP tools to extract metadata but to extract the tone and intention of a document's content. This application uses sentence detection, parts of speech tagging along with other NLP techniques to extract this metadata.

2.3 General Challenges of NLP Projects & Developing MyFido

MyFido was born from a conversation I was having with a friend about all of the online reading I do. Usually what starts out as a casual session of reading and web surfing, turns out to be hours of new discoveries and tons of new knowledge for me to internalize and process. I joked with my friend that it would be great to have an application that would watch me surf the web, determine my topics of interests, gather articles and papers from the web and present those articles as a custom RSS feed. Basically it would do the surfing for me based on what it thinks I like.

My interests in artificial intelligence led me to pursue making this application a reality, but I had no idea if it was possible to build such an application, and even after my initial research that proved this was possible, I had no idea of how to make it a reality.

When I started exploring building MyFido, the challenges and problems that needed to be solved became very apparent. However, I did not discover these challenges all at once. At every stage of development (especially in the early stages) I would run into a problems such as: how would I give MyFido the functionality to identify the structure of a particular web site's articles or what is the best way to store and present the knowledge base, and how will I represent the relationships between the terms in the knowledge base.

I also discovered other researchers were thinking of ways to combat information overload by implementing user suggestion systems that attempt to passively identify user interests by using Natural Language Processing and other AI related techniques. Though the specific problems these applications are attempting to solve are different and the approaches vary, I found the challenges that need to be addressed were similar across the board. Defining the domain of knowledge for the application or software agent, providing a mechanism to observe user interests, analyzing the structure of web pages, determining the context or “meaning” of words and phrases in a document, making suggestions to users based on observed interests and storing the application’s “knowledge” of the world around it are just a few of these issues that need to be addressed.

In the sections below I will further discuss these issues with respect to developing MyFido. I will also note research projects that inspired some of my approaches to these issues.

Keep The Domain Small Stupid: Scope and Knowledge Domain Management

Software can be designed to be intelligent, but it's intelligence is modeled after human intelligence and our approaches to problem solving. A web browser or word processor has no clue what a computer is for example. They have no reference or knowledge of this term unless we as developers give it to them.

When I first approached Professor Benjamin with the idea for this project, he constantly reminded me to keep
the scope small. I had no clue what he really meant by that, and in the beginning it was very frustrating. I figured, how hard could this actually be? Without taking the time to think, sure I know what Google is but how would I get a software application to “know” what Google is. I took for granted the “God’s eye” view of problems and documents we humans have. The typical human reader, uses its knowledge of the world around it to comprehend and determine the context of a document. I realized I had to mimic this behavior in MyFido. In order to figure out the context of an article a user read, MyFido needs knowledge of the outside world, otherwise it would not be able make suggestions to users based on what it observed them read. So I was stuck with the question: how would I give MyFido knowledge of the outside world so that it would be able to analyze the context of observed articles?

They say lessons learned on your own are the ones you never forget, and this still holds true with things I learned while planning and implementing this project. What Professor Benjamin was trying to drive home to me was to keep the scope or domain of knowledge that I would like to focus on small. This was not only helpful in terms of the ease of developing a proof of concept, it is also necessary in the initial development of Natural Language Processing applications. My initial idea was to be able to determine the context of articles on a broad scope of computer and technology related topics. After developing a small hierarchy of terms and phrases for MyFido’s initial knowledge base, I now understand this would have been a daunting task to say the least. Though in it’s initial implementation the knowledge base is small, given a good base it can be grown organically through an implemented learning system. The key is to start small and build toward a larger knowledge base over time.

Using a Category Hierarchy To Represent MyFido’s Knowledge Base

After establishing the need for a knowledge base, the following question arose: Should I build a system defined knowledge base or go with an existing solution? If you do build your own, how will you store this data and how will you represent it programatically so that it is easy to search?

My initial approach was to use WordNet (see Section 3.1: Technologies & Tool > Natural Language Processing Tools), as Professor Benjamin warned me that building your own knowledge base can be a very large undertaking and is essentially a project in its self. “WordNet® is a large lexical database of English. “ I needed to programatically access WordNet as I would need to match my terms and phrases from observed articles against it. Also, because I was sticking to the domain of technical and information technology terms, I needed a means to evaluate WordNet for the volume of technical and IT terms it contained. I used JWI, a Java based library for accessing WordNet (see Section 3.1: Technologies & Tool > Natural Language Processing Tools) WordNet can be accessed via the web as well as via dictionary files. To evaluate WordNet, I took two files of random technical terms, iterated through the lists and tested each term against WordNet. The library returns true if there is a match.

From a list of 7556 terms, WordNet matched 1372 terms (18% of terms).
From another list of 982 terms, WordNet matched 393 terms (40% terms).

I wasn’t satisfied with the volume of technical terms and keywords found in WordNet. Next, I tried to find a similar English language lexicon that had a focus on technical terms. I was unable to find such a lexicon. I hence decided to build my own knowledge base.

I decided to take a different approach from that found in WordNet. WordNet stores definitions and sense of words. It is essentially a dictionary of synonyms. I came to the conclusion that although it is necessary in some NLP projects to have the definitions and sense of words, it was not currently necessary for MyFido. Though there is a possibility of using WordNet as a knowledge base to match against, in order to add knowledge of general non-technical words and phrases to MyFido as a system enhancement. Please see the Conclusions section of this paper for a further discussion of this possible enhancement.
I settled on developing a Category Hierarchy as the ideal solution for representing the MyFido knowledge base. Yannis Labrou and Time Fini from the University of Maryland used Yahoo!'s categories as an ontology for describing documents [6]. They used Yahoo!'s categories because they are arranged in a hierarchical index; it is a pre-built index that could be scrapped and stored locally. They also concluded that “computer programs ...benefit from a standardized way of describing the content or the nature of things...the idea was to not use semantically deep descriptions of things but rather a headline-like accounts of their nature.”[6]

Another research team from Depal University used “a modular concept hierarchy” in a system used to enhance web queries by building “a client side web agent that uses domain specific concept hierarchies and interactive query information to create a less ambiguous” web search query.[7]

By using a category hierarchy to represent MyFido's “hardwired” knowledge base, it allows me to map relationships between various terms; these relationships could then be used to further describe the nature or “meaning” of these terms.

MyFido's initially “hardwired” knowledge base is represented as a directed acyclic graph. All the terms and phrases in the graph are linked by a predefined set of relationships. These relationships are helpful in making suggestions to MyFido users of topics that are related to their observed topics interest. They are also helpful in some cases for defining what a term or phrase is. Because graphs are natural models for representing arbitrary relationships among data objects, it was an ideal solution for representing a system defined category hierarchy of knowledge for MyFido. [8] The idea to use directed acyclic graphs was borrowed from the world of ontology matching, where large ontologies are stored as directed acyclic graphs and decomposed from “top to bottom”. [9]
Figure 2.1 Example of Terms & Phrases Mapped in the MyFido Category Hierarchy Graph

This figure shows an example of terms and phrases mapped in MyFido’s category hierarchy graph. All the arrows represent relationships between the terms and phrases. The phrases in green are base entities.

Some examples of relationships defined in the above graph snippet:

- Video Games are created by EA Sports
- Video Games are created by Nintendo
- Video Games are created by Atari
- Wii is created by Nintendo
- Video Games are a type of Software
- Rockstar Games are a type of Software Company
- Video Games are a type of Software Application
- Machine Learning is related to the field of & associated with the field of Artificial Intelligence
- AI is an alias of Artificial Intelligence

This category hierarchy contains 160 keywords and phrases that are all related to information technology and computer technology. All terms and phrases are mapped by various types of relationships. There are four tables in MyFido’s relational database that make up the category hierarchy. First there is a table to store a master list of terms and phrases; all terms are classified as a person, place or thing. This table acts as the knowledge base “dictionary”. The second table defines all classification types. The third table is a relationship type table that defines all the types of relationships that exist in the category hierarchy and lastly there is an edge table that stores all graph edges. The knowledge base graph, currently contains over 544 edges. The edges are linked together by their relationship types.
RelationshipTypeId | RelationshipNames
--- | ---
1 | is related to the field of
2 | is an alias of
3 | is associated with (term or phrase)
4 | is a type of / is a
5 | is created by
6 | is referred to as

**Figure 2.2  Graph Relationships As Stored In The MyFido Database**

Overtime MyFido’s knowledge of the outside world will grow the more the system is used. Because this could decrease the graph’s effectiveness over time, system enhancements can be made to combat this and will be discussed later in this paper. Any terms and phrases observed in articles that are not stored in it’s category hierarchy will then be added to that master database table that stores all it’s phrases and keywords. This will allow MyFido to gain further knowledge of technical terms as well as more general terms in the English language. For the time being all relationships will have to be mapped manually over time by a human being. It is my hope that in the future the relationship mapping will not be a manual process, but that is another project all together.

**Figure 2.3  Sample Text - Examples of Similar (matched) Terms and Related Terms**

*This figure is an example of actual output (debugging) text; it shows the similar (matched) and related terms found when a collection of keywords and phrases parsed from observed documents is passed into the KnowledgeAccess class and tested against MyFido’s category hierarchy graph. This is some of the output for the article chunked in Figure x.x.*

*Note: termToTest is the term or phrase passed in to the KnowledgeAccess class’ findRelatedTermsInCategoryHierarchy() method. These terms and phrases were passed into this method as a collection of strings. These were found by rule matching and logic found in the ContextAnalyzer class. Associated terms and phrases found will be used to provide suggestions to users, by suggesting articles that are related to or are “about” these topics.*

termToTest: Web browser( 37 )
  get associated end vertices: software application
  get associated end vertices: software
  get associated end vertices: Internet
  get associated end vertices: web
  get associated end vertices: world wide web
  get associated end vertices: world wide web
  get associated end vertices: web
  get associated end vertices: online
termToTest: Internet Explorer( 112 )
  get associated end vertices: Microsoft
  get associated end vertices: web browser
  get associated end vertices: software
  get associated end vertices: software application
termToTest: Internet( 38 )
termToTest: Web( 40 )
  get associated end vertices: world wide web
termToTest: Firefox( 111 )
  get associated end vertices: Mozilla
  get associated end vertices: web browser
  get associated end vertices: software
  get associated end vertices: software application
termToTest: Apple( 70 )
  get associated end vertices: software company
termToTest: Safari( 100 )
  get associated end vertices: Apple
  get associated end vertices: web browser
Researching the history of natural language processing as a field, it's unsuccessful research efforts, why the efforts failed and what was learned from those failures, really helped drive home the idea of domain specific knowledge and the importance of keeping the domain of knowledge as small as possible, especially when developing and implementing knowledge based NLP software. Gathering and structuring this knowledge can be as challenging as writing the application itself. Devising a schema for gathering, structuring and storing this knowledge so that it is usable and scalable is not a small task. Just as humans as infants can't know everything about the world around them all at once, neither can a software agent or application.

What's An Operating System?: Giving Software The Ability to Analyze The Context and “Meaning” of Text

Software and hardware by nature lack the ability to “understand” human constructs and abstractions. Hence, “the goal of NLP...is to accomplish human-like language processing.”[1] This is often addressed by developers and researchers by attempting to give software the ability to analyze and determine the context of documents, articles etc.

MyFido does two types of analysis to determine the context of observed articles and RSS feed article summaries. The system's Document Analyzer analyzes the structure of an HTML document from a system supported website, based on document schema data stored in it's database. This schema is mapped by a site Id. I will discuss the Document Analyzer further in a later section of this paper. MyFido's Context Analyzer determines the context of the document.

MyFido's ability to determine the context of an article and RSS feed article summaries is one of it's primary functions; without this ability it would be unable to make suggestions to users based on their interests. This was one of the more challenging problems that needed to be solved, as well as the most important one. In essence MyFido basically needs to be able to “read” or analyze an article a user read and determine what it is about, then pull out the main topics of interest and store those in a user interests profile along with related topics.
MyFido's Context Analyzer

The Context Analyzer is the heart of MyFido. It attempts to model the steps that human readers follow in order to read an article and understand what it was about. It gives MyFido the ability to determine the context and the "main idea" of a document or article by using a tiered system of analyzation, along with a rules system to flag keywords and phrases in a document. The first tier of analyzation acts as the "reader", it analyzes the grammatical structure of a document, and the second tier acts as the “context analyzer” where it attempts to determine the topics covered or “main idea” of the article.

After a user reads an article, MyFido stores a “noiseless" version of that article as an HTML file on the local file system. When the user runs the analyzer, it first parses the observed document based on the document’s schema for the website the document was from. The document and it's elements are stored in Document and DocumentElements objects native to MyFido (see Section 2.2: Applications of NLP > Software Doesn't Have Eyes: Observing user interests...). Once the document is stored into these native MyFido objects, the document's elements are parsed into sentences and tokenized. Next, all tokens are tagged by a Parts Of Speech (POS) tagger and sentences are then broken up into noun and verb phrase chunks by a Treebank Chunker.

All sentence detection, chunking and tagging is handled by OpenNLP, a JAVA based NLP library (see Section 3.1: Technologies & Tool > Natural Language Processing Tools). OpenNLP natively supports the use of Maximum Entropy (MaxEnt) statistical modeling for POS tagging as well as sentence detection. MaxEnt modeling is used frequently in NLP, especially for POS taggers. Often taggers, including ones included in the OpenNLP library, are trained using annotated corpus such as the Wall Street Journal corpus (from Penn Treebank project) “to learn either probability distributions or rules and use them to automatically assign POS tags to unseen text.” [10] The principle of Maximum Entropy is one that attempts to set up probability distributions based on partial knowledge, hence making it ideal for use in systems that will process documents that contain words that did not appear in its training data.[10,11]

Once all tagging and chunking is completed, the second tier of analyzation flags keywords and phrases in order to determine the topics covered in the article. These keywords and phrases will later be cached and matched against the native Category Hierarchy. The "chunked" version of a document is then matched against a set of system-defined, noun phrase detecting rules. There is system logic to also determine nouns as well. These rules are 3 (3-PosGram) and 2 (2-PosGram) gram patterns of parts of speech tags that represent common patterns of noun phrases. These rules were created by using informal, visual observations of POS tagged test documents along with my knowledge of the English language.
Figure 2.4 Examples of POS-Tagging, Chunking of an observed document (output data / debugging text)

This figure shows debugging output text from an observed article. This document has been tagged, tokenized and chunked by MyFido’s ContextAnalyzer. Also shown in this figure are examples of noun phrases found by matching chunked documents against system defined rules. For a list of MyFido POSGram rules see Figure 5.16.

OpenNLP’s Maximum Entropy based name finder wasn’t as accurate as I would have liked with the given models that it came with out of the box. So I implemented system logic and rules that allows MyFido to detect proper nouns and other nouns in the article. This logic implements keyword matching.

My approach was loosely inspired by work done by Eric Brill and Mitchell Marcus, where they created a system “that is based upon the assumption that if two adjacent part of speech tags are distributionally similar to some single tag then it is probable that the two tags form a constituent...If a single tag is substitutable for a pair of adjacent tags, it is highly likely that the pair of tags makes up a syntactically significant entity i.e. a phrase.” [12]
Software Doesn’t Have Eyes: Observing user interests and analyzing the structure of a web page

As human beings we have a “God’s Eye” view of problems. Meaning we have the ability in some cases to see the solutions to a problem just by viewing it, without any additional effort to solve it. This can be true for simple search problems for example. This is also true in the case of reading documents. We can view the layout and structure of the document and it’s elements immediately without heavy analysis. Unfortunately, computers aren’t as lucky as we are. Even robots equipped with optical sensors can’t solve problems presented to them using these sensors without a developer giving it knowledge of the problem and knowledge of what they are “looking at.”

Hold the noise! Parsing HTML pages; keep only what you need

In order for MyFido to determine the context of an article and in turn offer suggestions to users based on their interests, the system must be able to identify and analyze the structure of a web page / online article it observes a user reading. One of the challenges with analyzing an HTML document, specifically when your goal is to determine it’s context, is eliminating the noise data that all web pages to a degree contain. Navigation links, comments sections, advertisements etc. are all examples of what I classify as noise data. This unwanted data could skew the results generated by the Context Analyzer and confuse it by providing it with additional information that has no significant relevance to the body of the document the user is reading.

To solve this problem I wanted to mimic the “God’s Eye” view of the HTML document analysis problem in MyFido by building intelligence into it’s parser that would allow it to recognize a web page from the particular website the user is reading, hence giving it the ability to parse only the relevant article data. To mimic this “God’s Eye” view of an observed web page, I thought about how we as humans on a basic level see documents and wondered how I could reproduce this in MyFido. I concluded that documents are nothing more than a collection of document elements, and that there are common characteristics; such as headline, author, publication date, and body of the article that all news and blog articles are comprised of.

After identifying these characteristics, I needed to represent them programatically, as well as in a relational database, as a document schema that could be recalled and used at any time. Because most web sites that provide news content have article pages that are generated dynamically, using the same template for pages over and over again the document schema approach would make it easy to document the structure of these pages and recall them as reusable schemas. The document schema gives the MyFido parser information regarding the structure of a particular site’s article pages and hence giving it information on how to parse a document to not only eliminate noise data but to parse it for further analysis.

MyFido's internal representation of documents and document elements

MyFido uses internal, system defined, generic objects that represent documents and document elements. MyFido views a document as an object that contains a collection of document element objects, among other properties such as documentId, siteId etc. The document element object has properties that describe that element. Properties such as the HTML tag name, HTML tag symbol and value (text parsed out of the node element) are native to MyFido’s document element object. These objects serve two purposes; to cache parsed data from an observed article as well as hold document schema information for MyFido supported website news / blog articles. They are all implemented as interfaces, then generic objects that implement these interfaces.

Though I could have used XML and an XML schema to represent document and document element objects, which arguably is a more standard approach, I felt that parsing data from one mark up language and storing it
into another mark up language was a bit overkill. Furthermore, using the XML approach to represent an object would be more of a necessity if there was a need to allow MyFido a means to share its representation of documents and its elements outside of the system; for example if I provided a web service to share these objects with other systems. In such a case, because of MyFido's modular design an XML based solution for such a purpose could be integrated into the system. The Document and DocumentElement objects can be extended so that each system supported website can have its own customized object modules, in order to reflect properties that are native to that particular site for example.

Giving MyFido the ability to identify articles from specific, system supported web sites: Gathering schema information

First, I needed to gather relevant information regarding these common document characteristics for all system supported web sites. For this proof of concept the only system supported site analyzed is Cnet.com. To gather the document schema information, I manually analyzed the DOM structure of a Cnet.com news article web page, using IE developer Toolbar. Any tool, such as FireBug, that allows you to examine the DOM structure of a website, could have been used for this as well. Next, I documented the names used for class and id attributes assigned to HTML document elements that held relevant article data such as div, p and header tags. I also made a note of these tags and HTML elements.
Lastly, all these characteristics / attributes native for Cnet news / blog pages will be stored in a relational database and recalled when needed via MyFido. Listed below are some of the primary benefits to this approach:

1. We will now have the ability to add support for various news sites in the future...any site that has dynamic content and where the layouts of the article pages don't change.

2. We will be able to note and store where the actual articles on these pages start and end, in order to build more intelligence into the parser so that we will only parse article data and eliminate having to
parse noise, aka the data we don't care about on the page, such as navigation, blog post comments etc.

Putting it all together: The Parser & Document Analyzer working together

MyFido's parsers makes use of the open source, Java based HTMLParser library (see Section 3.2: Technologies & Tool > Other Technologies). The HTMLParser library allows you to specify tags and class names, ids attributes among other things to provide a custom, granular means of parsing each system supported site's documents. Each system supported web site has it's own document schema stored in the MyFido database. The Parser component uses this schema information as parsing instructions.

The Parser component has two functions. It's used to initially parse out the noise data from observed documents, and it is used to parse all document elements from observed articles for further analysis. All initially parsed, observed documents are cached on the local file system; in this proof of concept application the user's local file system is used. In the production version of this system, these documents will be stored on a centralized file system. The directory structure to store observed documents reflects that all documents and system activity is mapped to the individual system user.

![Diagram of Directory Structure To Store Observed Documents](image)

This diagram shows the file structure to store observed documents on a local file system. This file system can be the user's machine, as in the proof of concept implementation, or on a centralized server's local file system.

Each user has their own folder, that acts as the root directory that holds their observed documents. The directory is named `user_userId`, where the userId is the user's unique identifier stored in the MyFido database. Each system supported site has a folder in each user's directory, where observed documents from that particular site. The site directories are named `site_siteId`, where siteId is the site's unique identifier stored in the MyFido database. Lastly,
each document is named by its `observeddate_uniquecount.html`, where the `uniquecount` is a number iterated by one and acts as a unique identifier for that particular document.

The Observer component saves an initially parsed version of each observed article on the local file system. The initial parsing is done based on the domain name of the web site the document is from, the corresponding siteId for this domain, and the document element that MyFido uses to identify the ArticleContainer (the HTML element that contains all document elements relevant to the article). Information regarding the ArticleContainer is stored as a document element, in the document schema for the corresponding siteId. The parser uses this information as instructions to parse noise data out of each document the user reads.

The second round of parsing is started by the Document Analyzer component. When the user clicks the “Analyze” button on the desktop interface. A new Document Analyzer object is initiated, and a schema for each system supported site is pulled from database and stored as a Document and DocumentElement objects. These shell objects are then passed to the Parser component and each document from the corresponding site directory on the file system is parsed accordingly using the schema for it’s particular site. Once all the documents and their elements are parsed, each document is added to a collection of document objects that is then passed to the Context Analyzer, where the article's grammatical structure and context are analyzed.

Possible issues with this approach...

By using the names of the classes and id attributes (used to identify styles in CSS stylesheets), we are able to uniquely identify the divs, p tags etc. that we want to parse vs the ones that contain noise data. This not only makes parsing the noise data easier, it also makes our parsing out the noise data more accurate. This approach could also lead to a problem if these class and id attributes' names change, then our parser will not know what to do with the document it needs to parse, but this could be solved by eventually building a web or admin interface that would allow the maintainers of the system to update the names of these attributes accordingly, either with programmatic or human page analysis every so often.

I Know What You Like: Making Suggestions To Users Based on Observed Interests

One major function of knowledge management systems and systems that attempt to combat the issue of information overload is to make suggestions to users by passively or non-passively observing users. Suggestion systems attempt to create a profile of the user's preferences and interests, and then attempt to suggest products, articles etc. based on these observed interests. Such systems are nothing new to consumers on the web. E-commerce sites such as Amazon.com and web based movie rental company, Netflix, use predictive algorithms, observations of a user's purchasing / rental habits and observations of their searching habits on their sites, in order to make suggestions of products / movies they may be interested in. Suggestion systems can be very helpful to users by presenting products, articles etc. to them that they may have otherwise never been exposed to.

Giving software the ability to observe a users actions, interests, and preferences

There is a unique challenge in building software that attempts to observe a user's interests by observing their behaviors and activities on the Internet. There are many approaches to implement this functionality, but many of them need a means of analyzing the context of data collected during observation in order to build a profile of a user's interests. The type of data as well as the source of the data collected during these observations varies from system to system, and helps to define the particular approach of the system's developers. In this section I will discuss a few approaches to implementing user systems that I discovered in my research, and in some way influenced my approach in developing MyFido's user suggestion / RSS feed aggregator
One approach to observing user interests is by recording and learning “surrogate” tasks of the user. By recording these surrogate tasks of the user, the system attempts to determine user interests by observing their behaviors during a particular browser session. [13] Surrogate tasks such as “number of hyperlinks clicked on by the user and the amount of scrolling (or mouse activity on a page) the user performed, and whether the user book marked the page;” [13] would be recorded, and depending on the level of user activity on the page, it would be marked as a page that the user has a negative or positive interest in the page (and the topics / subjects covered on a page). The HTML text of a page and the hyperlinks the user clicks on are also recorded (and analyzed) and used to determine the user’s level of interests in topics / subjects of the particular article / document.

Another notable approach I encountered in my research was a system called <!metaMarker>; a system that uses Natural Language Processing and Machine Learning to extract and process “textual data such as emails, discussion group postings, or chat group transcriptions,” as well as “explicit and implicit metadata elements including proper names, numeric concepts and topic/subject information.” It uses this data to automatically create a user preference profile.

Lastly, one interesting approach to observe user interests was developed by Fabio Gasparetti and Alessandro Micarelli at Roma Tre University. In this system they use data collected from web browsing histories and web browsing activities to create a user interests profile. They use an algorithm “that given a single browsing session extracts the textual content from Web pages and processes it in order to find information related to the user’s current needs.” [14] The textual data collected is thought to be related to the main topic of the page; this data includes, “anchor text of the link, the title of the pointed page and the context of the selected link in the current page.” [14] The text that contains this link is set as the context of a link. When the context of a link is established, there is a search of the current HTML document for similar text, if it is found then that text is cached in a buffer and that cached data is said to be “information related to the users needs.” [14] Though our approaches are different, I was influenced by some of the ideas presented in their research, especially those regarding eliminating irrelevant (noise) data from a page when you are trying to determine the context of a HTML document / article.

Using passive & non-passive observations to create a user interests profile

MyFido uses both passive and non-passive observations in order to generate a user interest profile which is used to suggest articles of interest in the form of a custom aggregated RSS Feed. The challenge in passively observing a user reading news articles on the web, was caching the article for later processing as well as filtering out the noise or irrelevant data from the article before caching the document. The Observer and Fetch components provide MyFido with the passive and non-passive observation functionality respectively. The Fetch also implements MyFido’s user suggestion system functionality.

The Observer, Context Analyzer, and Fetch components enable MyFido to create user interests profiles based on observed user interests. A user interests profile is made up of two masters lists of topic ids that represent all the matched and related topics of interest for a particular user. The user interests profile also includes a profileId and userId. The two master lists are created by aggregating archived historical topic of interest data. This snapshot data is recorded for each observer and fetch sessions for all system users. These user interests profiles are stored in the UserInterestProfile table of the MyFido system database.

Every user will have two set of comma separated lists of topicIds stored in the UserInterestProfile table. These lists are the ObserverMasterList and FetchedMasterList; they are created by aggregating Observer and Fetch snapshot data. The Observer snapshot data is all topicIds that correspond with articles that the user read
during an observer browser session, and the Fetch snapshot data is all the topicIds that correspond to
RSSFeed items (articles) that the user said they read and liked during a Fetch session.

<table>
<thead>
<tr>
<th>Id</th>
<th>UserId</th>
<th>UserObservedTopicsOfInterest</th>
<th>FetchedFeedBackTopicsOfInterest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>106,31,30,7,6,34,35,112,69,37,64,61,27,28,49,43,121,8,9,6,14,2,89,17,83,84,38,40,39,41,3,53,60,67,111,1,03,113,42,70,12,13,5,4,137,32,132,130,131,140,36,86,85,90,87,151,149,145,124,120,100</td>
<td>30,7,6,96,27,28,61,69,84,83,43,42</td>
</tr>
</tbody>
</table>

Figure 2.7  Example of Observer & Fetch MasterLists Stored In UserInterestProfile table

In this example, user 1 has an observer browser session where he read 15 articles. The 15 articles are analyzed.
A list of topic ids (minus any duplicates) is compiled of matched topics and terms. The topicIds of these terms
are then stored in the ObservedTopicsSnapshot table. The is stored along with a date stamp of when the
session took place, the userId, profileId and an Id of the snapshot records. A similar table, the
FetchedTopicsSnapshot table is created to store history of topicIds that correspond with all the articles the user
read and liked during each user's fetch sessions.

The ObservedTopicsSnapshot and FetchedTopicsSnapshot tables store the snapshot data for all users. This
snapshot data is mapped by both a user's UserId and ProfileId. These snapshot tables are meant to provide a
crude learning system for MyFido. The system collects and stores topicIds for corresponding articles read
during observer and Fetch sessions. An algorithm is then used to to evaluate and combine all the topicIds
stored in the snapshot table into a master list. This master list is then stored in it's respective field in the
UserInterestProfile table. This also adds a crude weighting schema, as Ids in the snapshot tables must
appear at least twice in order to be added to the master list.

<table>
<thead>
<tr>
<th>Id</th>
<th>UserId</th>
<th>ProfileId</th>
<th>DataSnapshotTaken</th>
<th>UserObservedTopicsOfInterest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>09/03/09</td>
<td>1,5,78,109,207,13</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>09/05/09</td>
<td>5,6,109,13</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>09/10/09</td>
<td>14,12,63,100,15</td>
</tr>
</tbody>
</table>

Figure 2.8  Example of Observer Snapshot Data

In this example only Ids 5, 109 and 13 will be added to this user's Observer Master List (UserObservedTopicsOfInterest
column) UserInterestProfile table. An Id must appear at least twice among all the listed snapshot data for a given
user if it is to be added to either the Observer or Fetched Master Lists.

MyFido, Fetch: The user suggestion system...aggregating a custom RSS Feed based on user interests profile

The end goal of MyFido is to aggregate a custom RSS Feed for a given user, based on that user's observed
interests. The system suggests 21 articles at a time. These 21 articles are drawn from a pool of up to 106 *
articles that are taken from 5 system supported RSS Feeds. Once these five feeds are fetched from Cnet.com,
they are parsed and cached in system defined objects that represent an RSSFeed. In MyFido, RSSFeed
objects store a collection of FeedItems along with other properties that define an RSS Feed such as feed URL
and name.

* Note, the number of articles in a feed varies from day to day, though as of the date of this publication Cnet has not included more
than 21 articles per RSS feed.
<table>
<thead>
<tr>
<th>Id</th>
<th>SystemSupportedSiteId</th>
<th>FeedName</th>
<th>FeedURL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>CNET News.com</td>
<td><a href="http://news.cnet.com/2547-1_3-0-20.xml">http://news.cnet.com/2547-1_3-0-20.xml</a></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>CNET News.com - Personal Tech</td>
<td><a href="http://news.cnet.com/2547-1040_3-0-5.xml">http://news.cnet.com/2547-1040_3-0-5.xml</a></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>CNET News - Apple</td>
<td><a href="http://news.cnet.com/8300-13579_3-37.xml">http://news.cnet.com/8300-13579_3-37.xml</a></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>CNET News - Gaming and Culture</td>
<td><a href="http://news.cnet.com/8300-13772_3-52.xml">http://news.cnet.com/8300-13772_3-52.xml</a></td>
</tr>
</tbody>
</table>

Figure 2.9 Current MyFido System Support RSS Feeds

This table shows all current system supported RSS Feeds. This table reflects the feed data as it is stored in the MyFido database.

After all the system supported feeds are fetched, parsed and stored in system defined objects, a collection of RSSFeed objects are passed to an overloaded version of the initContextAnalyzation() method in the ContextAnalyzer where the title and description for each FeedItem (article) in the feed will be tokenized, tagged, and chunked. Next, all the detected nouns and noun phrases will be matched against the KnowledgeBase category hierarchy graph. The comma separated values of the matched ids will be stored in the FeedItem's TopicsOfInterests property. This is a similar process used to analyze observed articles.

Note, I chose to only analyze the FeedItems (article's) title and description to limit the data matched against the category hierarchy. This will also ensure that MyFido is only analyzing data that is related to the main idea of a FeedItem (article). Perhaps, when MyFido is able to determine the main idea of an article, not just the general topics discussed in an article it maybe possible to allow the context analyzation process for the Fetch component, to analyze the whole article for each Fetched FeedItem.

Selecting Suggested Articles

Once all FeedItems for each RSSFeed object have been analyzed, they are ready to be used by the Fetch. The system suggests 21 articles and 14 of those articles are suggested by MyFido, the other 7 are randomly suggested. The 21 articles are selected from a pool of FeedItem's that were successfully matched against the system's knowledge base. MyFido must be able to determine the general topics discussed in a fetched article in order for it to be included in the pool of suggestible articles.

Suggested articles are selected by matching the FeedItem's TopicsOfInterests property against the logged in user's Observer and Fetch Master lists. Since the observer component is meant to crudely simulate a user surfing to a news site, and selecting articles they would like to read based solely on the criteria of the headline and perhaps article summary. In the instance of passively observing the user's interests, there is no way to determine if the user liked the article they read, hence MyFido captures and analyzes every article the user reads, unlike in the Fetch component where the system explicitly asks the user for feedback regarding the articles they enjoyed reading. MyFido gives more weight to FetchMasterList as it is generated using explicit feedback from the user, unlike the ObserverMasterList which is generated by passively observing the user reading articles that they may or may not have enjoyed.

To implement this precedence, the 14 suggested articles are actually matched against a CombinedMasterList. The CombinedMasterList is generated by matching the FetchedMasterList against the ObserverMasterList. All
the topicIds from the ObserverMasterList that match the topicIds in the FetchedMasterList are then added to the combined list. The remaining 10 (of the 14 system suggested articles) are then matched against a list that comprises all ObserverMasterList topicIds that were NOT found in the FetchedMasterList. In the case where a user only has an ObserverMasterList, for example if they have not had their first Fetch session, then all 14 of the system suggested articles will be chosen by matching the FeedItem's TopicsOfInterests property against the user's ObserverMasterList. If MyFido is able to suggest 14 articles by matching against the master lists from the user's user interest profile, then the last 7 articles are randomly selected. Finally, if the system is unable to suggest 14 of the 21 articles, the remaining number of articles to be suggested will be chosen randomly.

To view code snippets from TheFetch class please see: Appendix B – Section V  TheFetch Class
3. Technologies & Tools

The growth in the field of Artificial Intelligent research has led to the development of software tools and libraries to aid developers / researchers in incorporating AI techniques into software systems. These libraries and tools make the task of developing intelligent software a less daunting one, as they often provide common functionality needed for a variety of AI software projects. Hence allowing us as researchers and developers to focus on the particular problems we are trying to solve, instead of trying to reinvent the wheel developing functionality that in many cases would be a new project in itself.

Throughout my time developing MyFido, I evaluated several such software libraries. Libraries that would provide Natural Language Processing, HTML parsing, and string metrics services, among other things. All of these services were necessary to make MyFido a reality, but it would have been an extremely large undertaking to implement all of these services from scratch. In this section I will discuss the libraries used and evaluated while developing the MyFido system.

All of the libraries and interfaces used and evaluated are open-source projects, and come in a variety of popular programming languages, such as C#, Java C++, Perl etc. Though using these libraries was very helpful, it was very important to understand the principles behind their implementation in order to get the most out of using them. Furthermore, in many cases the documentation for some of these libraries was modest at best (and in some cases non existent), and since many of these libraries are used by developers and researchers in an academic / research space, there isn't a lot of in the way of example code to help you get started. Hence it was also important to have a more advanced knowledge of programming in order to use most of these libraries.

3.1 Natural Language Processing Tools

Natural Language Processing APIs: OpenNLP

OpenNLP library was written by Jason Baldridge, Tom Morton, and Gann Bierner. “The opennlp project is now the home of a set of java-based NLP tools which perform sentence detection, tokenization, pos-tagging, chunking and parsing, named-entity detection, and coreference ...using the OpenNLP Maxent machine learning package. These tools are not inherently useful by themselves, but can be integrated with other software to assist in the processing of text. “[15]

Though I evaluated both OpenNLP as well as the less mature C# port called SharpNLP by Richard Northedge [16], because of it's maturity, it's documentation and the larger developer user base (messages boards etc), I decided to use OpenNLP to aid MyFido's Context Analyzer in context analysis of observed documents and “fetched" RSS feeds.
**String Metrics Library : SimMetrics**

MyFido's Knowledge Access component uses string metrics to match phrases and keywords from observed documents and “fetched” RSS feeds against it's Category Hierarchy acyclic graph. By using string metrics, MyFido is able to detect similar relevant matches as well as exact matches against the category hierarchy. The approach of using string metrics was borrowed from the world of Ontology matching, as it makes use of string metrics in matching string properties of ontologies.

I evaluated three basic string metric distance algorithms, Jaro Winkler, Levenshtein distance and q-grams, to implement this matching functionality. I tested each for accuracy with a combination of possible variations of phrases and keywords, including phrases and keywords that were not similar at all. In addition to testing their accuracy, I was evaluating the similarity ratio returned by each algorithm in order to establish an initially acceptable similarity score range.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Strings</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>q gram test:</td>
<td>google map, google map</td>
<td>1.0</td>
</tr>
<tr>
<td>jaro winkler test:</td>
<td>google map, google map</td>
<td>1.0</td>
</tr>
<tr>
<td>levenshtein test:</td>
<td>google map, google map</td>
<td>1.0</td>
</tr>
<tr>
<td>q gram test:</td>
<td>macpro, mac pro</td>
<td>0.7058824</td>
</tr>
<tr>
<td>jaro winkler test:</td>
<td>macpro, mac pro</td>
<td>0.9666667</td>
</tr>
<tr>
<td>levenshtein test:</td>
<td>macpro, mac pro</td>
<td>0.85714287</td>
</tr>
<tr>
<td>q gram test:</td>
<td>google, google map</td>
<td>0.6</td>
</tr>
<tr>
<td>jaro winkler test:</td>
<td>google, google map</td>
<td>0.9878788</td>
</tr>
<tr>
<td>levenshtein test:</td>
<td>google, google maps</td>
<td>0.9090909</td>
</tr>
<tr>
<td>q gram test:</td>
<td>google, google map</td>
<td>0.9466666</td>
</tr>
<tr>
<td>jaro winkler test:</td>
<td>google, google map</td>
<td>0.0</td>
</tr>
<tr>
<td>levenshtein test:</td>
<td>google, google map</td>
<td>0.0</td>
</tr>
<tr>
<td>q gram test:</td>
<td>windows xp, xp</td>
<td>0.25</td>
</tr>
<tr>
<td>jaro winkler test:</td>
<td>windows xp, xp</td>
<td>0.19999999</td>
</tr>
<tr>
<td>levenshtein test:</td>
<td>windows xp, xp</td>
<td>0.19999999</td>
</tr>
</tbody>
</table>

The Jaro Winkler distance algorithm produced the most accurate results in terms of the similarity score it returns. This algorithm is most accurate with short strings such as a person's name, also two strings similarity score is adjusted for having common prefixes (strings with matching prefixes are giving a more favorable score). Since MyFido currently detects 2 and 3 word phrases (using 2-PosGram and 3-PosGram POS tag rules), and in MyFido's case keywords and phrases with the same prefixes (within reason) are favorable and can be considered similar enough to flag a match, I decided to use the Jaro Winkler distance algorithm for string similarity matching.

MyFido uses the SimMetrics implementation of the Jaro Winkler distance algorithm. I also used it's implementations of the Levenshtein distance algorithm and q-grams during my algorithm evaluations.
“SimMetrics is an open source extensible library of Similarity or Distance Metrics, e.g. Levenshtein Distance, L2 Distance, Cosine Similarity, Jaccard Similarity etc. SimMetrics provides a library of float based similarity measures between String Data as well as the typical unnormalized metric output.

It is intended for researchers in information integration, II, and other related fields. It includes a range of similarity measures from a variety of communities, including statistics, DNA analysis, artificial intelligence, information retrieval, and databases.” [17]

**JGraphT**

The MyFido Knowledge Base is stored in a relational database and is programmatically represented as a directed, acyclic graph as discussed in previous sections. Graphs are by nature well suited data structures for representing relationships between two vertices. The FidosBrain package holds all the objects that deal with the Knowledge Base and the objects that it's graph contains.

MyFido makes use of the JGraphT library. “JGraphT is a free Java graph library that provides mathematical graph-theory objects and algorithms. JGraphT supports various types of graphs including:

- **directed** and **undirected** graphs.
- graphs with **weighted** / **unweighted** / labeled or any user-defined edges.
- various edge multiplicity options, including: **simple-graphs**, **multigraphs**, **pseudographs**.
- **unmodifiable** graphs - allow modules to provide "read-only" access to internal graphs.
- **listenable** graphs - allow external listeners to track modification events.
- **subgraphs** graphs that are auto-updating subgraph views on other graphs.
- all compositions of above graphs.

Although powerful, JGraphT is designed to be **simple** and **type-safe** (via Java generics). For example, graph vertices can be of any objects. You can create graphs based on: Strings, URLs, XML documents, etc; you can even create graphs of graphs!” [18] JGraphT also has various algorithms for searching, traversing and testing graphs for cycles etc.

I am using the SimpleDirectedGraph object from this library. “A simple directed graph is a directed graph in which neither multiple edges between any two vertices nor loops are permitted.” [18] In order to use any of the graph objects in this library you must create your own vertex, edge and relationship objects in order to fully integrate it's graphs into your applications in order to accurately represent your data in graph form. The MyFidoEdge class extends the JGraphT’s DefaultEdge object. All objects including the MyFidoDictionaryEntry class, MyFiodRelationship, MyFidoVertex and MyFidoEdge classes have properties that mimic the properties represented in corresponding database table.

One primary benefit of using this library is because it offers a wide variety of graph types. I can easily swap out one graph for another because all graphs are based on the same base graph objects in the library. One possible enhancement will be to swap out the SimpleDirectedGraph for a WeightedDirectedGraph, where the relationships are weighted. Please see the Conclusion section of this paper for a further discussion of this and other possible system enhancements.
**WordNet**

“WordNet® is a large lexical database of English, developed under the direction of George A. Miller. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be navigated with the browser. WordNet is also freely and publicly available for download. WordNet's structure makes it a useful tool for computational linguistics and natural language processing.” WordNet was developed and maintained by the Cognitive Science Laboratory at Princeton University.

Though WordNet exposes their API in C, there are several open source libraries to interface with WordNet that are written in Java, C#, Perl, Python and many other languages. I will discuss two of the popular JAVA based interfaces evaluated for use in the section below. [16]

**WordNet Interfaces**

In order to programmatically access WordNet via the web or via the dictionary files stored on a local file system, you need a WordNet interface. Though there are several Java implemented WordNet interfaces, I evaluated JWI and JWNL for potential use in MyFido.

### 3.2 Other Technologies

There are a few software libraries that that are used in MyFido but are not related to the development of software that implements Artificial Intelligence or Natural Language Processing techniques.

**MozSwing**

When this project was still in the planning and research phase, my initial idea was to use FireFox's XUL and JavaScript as the interface for the web browsing portion of the application. “XUL (XML User Interface Language) is Mozilla's XML-based language that lets you build feature-rich cross platform applications that can run connected or disconnected from the Internet.” [20,21] XUL allows users to create applications that are embedded in the FireFox web browser for example. “The Mozilla Gecko layout engine provides a feature-rich implementation of XUL used in the Firefox Browser.” [20,21] The plan was to build an embedded interface within FireFox and use the wGet command line utility to grab and save HTML files from any website, to the users local file system. The wGet stand alone .exe (executable) file was to be called from JavaScript / using the XPCOM components to access the file system.
Security restrictions placed on a scripting languages in web browsers with respect to them calling an executable made it almost impossible to implement this solution. This coupled with cross browser restrictions, and potential system design issues made this a less than attractive option for implementing the Observer component. Though I was able to get this to work, I decided to embed a browser component / object into the MyFido GUI as a user interface of the Observer component.

Though I had chosen Java as my platform of choice, I was faced with a bit of a delima. The Microsoft .Net platform has a built in browser component that renders HTML etc. very similar to Internet Explorer but Java’s Swing Framework on the other hand only offered the JEditorPane. The JEditorPane only renders basic HTML at best and support for Flash, JavaScript etc is none existent (as of the date of this paper's publication). This would obviously make viewing any modern web site a painful experience, I didn't want to sacrifice usability and browsing experience in order to use Java as my development platform.

After doing some research, it was apparent that it would be difficult to find a non-commercial or relatively cheap embeddable browser Swing component. Finally I stumbled upon a small but well implemented library, MozSwing. “MozSwing project is integrating the Mozilla rendering framework XUL with the Java Swing GUI framework.” [22]
**HTMLParser**

MyFido's parser component uses the HTMLParser library to handle all of its basic parsing functionality. “HTML Parser is a Java library used to parse HTML in either a linear or nested fashion. Primarily used for transformation or extraction, it features filters, visitors, custom tags and easy to use JavaBeans. It is a fast, robust and well tested package.” [23]

**Rome**

MyFido uses the Rome library to handle the utility functionality of pulling system supported RSS feeds from the web. “ROME is a set of open source Java tools for parsing, generating and publishing RSS and Atom feeds. The core ROME library depends only on the JDOM XML parser and supports parsing, generating and converting all of the popular RSS and Atom formats including RSS 0.90, RSS 0.91 Netscape, RSS 0.91 Userland, RSS 0.92, RSS 0.93, RSS 0.94, RSS 1.0, RSS 2.0, Atom 0.3, and Atom 1.0.” [24]

**MigLayout**

Managing custom Swing layouts can be a challenging task. The need to nest Swing components in order to create moderate to complex layouts can be time consuming. “MiGLayout is a superbly versatile SWT/Swing layout manager that makes layout problems trivial. It is using String or API type-checked constraints to format the layout. MiGLayout can produce flowing, grid based, absolute (with links), grouped and docking layouts.” MyFido uses the MigLayout library to aid in laying out the panel that loads the custom aggregated RSS feed when the Fetch component is activated. [25]

4.1 System Components (Overview)

MyFido is a modular system and is designed as a two-tiered system. The first tier is the client application that runs on the user's desktop; the other is the centralized back-end where all processing and data storage is handled. This centralized back-end would consist of database and application servers. The major advantage to designing a modular system is scalability and code reuse. Though this project is a proof of concept and in its current iteration all components run in the client, the system and its components were designed with long term goals in mind. The system's modular design enables for components such as the Document Analyzer, Context Analyzer and The Fetch to be used on both the client and server ends of the system.

In the following sections, I will briefly discuss the responsibilities of all major system components as well as knowledge and document representation related system components.

Brief Description of Major System Component’s Responsibilities

MyFidoDriver

The MyFidoDriver class is located in the MyFidoMaster package along with the MyFidoSystem class. It is responsible for initializing system settings, the knowledge base graph, and the MyFido client side GUI.

MyFidoGUIDriver

The MyFidoGUIDriver is located in the MyFidoGUI package. The MyFido client GUI assembles the GUI components. It uses the singleton design pattern, hence there is only one instance of the GUI used throughout a user’s session.

MyFidoUserManagement

The MyFidoUserManagement package contains the Session, user and UserProfile Classes. This package’s classes are responsible for all actions related to the user system, which include user login, managing history of system use, creating and updating user’s interest profile.
**MyFidoSystemComponents**

The package that contains the Observer, Parser, DocumentAnalyzer, ContextAnalyzer and Fetcher classes.

**The Observer**

Is responsible for saving initially parsed HTML documents the user reads during a browsing session.

**The Parser**

Is responsible for initially parsing all observed HTML documents and storing them on the user's local file system. It also parses observed documents based on their site's schema and stores these documents into system defined document and document element objects. Lastly, it parses fetched RSS feed XML documents and stores them in system defined objects that represent a fetched RSS feed and the article summaries it contains.

**The Document Analyzer**

The DocumentAnalyzer class pulls schemas for system supported web sites, it uses the parser to get data from observed documents. It also acts as the driver class for the context analyzer.

**The Context Analyzer**

The ContextAnalyzer is one of the most important classes in the MyFido system. In this class a collection of observed documents or collection of RSS article summaries is tokenized, tagged using a parts-of-speech tagger and chunked. This class is also responsible for phrase/keyword/proper noun detection and acts as a driver class for the Knowledge Access class, where all found keywords and phrases are matched against the system defined category hierarchy graph.

**The Fetcher**

The Fetcher class is responsible for pulling RSS feeds, parsing them (XML files) and storing them in system defined objects that represent RSS feeds and RSS article summaries.

*Figure 4.1 Description of Major System Components Responsibilities*
This diagram shows MyFido’s current system design implementation. In this proof of concept system, the application as well as the database all live on the user’s (development) machine. MyFido’s components are designed and implemented using Object Oriented development practices; hence, this system will easily port to a client / server implementation. This also makes implementing new features easy.
This diagram shows the proposed design of a client/server system implementation. In this system, the bulk of the document parsing and processing will be handled on a system of centralized database and application servers. All parsed, observed documents would be stored on the local file system of an application server, using the system directory structure currently implemented in the proof of concept version of the system.
**Brief Description of Knowledge and Document Representation Related System Components**

*Figure 4.4 Description of Knowledge and Document Representation Related System Components*

**MyFidoInternalObjects**
This package contains interfaces and classes that are responsible for giving MyFido internal, system defined representations of documents, document elements, RSS feeds, and RSS feed items. This package contains the following system defined interfaces and classes: Document, DocumentCNET, DocumentContextSummary, DocumentElement, DocumentElementCNET, FeedItem, IDocument, IDocumentElement, IFeedItem, and RSSFeed.

**FidosBrain**
This package contains interfaces and classes that are involved in knowledge base representation. The objects in this package are all used to build MyFido's category hierarchy graph. This package contains the following system defined interfaces and classes: IEntity, IRelationship, IVertex, KnowledgeAccess, MyFidoDictionaryEntity, MyFidoEdge, MyFidoRelationship, and MyFidoVertex.

**FidosBrain.TheThinker**
This package holds all classes and interfaces responsible for MyFido's Rules system. The package contains the following interfaces and classes: IRule, Rule, and Rules.
5. My Fido In Action: System Walk Through

This section is a walk through of MyFido system functionality. In addition to demonstrating core functionality, this section will also serve to discuss major / and non major system components. System components play a role in implementing the system's core functionality.

Getting Started: Logging Into MyFido

MyFido makes suggestions to users based on their personal interests. In turn the system is building an ever expanding and changing user interests profile. In order to do so, every action a user makes must be mapped to that specific user. These actions include: MyFido observing the user reading articles in the built in web browser; analyzing observed articles; fetching system created, custom RSS feeds. All user related data is handled and processed by the classes in the “MyFidoUserManagement” package.

Figure 5.1 User logging into MyFido

*When a user logs into MyFido, behind the scenes the GUI is being loaded by the MyFidoDriver and MyFidoGUIDriver classes. While the GUI is loading, the category hierarchy knowledge graph is loading along with system settings.*
Figure 5.2  User System Component

The user’s log in credentials are checked against the MyFido database. If this check is successful, the user is logged into the system. All user system functionality is handled by the “User System” component.
Figure 5.3 State of MyFido GUI After Successful Login

If the login is successful: The date of the user's last successful log in (current date) is updated in the MyFido database, a new system defined Session object is set, and all dynamic text is loaded. Please note, that the Session object is a property of the majority of MyFido objects. Only one Session object is set per login, and this object is passed along to all system defined classes that need it.

To view a more complete code snippet from GUIDriver class please see: Appendix B – Section VII: GUIDriver Class – ActionListener: actionPerformed() - System Login

```java
if(isSuccess)
{
    user.setUsername(username);
    _session = new Session();
    _session.setSessionId(_session.createSessionId(user.getUserId()));
    _session.setCurrentUserId(user.getUserId());
    _session.setLoggedIn(isSuccess);
    // more code (see code snippet)

    // set all dynamic text in the GUI (username, etc)
    _allDynamicText();

    // enable GUI navigation buttons
    _enableAllGUITraversal();

    // set flag - user has not had a browser session yet as they
    // have just logged in
    _isBrowserSessionSet = false;
}
```
MyFido Observe Me!: Starting A New Observer Browser Session

In order for MyFido to observe the user reading articles, the user must "activate" the observer component. In future implementations, this functionality may not require interaction from the user to "activate" it. For example implementing the MyFido functionality as a FireFox plug-in or an Internet Explorer add-on would allow MyFido to always be on to observe them (if they so desire).

The user clicks the "Observe" button. Upon clicking the "Observe" button a new instance of the MozSwing browser component is loaded and added to the Swing border layout. Also, the date of the last browser session is updated in the database for this particular user. Please note, the date of the last browser session is the current date.

Please note, a new Observer object instance isn't created until after the user presses the "Go" button on the browser component, hence the previous page a user is reading is "observed" and initially parsed.
Figure 5.5  MyFido "Observing" – User reading an article from Cnet.com

This figure shows an example of an article loaded into the MozSwing embedded browser component. Once the user presses the "Go" button, this article is then initially parsed and saved to the local file system.

Please note, a new Observer object instance isn’t created until after the user presses the "Go" button on the browser component, hence the previous page a user is reading is "observed" and initially parsed.
The Observer component is responsible for calling the Parser component to initially parse and then save an observed article to the local file system.

**Figure 5.6 Observer Component**

The Observer component is responsible for calling the Parser component to initially parse and then save an observed article to the local file system.
```java
goButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        // initialize the observer object
        TheObserver observer = new TheObserver();

        if (moz.getURL().equalsIgnoreCase("http://www.nastygirlish.com/wYfGoObserverIntroPage.html") )
        {
            // DEBUG ME:
            System.out.println("GO GO GADGET OBSERVER!");

            // set TheObserver's session object to the current session for this user.
            observer.setObserverSession(_session);

            // the previous url that the user visited is the current url as far as TheObserver object
            // is concerned, set the current url for the observer
            observer.setCurrentURL(moz.getURL());
            String previousURL = observer.getCurrentURL();

            // save the article / HTML from the previously read
            // document to the local file system.
            observer.saveArticleHTMLToFile(previousURL, count);

            // clear url
            observer.setCurrentURL(null);
            count++;
            _currentSiteId = observer.getTempSiteId();
            _currentDocumentId = observer.getTempDocumentId();
        }

        // DEBUG ME: get the URL before there was any clicks
        System.err.println("get the URL BEFORE the goButton was clicked: " + moz.getURL());

        // DEBUG ME: get URL after the goButton was clicked
        System.err.println("get URL AFTER the goButton was clicked: " + jtf.getText());
        System.out.println("clicked go!");
    }
});
```

**Figure 5.7 “Go” button Code Snippet – Initializing Observer Object**

This code snippet shows the action listener's code for accessing the "Go" button, and initializing a new Observer object for each article the user reads.
This code snippet shows the `initialParseDocument()` method. This method in the `Parser` class is used to perform the initial parse of observed documents and is called while the user is being “Observed.”

```java
// Performs the first pass parsing of the file...By parsing out all
// of the noise data out of the document
public void initialParseDocument(String fileToParse, int siteId, int fileCount)
{
    String articleContainer = null;
    StringBuilder builder = new StringBuilder();
    String HTMLstring = null;
    int elementType = 7;
    this._currentSiteId = siteId;

    try
    {
        Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
        String url = "jdbc:odbc:MyFileSystem";
        connection = DriverManager.getConnection(url, "", "");
        pstmt = connection.prepareStatement("call dbo.GetArticleContainer(?,?)");
        pstmt.setInt(1, elementType); // pass in element type to the stored proc
        pstmt.setInt(2, siteId); // pass in siteId
        rs = pstmt.executeQuery();

        // get initial filter to parse article out of the HTML doc
        while (rs.next())
        {
            articleContainer = rs.getString("ClassAttributeValue").toLowerCase();

            // DESIGN ME (GET COLUMN NAMES / VALUES):
            System.out.println("ArticleContainer: " + articleContainer);
        }

        Parser parser = null;
        NodeList list = null;

        // we only want the the div that has the post data, this is the 8th of
        // of the page...the data we actually want
        HasAttributeFilter filter = new HasAttributeFilter("class", articleContainer);
        filter.setAttributeName("class");

        // DEBUG ME:
        //System.out.println("attribute: " + filter.getAttributeName());
        //System.out.println("value: " + filter.getAttributeValue());

        try
        {
            // apply filter to to be used with parser & parse file
            parser = new Parser(fileToParse);
            list = parser.parse(filter);
        }
    }

    // CODE SNIPPET - Initially Parsing Noise Data from Observed Articles
    This code snippet shows the `initialParseDocument` method. This method in the `Parser` class is used to perform the initial parse of observed documents and is called while the user is being “Observed.”
    This method, grabs the article container of an observed article (by on the site id, and elementType). The element is applied it to the attribute
```
Figure 5.9 The Parser Component

The Parser component provides MyFido with all of its HTML file parsing functionality. It utilizes the HTMLParser library.
**MyFido What Are These Articles About?: Starting the Context Analyzer**

In order for MyFido to make suggestions to users based on their observed interests, it must first analyze the grammatical structure as well as the context of observed articles that the user has read. The initially parsed files, are saved during observation and are mapped to the logged in user. Articles are stored on the file system using the following directory structure:

![Diagram of Directory Structure To Store Observed Documents](image)

**Figure 5.10  Diagram of Directory Structure To Store Observed Documents**

This diagram shows the file structure to store observed documents on a local file system. This file system can be the user's machine as in the proof of concept implementation or on a centralized server's local file system.

Each user has their own folder, that acts as the root directory that holds their observed documents. The directory is named `user_userId`, where the `userId` is the user's unique identifier stored in the MyFido database. Each system supported site has a folder in each user's directory, where observed documents from that particular site. The site directories are named `site_siteId`, where `siteId` is the site's unique identifier stored in the MyFido database. Lastly, each document is named by its `observeddate_uniquecount.html`, where the `uniquecount` is a number iterated by one and acts as a unique identifier for that particular document.
For each system supported website, MyFido reads each directory for initially parsed observed documents to further parse and analyze.

To “activate” the analysis process, the user clicks the “Analyze” button. This starts the process of grabbing the schemas for all system supported sites, that the Parser uses to parse observed documents, the article’s data is then stored in the system defined DocumentElement and Document objects.

A collection of Document objects is then passed to the ContextAnalyzer class where the document data is tokenized, tagged, chunked and lastly using the system defined 2-POSGram and 3-POSGram rules to identify noun phrases and proper nouns. The keywords and phrases found during this process are then passed to the KnowledgeAccess class to be matched against the category hierarchy graph. All matched terms are added to the user’s interests profile.
Figure 5.12  Context Analyzer (And Related Components)

The DocumentAnalyzer, ContextAnalyzer and TheParser all work together to give MyFido its analyization functionality. Data from parsed documents is added to Document objects, and passed as a collection of Document objects to the ContextAnalyzer class.
** Figure 5.13  Code Snippet – DocumentAnalyzer’s: Get Data From Observed Documents**

The `getDataFromObservedDocuments()` method, calls a method from the `Parser` class, that will return a collection of `Document` objects that hold parsed data. The `parseObservedDocuments()` takes in a given site’s document schema as a parameter.

To view this method and other code snippets from `TheParser` class please see: Appendix B – Section I: TheParser Class
The `analyzeAllDocumentContext()` from the `ContextAnalyzer` class initializes the document analyzation process. The `initContextAnalyzation()` method iterates through the collection of documents (which is a property of the `ContextAnalyzer` class) and starts the call rolling for tokenizing, parsing, and chunking document data.

Figure 5.14 Code Snippet - `DocumentAnalyzer` Class: Starting The Context Analyzer Process

The `analyzeAllDocumentContext()` from the `DocumentAnalyzer` class initializes the document analyzation process. Note, that the `initContextAnalyzation()` method iterates through the collection of documents (which is a property of the `ContextAnalyzer` class) and starts the call rolling for tokenizing, parsing, and chunking document data.

To see code snippets from the `ContextAnalyzer`, please see Appendix B - Section II: `ContextAnalyzer` class.

More Regarding Context Analyzation

The `ContextAnalyzer`'s `initContextAnalyzation()` method iterates through the collection of document objects. This collection is a property of the `ContextAnalyzer` object, and this property is set before the `init` method is called. Each document's headline and article body are tagged, tokenized, and chunked. During the chunking phase, all tokenized data is matched against 2-POSGram & 3-POSGram rules.

In this section, I will show examples of debugging output to demonstrate the process of chunking, and tagging as well as rule matching. Please note in order for chunking to take place all data from a document must be tokenized, and each token is tagged with a POS-Tag.
Figure 5.15 Sample Text - ContextAnalyzer Class: Examples of POS-Tagging, Chunking of an observed document (output data / debugging text)

This figure shows debugging output text from an observed article. This document has been tagged, tokenized and chunked by MyFido’s ContextAnalyzer. Also shown in this figure are examples of noun phrases found by matching chunked documents against system defined rules. For a list of MyFido POSGram rules see Figure 5.16.
POSGram rules are stored as a sequence of pipe delimited string. The ContextAnalyzer has a Rules object as property this object is initialized in the initContextAnalyzation() method. The Rules class has a method getRules() that returns a collection of Rule objects. A Rule object has the following properties: ruleId, ruleType, rulePattern, ruleDescription, and POSGramType. All rules (and their properties) are stored in the MyFido database.

To see code snippets from the Rules Class please see: Appendix B - Section III: Rules class.

Figure 5.16  MyFido POSGram Rule Patterns & Rule Descriptions

noun: Apple
noun: Safari
noun: downloads
noun: days
noun: browser
noun: times
noun: release
noun: Friday
noun: Windows
noun: Safari
noun: Sites
noun: beta
noun: February
noun: browsers
noun: Explorer
noun: tops
noun: loading
noun: HTML
noun: pages
noun: Nitro
noun: JavaScript
noun: executes
noun: IE
noun: open-source
noun: support
noun: offline
noun: technologies
noun: Standards
noun: Project
noun: test
noun: enhancements
noun: ability
noun: history
noun: Page
noun: Zoom
noun: bit
noun: download
noun: Mac

Figure 5.17  Sample Text - ContextAnalyzer Class:
Examples of nouns found in an observed document (output data / debugging text)

This figure shows nouns (proper and non-proper) found in article chunked in Figure 5.15.
Knowledge Access: Matching Terms & Phrases From Observed Documents With MyFido’s Category Hierarchy Graph

MyFido seeks to find the context of observed documents; it is basically trying to determine what an article is about. To do this, a collection of noun phrases and terms found via matching the POS-Gram rules against chunked article data to the KnowledgeAccess class. In this section, I will show code snippets and sample text that demonstrate similar and related results of matching terms and phrases against the category hierarchy.

```java
// step 3: send lists of words and phrases to rules method
_knowledgeAccessObj.findRelatedTermsInCategoryHierarchy(_allFoundPhrases);
_knowledgeAccessObj.findRelatedTermsInCategoryHierarchy(_allFoundWordsAndProperNouns);
```

Figure 5.18 Code Snippet – KnowledgeAccess Class: Starting Graph Matching Process

Here we have two calls to the findRelatedTermsInCategoryHierarchy() method that lives in the KnowledgeAccess class. These two calls are made at the end of the initContextAnalyzation(). This method lives in the ContextAnalyzer class and is used to start the process of analyzing the grammatical structure and context of observed documents. The findRelatedTermsInCategoryHierarchy() accepts collection of found noun phrases and keywords.
Figure 5.19 Sample Text - KnowledgeAccess Class: Examples of Similar (matched) Terms and Related Terms

This figure is an example of actual output (debugging) text; it shows the similar (matched) and related terms found when a collection of keywords and phrases parsed from observed documents is passed into the KnowledgeAccess class and tested against MyFido’s category hierarchy graph. This is some of the output for the article chunked in Figure x.x.

Note termToTest is the term or phrase passed in to the KnowledgeAccess class' findRelatedTermsInCategoryHierarchy() method. These terms and phrases were passed into this method as a collection of strings. These were found by rule matching and logic found in the ContextAnalyzer class. Associated terms and phrases found will be used to provide suggestions to users, by suggesting articles that are related to or are “about” these topics.

To see code snippets from the KnowledgeAccess Class please see: Appendix B - Section IV: KnowledgeAccess class.
When MyFido loads, before a user logs in the system caches a collection of all system supported RSS Feeds. These feeds are fetched from system supported RSS Feeds, parsed and put into system defined RSSFeed and FeedItem objects. A collection of these RSSFeed objects are then sent to the context analyzer. The title and description from each of the articles is tagged, tokenized, and chunked by the context analyzer. Once all the noun and noun phrases are detected for each feed, they are matched against MyFido’s knowledge base. Next all matched and related topicIds for that particular FeedItem (article) are stored in the FeedItem’s TopicsOfInterests property.

All of the feed caching functionality is done within the MyFidoSystem class, as MyFido treats the generated collection of RSSFeeds as a system wide property as it will be used by all system users. Once a user logs in all RSS Feeds have been analyzed and cached, and are ready for use by the Fetch component.
public MyFidoSystem()
{
    Properties properties = null;
    properties = loadPropertyFile();

    // set system properties : paths to OpenNLP model files
    MyFidoSystem.ChunkerModelPath = properties.getProperty("chunkerPath".Short);
    MyFidoSystem.CorefModelPath = properties.getProperty("corefPath".Short);
    MyFidoSystem.NameFinderModelPath = properties.getProperty("nameFinderPath".Short);
    MyFidoSystem.ParamModelPath = properties.getProperty("parserPath".Short);
    MyFidoSystem.SentenceDetectModelPath = properties.getProperty("sentDetectPath".Short);
    MyFidoSystem.TokenizeModelPath = properties.getProperty("tokenizePath".Short);

    // get all system supported syndicated rss feeds
    // these feeds are used by the Metadata object
    loadSystemSupportedSyndicatedRSSFeeds();

    // analyze feeds
    analyzeFeedItems();
}

private void loadSystemSupportedSyndicatedRSSFeeds()
{
    // step 1: get all system supported RSS Feed info and URLs
    getSystemSupportedRSSFeedInfo();

    // step 2: get syndicated rss feeds using URLs from step 1
    // this method also adds fetched feeds to system defined RSSFeeds / FeedItem objects
    getSyndicatedRSSFeedsFromWeb();
}

Figure 5.21 Code Snippet - MyFidoSystem Class: Fetching System Supported RSS Feeds

This figure shows the MyFidoSystem class' no-args constructor, calling the loadSystemSupportedSyndicatedRSSFeeds() method. This method makes calls one method which pulls data for system supported RSS feeds from the MyFido database and another method that fetches all system supported RSS feeds (and their articles) from the web. Once the feeds have been fetched from the web, they are parsed and the data is loaded into system defined RSSFeed and FeedItem objects.

To see code snippets from the MyFidoSystem Class please see: Appendix B - Section VI: MyFidoSystem class.
Figure 5.22 Code Snippet – Context Analyzer: Analyzing FeedItems From RSSFeed Collection

This code snippet below shows the ContextAnalyzer's overloaded initContextAnalysis() method. This method iterates through the collection of all RSSFeeds cached by MyFido. It calls methods that tokenize, tag, and chunk titles and descriptions for each FeedItem in an RSSFeed object. After all nouns and noun phrases have been detected, they are matched against the knowledge base category hierarchy. All matched and related terms for each FeedItem are stored as a comma separated list of topicIds. These comma separated lists are stored in the particular FeedItem's TopicsOfInterests property.

To see code snippets from the ContextAnalyzer, please see Appendix B - Section II: TheContextAnalyzer class

```java
/**
 * Overload of initContextAnalysis() method. This method
 * facet the same except it tags, tokenizes, and chunks data from FeedItems
 * in RSSFeed collections.
 * 
 * It gets a string of topicIds and adds it to the topicsOfInterest property
 * of the FeedItem object.
 * 
 * @param analyzeRSSFeedItems
 */
public void initContextAnalysis(boolean analyzeRSSFeedItems) throws IOException
{
    // get rid of used objects, just in case we haven't been careful
    System.gc();

    // get all phrase detection and other rules
    _rules = new Rules();

    String[] descriptionTokens = null;
    String[] descriptionTags = null;
    String[] titleTokens = null;
    String[] titleTags = null;

    if(analyzeRSSFeedItems == true)
    {
        // step 1: iterate through collection of RSSFeeds
        for(int p = 0; p < _allSupportedRSSFeeds.size(); p++)
        {
            ArrayList<FeedItem> items = _allSupportedRSSFeeds.get(p).getFeedItems();

            // DEBUG ME:
            System.err.println("FEED NAME: ", _allSupportedRSSFeeds.get(p).getFeedName());
            System.err.println("Items size(): ", items.size());

            // step 2: iterate through the collection of FeedItems, for each RSSFeed
            for(int i = 0; i < items.size(); i++)
            {
                // for each feed item, get the description and title, analyze it
                // and store the topic of interests
                String title = items.get(i).getTitle();
                String description = items.get(i).getDescription();

                // step 3: for each feed item, match the phrases and words
                // against the category hierarchy graph in the knowledge base class
                _allFoundPhrases = new ArrayList<String>();
                _allFoundWordsAndProperNouns = new ArrayList<String>();
```
// Step 4: tokenize, tag, detect phrases and words for each feedItem's
titleTokens = tokenizeData(title);
titleTags = posTagData(titleTokens);
detectPhraseAndWords(titleTokens, titleTags);

descriptionTokens = tokenizeData(description);
descriptionTags = posTagData(descriptionTokens);
detectPhraseAndWords(descriptionTokens, descriptionTags);

// System.err.println("_allFoundPhrases.size(): " + _allFoundPhrases.size());
// System.err.println("_allFoundWordsAndProperNouns.size(): " + _allFoundWordsAndProperNouns.size());

String phrases = _knowledgeAccessObj.findRelatedTermsInCategoryHierarchy(_allFoundPhrases);
String properNounsAndWords = _knowledgeAccessObj.findRelatedTermsInCategoryHierarchy(_allFoundWordsAndProperNouns);

String allTerms = ""

// DEBUG ME:
// System.err.println("phrases: " + phrases);
// System.err.println("properNounsAndWords: " + properNounsAndWords);

if(phrases != null && properNounsAndWords != null )
{
    allTerms = phrases + properNounsAndWords;
}
else if(phrases != null)
{
    allTerms = phrases;
}
else if(properNounsAndWords != null)
{
    allTerms = properNounsAndWords;
}

if(allTerms != null && !allTerms.isEmpty())
{
    String cleanedString = removeDuplicateIds(allTerms);

    // Step 5: add allTerms to the feedItem's topicOfInterests property
    items.get(i).setTopicOfInterest(cleanedString);

    // DEBUG ME:
    // System.err.println("cleanedString: " + cleanedString);
}
**Figure 5.23 Sample Text – Context Analyzer: Analyzed FeedItems**

This figure is an example of output (debugging) text of a few analyzed FeedItems.

To see code snippets from the ContextAnalyzer, please see Appendix B - Section II: TheContextAnalyzer class.
To see code snippets from the KnowledgeAccess Class please see: Appendix B - Section IV: KnowledgeAccess class.

Currently the system supports 5 RSS feeds from Cnet.com. Please note that because MyFido only analyzes the title and description from the feeds, it is possible for the system to currently support RSS feeds from almost any source. In future versions of the system the user may be able to specify the feeds they would like MyFido to choose articles from.

<table>
<thead>
<tr>
<th>Id</th>
<th>SystemSupportedSiteId</th>
<th>FeedName</th>
<th>FeedURL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>CNET News.com</td>
<td><a href="http://news.cnet.com/2547-1_3-0-20.xml">http://news.cnet.com/2547-1_3-0-20.xml</a></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>CNET News.com - Personal Tech</td>
<td><a href="http://news.cnet.com/2547-1040_3-0-5.xml">http://news.cnet.com/2547-1040_3-0-5.xml</a></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>CNET News - Apple</td>
<td><a href="http://news.cnet.com/8300-13579_3-37.xml">http://news.cnet.com/8300-13579_3-37.xml</a></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>CNET News - Gaming and Culture</td>
<td><a href="http://news.cnet.com/8300-13772_3-52.xml">http://news.cnet.com/8300-13772_3-52.xml</a></td>
</tr>
</tbody>
</table>

**Figure 5.24 Current MyFido System Supported RSS Feeds**

This table shows all current system supported RSS Feeds. This table reflects the feed data as it is stored in the MyFido database.
**MyFido Fetch!: Starting a New Fetch Session**

In order for MyFido users to see their custom aggregated RSS feeds they must “activate” the Fetch component, by clicking the “Fetch” button. When the user activates TheFetch component, a collection of FeedItems is built. This collection is of all FeedItems from the cached and analyzed collection of RSSFeeds that have TopicsOfInterest property that is not null or blank. If this property is not null or blank that means that MyFido was successfully able to determine the matched and related topics of that particular FeedItem (article). Since the system stores all the topicIds associated with articles the user liked during the Fetch session as Fetch snapshot data, it generates this new collection of FeedItems to ensure it will only suggest articles from a pool of FeedItems that have topicIds associated with them.

![Image of MyFido GUI](image-url)

**Figure 5.25 User “Activates” TheFetch Component – Loading Custom Aggregated RSS Feed**

*When the user “activates” TheFetch component, code in the GUI Driver class calls methods in TheFetch class that generates the custom aggregated RSS Feed. After the feed has been generated, it is then added to the MyFido GUI.*

To see code snippets from TheFetch Class please see: Appendix B - Section V: TheFetch class.
To see code snippets from the GUIDriver.getFetchPanel() please see: Appendix B - Section VII: GUIDriver class.
Figure 5.26 The Fetch (And Related Components)

The Fetch component works with ContextAnalyzer in order to fetch, analyze and aggregate custom RSS feeds for users.
MyFido Fetch!: Getting User Feedback

In order to read an article, the user clicks the “Read” button next to the corresponding headline. Clicking the “Read” button opens a tab in FireFox. When the user likes an article they will select the checkbox next to that article’s headline. After the user has read all the articles they are interested in, they will click the “Save Your Feedback” button. This will aggregate and save all the topicIds from the articles the user enjoyed into the FetchFeedbackSnapshot database table.

Figure 5.27  Aggregated Feed Loaded Into The MyFido GUI

Upon activating TheFetch component, users will click the “read” button next to the corresponding articles they would like to read. Clicking the read buttons opens a tab in FireFox.
6. Conclusion

6.1 Results: Testing, Stats, Proof of System’s Accuracy

MyFido has several components that enable it to ultimately aggregate a custom RSS feed for its users. Testing needed to be done in order to measure how well MyFido was doing its job, in other words I needed to test the user suggestion system’s accuracy. Since TheFetch implements the bulk of the user suggestion system, trial data needed to be collected regarding how accurately the system was suggesting articles the users liked and if it was getting better at suggesting more articles that the user liked over time.

Notes Regarding The Testing Process

There were 21 trials, conducted with 3 testers. There was one long term user, one intermediate time user and one short term user. Each tester’s technical interests and knowledge varied between novice (no prior interest) to expert level.

One session equals one trial and all testers had 1 observer session and at least 1 Fetch session. During the observer portion trials, users were given a list of roughly 50 articles from cnet.com. For each article they were only given the headline and the URL of the article. They were told to browse the list of articles and chose 15 of those articles to read as if they were casually browsing the home page of a news site. During the Fetch trials, users were told to browse the given headlines as if they were browsing a normal RSS feed, and read the articles that seem interesting according to the headline. If they liked the article they read, they were to check the checkbox that corresponds to that article. When they were done and decided there are no other articles from the list they were interested in reading, they click the Save Feedback Now button.

All relevant statistical data was captured automatically by MyFido and stored in the system database. By recording statistical data programmatically, I did not need to interact with the user while they were testing. It also ensured that recording this data would not negatively impact the user’s overall experience of using MyFido.

Summary of Results / Accuracy Statistics

Overall I am very pleased with MyFido’s performance. Though this is by no means a production ready system, the results are definitely a great starting point for this proof of concept application. The data supports the conclusion that MyFido is able to suggest articles a user will like based on their observed interests with a promising amount of accuracy. The data also indirectly supports the conclusion that MyFido, even in its infancy, is able to determine the topics discussed in an article or a RSS feed item with a competent level of accuracy.

As stated in previous sections, MyFido returns a total of 21 articles in every custom aggregated RSS feed, and of these 21 articles, 14 of them are suggested by MyFido and the other 7 are randomly selected. Also, note that of the 14 system suggested articles, 10 of those are selected by matching feed items against the user’s combined FetchedMasterList (see section that explains) and 4 are selected by matching feed items against
the rest of the topics in ObserverMasterList that were not found in the FetchedMasterList. So this means roughly 66% of aggregated custom feed's articles are suggested by MyFido and roughly 34% of them are randomly suggested.

**General Statistics**

Over the course of 21 trials, on average users liked 11 of the 21 articles presented in their custom aggregated feeds. This means on average user’s liked 52% of the total 21 articles presented to them. Also note that on average 8 of those 11 articles the user liked were system suggested articles, meaning roughly 72% of the average number of articles liked by a user were suggested by MyFido! Comparably, on average 3 of the 11 articles the users liked were randomly suggested, meaning roughly 27% of the average number of articles liked by the user were randomly suggested.

Over the course of 21 trials, on average users liked 8 of the 14 system suggested articles. This means on average user’s liked 57% of the total 14 system suggested articles. Comparably, on average users liked 3 of the 7 randomly suggested articles. Which means on average user’s liked 42% of the 7 total randomly suggested articles.

Over the course of 21 trials, on average 5 of the 14 system suggested articles user’s liked were suggested by being matched against the user’s (combined) FetchedMasterList. Meaning that roughly 35% of the total 14 system suggested articles user’s liked were suggested by being matched against the user’s (combined) FetchedMasterList. Comparably, on average 3 of the 14 system suggested articles user’s liked were suggested by being matched against the topicIds from the ObserverMasterList that were not found in the FetchMasterList. This means that roughly 21% of the total 14 system suggested articles user’s liked were suggested by being matched against the the topicIds from the ObserverMasterList that were not found in the FetchMasterList.
Figure 6.1 General Statistics Summary

<table>
<thead>
<tr>
<th>General Statistics Summary: Total articles liked</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users liked <strong>11 of the 21</strong> articles presented → hence users liked <strong>52%</strong> of the total 21 article presented.</td>
</tr>
<tr>
<td><strong>8 of those 11</strong> articles the user liked were system suggested articles → hence <strong>72%</strong> of the articles the user liked were suggested by MyFido.</td>
</tr>
<tr>
<td><strong>3 of the 11</strong> articles the user liked were randomly suggested → hence <strong>27%</strong> of the articles liked by the user were randomly suggested.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>General Statistics Summary: Randomly vs System Suggested</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users liked <strong>8 of the 14</strong> system suggested articles → hence users liked <strong>57%</strong> of the total 14 system suggested articles.</td>
</tr>
<tr>
<td>Users liked <strong>3 of the 7</strong> randomly suggested articles → hence users likes <strong>42%</strong> of the 7 randomly suggested articles.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>General Statistics Summary: FetchedMaster vs ObserverMaster lists</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>5 of the 14</strong> system suggested articles the users liked were suggested by being matched against the user's (combined) FetchMasterList → hence <strong>35% of 14</strong> system suggested articles, were suggested by matching the FetchMasterList.</td>
</tr>
<tr>
<td><strong>3 of the 14</strong> system suggested articles the users liked were suggested by being matched against the user's ObserverMasterList → hence <strong>21% of 14</strong> system suggested articles, were suggested by matching the ObserverMasterList.</td>
</tr>
</tbody>
</table>

Statistics From Noteworthy Trials

UserId 5 was the long term trial user, and completed a total of 14 trials. The overall data from UserId 5's trials suggest that MyFido was able to successfully identify some of the user's interests, and that overtime it was able to use these interests in order to suggest an increasing number of articles the user would like over the course of his trials. Below I will briefly discuss some of UserId 5's milestone trials.

Up until trial number 5, UserId 5 on average liked **9** of **21** articles presented in Fetch sessions. However, starting with trial 5, this changed and after that on average the user liked **13** of the **21** articles presented in Fetch sessions. This means that there was approximately a **19%** increase in articles liked by the user between trials 5 through 14.

The overall data for UserId 5 yielded positive results, there were two particular trials where the user actually liked more or the same number of randomly suggested articles as system suggested articles. These trials were trial number 9 and trial number 10. Because there aren't more of these types of trials it is difficult to point out why there was a break down in the number of system suggested articles that the user liked. One other notable phenomenon from these two trials is that in both at least half or a little over half of the linked articles were suggested by matching against the remaining topics in the ObserverMasterList. On average of the system suggested articles the user's liked more of them were matched against the (combined) FetchMasterList. (see Section 2.2: I Know What You Like: Making Suggestions To Users based on Observed Interests)

The last noteworthy trial for UserId 5 was trial number 11, where he liked all 14 of the system suggested articles and none of the randomly selected articles.
Lastly I wanted to compare the starting points for all three users. This chart shows trial one for each of the users. Please note as mentioned in a previous section of this paper, the user's first trial is the trial where the user has only completed one Observer session and one Fetch session. At this point only the ObserverMaster list has been created and used to suggest articles in TheFetch.

<table>
<thead>
<tr>
<th>UserId</th>
<th>System Suggested Articles Liked</th>
<th>Randomly Suggested Articles Liked</th>
<th>Liked Articles Matched Against FetchMaster</th>
<th>Liked Articles Matched Against ObserverMaster</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>4</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>5</td>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>

*Figure 6.2 First Trial Statistics For All Users*

**Future Statistical Goals**

The overall results from the first trial tests are very promising and I am pleased with MyFido's performance, but there is a lot of room for improvement. It is important to reiterate that this is by no means a production ready system and though the level of accuracy of the user suggestion system (TheFetch) is encouraging, the system's overall intelligence and learning capabilities can be improved (see: Proposed System Enhancements section).

Statistically the next goal for MyFido is where on average users like approximately 11 of the 14 system suggested articles, which is roughly 80% of the 14 system suggested articles.

**Summary of Results: Fetched Feeds Analysis**

In addition to statistical data collected to track user system's accuracy, MyFido recorded statistical data related to RSS feeds fetched from the web. Over the course of development and trial tests, MyFido fetched a total of 1,685 FeedItems (articles) from the web. Please note, at one point in development and testing MyFido only pulled an average of 63 articles at a time, at that time MyFido only had 3 system supported RSS feeds, at the time of writing this paper it now has 5 system supported feeds.

Of the 1,685 articles in total MyFido has fetched from the web, 1007 of those were able to be matched successfully (meaning matched at least one topic of interests from analyzing the FeedItem's title and description) against the category hierarchy knowledge base. On average, MyFido was able to successfully match 38 articles successfully against it's knowledge base, per trip to the web. On average, the system pulled roughly 64 articles per trip to the web.
6.2 Challenges / Issues Encountered

**Challenge: Finding The “Main Idea” of an Article**

One of the major challenges with MyFido and similar applications is determining what a document is about. It is no small task to give software this ability, essentially we are trying to give software the skill of reading comprehension. Currently MyFido can determine the general topics discussed or mentioned in an article, but it can not determine specifically what an article is about. For example, if there is an article about operating systems but for some reason Google is referenced in the article, Google will be matched and returned as a topic of interest for that particular document. This is a good start but MyFido isn't as accurate as it could be.

As mentioned early in this paper, MyFido uses a tiered approach to analyzing a document. The first tier of analysis acts as the “reader” and is where the grammatical structure of a document is analyzed. The second tier acts as the “context analyzer” where it attempts to determine the topics covered in the article. The ultimate goal is to add additional layers of analysis to the tiered system in order to give MyFido the ability to granularly determine what a document is actually about as well as the general topics discussed.

**Challenge: Limited Knowledge of the World, Limits Suggestions to Users**

MyFido's knowledge base consists of computer and information technology terms and phrases as well as their aliases. Gathering and mapping the relationships between these terms was no small task. It took a considerable amount of time and effort just to assemble the small collection of knowledge MyFido currently has. As of the time of this document's publication MyFido's knowledge base contains 157 terms and it's graph contains 544 edges.

MyFido's knowledge of the outside world plays an important role in how useful and accurate the user suggestion system is. MyFido's limited knowledge limits the application's ability to determine the context of articles on a broader range of topics, hence limiting the scope of articles it can suggest to users.

**Challenge: More Knowledge, More Problems - “The Google Problem”**

As of the date of this publication Google is one of the only commercially available and commonly used applications that attempts to read or “crawl” through a document to determine the context of the document's contents, and then it returns results based on user interests expressed by the user querying this gathered data. Though Google is an excellent search engine, as its base of knowledge grows so does the amount of irrelevant data returned in user searches. I call this issue the "Google problem." Though there are techniques for yielding better results from Google searches, the average users do not use or know these techniques.[25]

Though increasing MyFido's knowledge base will improve the user suggestion system, the chance of overwhelming users with too many suggestions increases as well. This could lead to more information overload, a problem this project is attempting to combat. Later I will present system enhancements that will allow for increasing MyFido's knowledge of the outside world without sacrificing its accuracy and usefulness.
**Challenge: Level of Human Labor Required**

Knowledge based intelligent software systems require a level of manual labor by human beings. Their knowledge about the outside world comes from us human beings. One important challenge to developing this system was the level of human labor that was required to give MyFido its intelligence. Gathering terms for the category hierarchy, mapping the relationships between those terms, analyzing the DOM structure of articles from system supported sites, and analyzing chunked documents to identify parts-of-speech tag sequences that make up noun phrases for rules system are all actions that required human labor.

The amount of human labor required adds a considerable amount of time to the development and maintenance of knowledge based AI systems. Though some of the above mentioned tasks can eventually be automated, some tasks such as gathering terms and mapping their relationships to one another would be very difficult to automate as the relationships between many of these terms are human constructs and require human understanding. Unless our reasoning for determining these relationships can be gathered and utilized in a rules based system, this will continue to be a manual task for a human being. As the category hierarchy grows, updating these relationships will prove to be a tedious and time consuming.

### 6.3 Proposed System Enhancements

Throughout the research and development of this system, I thought of possible enhancements that would greatly improve MyFido's functionality, design and usability. In this section I will discuss proposed system enhancements that will attempt to: improve system accuracy & quality of suggestions returned to users, improve system performance, improving system design & architecture and improve the overall user experience.

**Improving System Accuracy: Context Analyzation & User Suggestions Functionality**

The accuracy of context analysis directly affects the quality and accuracy of MyFido's user suggestion functionality. By improving noun phrase detection, matching functionality, adding terms to MyFido's lexicon and continuing to add layers of analysis to the tiered system of analysis you ultimately improve the user suggestion system because this improves the terms and phrases added to the user's interests profile.

**Using a Weighting Schema, Graph Traversal & Distance Preference to Fight The “Google Problem”**

Currently, the category hierarchy graph is a weightless, directed graph. This means that each relationship that links terms of an edge together all have the same weight. For the purpose of this discussion, let's assume the current weight for every relationship type is 1. At the time of this publication six relationship types exist in the category hierarchy graph.
<table>
<thead>
<tr>
<th>RelationshipTypeId</th>
<th>RelationshipName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>is related to the field of</td>
</tr>
<tr>
<td>2</td>
<td>is an alias of</td>
</tr>
<tr>
<td>3</td>
<td>is associated with (term or phrase)</td>
</tr>
<tr>
<td>4</td>
<td>is a type of / is a</td>
</tr>
<tr>
<td>5</td>
<td>is created by</td>
</tr>
<tr>
<td>6</td>
<td>is referred to as</td>
</tr>
</tbody>
</table>

Figure 6.3 Graph Relationships As Stored In The MyFido Database

To implement the weighted relationship schema, each relationship will be assigned a weight based on level of preference. Next, the graph data pulled from the database will be added to a weighted, directed graph. In addition to adding weighted relationships, instead of iterating through the edges of a graph during the term matching process, MyFido will traverse the graph and examine its edges. When terms found in observed articles are an exact (similar) match against an entity in the graph, only the related terms within the closest amount of “hops” (distance from one term in an edge to another) to the matched term along with related terms that have the higher weighted relationship types will be returned along with the exact (similar) matched terms are added to the user’s interests profile. There will also be care taken to not return duplicate results.

By adding distance preference and relationships weight preference, we combat the “Google Problem” because this will help decrease the number of results returned and added to the user’s interests profile. The hope is that this will also increase the quality / accuracy of custom RSS feeds returned to users based on their interests profile.

Improving Noun Phrase Detection

MyFido’s rules based noun detection system is part of the tiered system of analyzation. Currently the system has (x number of ) rules. To improve noun phrase detection and the quality of noun phrases returned, more complex POS Gram rules should be added to the rule set. In order to create these rules, a large number of training articles must be tagged, chunked and analyzed to identify important, more complex noun phrase patterns not yet stored in the MyFido rule set. Though this is a labor intensive process by adding new noun detection rules will improve the quality of matched phrases and in turn leads to a more accurate user interests profile and hence more accurate user suggestion system.

One thing to note is that the effectiveness of this enhancements is coupled with the action of increasing the current knowledge base.

Add Location Based Weighting System To Noun Phrase Detection

Another way to improve noun phrase detection is to add a location based weighting system where we place higher weight or preference on nouns and noun phrases detected in places in the document that are known to hold the “main idea” of the article. These locations will be determined by common reading comprehension rules. For example, the first, second or third sentence of the first paragraph usually tells you what the article is about and the last paragraph of the article usually confirms what the “main idea” of the article is. Terms and phrases found in these “priority” areas on the document will be given higher weight than those found else
where in the document and would be treated with more priority in terms of matching against the category hierarchy graph. This location weighting schema, in conjunction with the weighted relationships and graph distance preference, will work together to return the best possible matches to be added to the user's interests profile.

Further research is needed to determine how the above rules are formally defined so that they can be implemented into MyFido's rules system. By making these rules a part of the noun detection rules system, this would further enable MyFido to determine what an article is about, not just the general topics discussed in the article.

“Tone” of Article Detection

One possible system enhancement that would increase the accuracy of the user suggestion system is adding “tone” detection to the tiered system of analysis. This would give MyFido the ability to not only return articles on given topics of interests, but it could also return articles that reflect the user's “feelings” on particular topics of interest.

Improving System Performance, Design & Architecture

Though in its current iteration, MyFido is a proof of concept system, it is common practice to plan and develop a system with production level performance and scalability in mind. MyFido's functionality includes many resource intensive system processes such as document parsing, data processing, edge iteration and the proposed addition of graph traversal would require significant resources as the number of users, its knowledge base and complexity grows.

In this section I will discuss proposed solutions for improving system performance, design and architecture.

System Implementation With Centralized Backend

In a centralized backend system implementation, document parsing and processing would be handled on a system of centralized database and application servers. All parsed, observed documents would be stored on the local file system of an application server, using the system directory structure currently implemented in the proof of concept version of the system.

By leveraging server side processing for all major system components, data processing would be optimized, hence increasing responsiveness of the client side application which in turn would improve the user's experience.

Moving MyFido Into The “Cloud”

For applications that are memory intensive and have demanding data processing needs, the concept of cloud computing can be a cost effective way to meet these demands. It also allows for on the fly flexibility and scalability. MyFido's data processing needs would benefit from being in “the cloud”, especially in a production ready implementation with high volume of user traffic.

Instead of taking the approach of spawning multiple threads to handle processing a large number of observed documents or RSS feeds for example, in a cloud environment multiple instances of an application are spawned on demand to handle the task of data processing etc. This would enable me to develop a flexible, scalability production level implementation of MyFido without adding additional complexity of needing to spawn
and manage multiple threads to handle MyFido’s processing intensive tasks.

For some applications, implementing multi-threading is unavoidable to improve processing speed, but in the case of MyFido, it’s demands and the nature of the service it provides would benefit from living in “the cloud.”

**Improve the Overall User Experience**

So far I have discussed ways of improving the system architecture, technical design and functionality of key components that in essence make MyFido “intelligent.” All of these proposed enhancements will improve the overall user experience along with a few practical changes that will make for a richer user experience.

**Implementing Automated Data Processing & Fetching**

In the current proof of concept implementation of MyFido, all data processing and fetching is currently done by even driven actions. For example observed articles are processed by clicking the “Analyze” button. In a production implementation, data processing and fetching as well as document storage would be handled on centralized servers, where data processing and fetching would all be handled by scheduled, automated processes or services (if in a Windows server environment). These automated processes would run continuous on the server(s). MyFido user profiles will be updated accordingly, by an automated process and custom RSS feeds created for individual users will be cached and stored on the server, waiting to be fetched according to individual user defined “fetch schedules.”

**User Controlled Settings**

In a production ready implementation, MyFido will give users the ability to update and control their specific system settings. Actions such as updating username and password, and creating a “fetch schedule,” which their custom RSS feeds are fetched and loaded into the local interface.

**Bells & Whistles**

Some enhancements that improve the user’s experience would make MyFido a more convenient application to use. Two possible enhancements are adding a mobile component and MyFido Lite.

*MyFido “On-the-Go”* would enable users to fetch their custom generated RSS feeds on their mobile devices. These feeds will be generated and cached on a centralized server(s). A mobile feed viewer would need to be developed in order to implement *MyFido “On-the-Go”.*

MyFido Lite is a scaled down version of MyFido, where it is installed in web browser such as FireFox or Internet Explorer as a plug in, browser add-on or bookmarking functionality. So that observation and fetching take place in the user's native browser instead of in a stand alone interface. The user will have the capability to update and control their system settings.
6.4 Final Remarks

Developing MyFido was more than planning the system architecture and researching / evaluating software libraries. It was necessary to not only determine could this idea be implemented and if so, how? This project took me on a journey of researching the history of Natural Language Processing, studying how other researchers were approaching the problems of implementing user suggestion systems, passively observing user interests, ontology matching, determining the context of documents, implementing knowledge based systems, and how to represent that knowledge programmatically and how to store that data.

As researchers we were all trying to meet a different end, but I learned their were common problems that all of us faced in implementing knowledge based systems that attempt to make suggestions to users based on passively observed interests.

The research portion of this project continued throughout the planning and development of this system. Though my initial research for this project started in the middle of the Spring 2008 semester, it continued throughout the Spring 2009 semester. When my academic career at Pace University began, my interests in Artificial Intelligence centered around Robotics, and though I still have a definite interest in Robotics research, the research and work done to bring MyFido to life has peeked my interest in the field of Natural Language Processing.

NLP has many real world, practical applications and I believe it will play a major role in developing systems that will combat the ever growing issue of information overload and the need for knowledge management and user suggestion systems. The implications of successes in the field of Natural Language Processing reach far beyond that of the fields of information technology and computer science.

It is my goal to continue my research and development of the MyFido system now that my thesis project is complete. I plan to implement my proposed system enhancements with the hopes of leveraging my current work in order to turn MyFido into a production ready application.
7. Appendix A: References


[5] Paik, W., Yilmazel, S., Brown, E. Poulin, M., Dubon, S., Amice, C. Applying Natural Language Processing (NLP) Based Metadata Extraction to Automatically Acquire User Preferences. (p.116-122) Solutions-united, Inc. Syracuse, NY


[23] Rome. https://rome.dev.java.net/


Other Resources

Note, below are some resources that were read during research and development of MyFido but were not sited in the text.


Other Resources (Wikipedia – Topics)

Computational Linguistics, Principle of Maximum Entropy, Natural Language Processing, Machine Learning, Parts-Of-Speech Tagging, string metrics, and Information Foraging.
8. Appendix B: Code Snippets

This section contains code snippets and methods from the MyFido system, that shows key system functionality. Please note that because MyFido in it's current implementation is a proof of concept application and all code and functionality implementations are subject to change. Some values in rare instances are hard coded in order for faster development.

** Please note this section does not contain all MyFido source code, and should not be used as official system documentation.

Section I: TheParser Class

Please note, the class shown here is NOT the complete TheParser class and my not be the same as the finally submitted code. These snippets show methods that implement some of MyFido's parsing functionality.

```java
public ArrayList<Document> parseObservedDocuments(Document schema)
{
    String[] observedFiles = null;
    String path = "";
    ArrayList<Document> docs = new ArrayList<Document>();

    // Step 1: Get full path to files that were saved during
    // user's browser sessions (by parser and observer classes)
    path = getFullFilePath();

    // Step 2: Get an array of all files in the ObservedHTMLDocs directory
    if( (!path.isEmpty()) && path != null )
    {
        observedFiles = getAllFiles(path);
        Document doc = null;

        // Step 3: For each article, create a new document object
        // and parse the document elements and store their value in a document element's value property
        // we are passing the shell document object (schema) to the parse method because it will
        // give the parser direction on how to parse each element of an article
        for(int s = 0; s < observedFiles.length; s++)
        {
            doc = new Document();
            doc.setDocumentType(schema.getDocumentType());
            doc.setDocumentId(schema.getDocumentId());
            doc.setSiteID(schema.getSiteID());

            doc = parseDocByClassAttributes(schema.getDocumentElements(), doc, path, observedFiles[s]);

            // DEBUG ME:
            //System.out.println("Current Document: " + s);
            //System.out.println(doc.getDocumentElements().get(0).getValue());
            //System.out.println(doc.getDocumentElements().get(1).getValue() + "\r\n");

            docs.add(doc);
        }
    }

    return docs;
}
```
**
* Takes a given document schema (which is nothing more than document object
* of a specific to a given website).
* Properties of the passed in document object, contains information that will
* instruct the parser on the nodes to parser (based on class attributes).
* Next, it will add the text of the nodes to the value property of a
* document element.
* Lastly, it will return an ArrayList of documents (programmatic
* representations of the article data contained in the relevant parts of a site's
* articles).
* @param schema
* @return ArrayList<Document>
*/
private Document parseDocByClassAttributes(ArrayList<DocumentElement> elements, Document newDoc, String
path, String filename)
{
    DocumentElement element = null;
    ArrayList<DocumentElement> temp = new ArrayList<DocumentElement>();

    for(int s = 0; s < elements.size(); s++)
    {
        // for each document element in elements parameter, create a document element
        element = new DocumentElement();

        // copy document element properties from schema (shell document object) to new document element
        element.setClassName(elements.get(s).getClassName());
        element.setDocumentType(elements.get(s).getDocumentType());
        element.setDocumentID(elements.get(s).getDocumentID());
        element.setElementType(elements.get(s).getElementType());
        element.setHTMLTagName(elements.get(s).getHTMLTagName());
        element.setHTMLTagSymbol(elements.get(s).getHTMLTagSymbol());
        element.setIdAttributeValue(elements.get(s).getIdAttributeValue());
        element.setSiteID(elements.get(s).getSiteID());

        // get schema classAttributeValue and schema's tag
        String attribute = element.getClassAttributeValue();
        //String tagSymbol = schema.getDocumentElement().get(s).getHTMLTagSymbol();
        String tagName = element.getHTMLTagName();

        // parse the text out of the HTML file
        String value = parseValue(attribute,tagName,path,filename);

        // set schema's value property, to the current value parsed from the file
        element.setValue(value);

        // add element to the temp arraylist
        // the temp arraylist is a collection of document elements that will be
        temp.add(element);
    }

    newDoc.setDocumentElements(temp);

    return newDoc;
}
private String parseValue(String classAttributeValue, String tagName, String path, String filename) {
    String strValue = "";
    StringBuilder builder = new StringBuilder();

    // get value by classattribute value OR if the classattribute is null
    // AND the siteId is 1 then get parse by tags..grab <h1> tag's text
    // (this is true if the element type is Tags, Topics, H1 (headline))
    if (classAttributeValue != null) {
        Parser parser = null;
        NodeList list = null;

        // we only want the the div that has the post data, this is the meat of
        // of the page...the data we actually want
        HasAttributeFilter filter = new HasAttributeFilter("class", classAttributeValue);
        filter.setAttributeName("class");

        // second filter to strip out script and style tags
        NodeFilter filter2 = new NotFilter( new OrFilter ( new TagNameFilter("SCRIPT"),new
            TagNameFilter("STYLE") ) );

        try {
            // apply filter to to be used with parser & parse file
            parser = new Parser(path + filename);
            list = parser.parse(filter);
            list.keepAllNodesThatMatch(filter2, true);

            //DEBUG ME:
            // System.out.println("list size: " + list.size());
            for(int i = 0; i < list.size(); i++) {
                //DEBUG ME:
                // System.out.println("node: " + list.elementAt(i).toPlainTextString().trim());
                // add nodes to string builder object
                builder.append(list.elementAt(i).toPlainTextString());
            }
            // save initially parsed html of the actual whole article
            strValue = builder.toString().trim();
            // DEBUG ME:
            if (!classAttributeValue.equalsIgnoreCase("post") && !
                classAttributeValue.equalsIgnoreCase("postlinks") ) {
                //DEBUG ME:
                // System.out.println("attribute: " + filter.getAttributeName());
                //System.out.println("value: " + filter.getAttributeValue());
                //System.out.println("strValue: " + strValue + "\r\n");
            }
        } catch (ParserException e) {
            e.printStackTrace();
        }
    } else {
        try {
            // if we are that means we have to parse based on the TagName
            // since document element with data we need does not have a
            // class attribute value set (in the html file)
// i check for all element types here to plan for the future
// implementations here where myfido supports more than CNET articles,
// just in case the articles for another site doesn't have class
// attributes set for each of these generic document element types.

Parser parser = null;
NodeList list = null;

TagNameFilter filter = new TagNameFilter(tagName.toLowerCase());

// apply filter to to be used with parser & parse file
parser = new Parser(path + filename);
list = parser.parse(filter);

for (int i = 0; i < list.size(); i++)
{
    //DEBUG ME:
    //System.out.println("node: " + list.elementAt(i).toPlainTextString());
    // add nodes to string builder object
    builder.append(list.elementAt(i).toPlainTextString());
}

// save initially parsed html of the actual whole article
strValue = builder.toString().trim();

// DEBUG ME:
// System.out.println("TAG NAME: " + filter.getName());
// System.out.println("strValue: " + strValue + "\n");

} catch (ParserException ex)
{
    ex.printStackTrace();
}

//System.out.println("strValue: " + strValue + "\n");
return strValue;
}
Section II: TheContextAnalyzer Class

Please note, the class shown here is NOT the complete ContextAnalyzer class and my not be the same as the finally submitted code. These snippets show implementations of key MyFido system functionality.

```java
package MyFidoSystemComponents;

import FidosBrain.KnowledgeAccess;
import FidosBrain.TheThinker.Rule;
import FidosBrain.TheThinker.Rules;
import java.io.IOException;
import java.util.ArrayList;

// MyFido Imports
import MyFidoUserManagement.Session;
import MyFidoInternalObjects.Document;
import MyFidoMaster.MyFidoSystem;
import java.io.File;
import opennlp.maxent.MaxentModel;
import opennlp.maxent.io.PooledGISModelReader;
import opennlp.maxent.io.SuffixSensitiveGISModelReader;
import opennlp.tools.dictionary.Dictionary;
import opennlp.tools.lang.english.Tokenizer;
import opennlp.tools.lang.english.TreebankChunker;
import opennlp.tools.namefind.NameFinderME;
import opennlp.tools.postag.POSTaggerME;
import opennlp.tools.sentdetect.SentenceDetectorME;
import opennlp.tools.tokenize.TokenizerME;
import opennlp.tools.util.Span;

public class TheContextAnalyzer {
    // instance variables
    private Session _session = null;
    private ArrayList<Document> _documents;
    private MyFidoSystem _systemSettings;
    private static ArrayList<String> _allFoundPhrases;
    private static ArrayList<String> _allFoundWordsAndProperNouns;
    private Rules _rules;
    // object that holds category hierarchy graph and graph related objects
    private KnowledgeAccess _knowledgeAccessObj;

    public enum DocumentElementType {
        // yes these are constants and should be in all caps but I want to match the
        // case of the types as they are stored in the database
        Date, Headline, Author, BodyOfArticle, Tags, Topics, ArticleContainer, Links
    }

    // getters & setters
    public Session getContextAnalyzerSession() {
        return _session;
    }

    public void setContextAnalyzerSession(Session currentSessionObject) {
        _session = currentSessionObject;
    }

    public MyFidoSystem getSystemSettings() {
        return _systemSettings;
    }

    public void setSystemSettings(MyFidoSystem settings)
```
public ArrayList<Document> getDocumentsCollection()
{
    return _documents;
}

public void setDocumentCollection(ArrayList<Document> documents)
{
    _documents = documents;
}

public void setKnowledgeAccessObj(KnowledgeAccess obj)
{
    _knowledgeAccessObj = obj;
}

/**
 * THIS METHOD ACTS AS THE DRIVER METHOD FOR THE REST OF THE CLASS
 * THIS IS CALLED WHEN YOU WANT TO START ANALYZING THE CONTEXT OF A DOCUMENT
 * (AND IT'S ELEMENTS)
 * @throws java.io.IOException
 */

public void initContextAnalyzation() throws IOException
{
    // get rid of used objects, just in case we havn't been careful
    System.gc();

    // get all phrase detection and other rules
    _rules = new Rules();
    _allFoundPhrases = new ArrayList<String>();
    _allFoundWordsAndProperNouns = new ArrayList<String>();

    StringBuilder output = new StringBuilder();
    String[] bodyOfArticleTokens = null;
    String[] bodyOfArticleTags = null;
    String[] headlineTokens = null;
    String[] headlineTags = null;

    // DEBUG ME:
    //System.err.println("# of documents (context analyzer): " + _documents.size());
    //System.err.println("settings (context analyzer): " + MyFidoSystem.ChunkerModelPath);
    //System.err.println("session - username (context analyzer): " + _session.getCurrentUsername());
    //System.err.println("enum example (context analyzer): " + DocumentElementType.Author.toString());

    // iterate through all documents and analyze the gramatical context of the data in each
    // document element we are interested in
    // in this case we want the Headline and BodyOfArticle elements for now.
    for(int p = 0; p < _documents.size(); p++)
    {
        // STEP 1: POS TAGGING OF ALL ELEMENTS FOR EACH DOCUMENT
        // AND ADD THE TAGGED VERSION OF THE DOCUMENT ELEMENT'S VALUE
        for(int i = 0; i < _documents.get(p).getDocumentElements().size(); i++)
        {
            // step 1: get all document elements from a given document
            String elementType = _documents.get(p).getDocumentElements().get(i).getElementtype();

            // DEBUG ME:
            // System.out.println(elementType);

            // step 2: tokenize, tag and chunk the Headline and BodyOfArticle docuemnt elements
            if(elementType.equalsIgnoreCase(DocumentElementType.Headline.toString()))
            {
// get tokens of Headline document element
tokenizeArticleData(_documents.get(p).getDocumentElements().get(i).getValue());

// tag Headline document element
posTagDocElements(headlineTokens);

// chunk Headline document element
detectPhraseAndWords(headlineTokens, headlineTags);
}

else if(elementType.equalsIgnoreCase(DocumentElementType.BodyOfArticle.toString()))
{

// get tokens of BodyOfArticle document element
tokenizeArticleData(_documents.get(p).getDocumentElements().get(i).getValue());

// get tags of BodyOfArticle document element
posTagDocElements(bodyOfArticleTokens);

// DEBUG ME:
//splitSentencesTest(_documents.get(p).getDocumentElements().get(i).getValue());
detectPhraseAndWords(bodyOfArticleTokens, bodyOfArticleTags);
}

// step 3: send lists of words and phrases to rules method
_knowledgeAccessObj.findRelatedTermsInCategoryHierarchy(_allFoundPhrases);
_knowledgeAccessObj.findRelatedTermsInCategoryHierarchy(_allFoundWordsAndProperNouns);

/**
* Overload of initContextAnalyzation() method. This method
* acts the same except it tags, tokenizes, and chunks data from FeedItems
* in RSSFeed collections.
* It gets a string of topicIds and adds it to the topicsOfInterest property
* of the FeedItem object.
* @param analyzeRSSFeedItems
*/
public void initContextAnalyzation(boolean analyzeRSSFeedItems) throws IOException
{
  // get rid of used objects, just in case we haven't been careful
  System.gc();

  // get all phrase detection and other rules
  _rules = new Rules();

  String[] descriptionTokens = null;
  String[] descriptionTags = null;
  String[] titleTokens = null;
  String[] titleTags = null;

  if(analyzeRSSFeedItems == true)
  {
    // step 1: iterate through collection of RSSFeeds
    for(int p = 0; p < _allSupportedRSSFeeds.size(); p++)
    {
      ArrayList<FeedItem> items = _allSupportedRSSFeeds.get(p).getFeedItems();
      
      // get tokens of BodyOfArticle document element
tokenizeArticleData(items.get(i).getValue());

      // get tags of BodyOfArticle document element
posTagDocElements(items.get(i).getValue());

      // DEBUG ME:
//splitSentencesTest(items.get(i).getValue());
detectPhraseAndWords(items.get(i).getValue(), items.get(i).getValue());
    }

    // step 3: send lists of words and phrases to rules method
    _knowledgeAccessObj.findRelatedTermsInCategoryHierarchy(_allFoundPhrases);
    _knowledgeAccessObj.findRelatedTermsInCategoryHierarchy(_allFoundWordsAndProperNouns);
  }
}
// DEBUG ME:
System.err.println("FEED NAME: " + _allSupportedRSSFeeds.get(p).getFeedName());
System.err.println("items.size(): " + items.size());

// step 2: iterate through the collection of FeedItems, for each RSSFeed
for(int i = 0; i < items.size(); i++)
{
    // for each feeditem, get the description and title, analyze it
    // and store the topic of interests
    String title = items.get(i).getTitle();
    String description = items.get(i).getDescription();

    // DEBUG ME:
    System.out.println("title: " + title);
    System.out.println("description: " + description);

    // step 3: for each feed item, match the phrases and words
    // against the category hierarchy graph in the knowledge acess class
    _allFoundPhrases = new ArrayList<String>();
    _allFoundWordsAndProperNouns = new ArrayList<String>();

    //step 4: tokenize, tag, detect phrases and words for each feeditem's
    titleTokens = tokenizeData(title);
    titleTags = posTagData(titleTokens);
    detectPhraseAndWords(titleTokens,titleTags);
    descriptionTokens = tokenizeData(description);
    descriptionTags = posTagData(descriptionTokens);
    detectPhraseAndWords(descriptionTokens,descriptionTags);

    //System.err.println("_allFoundPhrases.size(): " + _allFoundPhrases.size());
    //System.err.println("_allFoundWordsAndProperNouns.size(): " +
    //_allFoundWordsAndProperNouns.size());
    String phrases =
        _knowledgeAccessObj.findRelatedTermsInCategoryHierarchy(_allFoundPhrases);         
    String properNounsAndWords =
        _knowledgeAccessObj.findRelatedTermsInCategoryHierarchy(_allFoundWordsAndProperNouns);
    String allTerms = "";

    // DEBUG ME:
    //System.err.println("phrases: " + phrases);
    //System.err.println("properNounsAndWords: " + properNounsAndWords);
    if(phrases != null && properNounsAndWords != null )
    {
        allTerms = phrases + properNounsAndWords;
    }
    else if(phrases != null)
    {
        allTerms = phrases;
    }
    else if(properNounsAndWords != null)
    {
        allTerms = properNounsAndWords;
    }

    if(allTerms != null || !allTerms.isEmpty())
    {
        String cleanedString = removeDuplicateIds(allTerms);
        // step 5: add allTerms to the feedItem's topicOfInterests property
        items.get(i).setTopicOfInterest(cleanedString);
        // DEBUG ME:
        System.err.println("cleanedString: " + cleanedString);
    }
}
private String[] tokenizeArticleData(String str) throws IOException {
    TokenizerME tokenizer = new Tokenizer("openNLP_models\tokenize\EnglishTok.bin.gz");
    String[] tokens = tokenizer.tokenize(str);
    return tokens;
}

/**
* This method detects phrases and words by chunking documents and then matching the
* tag sequences against stored rule patterns.
* @param tokens
* @param tags
*/
private void detectPhraseAndWords(String[] tokens, String[] tags)
{
    // DEBUG ME:
    //System.out.println("token length: " + tokens.length);
    try
    {
        MaxentModel maxEnt = getModel("openNLP_models\chunker\EnglishChunk.bin.gz");
        TreebankChunker chunker = new TreebankChunker(maxEnt);
        String phrase = "";
        String word = "";
        // GO GO gadget chunker!
        String[] chunks = chunker.chunk(tokens, tags);
        // step 2: besides formatting chunks so that we can debug things easier, we are iterating through
        // the chunks in order to detect phrases and keywords
        StringBuilder output = new StringBuilder();
        for (int currentChunk = 0, chunkCount = chunks.length; currentChunk < chunkCount; currentChunk++)
        {
            if (currentChunk > 0 && !chunks[currentChunk].startsWith("I-"") && chunks[currentChunk - 1].startsWith("O")
            {
                output.append(" ");
            }
            if (chunks[currentChunk].startsWith("B-"))
            {
                output.append(" [" + chunks[currentChunk].substring(2));
            }
            output.append(" " + tokens[currentChunk] + "/" + tags[currentChunk]);
        }
        // step 2: check tags and tokens against rule patterns for phrase detection
        if (currentChunk >= 0 && ((currentChunk + 1) < chunks.length) && ((currentChunk + 2) < chunks.length))
        {
            // iterate through rule count
            for (int ruleCount = 0; ruleCount < _rules.getRules().size(); ruleCount++)
            {
                Rule rule = _rules.getRules().get(ruleCount);
                String[] rulePattern = rule.getRulePattern().split("\\|\";
                // get each rule pattern and iterate through its fragments, match the
                // fragments against the POS tags (from tagging documents)
for(int k = 0; k < rulePattern.length; k++)
{
    if(rule.getPOSGramType() == 1) // match 3-POSgram
    {
        if( k >= 0 && ((k + 1) < rulePattern.length) && ((k + 2) < tokens[currentChunk + 2]);
        // if the rule pattern matches the next 3 tags then we
        // found a phrase, and should add it to the collection of found phrases
        if(tags[currentChunk].equalsIgnoreCase(rulePattern[k])
            && tags[currentChunk + 1].equalsIgnoreCase(rulePattern[k + 1])
            && tags[currentChunk + 2].equalsIgnoreCase(rulePattern[k + 2]))
        {
            phrase = tokens[currentChunk] + " " + tokens[currentChunk + 1] + " " +
            tokens[currentChunk + 2];
            // add phrases to phrase collection, if it doesn't already exist
            if(!_allFoundPhrases.contains(phrase))
            {
                // DEBUG ME:
                //System.err.println("phrase: " + phrase);
                _allFoundPhrases.add(phrase);
            }
        }
    }

    if(rule.getPOSGramType() == 2) //match 2-POSgram
    {
        if( k >= 0 && ((k + 1) < rulePattern.length) )
        {
            // if the rule pattern matches the next 3 tags then we
            // found a phrase, and should add it to the collection of found phrases
            if(tags[currentChunk].equalsIgnoreCase(rulePattern[k])
                && tags[currentChunk + 1].equalsIgnoreCase(rulePattern[k + 1]))
            {
                phrase = tokens[currentChunk] + " " + tokens[currentChunk + 1];
                // add phrases to phrase collection
                if(!_allFoundPhrases.contains(phrase))
                {
                    // DEBUG ME:
                    //System.err.println("phrase: " + phrase);
                    _allFoundPhrases.add(phrase);
                }
            }
        }
    }
}

// step 3: find keywords / proper nouns
// The NameFinder with the OpenNLP library works but isn't great
if(tags[currentChunk].equalsIgnoreCase("NNP")
|| tags[currentChunk].equalsIgnoreCase("NNPS")
|| tags[currentChunk].equalsIgnoreCase("NN") || tags[currentChunk].equalsIgnoreCase("NNS"))
{
    word = tokens[currentChunk];
    // add word to collection if it doesn't exist
    if(!_allFoundWordsAndProperNouns.contains(word))
    {
        // DEBUG ME:
        System.err.println("proper noun: " + word);
        _allFoundWordsAndProperNouns.add(word);
    }
}
if (chunks[chunks.length - 1] != "O")
{
    output.append(" "]);
}

output.append("\r\n");

// DEBUG ME:
System.out.println(output.toString());

//System.err.println("phrase collection length: " + _allFoundPhrases.size());
//System.err.println("word collection length: " + _allFoundWordsAndProperNouns.size());

catch (Exception ex)
{
    //ex.printStackTrace();
}

/**
 * This is the method used to tag document elements (6/13/2009)
 * @param test
 * @return
 */
private String[] posTagDocElements(String[] test)
{
    String[] tags = null;

    try {
        // DEBUG ME:
        // String test = "Google map tracks deadly Australia bushfires";

        MaxentModel maxEnt = getModel("openNLP_models\postag\tag.bin.gz");

        // GO GO gadget tagger!
        Dictionary dict = new Dictionary();
        POSTaggerME tagger = new POSTaggerME( maxEnt, dict);
        tags = tagger.tag(test);
    } catch (Exception ex)
    {
        ex.printStackTrace();
    }

    return tags;
}

.... (class code continues)
Section III: KnowledgeAccess Class

Please note, the class shown here is NOT the complete KnowledgeAccess class and may not be the same as the finally submitted code. These snippets show implementations of key MyFido system functionality.

```java
package FidosBrain;

import MyFidoMaster.MyFidoSystem;
import MyFidoUserManagement.Session;
import java.io.Serializable;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;
import org.jgrapht.alg.CycleDetector;
import org.jgrapht.alg.StrongConnectivityInspector;
import org.jgrapht.graph.SimpleDirectedGraph;
import uk.ac.shef.wit.simmetrics.similaritymetrics.JaroWinkler;
import uk.ac.shef.wit.simmetrics.similaritymetrics.Levenshtein;
import uk.ac.shef.wit.simmetrics.similaritymetrics.QGramsDistance;

/**
 * @author Teresa Brooks
 */
public class KnowledgeAccess implements Serializable {

    // TO DO: ADD serialVersionUID TO THIS CLASS

    // "constants" - these are used to established the acceptable range
    // of similarity when matching strings.
    private static final double EXACT_MATCH = 1.0;
    private static final double SIMILAR_MATCH = .96;

    // instance variables
    private Session _session = null;
    private MyFidoSystem _systemSettings;

    // instance variables - related to database connections and queries
    private Connection connection = null;
    private ResultSet rs;
    private PreparedStatement pstmt;
    private CallableStatement cstmt;
    private Statement stmt;

    // class variables - all static so that they will be used for any instance of this class

    // A simple directed graph. A simple directed graph is a directed graph in
    // which neither multiple edges between any two vertices nor loops are permitted.
    private static SimpleDirectedGraph<MyFidoVertex, MyFidoEdge> _graph;
```
// holds are vertex (entities where the key is the entityId / vertexId)
// will be able to look up the vertex by the key {id} so that we can add
// the vertex to the edge based on a vertex.
// will hold a (Long, MyFidoVertex) value object pair
private static Hashtable _vertexHashTable;

// will hold (Integer, MyFidoRelationship) so that it can be queried
// if necessary for the proper relationship to add to the Edge, where the key
// would be the Relationship Id
private static Hashtable _relationshipHashTable;

// collection of graph edges
private static ArrayList<MyFidoEdge> _edges;

// collection of dictionary entities
private static ArrayList<MyFidoDictionaryEntity> _entities;

// default, no-args constructor
public KnowledgeAccess()
{
    initKnowledgeAccess();
}

// getters & setters
public Session getKnowledgeAccessSession()
{
    return _session;
}

public void setKnowledgeAccessSession(Session currentSessionObject)
{
    _session = currentSessionObject;
}

public MyFidoSystem getSystemSettings()
{
    return _systemSettings;
}

public void setSystemSettings(MyFidoSystem settings)
{
    _systemSettings = settings;
}

private SimpleDirectedGraph<MyFidoVertex, MyFidoEdge> getGraph()
{
    return _graph;
}

private ArrayList<MyFidoEdge> getAllEdges()
{
    return _edges;
}

private Hashtable getAllRelationshipsHashTable()
{
    return _relationshipHashTable;
}

private Hashtable getAllKnowledgeDictionaryHashTable()
{
    return _vertexHashTable;
}

private void initKnowledgeAccess()
{
    // TO DO: PASS IN THE NUMBER OF VERTEX (ENTITIES IN THE KNOWLEDGE DICTIONARY TABLE)
    _vertexHashTable = new Hashtable();
    _relationshipHashTable = new Hashtable();
_graph = new SimpleDirectedGraph<MyFidoVertex, MyFidoEdge>(MyFidoEdge.class);

getAllKnowledgeDictionaryEntities();
getAllCategoryHierarchyEdges();

public void findRelatedTermsInCategoryHierarchy(ArrayList<String> str)
{
    Iterator<MyFidoVertex> iterator = new DepthFirstIterator<MyFidoVertex, MyFidoEdge>(_graph);
    MyFidoVertex v;
    MyFidoDictionaryEntity ent;

    // return collection {ArrayList<String>} of these terms and phrase that are related
    // to the start vertex (getting all end vertex's)
    for(int p = 0; p < str.size(); p++)
    {
        int count = 0;

        // step 1: get vertexId for given String {term/phrase},
        String termToTest = str.get(p);
        long vertexIdReturned = getVertexIdByEntryName(termToTest);

        // DEBUG ME:
        if(vertexIdReturned > 0)
        {
            System.out.println("termToTest: " + termToTest + "(" + vertexIdReturned + ")");

            // step 2: get MyFidoVertex object that corresponds with phrase/term (if it exists)
            //MyFidoVertex vertexObjReturned = getVertexFromHashTableById(vertexIdReturned);
            for(int k = 0; k < _edges.size(); k++)
            {
                long id = _edges.get(k).getStartVertexId();

                // get all end vertices that match the vertex object {
                //start vertex} that matches terms / phrases found
                if( id == vertexIdReturned)
                {
                    // DEBUG ME:
                    System.out.println("      get associated end vertices: " +
                                    _edges.get(k).getEndVertex().getDictionaryEntity().getEntityName());
                }
            } // DEBUG ME:
            System.err.println("    count: " + count + " (should be 157, as there are 157 entries in the
                            KnowledgeDictionary Table) ");
        }
    }
}
private void getAllKnowledgeDictionaryEntities()
{
    // local variables (entity)
    long entryId = 0;
    String entryName = "";
    int classificationType = 0;
    String classificationTypeName = "";

    long vertexId = 0;
    // local variables (objects)
    MyFidoDictionaryEntity entity;
    MyFidoVertex vertex;
    MyFidoRelationship relationship;
    MyFidoEdge edge;

    _entities = new ArrayList<MyFidoDictionaryEntity>();

    try
    {
        Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
        String url = "jdbc:odbc:MyFidoDev";
        connection = DriverManager.getConnection(url, "", "");
        stmt = connection.createStatement();
        rs = stmt.executeQuery("{call dbo.GetAllKnowledgeDictionaryEntries}");

        while (rs.next())
        {
            // step 1: get all Knowledge Dictionary entries
            entryId = rs.getInt("EntryId");
            entryName = rs.getString("EntryName");
            classificationType = rs.getInt("ClassificationType");
            classificationTypeName = rs.getString("ClassificationTypeName");

            // DEBUG ME:
            //System.err.println("entryId: " + entryId + " \nentryName: " + entryName + " \nclassificationType: " + classificationType + " \nclassificationTypeName: " + classificationTypeName);

            // step 2: initialize new Entity object
            entity = new MyFidoDictionaryEntity(entryId, entryName, classificationType,
                classificationTypeName);

            // step 3: add entity to collection (ArrayList)
            _entities.add(entity);

            // step 4: add entity to a new vertex object
            // note: the vertexId is the same as the entityId
            vertex = new MyFidoVertex(entryId, entity);
            vertexId = vertex.getVertexId();

            // step 5: add vertex objects to a hashtable
            // this hashtable will hold all the vertex objects for the
            // directed graph we are building in this class it will act as a reference
            _vertexHashTable.put(vertexId, vertex);
        }
    }
    catch (SQLException ex)
    {
        System.err.println(ex.toString());
    }
    catch (ClassNotFoundException clex)
    {
        System.err.println(clex.toString());
    }
    finally
    {
        // clean up database connections
    }
}
private void getAllCategoryHierarchyEdges()
{
    // local variables (edge)
    long edgeId = 0;
    long startVertexId = 0;
    MyFidoVertex startVertex;
    long endVertexId = 0;
    MyFidoVertex endVertex;
    MyFidoRelationship relationship;
    int relationshipId = 0;
    String relationshipDescription = "";

    MyFidoEdge edge;
    _edges = new ArrayList<MyFidoEdge>();
    boolean isEdgeAdded = false;

    try
    {
        Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
        String url = "jdbc:odbc:MyFidoDev";
        connection = DriverManager.getConnection(url, "", "");
        stmt = connection.createStatement();
        rs = stmt.executeQuery("{call dbo.GetAllCategoryHierarchyEdges}");
        while(rs.next())
        {
            edgeId = rs.getInt("EdgeId");

            // step 1: get startVertex object (by Id)
            startVertexId = rs.getInt("StartVertex");
            startVertex = getVertexFromHashTableById(startVertexId);

            // step 2: get endVertex object (by Id)
            endVertexId = rs.getInt("EndVertex");
            endVertex = getVertexFromHashTableById(endVertexId);

            // step 3: get relationship data and add to a relationship object
            relationshipId = rs.getInt("RelationshipType");
            relationshipDescription = rs.getString("RelationshipName");
            relationship = new MyFidoRelationship(relationshipId, relationshipDescription);

            // DEBUG ME:
            // System.err.println("edgeId: " + edgeId + "\nstartVertexId: " + startVertexId + "\nendVertexId: " + endVertexId + "\nrelationshipId: " + relationshipId + "\nrelationshipDescription: " + relationshipDescription);

            // step 4: add relationship objects to _relationshipHashTable hashtable
            _relationshipHashTable.put(relationshipId, relationship);

            // step 5: create new edge object
            edge = new MyFidoEdge(edgeId, startVertexId, startVertex, endVertexId, endVertex, relationship);

            // step 6: add edge objects to collection (ArrayList<MyFidoEdge>)
            _edges.add(edge);

            // step 7: last step...add start vertex, end vertex object and edge to _graph
            if(isEdgeAdded)
            {
                // DEBUG ME:
                // System.out.println("edge added successfully to graph! --> " + edge.getEdgeId());
            }
            else
            {

            }
        }
    }
}
// DEBUG ME:
//System.err.println("oops this is edge was not added: --> " + edge.getEdgeId());
}
}
}  
catch (SQLException ex)  
{
    System.err.println(ex.toString());
}
catch (ClassNotFoundException clex)
{
    System.err.println(clex.toString());
}
finally
{
    // clean up database connections
    cleanUpDBCConnections();
}
/**
 * Queries the _vertexHashTable hashtable, to get the vertex by given
 * key (vertexId)
 *
 * @param vertexId
 * @return MyFidoVertex
 */
private MyFidoVertex getVertexFromHashTableById(long vertexId)
{
    return (MyFidoVertex) _vertexHashTable.get(vertexId);
}
private long getVertexIdByEntryName(String paramStr)
{
    // this gets the entryId which is the same as the vertexId
    long returnId = 0;
    String testString = "";
    boolean isMatch = false;
    JaroWinkler jaro = new JaroWinkler(); // Jaro-Winkler similarity metric algorithm
    float similarityScore = 0;
    for(int p = 0; p < _entities.size(); p++)
    {
        testString = _entities.get(p).getEntityName();
        // format strings by making all lower case and trim possible left and right white space
        // this is done for the best possible matches
        testString = testString.toLowerCase().trim();
        paramStr = paramStr.toLowerCase().trim();
        // testing string similarity, if they are within the range of similarity
        // we assume the terms are the same or are
        // very similar, hence they are a match and we return the corresponding vertexId
        similarityScore = jaro.getSimilarity(testString,paramStr);
        if(similarityScore <= EXACT_MATCH && similarityScore >= SIMILAR_MATCH)
        {
            // DEBUG ME:
            //System.out.println("score: " + similarityScore);
            //System.out.println("testString: " + testString);
            //System.out.println("paramStr: " + paramStr);
            isMatch = true;
            if(isMatch)
            {
                returnId = _entities.get(p).getEntryId();
            }
        }
    }
}
//System.out.println("isMatch: " + isMatch);
//System.out.println("returnId: " + returnId);

break;
else
{
    // TO DO: THESE WORDS AREN'T MYFIDO'S VOCABULARY, LET'S ADD THEM TO THE KNOWLEDGE DICTIONARY.
}
}
}

return returnId;

/**
 * Builds the Category Hierarchy graph, returns true if graph didn't already contain
 * the given edge.
 * @param startVertex
 * @param endVertex
 * @param edge
 * @return boolean
 */
private boolean addVerticesEdgeToGraph(MyFidoVertex startVertex, MyFidoVertex endVertex, MyFidoEdge edge)
{
    boolean isAddedSuccess = false;

    // add start and end vertex objects (source, target) to the graph
    _graph.addVertex(startVertex);
    _graph.addVertex(endVertex);

    // add edge to the graph
    try
    {
        // The source and target (start and end) vertices must already be contained in this graph
        // returns true if graph did not already contain the specified edge.
        isAddedSuccess = _graph.addEdge(startVertex, endVertex, edge);
    }
    catch(IllegalArgumentException iae)
    {
        System.err.println(iae.toString());
    }

    return isAddedSuccess;
}

.... (class code continues)
public class Rules

   // instance variables
   private ArrayList<Rule> _rules;
   private Session _session = null;
   private MyFidoSystem _systemSettings;

   // instance variables - related to database connections and queries
   private Connection connection = null;
   private ResultSet rs;
   private PreparedStatement pstmt;
   private CallableStatement cstmt;
   private Statement stmt;

   // default, no-args constructor
   public Rules()
   {
      intRules();
   }

   // args constructors
   // getters & setters
   public ArrayList<Rule> getRules() {
      return _rules;
   }

   public Session getRulesSession() {
      return _session;
   }

   public void setRulesSession(Session currentSessionObject) {
      _session = currentSessionObject;
   }

   public MyFidoSystem getSystemSettings() {
      return _systemSettings;
   }

   public void setSystemSettings(MyFidoSystem settings) {
      _systemSettings = settings;
   }

Section IV: Rules Class

Please note, the class shown here is NOT the complete Rules class and my not be the same as the finally submitted code. These snippets show implementations of key MyFido system functionality.
/**
 * Get all rules out of the database, build rule objects and collection of rules
 */

public void intRules()
{
    // local variables
    Rule rule;

    try
    {
        Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
        String url = "jdbc:odbc:MyFidoDev";
        connection = DriverManager.getConnection(url, "", "");
        stmt = connection.createStatement();
        rs = stmt.executeQuery("{call dbo.GetAllRules}");

        _rules = new ArrayList<Rule>();
        while(rs.next())
        {
            // get rules
            rule = new Rule();
            rule.setRuleID(rs.getInt("RuleId"));
            rule.setRuleType(rs.getInt("RuleType"));
            rule.setPOSGramType(rs.getInt("POSGramType"));
            rule.setRulePattern(rs.getString("RulePattern"));
            rule.setRuleDescription(rs.getString("RuleDescription"));

            // add rules to collection
            _rules.add(rule);

            //System.err.println("adding rules!");
        }
    }
    catch (SQLException ex)
    {
        System.err.println(ex.toString());
    }
    catch (ClassNotFoundException clex)
    {
        System.err.println(clex.toString());
    }
    finally
    {
        // clean up database connections
        cleanUpDBConnections();
    }
}

.... (class code continues)
Section V: TheFetch Class

Please note, the class shown here is NOT the complete TheFetch class and my not be the same as the finally submitted code. These snippets show implementations of key MyFido system functionality.

```java
package MyFidoSystemComponents;
import MyFidoInternalObjects.FeedItem;
import MyFidoInternalObjects.RSSFeed;
import MyFidoMaster.MyFidoStatisticsRecorder;
import MyFidoMaster.MyFidoSystem;
import MyFidoUserManagement.UserInterestProfile;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;

/**
 * This class fetches, parses and prepares all system supported RSSFeeds for context analyzation, and creates system created custom feeds for users.
 */
public class TheFetch {
    // instance variables
    private Session _session = null;
    private MyFidoSystem _systemSettings;
    //private ArrayList<RSSFeed> _feedsWithTopicOfInterestsIds = new ArrayList<RSSFeed>();
    private ArrayList<FeedItem> _feedItemsWithTopicOfInterestsIds = new ArrayList<FeedItem>();
    private ArrayList<FeedItem> _customSuggestedFeedItems = new ArrayList<FeedItem>();
    // contains a comma seperated list of ObservedTopicsIds that matched FetchedTopicsIds
    // (by comparing the two master lists from the user interest profile of a given user)
    private String _topicsOfInterestsCombinedLists = null;
    // these are all the ObservedTopicIds that did not match the observed master list
    // was compared with the fetched master list:
    private String _nonMatchedTopicsOfInterestsObservedOnly = null;
    // object that records system statistics, used for recording testing results (accuracy testing)
    private MyFidoStatisticsRecorder _statsRecorder = null;
    // this stores a collection of RSS Syndicated feeds fetched from the "web"
    //private ArrayList<SyndFeed> _syndicatedFeeds = new ArrayList<SyndFeed>();

    // database connectivity instance variables
    private Connection connection = null;
    private ResultSet rs;
    private PreparedStatement pstmt;
    private CallableStatement cstmt;

    // default, no-args constructor
    public TheFetch(){}

    public enum MasterListType {
        Combined, ObservedOnly, FetchedOnly
    }

    // getters & setters
    public Session getFetchSession()
    {
    
```
public void initFetch()
{
    // step 1: create a collection of RSSFeeds that only have non-null or non-blank
    // topicsOfInterest properties.
    getFeedsWithTopicsOfInterests();

    // step 2: build custom feed
    //buildCustomSuggestedFeed();

    // DEBUG ME:
    //test();
}

private void getFeedsWithTopicsOfInterests()
{
    ArrayList<RSSFeed> allAnalyzedFeeds = _systemSettings.getAllFetchedFeeds();
    int totalNumberOfFetchedFeedItems = 0;
    for(int f = 0; f < allAnalyzedFeeds.size(); f++)
    {
        // DEBUG ME:
        System.out.println("feed name: " + allAnalyzedFeeds.get(f).getFeedName());
        ArrayList<FeedItem> items = allAnalyzedFeeds.get(f).getFeedItems();
        for(int i = 0; i < items.size(); i++)
        {
            String topicsOfInterestStr = items.get(i).getTopicsOfInterest();
        }
    }
}
if(!topicsOfInterestStr.equalsIgnoreCase(""))
{
    // DEBUG ME:
    System.out.println("title: " + items.get(i).getTitle());
    System.out.println("topicsOfInterestStr: " + topicsOfInterestStr + "\n\n");
    _feedItemsWithTopicOfInterestsIds.add(items.get(i));
}

    totalNumberOfFetchedFeedItems++;  
}

// SET STAT RECORDER PROPERTY
    _statsRecorder.setToalFetchedItemsMatchedCategoryHierarchy(_feedItemsWithTopicOfInterestsIds.size());
    _statsRecorder.setTotalFetchedArticles(totalNumberOfFetchedFeedItems);

    // DEBUG ME:
    System.out.println("counter (total feeds with topicIds): " + _feedItemsWithTopicOfInterestsIds.size());
}

public boolean buildCustomSuggestedFeed()
{
    // get current user's user interests profile
    UserInterestProfile profile = _session.getUserInterestProfile();

    String userObservedMasterList = profile.getUserObservedTopicsOfInterest();
    String fetchedFeedbackMasterList = profile.getFetchedFeedBackTopicsOfInterest();
    boolean observedIsEmpty = false;
    boolean fetchedIsEmpty = false;
    String[] observed = null;
    String[] fetched = null;
    String[] combined = null;
    boolean isAbleToSuggestFeed = false;
    if(userObservedMasterList == null)
    {
        observedIsEmpty = true;
    }
    if(fetchedFeedbackMasterList == null)
    {
        fetchedIsEmpty = true;
    }

    //DEBUG ME:
    System.out.println("observedIsEmpty: " + observedIsEmpty);
    System.out.println("fetchedIsEmpty: " + fetchedIsEmpty);

    // step 1: determine which set of master lists to use and how
    // depending on which lists are null
    if(fetchedIsEmpty == false && observedIsEmpty == false)
    {
        // if both the observedMasterList & fetchedFeedbackMasterList are not
        // null, then compare the two lists and return the list where
        // only the observed topics of interests that matched the fetched list
        // TO DO: WRITE METHOD TO CALL THAT RETURNS STRING OF IDS
        mergeMasterLists(userObservedMasterList, fetchedFeedbackMasterList);
        if(_topicsOfInterestsCombinedLists != null)
        {
            combined = _topicsOfInterestsCombinedLists.split("","");
            // TO DO:
            // compare feeditems with combined list ids
            // if matched add to _customSuggestedFeedItems <FeedItem> collection
            isAbleToSuggestFeed = matchMasterAgainstFeedItemTopics(combined, MasterListType.Combined);
        }
    
    return isAbleToSuggestFeed;
}
private boolean matchMasterAgainstFeedItemTopics(String[] topicIdsFromMasterList, MasterListType type) {
    boolean okToAdd = false;
    boolean doneAddingFirstTen = false;
    boolean doneAddingLastFour = false;
    boolean allFourteenAdded = false;
    int totalArticlesToSuggest = 21;
    int numberOfArticlesMyFidoToSuggest = 14;
    int numberOfRandomArticlesToSuggest = 7;
    boolean isAbleToSuggestFeed = false;
    int currentIndex = 0;
    int currentRandomRangeMax = 0;
    int randomIndex = 0;

    // stats related local variables
    int totalSuggestedByFetch = 0;
    int totalSuggestedByObserved = 0;
    int totalSuggestedRandom = 0;

    // if we have 14 or more articles that were analyzed and have
    // topicOfInterests properties that are not null or blank we are able to
    // build a custom feed and suggest articles to the user.
    if(_feedItemsWithTopicOfInterestsIds.size() >= totalArticlesToSuggest) {
        isAbleToSuggestFeed = true;
    }
    return isAbleToSuggestFeed;
}
if(type.toString().equalsIgnoreCase("Combined"))
{
    //DEBUG ME:
    System.out.println("type: " + type.toString());

    String[] nonMatchedObserved = _nonMatchedTopicsOfInterestsObservedOnly.split(",\"\"");
    System.out.println("topicIdsFromMasterList.length: " + topicIdsFromMasterList.length);
    System.out.println("nonMatchedObserved.length: " + nonMatchedObserved.length);

    // iterate through collection of FeedItems where all FeedItems
    // have a TopicOfInterest property that is not null and not blank
    for(int f = 0; f < _feedItemsWithTopicOfInterestsIds.size(); f++)
    {
        FeedItem item = _feedItemsWithTopicOfInterestsIds.get(f);

        //DEBUG ME:
        System.out.println("f (index) \[combined\]: " + f);

        // current feed's topics of interests property
        String[] feedItemsTopicsOfInterest = item.getTopicsOfInterest().split(",\"\");

        // this keeps use from checking the combined list over and over again
        // wether we add the first ten here or not.
        if(doneAddingFirstTen == false)
        {
            // iterate through all topic ids from master list and compare
            // the topic ids for the current FeedItem (matching against Fetched /Combined Master
            // List)
            for(int i = 0; i < topicIdsFromMasterList.length; i++)
            {
                String combinedMasterTopicId = topicIdsFromMasterList[i];

                // iterate through all topics of interests for a given feeditem
                for(int k = 0; k < feedItemsTopicsOfInterest.length; k++)
                {
                    String feedItemTopicId = feedItemsTopicsOfInterest[k];

                    // if the id's match add to collection of FeedItems
                    // to suggest to the user
                    if(combinedMasterTopicId.equalsIgnoreCase(feedItemTopicId))
                    {
                        // PERSONAL DEVELOPMENT NOTE (REMOVE LATER)
                        // => 5 because we are counting backwards
                        // from 14 to 1, 5 is the last number before we add
                        // the last 4 articles.
                        if(numberOfArticlesMyFidoToSuggest >= 5)
                        {
                            // make sure this feeditem hasn't been added to
                            // the suggested collection
                            if(!_customSuggestedFeedItems.contains(item))
                            {
                                okToAdd = validateFeedItem(item.getTitle());
                                if(okToAdd)
                                {
                                    item.setSelectedByMyFido(true);
                                    item.setMatchedInFetchedMasterList(true);
                                    item.setRandomSelected(false);
                                    _customSuggestedFeedItems.add(item);
                                    numberOfArticlesMyFidoToSuggest--;

                                    // STAT VARIABLE
                                    totalSuggestedByFetch++;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
// DEBUG ME:
System.out.println("   numberOfArticlesMyFidoToSuggest[combined!]: " + numberOfArticlesMyFidoToSuggest);
}
} else {
    doneAddingFirstTen = true;
    break;
}
}

// if we have added the first 10 articles that MyFido is to suggest
if(doneAddingFirstTen == true) {
    // DEBUG ME:
    System.out.println("**** COMBINED BRANCH: FIRST 10 FROM FETCHED-COMBINED LIST DONE! ****");

    break;
}
}

// DEBUG ME:
System.out.println("# of articles added {matched combined}: " + _customSuggestedFeedItems.size());

//System.out.println("offset: " + offset);
System.out.println("numberOfArticlesMyFidoToSuggest: " + numberOfArticlesMyFidoToSuggest);

for(int g = 0; g < _feedItemsWithTopicOfInterestsIds.size(); g++) {
    FeedItem item = _feedItemsWithTopicOfInterestsIds.get(g);

    // now iterate through nonMatchedObservedTopics and add last 4 feed items
    // to suggested feed item collection (this collection is full of feed items
    // that are to be suggested to the user)

    for(int j = 0; j < nonMatchedObserved.length; j++) {
        String nonMatchedObservedId = nonMatchedObserved[j];

        // current feed's topics of interests property
        String[] feedItemsTopicsOfInterest = item.getTopicsOfInterest().split(",");

        // iterate through all topics of interests for a given feeditem
        for(int s = 0; s < feedItemsTopicsOfInterest.length; s++) {
            String feedItemTopicId = feedItemsTopicsOfInterest[s];

            if(nonMatchedObservedId.equalsIgnoreCase(feedItemTopicId)) {
                // add last 4 feed items to suggested collection
                // if first 10 articles were added by matching the combined list
                // only 4 will be added here
                if(numberOfArticlesMyFidoToSuggest >= 1) {
                    // make sure this feeditem hasn't been added to
                    // the suggested collection
                    if(!_customSuggestedFeedItems.contains(item)) {
                        okToAdd = validateFeedItem(item.getTitle());
                        if(okToAdd) {

                            // DEBUG ME:
                            System.out.println("**** COMBINED BRANCH: FIRST 10 FROM FETCHED-COMBINED LIST DONE! ****");

                            break;
                        }
                    }
                }
            }
        }
    }
}
item.setSelectedByMyFido(true);
item.setMatchedInFetchedMasterList(false);
item.setRandomSelected(false);
_customSuggestedFeedItems.add(item);

numberOfArticlesMyFidoToSuggest--;

// STAT VARIABLE
totalSuggestedByObserved++;

//(pass to random number generating method)
currentIndex = g; //current feed item

// DEBUG ME:
System.out.println(" numberOfArticlesMyFidoToSuggest [non-matched observed!]": " + numberOfArticlesMyFidoToSuggest);
}
}
}

if(numberOfArticlesMyFidoToSuggest == 0)
{
    // DEBUG ME:
    System.out.println(" numberOfArticlesMyFidoToSuggest (is 0): " + numberOfArticlesMyFidoToSuggest);
    doneAddingLastFour = true;
    break;
}
}

if(doneAddingLastFour == true)
{
    // DEBUG ME:
    System.out.println("**** COMBINED BRANCH: ADDING MATCHES FROM --> NON-MATCHED FROM OBSERVED LIST LIST DONE! ****");
    break;
}

if(_customSuggestedFeedItems.size() == 14)
{
    // all done!
    allForteenAdded = true;

    // DEBUG ME:
    System.out.println("**** COMBINED BRANCH (allForteenAdded ?): " + allForteenAdded + "*****");
    System.out.println("**** _customSuggestedFeedItems.size(): " + _customSuggestedFeedItems.size() + " *****");
    break;
}
}
else
{
    //DEBUG ME:
    System.out.println("type: " + type.toString());

    // if we are here then then, only the ObservedMasterList or FetchedMasterList
    // has topic ids but not both
    // iterate through collection of FeedItems where all FeedItems
    // have a TopicOfInterest property that is not null and not blank
    for(int f = 0; f < _feedItemsWithTopicOfInterestsIds.size(); f++)
{  FeedItem item = _feedItemsWithTopicOfInterestsIds.get(f);
  String[] feedItemsTopicsOfInterest = item.getTopicsOfInterest().split(",");
  // DEBUG ME:
  System.out.println("f (index): " + f);
  for(int i = 0; i < topicIdsFromMasterList.length; i++)
  {
    String masterTopicId = topicIdsFromMasterList[i];
    // iterate through all topics of interests for a given feeditem
    for(int s = 0; s < feedItemsTopicsOfInterest.length; s++)
    {
      String feedItemTopicId = feedItemsTopicsOfInterest[s];
      if(masterTopicId.equalsIgnoreCase(feedItemTopicId))
      {
        if(numberOfArticlesMyFidoToSuggest >= 1)
        {
          // make sure this feeditem has only been added once
          // it could be added more than once if it master list contains
          // more than one of the feeditem's topicOfInterestIds
          // note: currently we are adding the FeedItem to our suggested collection
          // if we match on at least one topicId
          if(!_customSuggestedFeedItems.contains(item))
          {
            // before adding make sure the article isn't in this collection already
            // To do this I will check on it's title, as the artilce could have
            // appeared in more than one
            // of the synidated feeds fetched from the web, this is done to avoid
            // having duplicate articles show up in the feed we suggest to the user.
            okToAdd = validateFeedItem(item.getTitle());
            if(okToAdd)
            {
              // set feeditem's properties
              item.setSelectedByMyFido(true);
              if(type.equals(MasterListType.FetchedOnly))
              {
                item.setMatchedInFetchedMasterList(true);
                // STAT VARIABLE
                totalSuggestedByFetch++;
              }
              else
              {
                item.setMatchedInFetchedMasterList(false);
                // STAT VARIABLE
                totalSuggestedByObserved++;
              }
              item.setRandomSelected(false);
              // add to suggested feeditem collection
              _customSuggestedFeedItems.add(item);
              // (pass to random number generating method)
              currentIndex = f; //current feed item
              numberOfArticlesMyFidoToSuggest--;
              // DEBUG ME:
              System.out.println("numberOfArticlesMyFidoToSuggest: " +
                                  numberOfArticlesMyFidoToSuggest);
            }
          }
        }
      }
    }
  }
}
if (numberOfArticlesMyFidoToSuggest == 0)
{
    // DEBUG ME:
    System.out.println("numberOfArticlesMyFidoToSuggest (is 0): " + numberOfArticlesMyFidoToSuggest);
    allForteenAdded = true;
    break;
}

// break out of loop if MyFido's 14 suggested articles have been
// added to the suggested collection
if (allForteenAdded == true)
    break;

// DEBUG ME:
System.out.println("   allForteenAdded: " + allForteenAdded);

// break out of outer loop if MyFido's 14 suggested articles have been
// added to the suggested collection
if (allForteenAdded == true)
    break;

// TOOK RANDOM STUFF OUT THERE!!!!

}

// DEBUG ME:
System.out.println(" ** RANDOM BRANCH! *** ");
System.out.println("numberOfArticlesMyFidoToSuggest (# left to be suggested by MyFido): " + numberOfArticlesMyFidoToSuggest);

currentRandomRangeMax = (_feedItemsWithTopicOfInterestsIds.size() - 1);

// did MyFido suggest 14 articalcs?
if (allForteenAdded == true)
{
    // if yes, use numberOfRandomArticlesToSuggest as index
    randomIndex = numberOfRandomArticlesToSuggest;
}
else
{
    // if no, get number of feeditems that need to be added to make suggestion collection
    int currentLengthOfSuggestionCollection = _customSuggestedFeedItems.size();
    int totalNumberToSuggest = 21;
    randomIndex = totalNumberToSuggest - currentLengthOfSuggestionCollection;

    // DEBUG ME:
    System.out.println("    suggested collection's current length (not all 14 added!): " + currentLengthOfSuggestionCollection);
}

for(int x = randomIndex; x >= 1; x--)
{
    // DEBUG ME:
    System.out.println("x index: " + x);
// make sure the random number generator includes these two values as options.
int index = randomNumberGenerator(currentIndex, currentRandomRangeMax);

FeedItem itemRandom = _feedItemsWithTopicOfInterestsIds.get(index);
boolean addItem = validateFeedItem(itemRandom.getTitle());
if(addItem)
{
    itemRandom.setSelectedByMyFido(false);
    itemRandom.setMatchedInFetchedMasterList(false);
    itemRandom.setRandomSelected(true);
    _customSuggestedFeedItems.add(itemRandom);
    // STAT VARIABLE
    totalSuggestedRandom++;
}
else
{
    x = x + 1; // roll back 1 in order to retry
    // DEBUG ME:
    System.out.println("x index (rollback): " + x);
}

//DEBUG ME:
System.out.println("_customSuggestedFeedItems.size: " + _customSuggestedFeedItems.size());

// SET STAT PROPERTIES
_statsRecorder.setTotalArticlesSuggestedRandomly(totalSuggestedRandom);
_statsRecorder.setTotalArticlesSuggestedByMyFido(totalSuggestedByFetch + totalSuggestedByObserved);
_statsRecorder.setTotalSuggestedMatchedAgainstFetchMaster(totalSuggestedByFetch);
_statsRecorder.setTotalSuggestedMatchedAgainstObserverMaster(totalSuggestedByObserved);
_statsRecorder.setTotalSuggestedArticles(totalSuggestedRandom + totalSuggestedByFetch + totalSuggestedByObserved);

return isAbleToSuggestFeed;
}

private boolean validateFeedItem(String title)
{
    boolean okToAdd = true;
    for(int i = 0; i < _customSuggestedFeedItems.size(); i++)
    {
        String titleToCompare = _customSuggestedFeedItems.get(i).getTitle();
        // match on title (as we never edit the title)
        // if the title matches a title of a FeedItem in the customSuggestedFeedItems
        // then return false
        if(titleToCompare.equalsIgnoreCase(title))
        {
            okToAdd = false;
        }
    }

    return okToAdd;
}

private int randomNumberGenerator(int min, int max)
{
    int range = (max - min) + 1; // plus one includes the max in the range of numbers that can be selected
    int ranToReturn = min + (int)(Math.random() * range);
    // DEBUG ME:
    System.out.println("RANDOM INDEX GENERATED: " + ranToReturn);
}
private void mergMasterLists(String observedMasterList, String fetchedMasterList) {
    ArrayList<String> tempToReturn = new ArrayList<String>();
    ArrayList<String> nonMatchedObserved = new ArrayList<String>();
    String[] observed = observedMasterList.split(",");
    String[] fetched = fetchedMasterList.split(",");
    StringBuilder sb = new StringBuilder();
    StringBuilder sb2 = new StringBuilder();
    boolean isMasterMatched = false;
    boolean isAddedToNonMatchedList = false;

    for(int i = 0; i < fetched.length; i++) {
        for(int k = 0; k < observed.length; k++) {
            if(fetched[i].equalsIgnoreCase(observed[k])) {
                // if the observed id matches the fetched id, add arraylist
                // that is returned
                tempToReturn.add(fetched[i]);
                // if at least one match then this is true
                isMasterMatched = true;
            } else {
                nonMatchedObserved.add(observed[k]);
                isAddedToNonMatchedList = true;
            }
        }
    }

    // generate comma separated list (non-matched observed ids)
    for(int n = 0; n < nonMatchedObserved.size(); n++) {
        sb2.append(nonMatchedObserved.get(n));
        if(n != (nonMatchedObserved.size() - 1)) {
            sb2.append(",");
        }
    }
    if(isAddedToNonMatchedList == true) {
        // add to non matched string of ids (global properties)
        _nonMatchedTopicsOfInterestsObservedOnly = sb2.toString();
    }

    // generate comma separated list (ids matched from comparing master lists)
    for(int m = 0; m < tempToReturn.size(); m++) {
        sb.append(tempToReturn.get(m));
        if(m != (tempToReturn.size() - 1)) {
            sb.append(",");
        }
    }
    if(isMasterMatched == true) {
        // add matched string of ids (global property)
        // this is the list of ids where ids matched from
        // observed and fetched master lists
    }
}
_topicsOfInterestsCombinedLists = sb.toString();
}
}

.... (class code continues)
}
Section VI: MyFidoSystem Class

Please note, the class shown here is NOT the complete MyFidoSystem class and my not be the same as finally submitted code. These snippets show implementations of key MyFido system functionality.

/*
 * This class loads properties from the MyFidoSystem.properties file.
 * This file contains various system properties for MyFido.
 * This class also sets any system properites, that are needed throughout
 * the system. These properites include, fetched and parsed system supported
 * RSS feeds. These feeds are stored in a RSSFeeds collection.
 */

package MyFidoMaster;

import FidosBrain.KnowledgeAccess;
import MyFidoInternalObjects.FeedItem;
import MyFidoInternalObjects.RSSFeed;
import MyFidoSystemComponents.TheContextAnalyzer;
import com.sun.syndication.feed.synd.SyndEntryImpl;
import com.sun.syndication.feed.synd.SyndFeed;
import com.sun.syndication.io.SyndFeedInput;
import com.sun.syndication.io.XmlReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.net.URL;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

public class MyFidoSystem
{

    public static String ChunkerModelPath;
    public static String CorefModelPath;
    public static String NameFindModelPath;
    public static String ParserModelPath;
    public static String SentenceDetectModelPath;
    public static String TokenizeModelPath;
    private ArrayList<RSSFeed> _allSupportedRSSFeeds = new ArrayList<RSSFeed>();
    private ArrayList<String> _systemSupportedFeedURLs = new ArrayList<String>();

    // this stores a collection of RSS Syndicated feeds fetched from the "web"
    private ArrayList<SyndFeed> _fetchedSyndicatedFeeds = new ArrayList<SyndFeed>();

    // object that holds category hierarchy graph and graph related objects
    private KnowledgeAccess _knowledgeAccessObj;

    // holds default names of system supported feed names
    ArrayList<String> _defaultSystemSupportedFeedNames = new ArrayList<String>();

    // database connectivity instance variables
    private Connection connection = null;
    private ResultSet rs;
private PreparedStatement pstmt;
private CallableStatement cstmt;

public MyFidoSystem()
{
    Properties properties = null;
    properties = loadPropertyFile();

    // set system properties : paths to OpenNLP model files
    MyFidoSystem.ChunkerModelPath = properties.getProperty("chunkerPath");
    MyFidoSystem.CorefModelPath = properties.getProperty("corefPath");
    MyFidoSystem.NameFindModelPath = properties.getProperty("nameFindPath");
    MyFidoSystem.ParserModelPath = properties.getProperty("parserPath");
    MyFidoSystem.SentenceDetectModelPath = properties.getProperty("sentdetectPath");
    MyFidoSystem.TokenizeModelPath = properties.getProperty("tokenizePath");

    // get all system supported syndicated rss feeds
    // these feeds are used by TheFetcher object
    loadSystemSupportedSyndicatedRSSFeeds();

    // analyze FeedItem data
    analyzeFeedItems();
}

//getters & setters
public ArrayList<RSSFeed> getAllFetchedFeeds()
{
    return _allSupportedRSSFeeds;
}

public void setAllFetchedFeeds(ArrayList<RSSFeed> feeds)
{
    _allSupportedRSSFeeds = feeds;
}

public ArrayList<SyndFeed> getSyndicatedFeeds()
{
    return _fetchedSyndicatedFeeds;
}

public void setSyndicatedFeeds(ArrayList<SyndFeed> feeds)
{
    _fetchedSyndicatedFeeds = feeds;
}

public ArrayList<String> getAllSystemSupportedFeedURLs()
{
    return _systemSupportedFeedURLs;
}

public void setAllSystemSupportedFeedURLs(ArrayList<String> feeds)
{
    _systemSupportedFeedURLs = feeds;
}

public void setKnowledgeAccessObj(KnowledgeAccess obj)
{
    _knowledgeAccessObj = obj;
}

private Properties loadPropertyFile()
{
    // Read properties file.
    Properties properties = new Properties();

    try
    {
        properties.load(new FileInputStream("MyFidoSystem.properties");
    }
catch (IOException e)
{
   e.printStackTrace();
}

return properties;
}

private void loadSystemSupportedSyndicatedRSSFeeds()
{
   // step 1: get all system supported RSS Feed info and URLs
   getSystemSupportedRSSFeedInfo();

   // step 2: get syndicated rss feeds using URLs from step 1
   // this method also adds fetched feeds to system defined RSSFeed / FeedItem
   // objects
   getSyndicatedRSSFeedsFromWeb();

   // step 3: parse syndicated feeds, store data in RSSFeed and FeedItem
   // objects.
   //addFeedItemsToRSSFeedObjects();
}

/**
* This method uses TheContextAnalyzer to analyze RSSFeed FeedItems.
* Once the feeds are analyzed their topic of interest ids with each FeedItem.
*/
public void analyzeFeedItems()
{
   try
   {
      // analyze descriptions and titles of all RSSFeed FeedItems
      // after the feedItem's are analyzed, each feeditem will be given a
      // string containing the topicsOfInterestsIds that are related to that
      // particular feeditem (article). These are then stored
      // in the feeditem's topicsOfInterestsIds property
      TheContextAnalyzer contextAnalyzer = new TheContextAnalyzer();

      // Note: set properties > do not need to set all properties when
      // analyzing feed data
      // set RSSFeed collection
      contextAnalyzer.setAllFetchedFeeds(_allSupportedRSSFeeds);

      // set ContextAnalyzer KnowledgeAccess property (holds category hierarchy graph and graph related
      services and objects)
      contextAnalyzer.setKnowledgeAccessObj(_knowledgeAccessObj);

      // starts the ball rolling in analyzing the context of RSSFeeds
      // passing in true, lets us use the overloaded version of this method
      // which, analyzes RSSFeed objects and their FeedItem collections
      // the other version of this method analyzes collection of documents
      // and their document element collections
      contextAnalyzer.initContextAnalyzation(true);
   }
   catch (IOException ex)
   {
      System.out.println("Error: " + ex.getMessage());
   }
}

/**
* Method gets system supported RSS feed information, this information includes
* the URL of these rss feeds.
*/
private void getSystemSupportedRSSFeedInfo()
{
   // DEBUG ME:
   System.out.println("Getting System Supported RSSFeed Info ");
try {
    Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
    String url = "jdbc:odbc:MyFidoDev";
    connection = DriverManager.getConnection(url, "", "");
    cstmt = connection.prepareCall("{call dbo.GetAllSystemSupportedRSSFeedInfo}");
    rs = cstmt.executeQuery();
    while (rs.next()) {
        int feedId = rs.getInt("Id");
        int siteId = rs.getInt("SystemSupportedSiteId");
        String feedName = rs.getString("FeedName");
        String feedURL = rs.getString("FeedURL");

        // cache feed names, will need this when we add feed items to
        // system defined FeedItem and RSSFeed objects.
        _defaultSystemSupportedFeedNames.add(feedName);

        // DEBUG ME:
        // System.out.println("  feedId: " + feedId);
        // System.out.println("  siteId: " + siteId);
        // System.out.println("  feedName: " + feedName);
        // System.out.println("  feedURL: " + feedURL);
        // System.out.println("----------------------------");

        RSSFeed rssFeed = new RSSFeed();
        rssFeed.setFeedId(feedId);
        rssFeed.setAssociatedSiteId(siteId);
        rssFeed.setFeedName(feedName);
        rssFeed.setFeedURL(feedURL);

        // add to collection of RSSFeed objects
        _allSupportedRSSFeeds.add(rssFeed);
    }
} catch (SQLException ex) {
    System.err.println(ex.toString());
} catch (ClassNotFoundException clex) {
    System.err.println(clex.toString());
} finally {
    // clean up database connections
    cleanUpDBConnections();
}

/**
 * Grab's RSS Feeds from the web, using Rome library and it's objects.
 * Also adds these feeds to a collection of SyndFeed objects
 */
private void getSyndicatedRSSFeedsFromWeb() {
    // iterate through collection of system defined RSS feed objects
    // these objects contain the URLs for feeds we need to fetch from the web
    for (int f = 0; f < _allSupportedRSSFeeds.size(); f++) {
        // DEBUG ME:
        String feedName = _allSupportedRSSFeeds.get(f).getFeedName();
        try {
            // get feed's URL
            String strURL = _allSupportedRSSFeeds.get(f).getFeedURL();
            URL feedURLObj = new URL(strURL);
            } catch (MalformedURLException ex) {
                System.err.println(ex.toString());
            }
            // DEBUG ME:
            System.out.println("feedName: " + feedName);
            // DEBUG ME:
            System.out.println("feedURL: " + feedURLObj.toString());
            // DEBUG ME:
            System.out.println("----------------------------");
            }
            }
        } finally {
            // clean up database connections
            cleanUpDBConnections();
            }
            }
            }
        }
        }
    }
    }
} finally {
    // clean up database connections
    cleanUpDBConnections();
}
// get syndicated feed from web
SyndFeedInput input = new SyndFeedInput();

// About XML Reader: Character stream that handles (or at least attempts to) all the
// necessary Voodoo to figure out the charset encoding of the
// XML document within the stream.
SyndFeed feed = input.build(new XmlReader(feedURLObj));

// DEBUG ME:
// System.out.println(feed);
// System.out.println("------------------------------");

// add fetched SyndFeed object's to collection
_fetchedSyndicatedFeeds.add(feed);

// if entries list isn't empty, iterate through
// collection of entities
if(!feed.getEntries().isEmpty())
{
  List<SyndEntryImpl> entry = feed.getEntries();
  // iterate through collection of feed entries and store
  // data in FeedItem objects and add to the RSSFeed object's
  // collect of FeedItems
  for(int e = 0; e < entry.size(); e++)
  {
    int feedItemId = e;
    String pubDate = entry.get(e).getPublishedDate().toString();
    String feedItemTitle = entry.get(e).getTitleEx().getValue();

    // some of the feeds embed image tags for holding ads in the description
    // so we need to do a little clean up before storing the description
    String description = entry.get(e).getDescription().getValue();
    description = description.replaceAll("<img.*?>", "").trim();

    String url = entry.get(e).getUri();
    // DEBUG ME:
    // String test = description.replaceAll("<img.*?>", "");
    // System.out.println(" test: " + test);

    FeedItem item = new FeedItem();
    item.setItemId(feedItemId);
    item.setPublicationDate(pubDate);
    item.setTitle(feedItemTitle);
    item.setDescription(description);
    item.setArticleURL(url);

    // add feed item to RSSFeed's collection of FeedItems
    // this data will later be analyzed by the context analyzer
    // during a fetch session.
    _allSupportedRSSFeeds.get(f).getFeedItems().add(item);
    // DEBUG ME:
    // System.out.println(" pubDate: " + pubDate);
    // System.out.println(" feedItemTitle: " + feedItemTitle);
    // System.out.println(" description: " + description);
    // System.out.println(" url: " + url);
    // System.out.println("------------------------------");
  }
}

// DEBUG ME:
// System.out.println("FEED TITLE: " + feedName);
} catch (Exception e)
{
  System.err.println("error: " + e.getMessage());
}
Section VII: GUIDriver Class – getFetchPanel() Method

Please note, the class shown here is NOT the complete GUIDriver class and my not be the same as finally submitted code. These snippets show implementations of key MyFido system functionality.

* This is the method that iterates through collection of FeedItems that are suggested to the user, adds them to the FetchPanel and returns them to the calling method so that they are added to the GUI.

```java
private JPanel getFetchPanel(ArrayList<FeedItem> collection, final int userId, final int profileId)
{
    // will use the mig layout manager for swing...helps layout
    // components easily w/o using an editor wow wish I had found this before :)
    // http://www.miglayout.com/
    MigLayout mig = new MigLayout("wrap 2");
    fetchPanel2 = new JPanel(mig);
    fetchPanel2.setBackground(new java.awt.Color(255, 255, 255));
    fetchPanel2.setBorder(javax.swing.BorderFactory.createLineBorder(new java.awt.Color(0, 0, 0)));
    fetchPanel2.setPreferredSize(new Dimension(730,543));
    fetchPanel2.setVisible(true);

    // hide other panels loaded into the Center space of the layout before returning this panel
    if(moz != null)
    {
        moz.setVisible(false);
    }
    if(blankPanel != null)
    {
        blankPanel.setVisible(false);
    }

    // add instructional copy to panel
    String instructionStr = "<html><strong>MyFido created this custom RSS feed just for you! Please select
    the checkbox next to the article you enjoyed reading, and press the \"Save Feedback\" button.</br></br></br>Note: If you do not enjoy reading any of the articles MyFido selected, do NOT
    select any articles and press the \"Save Feedback\" button.</strong></html>";
    JLabel instructionalCopy = new JLabel(instructionStr);
    fetchPanel2.add(instructionalCopy, "wrap 15");

    String instructionStr2 = "<html><strong><font color=##ff0000>Note: If you do not enjoy reading any of the
    articles MyFido selected, do NOT select any articles and press the \"Save Feedback\" button.</font></strong></html>";
    JLabel instructionalCopy2 = new JLabel(instructionStr2);
    fetchPanel2.add(instructionalCopy2, "wrap 15");

    // this collection will store all of topicIds of the feed articles the user
    // read and liked!
    final ArrayList<String> likedCollection = new ArrayList<String>();

    // STAT VARIABLE
    final ArrayList<FeedItem> likeFeedItemCollection = new ArrayList<FeedItem>();
```
// get custom feed collection, this is feed that is suggested to the user based on their interest
ArrayList<FeedItem> customFeedItems = collection;

// iterate through the feed and add each feed item to the panel for display
// also show checkbox and read me button that opens the article's url in the
// user's default browser
for(int x = 0; x < customFeedItems.size(); x++)
{
    final String articleUrl = customFeedItems.get(x).getArticleURL();
    final String topicsOfInterest = customFeedItems.get(x).getTopicsOfInterest();
    final String articleTitle = customFeedItems.get(x).getTitle();
    final FeedItem currentFeedItem = customFeedItems.get(x);

    // create checkbox that will hold the topic of interest ids of it's respective
    // feed item in it's name property this will act as an identifier
    // (note: title may act as identifier if needed as well)
    final JCheckBox feedItemCheckBox = new JCheckBox(articleTitle);
    feedItemCheckBox.setBackground(new java.awt.Color(255, 255, 255));
    feedItemCheckBox.setName(topicsOfInterest);
    feedItemCheckBox.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {
            boolean isLiked = feedItemCheckBox.isSelected();
            String likedTopicOfIds = feedItemCheckBox.getName();

            // DEBUG ME:
            System.out.println("CHECKBOX SELECTED:");
            System.out.println("isSelected:");
        
            if(isLiked)
            {
                likedCollection.add(likedTopicOfIds);
                // SET FOR TESTING
                likeFeedItemCollection.add(currentFeedItem);
            }
            else
            {
                // if this article is not selected and it is
                // in the collection we must remove it!
                if(likedCollection.contains(likedTopicOfIds))
                {
                    // DEBUG ME:
                    System.out.println("[REMOVED] : " + likedTopicOfIds);
                    likedCollection.remove(likedTopicOfIds);
                    // SET FOR TESTING
                    likeFeedItemCollection.remove(currentFeedItem);
                }
            }
        }
    });
}
// create new button for each feed item
// this button will open the given url for it's corresponding feed item
JButton readButton = new JButton("read");
readButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            // gets the default browser (or IE if it's the default browser)
            // this gets the default browser and executes the app and passes the url to open to it
            Runtime.getRuntime().exec("rundll32 url.dll,FileProtocolHandler " + articleUrl);
        }
        catch (IOException ex) {
            System.out.println(ex.getMessage());
        }
    }
});
fetchPanel2.add(feedItemCheckBox);
fetchPanel2.add(readButton);

JButton saveButton = new JButton("Save Your Feedback!");
saveButton.addActionListener(new ActionListener() { 
    public void actionPerformed(ActionEvent e) {
        // TO DO: SET SOME OTHER STAT PROPERTIES HERE & SAVE STATS
        _statsRecorder.setLikedCollection(likeFeedItemCollection);
        _statsRecorder.saveAllStats();
        // build string of topicIds, these strings are associated with
        // feed articles that user said they enjoyed reading
        // this string will be stored in the FetchedTopicsSnapshot database table
        final String allFetchedTopicsOfInterest = buildFetchedSnapshotTopicsOfInterest(likedCollection);
        // DEBUG ME:
        System.out.println("allFetchedTopicsOfInterest: " + allFetchedTopicsOfInterest);
        // when user clicks save: do this..save fetched topic snapshot
        _session.getUserInterestProfile().saveTopicsToSnapshot(allFetchedTopicsOfInterest, userId, profileId, SnapshotType.FETCHED);
    }
});
fetchPanel2.add(saveButton, "gaptop 20px");

return fetchPanel2;
Section VII: GUIDriver Class – ActionListener: actionPerformed() - System Login

Please note, the class shown here is NOT the complete GUIDriver class and may not be the same as the finally submitted code. These snippets show implementations of key MyFido system functionality.

* GUIDriver login code

```java
boolean isSuccess = false;

String username = usernameTxtField.getText();
char[] arrPassword = passwordField.getPassword();
String password = "";
for(int i = 0; i < arrPassword.length; i++)
{
    password += arrPassword[i];
}

// logIn()
_user = new User();
isSuccess = _user.logIn(username, password);

if(isSuccess)
{
    _user.setUsername(username);
    _session = new Session();
    _session.setSessionId(_session.createSessionId(_user.getUserId()));
    _session.setCurrentUsername(username);
    _session.setCurrentUserId(_user.getUserId());
    _session.setIsLoggedIn(isSuccess);

    // get user interest profile
    UserInterestProfile profile = new UserInterestProfile();
    profile.getObserverSnapshotData(_user.getUserId());
    profile.getFetchedSnapshotData(_user.getUserId());

    // saves master lists to UserInterestProfile table
    profile.updateUserInterestProfile(_user.getUserId());

    // get current user interest profile
    profile.loadUserInterestProfile(_session.getCurrentUserId());

    // set Session & User object's UserInterestProfile property
    _session.setUserInterestProfile(profile);
    _user.setUserInterestProfile(profile);

    // set all dynamic text in the GUI (username, etc)
    setAllDynamicText();

    // enable GUI navigation buttons
    enableAllGUINavigation();

    // set flag - user has not had a browser session yet as they // have just logged in
    _isBrowserSessionSet = false;
}
```
Section VIII: XUL Application – Test Code

Please note, the code below was created to evaluate the basic capabilities of the XUL framework. This code is not complete and may vary from its final iteration.

XUL – XML (GUI)

```xml
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<?xml-stylesheet href="stats.css" type="text/css"?>
<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul" id="statswindow"
title="View Web page stats">
  <script type="application/x-javascript" src="stats.js"/>
  <vbox flex="1">
    <hbox>
      <label value="Enter a URL:"></label>
      <textbox id="url" flex="1" value="http://www.ibm.com/developerworks/web"/>
      <button label="Go!" oncommand="change_url(event)"/>
      <button label="you" oncommand="callLight()"/>
    </hbox>
    <description value="Current page:"></description>
    <hbox flex="1">
      <iframe id="contentview" src="http://www.ibm.com/developerworks/web" flex="2"/>
      <vbox>
        <groupbox>
          <caption label="Stats"></caption>
          <grid flex="1">
            <columns>
              <column flex="1"/>
              <column flex="1"/>
            </columns>
            <rows>
              <row>
                <label value="Word count"></label>
                <textbox class="count" id="wordcount" value="N/A" readonly="true"/>
              </row>
              <row>
                <label value="Character count"></label>
                <textbox class="count" id="charcount" value="N/A" readonly="true"/>
              </row>
              <row>
                <label value="Element count"></label>
                <textbox class="count" id="elemcount" value="N/A" readonly="true"/>
              </row>
            </rows>
          </grid>
        </groupbox>
        <spacer flex="1"/>
      </vbox>
    </hbox>
  </vbox>
</window>
```
XUL - JavaScript Code

//Invoked in response to a click on the "Go!" button
function change_url(event)
{
    //Variables for convenient access to specific elements in the XUL
    var urlbox = document.getElementById("url");
    var contentview = document.getElementById("contentview");
    var wordcountbox = document.getElementById("wordcount");
    var charcountbox = document.getElementById("charcount");
    var elemcountbox = document.getElementById("elemcount");

    //test2();
    //test(urlbox.value);

    //Setting this attribute, happens to change the visible contents of the panel
    contentview.setAttribute("src", urlbox.value);
    alert(urlbox.value);

    //Fake up the update code for now, to allow running in Firefox
    wordcountbox.previousSibling.value += " (fake)";
    wordcountbox.value = "1000";
    charcountbox.previousSibling.value += " (fake)";
    charcountbox.value = "100";
    elemcountbox.previousSibling.value += " (fake)";
    elemcountbox.value = "10";
}

function test()
{
    netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");

    // create an nsILocalFile for the executable
    var file = Components.classes["@mozilla.org/file/local;1"].createInstance(Components.interfaces.nsILocalFile);
    file.initWithPath("C:\wgetLives\wget.exe");
    alert("file: " + file.value);

    // create an nsIProcess
    var process = Components.classes["@mozilla.org/process/util;1"].createInstance(Components.interfaces.nsIProcess);
    process.init(file);

    // Run the process.
    // If first param is true, calling thread will be blocked until
    // called process terminates.
    // Second and third params are used to pass command-line arguments
    // to the process.
    var args = ["-oa test.txt","http://www.google.com"];
    process.run(true, args, args.length);
}
function test2()
{
    netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");

    //const path = "C:\\WINDOWS\\CHARMAP.EXE";
    const path = "launchThisFile.bat"

    var file = Components.classes["@mozilla.org/file/local;1"].createInstance(Components.interfaces.nsILocalFile);
    file.initWithPath(path);
    file.launch();
}

function callLight()
{
    netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");

    var getWorkingDir = Components.classes["@mozilla.org/file/directory_service;1"].getService(Components.interfaces.nsIProperties).get("CurProcD", Components.interfaces.nsIFile);

    var lFile = Components.classes["@mozilla.org/file/local;1"].createInstance(Components.interfaces.nsILocalFile);

    var lPath = getWorkingDir.path + "\TestFidoObserver\wget.exe"
    alert("file: " + lPath);

    lFile.initWithPath(lPath);
    var process = Components.classes["@mozilla.org/process/util;1"].createInstance(Components.interfaces.nsIProcess);
    process.init(lFile);

    var args = ["-o test.txt","http://www.google.com"];
    process.run(true, args, args.length);
}