The Search for Meaning in Large Text Databases

by

Daniel Frederick Gruhl

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

Doctor of Philosophy in Electrical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2000

© Massachusetts Institute of Technology 2000. All rights reserved.

[Signature]

Department of Electrical Engineering and Computer Science
December 10, 1999

[Signature]

Marvin Minsky
Toshiba Professor Of Media Arts And Sciences
Thesis Chairman

[Signature]

Arthur C. Smith
Chairman, Department Committee on Graduate Students
The Search for Meaning in Large Text Databases

by

Daniel Frederick Gruhl

Submitted to the Department of Electrical Engineering and Computer Science on December 10, 1999, in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Electrical Engineering

Abstract

The task of retrieving information is becoming increasingly complicated as larger corpora are searched. Systems that employ single-pass statistical methods often swamp the user with an excessive number of hits or fail to find any exact matches and return none. This thesis examines a system that employs richer representation of the texts in its corpus. This understanding is developed by a collection of experts that examine articles in the database, looking at both the text itself and the annotations placed by other agents. These agents employ a variety of methods ranging from statistical examination to natural-language parsing to query expansion through specific-purpose knowledge bases. Retrieval servers use these augmented feature sets to provide more useful responses to user queries.

Thesis Chairman: Marvin Minsky
Title: Toshiba Professor Of Media Arts And Sciences
Acknowledgments

There are many people who deserve thanks for helping to make this thesis a reality. First and foremost is Walter Bender, for providing an opportunity to work in one of the most interesting of research groups, as well as for his continual support and guidance as this process threaded its way through the intricacies of the Course VI graduate program.

Second, I would like to thank my entire committee for their time in reviewing this research and critiquing the approach. Especially I would like to thank Ken Haase who has fielded no end of questions and served as an invaluable resource throughout this process.

Research is not a solitary process, and I would like to thank the entire Electronic Publishing Group of the MIT Media Lab, past and present for their insights and assistance, particularly Jon Orwant, Sunil Vemuri and Warren Sack, as well as Felice Gardner who manages assure that every-thing runs much more smoothly than it has any right to.

This thesis has been at times a convoluted collection of networked hardware impinging on an already taxed computing infrastructure. I owe those who maintain the labs network and computing resources a debt of gratitude, especially Jane Wojcik, Michail Bletsas, Will Glesnes and Jon Ferguson.

The sponsors of the Media Lab are always a great resource and their feedback has helped to guide this work in ever more productive directions. I would in particular like to thank John Coate and the entire staff of SFGate for their willingness to try Fishwrap and ZWrap, and their encouragement throughout this thesis.

Outside of the lab, I owe thanks to IBM; particularly Rakesh Agrawal for his support and flexibility as this degree process concludes, David Boor for his work as the liaison to the Media Lab, and the entire QUEST team for introducing me to Data Mining and explaining the intricacies of the field.

Personally, I would like to thank those who helped keep me sane throughout this degree, particularly the MIT Shakespeare Ensemble, and especially Ann-Marie White, Abby Spinak, Portia Vescio, Liz Stoher and Erin Lavik. Also, Leah Leier, Tony Leier, Hani Sallum, Joe Calzaretta, and Jen Mills for the weekly dose of escapist entertainment. And of course, Paula Wirth, for the wonderful illustrations, but more importantly for reminding me there is life beyond thesis.

Lastly, and most importantly, my family, without whose support, encouragement, and love I never would have attempted, much less completed this degree.
Contents

1 Introduction 12
   1.1 Advantages of a human editor .................................. 13
       1.1.1 "In today's Journal of the American Medical Association...." 13
       1.1.2 "Dear Editor," ............................................. 13
       1.1.3 On your doorstep .......................................... 14
       1.1.4 The Alta Times? .......................................... 14
       1.1.5 I don't know what I want! .............................. 14
       1.1.6 I want what he's got! .................................. 14
   1.2 The Big Picture .................................................. 15
       1.2.1 Pre-cognition ............................................ 15
   1.3 Thesis Statements ................................................ 16
   1.4 Organization ..................................................... 16

2 Blackboard Systems 18
   2.1 Introduction .................................................... 18
   2.2 Blackboards and News ........................................... 20
   2.3 Implementation .................................................. 20
       2.3.1 Dependency .............................................. 22
   2.4 Blackboards vs. Rule Based Systems ........................... 23
   2.5 Conclusion ...................................................... 25

3 Data Representation 26
   3.1 Introduction ..................................................... 26
   3.2 Searching ........................................................ 28
   3.3 Implementation .................................................. 28
4 Network Protocol
4.1 Introduction
4.1.1 The Dtypes Concept
4.2 Expert Distribution
4.3 Efficiency
4.4 Other DType Services
4.5 Conclusion

5 Searching in Augmented Framestores
5.1 Boolean Searches
5.2 Alta Vista style searches
5.3 Activation/Evidential Searches
5.4 Relevance Feedback
5.5 News Channels
5.6 Additional tools for searching
5.6.1 Phrase Searching
5.6.2 Description Logics/Named Subqueries
5.6.3 Experts
5.7 Conclusion

6 Statistics
6.1 Introduction
6.2 Why Use Statistical Techniques?
6.3 Techniques
6.3.1 Simple Apriori Occurrence Expectation
6.3.2 Related Articles
6.3.3 Association rule data mining
6.4 Clustering
6.4.1 Presentation
6.4.2 Cluster Management
6.4.3 Dimensionality Reduction
6.5 Conclusion
7 User Interface

7.1 Share Everything ........................................... 55
7.2 Marginal Notes ............................................. 56
7.3 What was it thinking? ........................................ 56
7.4 Truth in Advertising ......................................... 59
7.5 Clipping Basket ............................................. 59
7.6 Channel Structure ........................................... 60
7.6.1 Tabs ..................................................... 60
7.7 Alternative Delivery Mechanisms ............................. 60
7.8 Conclusion .................................................. 61

8 Implementation .................................................. 62

8.1 Introduction ................................................. 62
8.2 Projects ..................................................... 62
8.3 General System Flow ........................................ 63
8.4 News Streams ............................................... 64
8.4.1 Wire Services ............................................ 64
8.4.2 Web news sources ........................................ 64
8.4.3 Net News ................................................ 65
8.4.4 Internal Discussion ........................................ 65
8.4.5 Email/Calendar ........................................... 65
8.5 Framestores .................................................. 66
8.5.1 Flat file .................................................. 66
8.5.2 DBM Style ............................................... 67
8.5.3 SQL ..................................................... 68
8.5.4 Inheritance and Reference ................................. 69
8.6 Indexing ..................................................... 69
8.6.1 FramerD and ZWrap Style Indices ....................... 69
8.7 Search Engines ............................................... 71
8.8 Experts ...................................................... 71
8.9 Statistical Examination ..................................... 72
8.9.1 Association Rules ........................................ 73
8.9.2 Occurrence Statistics .................................... 73
8.9.3 Clustering, Cluster Naming ............................... 74
8.9.4 Nearest Neighbor ........................................... 74
8.9.5 The Difference Engine ..................................... 74
8.10 Tools to assist presentation and user interface .......... 74
  8.10.1 Top Level Presentation .................................. 75
  8.10.2 Searches vs. Repurposed Information .................. 75
  8.10.3 Channel Creation ......................................... 76
  8.10.4 Tools for Channel Analysis ............................... 76
8.11 Email, Pagers, etc. ........................................... 78
8.12 System Status .................................................. 78
8.13 Conclusion ...................................................... 78

9 Results ..................................................................... 80
  9.1 Blackboard Approach .......................................... 81
      9.1.1 Flexibility .................................................. 82
      9.1.2 Scalability While Preserving Parallelizability ....... 83
  9.2 Machine Understanding Informing Statistics ............... 85
      9.2.1 Experimental Protocol .................................... 87
      9.2.2 Experiment Results ....................................... 88
  9.3 Statistics Informing Machine Understanding ............... 91
      9.3.1 Closing the Circle ......................................... 94
  9.4 Conclusion ....................................................... 94

10 Conclusions ........................................................ 95
  10.1 Contributions .................................................. 96
  10.2 Future Work .................................................... 97

A Dtypes Specification .................................................. 99

B Framestore Protocols ............................................... 107
  B.1 FramerD Framestore Access .................................... 107
     B.1.1 pool-data .................................................... 107
     B.1.2 get-load ..................................................... 107
     B.1.3 new-oid ...................................................... 107
     B.1.4 lock-oid ...................................................... 108
     B.1.5 unlock-oid .................................................... 108
B.1.6 clear-lock ................................................. 108
B.1.7 oid-value .................................................... 108

B.2 ZWrap Framestore Additions ................................. 108
  B.2.1 frame-create ........................................... 109
  B.2.2 fkeys ..................................................... 109
  B.2.3 fget ....................................................... 109
  B.2.4 fadd ........................................................ 109
  B.2.5 fzap ........................................................ 109
  B.2.6 fdrop ....................................................... 109
  B.2.7 fset ........................................................ 110
  B.2.8 ftest ....................................................... 110
  B.2.9 fsearch (optional) ....................................... 110
  B.2.10 ifget (optional) ......................................... 110

B.3 FramerD Index Server ........................................ 110
  B.3.1 iserver-get ............................................... 111
  B.3.2 iserver-add ............................................... 111
  B.3.3 iserver-get-size ......................................... 111
  B.3.4 iserver-writable ......................................... 111
  B.3.5 iserver-keys .............................................. 111

B.4 ZWrap Index Server ............................................ 111
  B.4.1 iserver-ordered-get ...................................... 111
List of Figures

2-1 On the left, lots of experts, all standing around the same blackboard. They all look at the blackboard and, when one has something to contribute, they grab the chalk (if it is available) and write their thoughts down. All communication is done via the blackboard and only one expert is allowed to write at a time. On the right as you might imagine, this can lead to a lot of frustrated agents! ........................................... 19

2-2 With a group of blackboards, experts can wander around and work on whichever blackboard is free. ................................................................. 21

2-3 The proper-noun expert adds a list of the proper nouns found in a story. The person expert examines this list to identify people's names and writes them into the story also. 22

2-4 Before the Person expert operations on a set of frames, it checks the dependency scratch space to see if the Proper Noun expert has already visited it. This structure supports single instances of each agent, but can be easily modified to support more. 23

2-5 Differing topologies for rule based(left) and ZWrap expert based(right) systems... 24

3-1 A frame describing the author. ............................................................... 26

3-2 SQL table used to store frames. Note, set valued attributes are stored on multiple table rows. ................................................................. 30

6-1 On the left, traditionally statistical analysis on text is done by first creating a word-document matrix on which to perform analysis. ZWrap uses feature-document matrices, such as the one on the right. ........................................... 47

7-1 A portion of the raw frame for a story, with the keys in bold and the values on the lines that follow. ............................................................... 57

7-2 When an article is displayed, additional information is included as marginal notes (dotted boxes added here for illustration only). ........................................... 58
8-1 Overall system diagram for the approach presented in this thesis

8-2 A flat-file database. The index at the front of the file refers to each frame that is stored in a contiguous block later in the file.

8-3 The ZWrap index uses a DBM style file to hold positional information, and a flat file to hold the sorted index. Note, space is left at the end of the index vector to accommodate more entries without the need to do a recopy.

8-4 The top level MyZWrap presentation is done as channels. Here are 3 channels from Cameron Marlow's paper.

8-5 Feature analysis for the Kosovo topic. The higher level derived features appear on the left. On the right are the low level (stemmed word) features.

8-6 Keeping a ZWrap system running requires a way to track which servers are up and which are down. MyZWrap employs the "Big Brother" system monitor package to accomplish this task.

9-1 ZWrap allows agents to be distributed among arbitrary numbers of machines. This is helpful up to a point. If the above system is performance bound by the stemmer (running by itself on one machine) then spreading the other agents to more machines will not result in a performance improvement.

9-2 The multiple-blackboard architecture provides a natural way to break up the problem. Multiple instances of slower experts (e.g., the stemmer) can all be working on different parts of the framestore at the same time.

9-3 If one particularly slow expert is amenable to parallization itself, then a particular node can be replaced with a parallel computation structure such as a Beowulf or SP2.

9-4 One possible scheme for distributing the load on the framestore across three machines based on a trivial hash of the frame object ID. Due to the access patterns on ZWrap, a better algorithm might be \( \lfloor \text{oid} / 128 \rfloor \% 3 \) (supporting bulk reads and writes), but of course this is application dependent.
List of Tables

5.1 Evidential searches use various names to express degrees of certainty. These names can be modified by the expression “supports”, which means the score is added, or “refutes”, which means it is subtracted. ........................................... 42

8.1 Associations between Richard Butler and other proper nouns over a two week period in February. This is the complete rule list generated with a support requirement of 20 articles and a confidence requirement of 50%. .................................................. 73

9.1 Categorization results for North America. .................................................. 88
9.2 Categorization results for South America. .................................................. 89
9.3 Categorization results for Asia. ................................................................. 89
9.4 Categorization results for Middle East. ...................................................... 89
9.5 Categorization results for Middle America. ................................................. 90
9.6 Categorization results for Europe. ............................................................. 90
9.7 Suggested rules for 'Christopher Hill'. The exact nature of the implication would have to be developed independently of the simple association rule discovery. .......................... 93
9.8 Rules proposed for Yugoslavia. ................................................................. 93
9.9 West Bank rules. Note, the statistical system does not “know” the relationship between Israel and Israeli. This is one place where a knowledge base would allow for better rule mining (perhaps built by noting the bijection 'Israel' ↔ 'Israeli' has a high confidence and support). ................................................................. 94
Chapter 1

Introduction

News editing is an exacting problem. Many factors contribute to making the selection and presentation of timely, relevant information a task as daunting as it is necessary. People want to be kept informed of events and occurrences that impact them, but at the same time don’t want to wade through the tens of thousands of news stories available every day to find what they need.

And it’s not just a matter of deciding which stories may be of interest. How much is enough, and how much is too much is a delicate balance to strike. One Lewinsky story a week might be interesting. Fifty might not be. A person’s source of news needs to express “what’s new”, rather than just “what’s happened”. However, if someone has relatives in Algiers, then every story about a revolution that occurs there may be of interest.

The average person has many interests, that compete for their time. The time one can spend on the news each day is more or less fixed and decisions must be made regarding what to present, in what order, and in what way. It requires understanding on the editor’s part as to not only what a given story is about, but also the context\(^1\), how particular story relates to other stories, as well as

---

\(^1\) According to Webster, understanding means “an act or result of interpreting” and context “the parts of a discourse that surround a word or passage and can throw light upon its meaning”. However, different fields have taken these words to have much more specific, technical meanings that unfortunately do not always agree. This thesis, since it spans several fields, must thus make clear what definition it assigns to these two words.

I take the understanding of an article to be the result of examining the article and identifying features in it. These features may be as simple as the individual words that appear, or as complex as the actors and actions they perform in an article, or the bias with which a particular article is written.

By context, I mean how these features relate to each other, most especially the way in which they tend to occur in a large number of articles. This can include, for example, co-occurrence of features, association and implication of features, or ways in which some features tend to perform some actions on some other features.

Prof. Bender, noting the ambiguity in the use of these terms, has suggested *intra*standing and *inter*standing as better labels for these two concepts. However I will sacrifice this additional level of precision in the interest of readability and employ “understanding” and “context” in the sense defined above.
how it relates to the reader.

The task of an editor then is to examine the news for a given day and try to find the "meaning" in it, not only to understand the article, but also to understand the context. What does the reader need to know about? What would the reader like to know about? What is interesting and why? How many stories on a topic are appropriate? If the answer is not "all of them" then which to keep and which to discard?

1.1 Advantages of a human editor

When considering an on-line paper as a primary news source it makes sense to consider what an editor does, what issues they face, what are their strengths, as well as their weaknesses.

1.1.1 "In today’s Journal of the American Medical Association....”

Not all stories of interest come from the same source. In fact, unless someone’s interests are exactly focused on a particular publication, they probably will need to consult several sources of information each day to find what they need. For example, the TV weather report in the morning, the local newspaper on the bus ride in to work, the radio for the outcome of the afternoon Sox game, and an on-line news service for stock information. Each of these sources presents information in a different format and they need to be understood and considered together if stories are being selected from each of them in order to present a uniform window on the news.

1.1.2 "Dear Editor,”

Editors don’t work in isolation. They receive feedback from the community they serve in a number of ways: letters to the editor; comments from their colleagues; focus and market surveys; advertiser's preferences; as well as simple hard numbers such as news stand sales when a particular headline is run. This feedback allows an editor to better serve the needs of their community.

Note that none of this feedback is actively requested from the readership. Rather, these are observations that can be made passively, or as a result of the user choosing to comment. There is something to be said for assuming that if there are no complaints, then something is going right and need not be modified.
1.1.3 On your doorstep

One other aspect of the “real world” editorial process is that the reader does not “wait” while the editing is done. When one reaches for a paper there is no delay. The fact that a paper may represent a production cycle of over 24 hours, thousands of person-hours of preparation, and a variety of news sources is inconsequential. When you want the news, there it is.

This is especially important when contemplating on-line editorial approaches that require significant time. It may seem obvious, but the right time for an editing system to begin to consider the news is not the first time someone asks for a copy of the paper.

1.1.4 The Alta Times?

When we type a query into a search engine, we are making a request that such an engine consider a large number of possible “stories”, select, and present those stories for our consideration. This is nothing more nor less than an editorial process. Many search engines return results that would be considered poorly edited. Sometimes they return nothing, without any explanation of what was too restrictive in the request. Other times they return far too many results, swamping the user with a plethora of information they must then wade through and cull for themselves. Neither of these alternatives is particularly attractive to the user.

Little wonder then that most people would be unwilling to accept an on-line newspaper where the stories were selected by the likes of Alta Vista when they have the opportunity to read one where the selection is done by human editors.

1.1.5 I don’t know what I want!

Defining searches is a difficult task. It is even more difficult when trying to define what the search should be about. One of the reasons a person may subscribe to a paper is that they trust the editors will provide those pieces of information that they need. An on-line paper will need to give a user with some reasonable starting point that provides what they need, even if they themselves don’t know what that is.

1.1.6 I want what he’s got!

One role a newspaper fills is that of providing a sense of community and shared world-view. The conversation started with “Did you see the front page of the paper?” is absent from a world where each newspaper is custom focused on the individual. The common context provided by a shared
information source is important, for without it people lose the common reference point with which to discuss what is happening in the world.

1.2 The Big Picture

This thesis examines an approach by which a computer system can develop meaning in a corpus of news. Beyond developing understanding of the articles, the system will attempt to find context for each article by examining how it fits into the larger picture of the news. It will consider what technologies are needed to keep this context current in the face of a changing external world. It will examine ways in which non-information-retrieval experts can share their understandings with the system and act as a source of common sense for it. The goal is to develop a system that can assist, simplify, and automate the kinds of editorial decisions that a human editor must face and resolve, doing so in a way that is amenable to testing in an on-line news environment. This system will be referred to in this paper as ZWrap.

To this end, two prototype systems (MyZWrap and Panorama) have been developed to explore and evaluate this approach as compared to other existing technologies. It is worthwhile to note at this point that, in many cases, existing technologies were grafted into the architecture. Since this thesis is about architecture and organization, it was deemed less than useful to attempt to reinvent the wheel at every turn. In cases where a particular technology was not readily available, simple “mock ups” were implemented with the understanding that they could be replaced later with more efficient or more accurate versions.

The result is a system that is easy to assemble, easy to maintain, easy to improve on, and performs the task of developing and discovering meaning in a reasonably efficient and interesting manner.

1.2.1 Pre-cognition

One theme that will arise throughout this work (especially in conjunction with the “at your doorstep” issue) is the constraint on any system which interacts with users; that is the need for short response times when a user is waiting. This requirement limits the amount of computation that can be done while servicing a particular request, and thus would seem to limit the complexity of the understanding that can be attempted.

ZWrap addresses this by “thinking” about the articles before the request arrives. These “thoughts” are stored with the article, indexed, and can be used to quickly select and process stories. Since
the computationally intensive work has already been performed (e.g., by caching a parsed version of an article with the article itself) fairly complicated approaches can be employed without forcing the user to wait for a response.

1.3 Thesis Statements

As will be developed in the following chapters, ZWrap seeks to address the development of understanding using a blackboard system of experts—some more traditionally found in the machine understanding community and some from the statistical information retrieval community. Thus, the following statements will be made, and shown in the course of this thesis:

1. The blackboard system approach allows for the easy construction of efficient, distributed machine-understanding systems for developing meaning in large, dynamically evolving corpora such as news.

2. Statistical and machine-understanding techniques can inform each other if they share a common representation that supports such sharing.

Demonstrating the first statement will require addressing the traditional efficiency problems from which blackboard systems suffer by reformulating the problem in the context of news and by incorporating some variations to the typical approach. The second statement will be demonstrated by example of one camp informing the other and vice-versa.

1.4 Organization

- Chapter 1 - Introduction
- Chapter 2 - Blackboard Systems/Overall architecture
- Chapter 3 - Frames/Data representation and management
- Chapter 4 - Dtype/Data Networking, distribution, parallel computation
- Chapter 5 - Searching and sorting in an augmented database
- Chapter 6 - Statistical examinations/Data Mining/Finding new rules
- Chapter 7 - User Interface and Presentation
• Chapter 8 - Implementation notes
• Chapter 9 - Results and Evaluation
• Chapter 10 - Conclusion
Chapter 2

Blackboard Systems

2.1 Introduction

Blackboard systems are an old idea, proposed by Allen Newell in 1962 and used in 1971 on the Hearsay system for speech-recognition systems. Roughly speaking, a blackboard system is one that employs a collection of experts that independently look at a blackboard and judge when they have something worthwhile to add[19].

The blackboard architecture begins addressing each problem by writing it on the blackboard. A group of experts sits in front of this blackboard, examining everything on it. There is one piece of chalk and, when an expert has something to contribute to the understanding of a problem, they take the chalk and write their observation on the board. Experts are generally not allowed to talk to each other; all communication is done via the blackboard.

Each of the experts sitting in front of the blackboard is assumed to have a specialized area of knowledge, but they may make use of observations written on the blackboard by other experts. Thus, the code necessary to integrate an expert into the system is fairly simple. The expert needs to be able to read the blackboard, grab the chalk, and write observations on the blackboard. Since all communication is done via the blackboard, no other inter-expert protocols are necessary.

Since the experts don’t interact with each other except via the blackboard, adding, removing or changing an expert should have minimal impact on the overall system (although, if one agent depends on the work of another, it will obviously become impotent if this work is never done). This allows the development and improvement of experts to be an ongoing process. Additionally, since the experts don’t need to interact except through the blackboard, they can all “think” about the problem simultaneously. This opens the possibility of parallelizing the “cognition” process.
Figure 2-1: On the left, lots of experts, all standing around the same blackboard. They all look at the blackboard and, when one has something to contribute, they grab the chalk (if it is available) and write their thoughts down. All communication is done via the blackboard and only one expert is allowed to write at a time. On the right as you might imagine, this can lead to a lot of frustrated agents!

From a theoretical standpoint, this architecture allows for the development of a "society of agents" as suggested by Minsky[34], with a number of very specialized experts contributing their observations to a problem in hopes of finding some kind of understanding about it. Each of these experts can "evolve" independently, new ones can be added at any time, and ones that are found to be less useful dropped in favor of those more so.

Despite all these benefits, blackboard systems have fallen into disrepute. Perhaps the biggest difficulty with this architecture is its efficiency. It has been observed that in general, most experts wait on the actions of another expert. If agent B needs to look at agent A's observations, until agent A makes those observations, agent B can do except consume processor cycles looking at the blackboard and waiting. As a result, a blackboard, especially one with agents working at different levels of abstraction, tends to become serialized, with agents waiting to operate on a story in a (relatively) fixed order.

The necessity of a single piece of chalk, with atomic locking on the "grab" and "release" process, makes writing to the blackboard somewhat expensive, since atomic operations are often quite time consuming in a multi-process system. Thus, when more than one expert has something to say, a fight over the chalk ensues. When one expert gets it, the other experts are obligated to reconsider the blackboard in light of whatever is written before once again trying to make their annotations. Ultimately, it has been observed that between the serialization due to dependency and the "fighting" due to the atomic nature of writing, these systems perform less efficiently than many alternative
architectures.

Why is efficiency such an issue, given that a blackboard system can help to reduce development time, sometimes dramatically? The majority of blackboard systems to date have been in some way tied to military analysis, evaluation, or planning. Applications such as speech processing[20], sonar contact analysis[36], and fighter-aircraft-intercept evaluation[27] are all cases where there is a single or small number of problems being considered, an element of time pressure (necessitating efficient processing), as well as an environment that does not encourage “in-the-field” modifications. In other words, the typical blackboard system was being used in an environment where its weaknesses where highlighted.

2.2 Blackboards and News

Developing understanding of news is a problem that shares many elements with traditional blackboard problems. A complex problem is under consideration, where it makes sense for many agents to work on several different approaches at once. There are, however, a number of key differences that make news an ideal environment for using blackboards.

In the case of the traditional blackboard, there is a single (or perhaps a small number) of problems being considered simultaneously. In a single day, the MyZWrap system sees on the order of 10,000 stories. Unlike the military cases, where a single problem might be relevant for only a matter of minutes or hours, news stories often retain their relevance for days or weeks.

As a result of both the large scale and longer period of relevance in the news domain, the blackboard architecture can be modified. Instead of several dozen experts standing around a single blackboard, one can instead imagine a room with thousands of blackboards (see Figure 2-2). Several dozen experts wander around the room, making observations on each of the problems “in process”. If a board has an expert standing in front of it, another expert can just pass it by, coming back to it later, when it is free. If each expert has a different color chalk, then those agents that depend on the work of others can just visit boards that already have the appropriate colored marks on them.

In short, most all of the problems with the original blackboard architecture either don’t arise or can be avoided in the case of on-line news systems.

2.3 Implementation

The ZWrap approach uses a blackboard architecture for developing its in-frame understanding of stories. Agents watch a variety of news sources. When a story arrives, a new blackboard is created
and the story is posted on it. Once a blackboard exists, it is then visited by a number of experts. For simplicity of discussion, these experts will be divided into three “crowds”.

The first crowd of experts examines the stories for purely structural elements. They extract the headline, the dateline, the author, the body text, the time the story was posted, etc. and place this information on the blackboard. These experts mitigate the problem of different news sources providing news having different structures. After these agents have visited, all stories will have more or less the same story elements broken out and tagged for later agents (e.g., BODY, HEADLINE, DATE, etc.)

The second crowd of agents performs information extraction, parsing, and evaluation on the uniform story elements extracted above. They perform tasks such as proper-noun identification, word stemming, date and time spotting, parsing to find references to places, spotting country names, etc. All of these facts (e.g., the list of stemmed words, the list of proper nouns) are written back onto the blackboard for the article.

The third crowd uses these derived features to perform high-level augmentation to the stories. For example, the people spotter looks at the list of proper nouns and decides (through a simple heuristic) which are people’s names (see Figure 2-3). The geography expert uses a list of rules
applied to the country feature to decide which regions and continents are being discussed in a story. Also included in this crowd are agents that use even these derived features to produce more features. For example, the Media Lab agent looks at the PEOPLE feature to see if any are Media Lab professors, as well as looking for the name MIT Media Lab in the PROPER_NOUN feature.

Ultimately, this parade of experts takes a story as a monolithic piece of text and transforms it into a richly annotated story, full of identified high-level features related to story content.

### 2.3.1 Dependency

A word about dependency is necessary. In Figure 2-3 the Person expert could do no work until the Proper-Noun expert had visited the article. This could lead to inefficiency if the Person expert kept checking frames to see if they had been touched.

MyZWrap deals with this through the use of a dependency scratch space (see Figure 2-4). When an expert is finished with a set of frames, it notes this in the dependency scratch space. This allows the other experts to see if it is worth checking the frames or if they should wait.

More complicated dependency, scheduling and control approaches are possible, but by allowing each expert to choose what its conditions are for running the task of updating experts is simplified. The expense of this approach is minimal, a small overhead associated with checking the scratch space more often than strictly necessary. The only issue difficult to check with this scheme is cyclic dependency of experts, although the fact that none of the agents in the cycle are running is an indication that this might be an issue.
Additionally, this approach allows experts to run at their own pace, simplifying the scheduling and allowing experts to easily be changed (since the code that does the scheduling is in the same place as the code that needs various other experts to be done). As will be discussed later, this also allows more than one instance of each expert to be running if the computational resources are available.

Lastly, while the experts conceptually visit one frame at a time, the realities of SQL databases makes it much more efficient to request groups of articles and then batch the additions (INSERTs are one of the most costly SQL commands, and as a result many SQL databases implement efficient batch loading functions). The scratch space approach is amenable to this multi-frame batching approach.

### 2.4 Blackboards vs. Rule Based Systems

In some respects, this blackboard approach strongly resembles rule based systems, in that with ZWrap an expert examines the existing features of a “problem” and if certain conditions are met adds new features to the problem definition (the Black Board). Thus, it makes sense to consider if ZWrap will encounter problems similar to those which hamper rule based systems, especially the issues involved with incorrect or partially correct rules, and the results of chaining large numbers of rules together.
ZWrap differs from traditional rule based systems in three important ways; “rule” complexity, bushiness of the “rule” dependency tree and the topology of this tree. The complexity issue is minor, but in general ZWrap experts may perform much larger amounts of computation in deciding whether or not to add a new feature, than one might find in a simple first-order-logic-type rule system. This is a difference more in philosophy than in architecture, since a rule based system could use arbitrarily complicated rules to achieve their result.

The second difference is in the “bushiness” of the “rule” dependency. ZWrap tries to develop superficial understanding of a news corpus that may span many topics (e.g., politics, sports, living) and as a result has a large number of experts that work either on the initial text, or perhaps one or two levels removed.

Rule based systems (often because of smaller scope) tend to have “deeper” nestings of rules, with conclusions extending many rules out from the source. In cases where there is some uncertainty or error in the rules, this results in a quickly decreasing reliability as the chain extends (see [30], especially the section on MYCIN). Because of its tradeoff towards a broader selection of experts, ZWrap has yet to encounter issues inherent in extensive rule chaining.

Lastly, there is the issue of topology (see Figure 2-5). With a rule based system, any of several rules may result in the inclusion of a new feature. As a result, if a new feature is added incorrectly, it is difficult to identify (from examining features) which rule is to blame. This makes the automatic feature selection (discussed in Section 9.2.1) much more difficult. With ZWrap, each feature is added only by one expert. Thus, it is possible to evaluate experts using only the information in the frames themselves (i.e., that information available to the statistical methods).
Taken together, these three points illustrate that while having some similarities to rule based systems, ZWrap also has some important differences, and these differences help it avoid some of the issues common to rule based systems.

2.5 Conclusion

By moving to a multi-blackboard approach many of the efficiency issues of a traditional blackboard system are addressed. The simplified development of blackboard systems combined with the opportunities for parallelism make them ideal candidates for developing meaning in large, evolving text corpora such as news.
Chapter 3

Data Representation

3.1 Introduction

The use of a blackboard system for developing understanding simplifies the question of data representation for the system, as the only well defined representation needed is that of the blackboard. This representation needs to support several actions efficiently: the reading of a particular blackboard's contents; the addition of a note or notes to a given blackboard; and the search of all blackboards for particular notes, or classes of notes.

The requirements of both flexibility and speed of access argues for the use of frames as the data storage medium. Frames date back to Minsky[33] who was formalizing an idea that was already known at the time. A frame is basically a collection of key/value pairs, known as terminals, which describe a particular object or concept. For example, a frame describing the author might contain

Frame #1234
Name: Daniel Gruhl
Height: 6'2"
Degrees: {SB, MEng}
ISA: #0011(Generic Student)

Figure 3-1: A frame describing the author.
a number of terminals listing my characteristics (see Figure 3-1).

This format has a number of advantages, not the least of which is its flexibility. New key/value pairs can be added at whim and clients looking for a particular datum can ignore those terminals that they do not understand or care about. For example, we can add the terminal FAVORITE.COFFEE.DRINK with the value "Café Mocha" without impacting the name-listing agent who examines only the NAME terminal of the frames. For experts, this ability to ignore what they don’t need is important, since it allows the addition of arbitrary experts to the mix without the need to modify every agent in the system.

Another advantage to this format is that not all the terminals need be stored if there is an easy way to generate them. For example, if an expert is looking for the terminal METRIC.HEIGHT and it is missing from a particular frame, then it might use a rule that examines the HEIGHT terminal and does a conversation to generate the value “186.96 cm”. This is especially important in implementations where storage space is not unlimited. In the MyZWrap system, the list of stemmed words is indexed but not stored in the frame. Since it is easy to regenerate, the framestore recreates it from scratch every time it is requested1.

One issue to consider is how complex a value to allow for the value of each key in a terminal. The FramerD system[25] allows perhaps the most complex option, an arbitrary Scheme value, thus providing for everything from lists to procedures to be stored. While granting an enormous amount of flexibility, if all data is stored as Scheme values then all clients are required to have a Scheme parser and interpreter available. This is somewhat cumbersome requirement for ZWrap clients, so a restriction to only string values has been made for the initial implementation2.

Another issue is one of allowing sets of values rather than only a single value for each key. In this case, the advantages outweigh the costs. Set values have no inherit order, which allows for an efficient underlying data representation. Multiple values are appropriate in cases such as:

ACADEMIC_DEGREES: "B.S.", "M.Eng."

that are useful in the ZWrap system, as it is common for multiple COUNTRYs or PERSONs to be mentioned in a story.

One last feature borrowed from FramerD is the use of a universally unique identification number (called an Object IDentification number or OID) to refer to every frame. This 64-bit number allows

---

1 Even with inexpensive storage, access time of frames tends to suffer if the framestore becomes too large. This is an artifact of seeking around very large files and cannot be addressed easily in code, since it often has to do with physical positioning of a head on a disk.

2 Perhaps the best compromise would be to allow for the storing of ASCII representations of Scheme expressions.
the frame to be referred to in a succinct, unambiguous manner. Since OIDs are never recycled, they can be used as a "pointer" to a story and it is assured that the story being pointed to will never change⁴.

In the blackboard context, each blackboard is represented by a frame. Experts examine a frame to see what observations have been made and make their own observations by adding terminals to it. A mechanism exists to allow for locking a blackboard when an expert is looking at it (inherited from FramerD), thus preventing race conditions that might occur as two experts attempt to simultaneously modify a frame based on the other's observations.

When a story enters the system, a frame is allocated and the raw story is entered as the only terminal, under the key TEXT. As experts examine the frame in turn, they add more and more terminals, each containing additional high-level information about the story, on that blackboard.

### 3.2 Searching

One engineering requirement of the framestore (a persistent database of frames) is that there be efficient methods for finding both which blackboards have a particular key on them, and which blackboards have a particular terminal (key/value pair). When a user is waiting for results, sequentially scanning several million frames looking for a particular key/value is not a practical option. Indexing is needed to make large framestores useful.

### 3.3 Implementation

There are several ways one can implement a framestore. The existent one at the inception of ZWrap was the one developed by the FramerD project. FramerD uses a single file to store all of the frames in a given contiguous range of OIDs. The first part of the file is filled with offset and length information for the frames themselves, which are stored as binary "slotmap" dtypes (see Appendix A). When a frame is to be accessed, the server seeks to the specified point in the file and reads the specified number of bytes, returning them over the network connection. When a change is made to the frame, it is rewritten in its entirety at the end of the current data area in the file, and the offset and length information are updated to indicate the new location and size.

This is an excellent storage approach when the data rarely changes. It runs into problems when a large number of agents will be adding keys to an existing frame, as each addition will result in the

---

³It is possible that more augmentations may be added or that in the interest of space some frames may be deleted, but an OID will never point to a different frame than it originally did.
complete rewrite of the frame. Since the first thing added to the frame is the full text of the article, this very quickly results in many copies of the full text appearing in the datafile. This overhead can run to hundreds or thousands of times the size of the frame itself. While disk storage is cheap, it is not that cheap, so another approach was needed for ZWrap.

Another issue with the FramerD approach was that of indexing. FramerD creates an inverted dictionary of each unique key/value pair that appears in the store. It returns this list as an unordered set of oids when queried. Since many searches require some kind of Boolean or set operations on the indexed values, it is more efficient if this set can be returned in sorted order.

It would be possible to store indices in sorted order in FramerD, but the potential need to resort each index every time another oid was added could become expensive. This problem is exacerbated by the fact that, in general, there will be many more inserts to an index than searches on it (e.g. there will be many more stemmed words added to the list than likely will be searched on).

When not using the FramerD approach, ZWrap employs several different data storage methods for its framestores, each built on top of a specialized structure that already handles the storage issues indicated above. Initial work was done using GDBM databases (the GNU implementation of the Berkeley DBM[21]). This worked well until database size grew above around 100 megabytes, at which point internal architecture problems arose from the interaction of GDBM with the file system, causing speed problems. A switch was made to a filesystem-based storage method with more efficient file-access strategies. This proved to be successful, but was somewhat brittle and not terribly portable, as it relied on features of the Linux Ext2 filesystem and did not work as well on other filesystems.

The final move was to an SQL database to store the data. This technique uses a single table of the form shown in Figure 3-2 to store the data. Each terminal entry becomes a record in this database. Set valued attributes are just those where there is more than one row with the same OID/Key entries. The database itself handles the internal indexing for fast retrieval. As a result, frames can be selected with:

SELECT DISTINCT key, val FROM framestore WHERE oid=1234;

Searches for which frames have been “touched” by an agent can be done with:

SELECT DISTINCT oid FROM framestore WHERE key='sample' ORDER BY oid;

and index searches by:

4 Since this was initially written support for sorted indices has been added to FramerD, realizing at times a three order of magnitude speed-up in some searches.
Figure 3-2: SQL table used to store frames. Note, set valued attributes are stored on multiple table rows.

```
SELECT DISTINCT oid FROM framestore WHERE key='foo'
    AND val='bar' ORDER BY OID;
```

Adding terminal entries to the database is a matter of:

```
INSERT INTO framestore VALUES (1234, 'foo', 'bar');
```

All of these basic operations are ones which SQL databases typically perform efficiently. Since most platforms have a "tuned" SQL database available, there is no requirement that the framestore abstraction do so.

It should be noted, however, that an SQL database takes more time to search and retrieve than the "precomputed" responses of a FramerD-style framestore. It makes sense in a production system to have some method of "migrating" frames out of the SQL database and into a more FramerD-like one once they settle (i.e., when it is unlikely they will change much any more).

### 3.4 Conclusion

Frames are an ideal mechanism for data representation in a blackboard system. They allow experts to add keys easily and to only examine the parts of the blackboard that are of interest to them.
There exist efficient ways to store and manipulate frames and, with an index mechanism, they can be efficiently searched and sorted. Their resemblance to XML assures that there will be continuing work on developing efficient storage mechanisms for them, at least for the foreseeable future.
Chapter 4

Network Protocol

4.1 Introduction

Given that the system we are envisioning will use a framestore to represent its blackboards, the next question is how will the agents communicate with the framestore to read the blackboards and place their annotations. There are certain characteristics such a communication system should have – some implied by previous assumptions made, and others that simply enhance the usability:

1. There must be the ability to support “atomic” accesses to the framestore. In some way, experts must be able to “grab the chalk” while they are writing on a board (or perhaps even when doing deep thinking about the state of the board where any change to the board would potentially invalidate their line of reasoning.)

2. Access to the framestore needs to be efficient but, in many cases, the time taken for this access will not be the bottleneck for programs augmenting the articles. Experts tend to be computationally expensive, so extremely fast access (e.g. such as that found in memory on a single machine) is probably unnecessary. When it is, this may be addressed through prefetching and caching (especially since in many cases experts access frames sequentially).

3. Access code should be “light weight”. The primary concern of someone coding an expert should not be how that agent gets its data. Additionally, the computation and memory associated with the frame access should be minimal.

4. The blackboard system gains much of its performance through allowing many experts to access the framestore simultaneously. The extension of multiple blackboards discussed in Chapter 2
almost demands that there be a way for multiple experts to work on different blackboards in
the store at the same time. Atomic access is complicated by this requirement, but multiple
expert access is the basis of the parallel understanding architecture being proposed.

Taken as a group, these characteristics suggest that the central blackboard repository be provided
as a service. If experts connect to a service, then they can allow the service provider to worry about
issues of concurrent access, lightening the load on the expert and those authoring them.

Given the prevalence of networks in today’s computing environments, it makes sense to consider
that such a service might be provided over a network connection. There has been extensive work done
in allowing many clients to connect to a network server (e.g., mail handlers), which has many of the
same issues as a blackboard system (e.g., atomic access and multiple clients)\(^1\). A network approach
also allows many computationally-intensive agents to run on different computers, distributing the
load and allowing a more efficient use of computing resources.

A side effect of connecting over a network is that any program that can obtain a network connec-
tion and speaks the framestore protocol can operate as an expert. This opens the door for experts
to be written in whatever language is appropriate for the understanding they seek to perform and
to be run on whatever hardware best facilitates their execution (e.g. Pentiums, Sparcs, RS6000s,
Lisp Machines).

Lastly, the network approach allows data to be shared between experts and the framestore at
fairly impressive rates, at least as long as they are within the same subnet. Use of ATM networks
which allow up to 80Mbps throughput and the newer gigabit networks see a real throughput on
the order of 300Mbps. These numbers compare well compared even to disk burst transfer rates (33Mbps for EIDE, or 80Mbps for Ultra-Wide SCSI).

All that is needed to employ a networked-framestore server is a protocol by which the clients
can read from, write to, and lock blackboards. Again, an examination of useful characteristics is in
order. Such a protocol should be:

1. Easy to implement. Preparing messages to go over the network and decoding the response
   should be simple for the programmer. The easier it is to implement an expert, the more likely
   it is that many experts will be created.

2. There should be a low overhead in terms of the computation needed to prepare the queries.
   Additionally, the percentage of bandwidth dedicated to the protocol should be low. Ideally

\(^1\)And raises the interesting, but not explored in this thesis, question of experts that send and receive email or use
netnews to communicate with the blackboard.
using this protocol will not slow the expert down.

3. Extendibility of the protocol is important. New commands and structures should be easy to add, and not require the need to rewrite old experts (e.g. when the ability to grab “blocks” of frames is added to allow experts to operate more efficiently).

4. Expressiveness is key for experts that might be putting up nearly any kind of observation on the blackboards. The protocol should support arbitrary nesting, combination, and extensions of the basic data set.

5. While not strictly necessary, the existence of an ASCII representation is extremely useful. This enables quick and easy debugging and documentation of the protocol, as there is a mapping between it and human readable representations.

6. Given the current design strategy, a protocol that works well over a network, given the peculiarities of network packet transport, dropouts, concurrent connections, etc., is important.

It is fortunate that a protocol meeting almost all of these criteria existed at the inception of the ZWrap project in the form of Dtypes. Dtypes was originally developed by Nathan Abramson[1], later extended by Klee Dienes[15], and subsequently formalized and extended by Kenneth Haase and the author. The resulting protocol (found in Appendix A) meets all of the above criteria and serves as the underlying communication system for ZWrap.

4.1.1 The Dtypes Concept

The idea behind Dtypes is to emulate a scheme R4[40] read/eval/print loop. A client connects to a server (usually using a TCP/IP connection) and sends a binary representation of a scheme expression (a Dtype) to the server, which “evals” it and sends back the resulting scheme expression. E.g.,

Client→Server: (+ 1 1)  
Server→Client: 2
Client→Server: (fget 01234/5678 'HEADLINE)
Server→Client: "Dewy Defeats Truman"

It is important to note that while the server is emulating a scheme interpreter, it need not actually implement one. A framestore might instead reply

Client→Server: (+ 1 1)  
Server→Client: [#error "The function + is not implemented on this server."]
since it is not an arithmetic server. The program may have looked at the first element in the list
sent (+) and compared it to a look-up table of known symbols. Not finding it, the server can simply
responds with a stock error Dtype.

That aside, in cases where the server does implement a full scheme interpreter some interesting
interactions can be initiated by the client, e.g.,

Client->Server: (define (sample x) (* (+ 1 x) 2))
Server->Client: #undef
Client->Server: (sample 4)
Server->Client: 10

Thus the client is free to “upgrade” the server with whatever code it wants. This opens a host
of possibilities as an extension language as well as a even more issues related to security, which are
beyond the scope of this document, but have been treated in Mr. Dienes dtcore implementation.

The Dtype protocol meets all of the above requirements, in addition to having the important side
advantage that all undergraduate in computer science at MIT must learn Scheme, vastly shortening
the time it takes to bring undergraduate researchers up to speed on the protocols used.

The Dtype protocol uses a binary representation, instead of the standard ASCII one, for commun-
ication, due to the overhead involved in parsing ASCII scheme expressions. In this representation
each low-level Scheme datatype (e.g., integers, strings, pairs) is represented by a single byte tag
identifying the type followed by an encoded representation of the data, in network byte order where
appropriate.

On top of this low-level protocol, sets of “known” functions are implemented to provide standard
sets of services. A “framestore” is a dtype server that can execute the following “Scheme” commands:

(pool-data <session key>) => (<base oid>, <pool size>, <read only>, <description>)

Returns basic information about the framestore.

(get-load) => <number of frames currently in the pool>

Returns the number of frames in the framestore

(new-oid) => The next oid value.

Allocates a new frame.

(oid-value <oid>) => <frame at that oid>
Simply reads the value of a given frame.

(lock-oid <oid> <session key>) => <the frame at that oid>

Allows a client to "grab" a frame. No other client may write to it until it is unlocked.

(unlock-oid <oid> <session key> <new value>) => <success>

Sets a frame to a new value, releasing it for other experts to use.

(clear-lock <oid> <session key>) => <success>

Allows a lock to be removed without changing the frame.

Any server that implements these functions is a framestore and can thus act as a blackboard repository. What code underlies each of these functions and how the data is actually managed is an implementation issue, left to be performed in the best way possible for the task at hand.

The full specification for the framestore protocol can be found in Appendix B.

4.2 Expert Distribution

Any program that can connect over the network to the server and send the above commands is a potential "expert" in this blackboard system. These can run on any machine. At the moment, MyZWrap uses 4-6 machines to augment and present the news. These machines have been (over the course of this investigation) RS6000s, Suns, Macintoshes, Next Machines, Linux Intel machines, Windows NT machines and Windows 95/8 machines. Since the dtype library is so simple to implement, versions exist in Java, Perl, C, C++, Scheme, and Lisp, allowing experts to be implemented in whatever language that best supports the task they will undertake.

4.3 Efficiency

This structure is sufficient to implement the blackboard system as outlined in Chapter 2, but it suffers from problems of inefficiency and the potential for time-wasting "chalk fights". Fortunately, these are easily addressed.

As explained in Section 2.3, the experts in ZWrap tend to work in crowds, with dependency working (generally) in just one direction (that is, dependency of experts form a directed acyclic graph). Thus, there is no need for later experts in the dependency graph to run until earlier ones have. To facilitate this, ZWrap provides a blackboard where agents may post "I have completed" messages.
stating the ranges of articles they have examined. Another expert can look at this blackboard, examining the messages from the experts it depends on, and choose which frames to consider.

With this mechanism in place, there are two types of interaction supported with the framestore, locking and non-locking (See Appendix Section B.1 and Section B.2). If the dependency tree will provide sufficient protection against concurrency problems, experts may use the more efficient non-locking versions of the protocol. If an agent truly does need to completely hold access to a blackboard (due to the amount of time it is going to invest in “thinking” about a problem, or if multiple agents write to the same key, or if there are circular dependencies in the expert chains), the locking versions may be used.

Another feature to note is the lack of a defined notification system to tell an expert when to run. This is a deliberate choice, as almost all of ZWrap is based on a “pull” architecture rather than a “push” one. The standard Dtype protocol allows a server to respond only to requests, not initiate them.

This is not to say that if needed they could not be implemented. A frame could have a NOTIFY.ME key and the framestore could be modified to initiate a connection back to the expert (which would act like a server in this interaction) to notify it when a frame has changed.

4.4 Other Dtype Services

Framestores are not the only dtype servers around. Closely related is the inverted key/val pair Index Server (See Appendix Sections B.3 and B.4) that keeps track of which frames a particular key/val pair appears in to facilitate fast searching.

More distantly related is the NLP server, which performs parsing and certain types of information extraction on request. By running as a service, this server does not have to reload all of it’s dictionaries every time a request is made, making this a faster approach than having a native NLP in the expert start when needed. Additionally, since Perl access libraries exist to talk to dtype servers, people who don’t know Scheme can still make use of them. Lastly, servers such as the “weather server”, “zip-code server”, “dictionary-definition server”, and others perform a variety useful functions that experts may call upon.

4.5 Conclusion

The use of the Dtype protocol facilitates distributed, concurrent interactions needed to implement the blackboard system proposed in this thesis. This approach allows the development of experts
in an incremental manner without incurring a performance penalty. The ability to run experts on multiple mid-range consumer-grade computers allows the use of many computationally expensive experts without the need to resort to expensive special-purpose super-computing solutions.

From a theoretical standpoint, the distribution of agents over a network allows many very different approaches to understanding to be pursued simultaneously, increasing the chance that the system will be able to develop interesting and relevant observations for every article.
Chapter 5

Searching in Augmented Framestores

The augmentation done by experts on blackboards is used for three different purposes. It is used by search engines for selecting articles, by presentation engines in deciding how to augment their presentations, and by a variety of clustering and data mining techniques that look for patterns and trends in the articles. This chapter will deal with the first of these, as searching for potential articles is the first step of the editing task.

As mentioned in Chapter 1, editing is not easy. In fact, it is difficult enough that editors are a class of professionals and people are willing to pay others to select news stories for them. ZWrap will address the higher level editing task of selecting which stories to present, and will leave the question of in-story editing (e.g., summarization) to future work.

Even with this simplification, there exists the question of what skills and knowledge are needed to select stories well? One important factor is a general understanding of the world. Second is an understanding of the particular topics for which the articles are being selected for. Lastly is an understanding of what the readership needs and is looking for.

For a computer to compete with a human editor, it would need to be able to perform all three of these tasks well. Research has shown that all of them are extremely difficult. Projects that try to assemble "generic" world knowledge (i.e., common sense) tend to be very large and at the same time suffer from spotty coverage (such as the CYC project[28]). Computers fare somewhat better in the field of specialized domain knowledge, but have trouble "keeping up" with quickly changing fields. When it comes to understanding people, user modeling is still a young field.
Rather than approach this task head on, ZWrap attempts to side step it. Most people have an excellent supply of common sense, a fairly detailed knowledge of topics that interest them, and a good understanding of what they want and need, if only in a “thumbs up/thumbs down” sense. ZWrap seeks to provide a user with the tools they need to quickly and easily identify news articles they are interested in. The system can then use rules developed from these interactions to select news that may be of interest. The next chapter will explore how, once these rules are created, they can be kept current, but here the focus will be on how selection is specified.

Using interaction with the user to provide guidance for a search can be viewed as either “training” in the traditional AI sense or, in an application sense, as a way that a person can become more involved in the editorial process. Applying user-provided training to identify of prospective articles is not a new approach. Nearly anyone who uses the web is familiar with an interaction where they provide some explanation of their interest to a program that uses this explanation to select articles – that is a traditional search engine.

ZWrap’s search engines have an advantage over more traditional approaches in that they have a whole host of augmentations and understandings to help guide the search. The rest of this chapter will explore how several traditional search approaches have been adapted and implemented for use in this enhanced environment.

5.1 Boolean Searches

Boolean searches are the “base” case for many search engines. This type of search traditionally examines articles to see if a Boolean expression describing the presence of certain words is true. For example, the Boolean expression

("Bill" OR "Hillary") AND "Clinton"

seeks articles that mention the word Bill or the word Hillary and also mention the word Clinton. MyZWrap implements this type of query using the operators AND, OR and NAND\(^1\) However, ZWrap operates on terminals, not word, when searching. Since the set of stemmed words is written back into the frame, this search (in ZWrap) is actually implemented to form the query:

\[
( (\text{STEMMED\_WORD} \cdot "Bill") \text{ OR } (\text{STEMMED\_WORD} \cdot "Hillary") ) \text{ AND } (\text{STEMMED\_WORD} \cdot "Clinton")
\]

\(^1\)In actual implementation, MyZWrap uses set operations to perform these joins, and the set of NOT some term can be prohibitively too large to realize directly, hence NAND. This could also be addressed with a rule rewriting system that transformed the NOTs.
But \textit{STEMMED-WORD} is not the only feature on an article’s blackboard. As an example, consider what can be done with the output of three experts. Erik Mueller and Warren Sack have written a \textit{ZWrap} expert that spots references to food and notes it in the article’s frame under the \textit{FOOD} terminal. From \textit{Fishwrap}[10], \textit{MyZWrap} has inherited a number of experts, one of which spots morbid news stories (e.g., stories where death, serious injury, or grievous harm occur)\footnote{This was implemented as a “stop” topic for people who didn’t want to see this kind of story.}. Lastly, a geography agent places information on continents implicitly mention in an article into the frame (it does so by examining other geographic features explicitly mentioned). Taken together, these can be used as:

\begin{verbatim}
  ((FOOD . "beer") OR (FOOD . "wine"))
AND (FISHWRAP.TOPIC . "morbid")
AND (CONTINENT . "Europe")
\end{verbatim}

which finds (mostly) stories about drunken driving in Europe. Thus, with a fairly detailed knowledge of what agents are being employed and some clever writing of search expressions, reasonably complex concepts can be expressed using only Booleans.

There are two problems with using this approach in an on-line news system. First, there is a considerable onus on the user to understand the details of the system such as what features experts are spotting and what their values typically are. Secondly there is no concept of how “on topic” a story is. It either matches the Boolean or it doesn’t. This means there is very little in the way of “hints” that can be passed on to the display engine to help decide how the stories should be presented.

\subsection*{5.2 Alta Vista style searches}

The \textit{Alta Vista} search engine employs the concept of “must” (by prepending a “+” to a search term) and “must not” (by prepending a “-”), allowing binary query weighting. Some features are thus infinitely more important than others but, once all of the “musts” have been met, other terms can contribute to the fitness of an article for selection. For example,

\begin{verbatim}
  +Bill Hillary +Clinton -Chelsea
\end{verbatim}

finds stories that have Bill and Clinton in them but do not have Chelsea. From this set, stories that also mention Hillary are considered a better match than those that do not. This allows for a
Table 5.1: Evidential searches use various names to express degrees of certainty. These names can be modified by the expression “supports”, which means the score is added, or “refutes”, which means it is subtracted.

“ranking” of the returned results, although there is no way to indicate relative importance of various terms beyond the +/- ranking.

Of course, as before, the search terms can reference any of the derived features, not just stemmed words.

5.3 Activation/Evidential Searches

Activation-style searches are fully weighted searches. The presence or absence of a feature contributes or detracts from the selection score of an article. For example:

The presence of the PROPER_NOUN "Kosovo" strongly supports selection.
The presence of the STEMMED_WORD "Albanian" does support selection.
The presence of the PROPER_NOUN "NATO" may support selection.
The presence of the PROPER_NOUN "United Nations" may support selection.

For readability, words are used instead of numeric weights, but this is just an arbitrary assignment. A very large weighting (certainly) can be used to allow features to be "stop" features by preventing selection if they occur. The mapping used is show in Table 5.1.

The name “Activation Search” come from imagining a database of all the key/value pairs with connections to all of the articles that mention a particular feature. A search is then performed by activating the stated concepts and selecting those articles that are sufficiently activated in turn through these connections. The result is a list of articles that can be ranked by the level of activation. One should note that there is no need to be restricted to the weights mentioned above. If a more accurate numeric weight is appropriate it can be used.

This particular kind of search is very easy to construct out of a large number of example articles. The system examines the frequency of occurrence of each feature in the training set. If enough
appropriate “high level” features are present, then at least some commonality is sure to exist (e.g., a set of stories may only share the fact that they all mention money). If no commonalities are found, then the system requires those concepts that those stories have in common explained through the authoring of new experts.

5.4 Relevance Feedback

The ease of construction from example suggests that relevance feedback might be a useful approach for designing searches: here, the user performs an initial search to identify articles similar to the ones they are looking for as above. The system then looks for similarities between these articles and uses these as a search criteria. The user examines the results of this new search and identifies articles that seem most relevant. This process continues until the user has found the articles of the class for which they are looking.

Relevance feedback searches need not be explicit. As was explored in the FlyPaper project, if a system can observe a users' interactions and infer which articles are of interest, refinements can be made without specifying explicit rankings.

Traditionally, this type of feedback has used words in the articles or, in some cases, certain derived features (FramerD experiments where performed by Kenneth Haase using lists of proper nouns, names, time words, and places). With ZWrap, every derived feature is a candidate for finding relevance. This allows convergence on a useful search criteria more quickly if the system has augmentations in the appropriate domain. It is more efficient to see that a group of stories have all been tagged with a REGION: South.America, than it is to realize that all of the stories being mentioned all have at least one word from a set of twenty (the list of the names of all of the countries in South America).

5.5 News Channels

ZWrap borrows the concept of channels for news display from Fishwrap, PointCast and MyExcite. All articles in a channel are part of a specific, hopefully well defined, topic. Display issues aside, this means that in general articles are selected for a channel by searching. Thus, any search performed by the user is a candidate for being turned into channel for the newspaper. This allows a user to employ search skills they already have to the task of automating the editing for their newspaper.
5.6 Additional tools for searching

The above are basic search schemes that can be used to identify articles. In addition, a number of tools can be used to help simplify the searching process.

5.6.1 Phrase Searching

It is surprising how difficult it is to implement phrase searching efficiently, yet it is a very important feature of any easy-to-use search engine. Many more articles mention "Johnson" than mention "Johnson & Johnson". Without a way to specify the later, a system runs the risk that many false hits will be generated.

The difficulty with this type of search is that currently ZWrap gathers the entire set of hits for each term before combining them. This means that a simple Boolean occurrence index can only be used to identify candidate articles, which must then be retrieved from the framestore and sequentially scanned for presence or absence of the full phrase.

There are several ways this problem could be addressed. The first is to enhance the index with positional information, allowing proximity to be computed without forcing full retrieval. Another approach would be to memoize the search results, allowing articles to be considered for inclusion only as needed (since typically only the first few dozen or hundred articles are needed for the presentation system that is requesting the search). Another approach would be to cache the result of phrase searches, so that when researched, a phrase engine would only have to consider those articles which have arrived since the last searches.

5.6.2 Description Logics/Named Subqueries

Within a given domain, certain parts of a search may appear frequently. For example, a drug researcher who commonly looks for articles on pain killers may have

(Advil OR Tylenol OR Aspirin OR Bayer OR Morphine OR ....)

appearing in many of their searches. In these cases, it is convenient to be able to name these subqueries. The above could be named

sq:Analgesics

reducing the amount of retyping. This has the additional advantage of encouraging a much more extensive list of pain-killers being developed, since it only need be typed once and the resulting subquery may be shared between people working in the same area.
More complicated subqueries such as “recent”, meaning occurring within the last 24 hours, or “reputable”, meaning not coming from The Onion or The Weekly World News are useful as well. These more complicated subqueries begin to encroach on an area of research known as Description Logics[6]. This subset of first-order predicate logic can be solved easily and maintained efficiently, allowing exploration of classes of objects defined by queries.

These subqueries have the advantage of being dynamic, meaning that they can be edited and the results immediately seen. Experience with the Fishwrap system revealed that designing a tagger and having to wait days or weeks to see results contributed to a slow development cycle for channels or augmentations.

On the downside, however, is the fact that the classification and statistic systems may not know about a particular subquery. In the above example, the system would never know that all the stories were about “analgesics”, since this fact was not written into the frames. Thus, named subqueries are a useful development tool, but a path is needed to migrate them into full-fledged experts.

5.6.3 Experts

Experts are the full-fledged agents, operating autonomously and tagging story frames with additional data. They can employ subqueries from above, additional knowledge bases (either local or accessed through a network), and be as complicated as desired. Once an agent writes something into a frame, this additional knowledge can be used by all other experts in the system, as well as examined statistically.

There is a clear path of evolution ending in such an agent. First, a simple query might be developed. Over time, that query may be refined, and commonalities between queries formalized into subqueries. If a subquery is useful enough, it is a candidate for being turned into an agent.

5.7 Conclusion

By employing higher-level derived features, searching becomes easier and commonalities can be detected with far fewer examples, increasing accuracy and reducing the burden on the user to provide training examples. All of the traditional search techniques have immediate application in an enhanced domain and many of them perform even better with richer information bearing features to work with.

---

3 The problem is even worse in commercial environments, where adding additional topics can take 8 months, and thus is almost never done.
Complicated query building can be simplified by techniques such as named subqueries, and query by example. Users can define their own concepts for use in querying and these concepts can eventually become experts, at which point they can be automatically brought to the attention of other users, who might benefit from them.

Lastly, a query need not be a "one-shot" event, for once a way of finding a particular type of information is developed, it can be turned into a standing request for information as a channel in an on-line newspaper.
Chapter 6

Statistics

6.1 Introduction

Up to this point we have considered how ZWrap draws on a society of experts to augment a single article. Additionally, we have seen how traditional searching techniques perform better when applied to these augmented frames. In short, we have an approach that can develop considerable understanding about each article it sees and take advantage of that understanding to better inform the user.

This is only half the story though. In Chapter 1 we set out to find meaning and defined meaning as not only understanding, but also context, or how the articles interrelate. While work has been done by Chakrabarti et al. [8] on the use of explicit cross-referencing information, ZWrap will explore a document space where such cross references are absent.

Without these explicit hints, ZWrap develops understanding of how the frames relate to each other through a variety of statistical examinations of the corpus, looking for patterns and trends.

Figure 6-1: On the left, traditionally statistical analysis on text is done by first creating a word-document matrix on which to perform analysis. ZWrap uses feature-document matrices, such as the one on the right.
that develop between high-level features.

Statistical and pattern-recognition techniques have a long history of use in the context of information retrieval. In order to apply them, some mapping is needed between the articles and a vector of features that represent the article. The most common such mapping is one that takes the list of words that appear in the article and maps them to individual elements in the vector. These vectors can then be collected into a single matrix which represents the corpus, known as a "word-document matrix" (see Figure 6-1).

What makes this approach difficult is that each word carries (usually) only a small amount of information. English being what it is, there are often several words that have more or less the same meaning and can be used interchangeably (e.g., doctor and physician). The result is a set of very long vectors and the need for a large number of example articles. This in turn leads to slow computation.

Many attempts have been made to reduce the size of this matrix, from the use of stop-word lists, heuristic sub-sampling of the word list, stemming, and thesauri, to dimensionality reduction, all with some success. ZWrap takes a different approach. By the time an article arrives at the point where it is to be examined statistically, it should have a host of additional features that have been spotted, computed, or otherwise added to the frame. These features are presumably more important than the words that appear in the frame. It should come as no surprise then that techniques that work well on word sets work even better on augmented frames, since the frames contain features that carry more information (See Section 9.2 for specific examples).

As always, the question that needs to be addressed here is the reliability of the experts which place the additional augmentations and the negative impact of incorrectly augmented articles on the performance of the statistical techniques employed (See Section 9.2.1 for a discussion of how this negative impact can be avoided).

6.2 Why Use Statistical Techniques?

Since ZWrap already has a good searching and selection mechanism, why consider using statistical methods at all? Well, one reason is that developing good searches by hand is expensive. ZWrap seeks to capture this work by allowing searches to be turned into news channels, where they can be used for an extended period of time.

This effort is confounded by the tendency for topics to "drift" over time. Few would have guessed at the beginning of the Whitewater investigation that the feature PERSON: Monica Lewinsky
would become an important feature of that channel. Simple statistical examination of articles in the Whitewater channel in January 1998 would have revealed the presence of a new feature that appeared consistently enough to warrant being drawn to the channel maintainers attention for possible inclusion in the search criteria for that channel.

Taken a step further, statistics can be used to recognize when “something new” happens. If an article arrives in a channel whose features deviate appreciatively from what is normal for that channel, it may be worth bringing that story to a reader’s attention. There always seems to be a news story where the amount of ongoing reporting exceeds a readers interest. If nothing new has happened, it would be better if the story were not mentioned at all.

A third application (and perhaps the most powerful) is the ability to use statistics as a guide to when other approaches should be tried to develop understanding. The first application mentioned (detection of new feature occurrence) might be a good indicator that further analysis is appropriate (e.g., action/action, bias, etc.) This is especially useful when analysis takes significant time. If 10,000 articles arrive in the system every day, then an agent needs to be able to process stories at a rate of at least 7 stories per minute or it will fall behind. If 90% of stories can be removed from consideration by efficient statistical methods, then this allows an understanding agent to take up to 85 seconds per story, and still not lag behind. It has been observed that most discoveries don’t begin with “Eureka”, but rather “that’s odd...”. Statistics can provide more traditional machine-understanding techniques with the realization that something is odd, drawing attention to it for further consideration.

### 6.3 Techniques

Statistical examination being useful, what techniques are available for application to ZWrap? Some existing techniques work well in the on-line news context being developed, while others are inappropriate in the ZWrap domain. Before delving into specific examples, we should first define what is meant by “Statistical Technique”.

For the purpose of this document, a statistical technique is one where each article is represented as a vector. We will call \( a_i \) the feature vector for article \( i \), and \( a_i(n) \) the \( n \)th entry of that article’s vector. The mapping of features to entry is arbitrary but fixed for the corpus. For reasons of efficiency, features with insufficient support may be dropped from this mapping (if a feature occurs only once, then it is not of much help in classification) and likewise, overly common features may also be dropped (a feature is equally useless if it occurs all the time). For simplicity, ZWrap uses
Boolean features. This means that \( a_i \) is filled with only ones and zeros, representing a feature being present or not in a given article.

What makes certain techniques appropriate and others not? As outlined in Chapter 1, ZWrap seeks to share its understanding with the user at all times. This means that techniques whose end result is an arbitrary array of probabilities is probably not appropriate, since a user can't glance at the array and tell the computer where it is confused. As a result, in several places, ZWrap uses less than mathematically optimal approaches in the interest of feedback from the user.

It is worth noting that "mathematically optimal" is an interesting expression in this domain. It means doing the most with the information the system has. But a technique that works in such a way as to elicit additional user input may out-perform one that tries to make do with the information it starts with, since the user input represents more information entering the system. The idea of using the user to teach and train a system is a very old one (see McCarthy[32]) but one often overlooked in the more mathematically rigorous information retrieval techniques. It it import to remember that with an on-line news system, the concept of optimal needs to include the fact that the user is a resource that may be periodically employed and in general will have a better base of "common-sense" than the system does.

### 6.3.1 Simple Apriori Occurrence Expectation

While perhaps the simplest approach to statistical examination, simple apriori occurrence expectation is also the most powerful and appropriate in the ZWrap context. This approach looks at a domain (e.g., within a channel, within the whole corpus, etc.) and develops apriori statistics on feature occurrence. For example, let \( \bar{A} \) be the normalized, average article in a section. \( \| A - \bar{A} \|_2 \) or \( \cos(A \cdot \bar{A}) \) is then a measure of how "odd" a particular article is when compared to what is typical for the section.

The set of \( \bar{A}s \) for the different sections in a paper represent the typical or expected articles for that section and a new article can be compared to all of these "stereotypical" articles for deciding which section to place them in. This provides a reasonable approach for article selection by example.

### 6.3.2 Related Articles

We would expect that articles covering the same topic would have similar features. Thus, a simple distance metric such as cosine angle between the normalized feature vectors would give some sense of how related articles are. This nearest-neighbor analysis allows automatic identification of related articles. It also can be used to find near duplicate articles, for related articles are close, but not too
close. This is especially true for news streams that tend to repeat stories with small changes from hour to hour. In these cases, just presenting the most recent story is probably sufficient.

6.3.3 Association rule data mining

The next step up from simple occurrence is co-occurrence, looking at what features occur frequently together in the same article. Association-rule mining seeks to find the “associations” between groups of features, for example that \( A \land B \to C \), where \( A, B, \) and \( C \) are particular features in the corpus. It is easy to imagine this process becoming somewhat time consuming, as at first blush it would seem there is the need to consider the full superset of features. Two requirements, however, allow this problem to be reduced to tractability.

In the rule \( X \to Y \), where \( X \) and \( Y \) may be sets of features, we define two values. The first is the number of times the set \( X \) appears in the corpus. This is called the “support”. If \( X \) doesn’t appear frequently enough, then it is not very useful, both because the rule will not be triggered often and because there will not be enough examples to develop a statistically meaningful statement about co-occurrences with \( X \). The second is the accuracy of the statement \( X \to Y \). The observed fraction of the time this is true is called the confidence. By setting minimum levels of confidence and support, an association rule system can quickly reduce its problem space to a solvable one by pruning the subset tree when the requisite support gives out[2].

One metric that doesn’t reduce computation, but does reduce how many rules are presented to the user, is “interestingness”. Given \( X \to Y \) again, it is possible to compute what the expected confidence would be if \( X \) and \( Y \) were statistically independent. Just because ‘a’→‘the’ 100% of the time in our corpus doesn’t mean it’s an interesting rule. Interesting rules are ones where the confidence differs significantly from what would be expected if the values were indeed independent.

Generating rules in the form \( X \to Y \) also is a good match to rules in rule-based systems. This helps make for simple transitions from discovered rules to experts who can employ those rules, for example, adding the rule for searching 'Lewinsky'→'Clinton'. When this isn’t appropriate, other approaches can be used, such as bringing the rules to the attention of the users and allowing them to best decide how the information should be employed.

Developing rule-based systems from statistical observations isn’t a new approach. Drescher[17] developed an entire learning construct that performed more or less random actions and then learned which ones led to predictable results. Codifying these rules automatically gave a rule-based system the rules it needed to solve problems. This approach could be used with other rules to create a more complex article-selection agents, but this is beyond the scope of this thesis.
6.4 Clustering

One area where statistical methods do quite well is that of clustering (see Therrien[42]). There are many different clustering algorithms available, ranging from simple K-Means to the more complicated Simulated Annealing and Aglutative methods. With all of these methods, the goal is to take a large number of items and sub-divide them into groups. Often, the number of groups is fixed initially or modified by a heuristic during the application of the algorithm.

Considerable work was done on the ZWrap system in the spring of 1998[24] developing ways in which an augmented frameset can be used to “explain” why K-means developed clusters the way it did. The K-means was run on a set of LSI-dimensionality-reduced vectors generated from the stemmed word-document vectors through singular-value decomposition. Initially, although very fast, the system could give the user no indication of why a cluster had been created.

This deficiency was addressed by looking at what high-level features were in common among the articles within the cluster and revealing them to the user, rephrasing the numerical confidences into more “conversational” terms (e.g., most of the articles mention Madeline Albright and many of them take place in the Middle East.)

6.4.1 Presentation

Clustering is perhaps most useful in deciding what articles to present to a user. If several dozen articles are candidates for presentation within a particular topic, one approach is to cluster them and select then “representative” articles from each cluster for presentation. The user can then ask the system to expand on an article to indicate that they are interested in a particular cluster.

These better-formatted presentations can be of use not only during article display, but also during relevance feedback. If three articles “span the space” of the query results, then having the user select from among them is easier, and likely just as useful, as having them pick the most appropriate article from a list of hundreds.

6.4.2 Cluster Management

Most clustering algorithms operate on a fixed number of clusters. This is a difficult number to fix, especially if there is no apriori reason to suspect how many cluster there are in a set. As a result, there are several heuristics that can be applied to determine when a cluster should be split (e.g. when there appears to be two or more strong sub-cluster within it) or when two clusters need to be joined (there is not much difference between them).
While useful for the clustering step itself, these heuristics can also be used to examine when channels might need to be split into two or more or when two channels might warrant being joined, simply by examine the articles selected into each as features. This is just one way in which statistics are used at a “meta” level in ZWrap. Statistical techniques such as automated collaborative filtering can examine which articles a user likes and recommend others that similar users liked, but it can also look one level higher and examine what channels a user subscribes to and use this information to recommend other channels.

6.4.3 Dimensionality Reduction

Dimensionality reduction is an area where the design goal of being able to share understanding with the user is undermined. Dimensionality reductions (such as those achieved through LSI, PCA, and SVD methods\(^1\)) seek to map a given feature space to a space of much smaller dimensionality through projection, where the “important information” is preserved. In this smaller space, the hope is that operations such as finding nearest neighbor or clustering can be performed much more efficiently.

The difficulty is that dimensionality reductions are somewhat arbitrary mappings into subspaces. Each dimension in these subspaces is a collection of bits and pieces of other features, assembled in such a way as to conserve as much of a mathematical quantity as possible (often the variance). The result is that it is very difficult to present these results in any reasonable way.

In ZWrap, dimensionality reduction needs to be used carefully and never as the main feature in an expert. As noted above, as a preprocessing step to clustering dimensionality reduction can work well, as long as care is taken that the final step in an algorithm results in a human-understandable result.

A form of dimensionality reduction that works in a way more in line with the ZWrap philosophy would be one that performed feature selection, taking or rejecting features as a whole rather than creating mathematical constructs that combine arbitrary fractions of them.

6.5 Conclusion

The machine-understanding techniques based on the blackboard architecture developed in the first half of this thesis concern themselves with the understanding and augmentation can be done within frames. This chapter looked at how the system can compare frames to develop the context needed to

\(^1\)See either Golub and Van Loan [22] or Devijver and Kittler[14] for a discussion of the maths behind these approaches.
keep the system evolving. Statistics provide a tool by which the system can discover rules about the apparent interrelationships in the stories it sees and can suggest modifications to existing knowledge bases when it appears reality has changed. It can bring novel and diverse information to the user's attention in an appropriate manner, being better able to choose what to present. Lastly, the system can more efficiently make use of its computational resources by allocating them only when it has reason to believe there will be a sufficient return on investment.
Chapter 7

User Interface

This chapter is shorter than many others, as the user interface was not a focus of the research in this thesis. Rather, the user interface was developed to allow diagnostics and debugging of the underlying system, as well as to develop insight into how the ZWrap architecture impacts the development of an on-line news system.

While it was found that rich presentation of articles was fairly easy to achieve with the augmented-frame model used in ZWrap, to best take advantage of the additional information developed requires a model of the user to whom news is being presented. MyZWrap currently implements only the most cursory of user models, although it is designed to enable the inclusion of a Doppelgänger[37] style user model with very little modification.

With these reservations in mind, however, it is worth examining some of the design principles, decisions, and observations that went into developing the MyZWrap user interface.

7.1 Share Everything

One of ZWrap’s design principles is to represent all information internally in a form that is human understandable. Having gone to this trouble, it only makes sense to then share as much of this information as possible with the user.

ZWrap shares its understanding with the user for two reasons. First is to make articles more useful by helping the user fill in gaps in their understanding. The second is to allow the user to notice when the system is “confused” and take steps to address it. Part of the user interface is providing back channels for the user to give these corrections to the system.

In short, the philosophy guiding the design of the user interface is that understanding is a collab-
oration between the system and the user, and like all collaborations, the better the communication the better they work.

There are various ways the system seeks to do this sharing. The first (and simplest) is to always allow the user to access the full internal representation of any article they are viewing (see Figure 7-1). While ZWrap is able to show everything it knows about an article, this is not optimal. If 15 cities in Eastern Europe are mentioned in an article, a map might be a better way to display this information than a list of city or country names. If the occasional odd word appears in an article, perhaps a dictionary definition would be useful; with people, a short biographic sketch. This kind of augmentation requires a knowledge base and specialized agents.

PLUM[18] is an example of how effective the presentation of augmentation information can be to the understanding of an article, provided the system understands both the article and the user.

7.2 Marginal Notes

From this most general philosophy, we move to a specific implementation method. One complaint heard all too often with on-line information systems is that they tend to swamp the user with more information than can possibly be used.

Part of this problem can be dealt with by changing how supplemental information is presented. Rather than in-text links, MyZWrap opts for a “marginal note” approach to displaying additional information (see Figure 7-2). By placing information to the sides of the article, rather than directly in the “line of sight”, the reader can ignore the augmentations until they are needed, at which time a quick glance will allow the user to find what they need.

The next question is what types of information to include in the marginal notes. The answer depends on the user. As will be discussed in Section 8.9.4, one possible augmentation to articles is the inclusion of the Grateful Dead lyric most appropriate for the article. In demonstrations, some users found this to be an extremely desirable feature for MyZWrap (there are deadheads everywhere...), some users found this to be inappropriate for a journalistic environment, and most ignored it. Without an effective user model, the system can only guess at what to provide, using a generic model of what augmentations are interesting.

7.3 What was it thinking?

One of the most frustrating characteristic of many information retrieval services is how hard it is to figure out why a particular document was selected for presentation. This is not just a trivial
Frame #1279986

LA GRANDE, Ore. (AP) - An invasion of tree-chomping moths threaten to devour thoua

Figure 7-1: A portion of the raw frame for a story, with the keys in bold and the values on the lines that follow.
UN cracks down on lawlessness; Rugova to work with rivals

Pristina, Yugoslavia—Plagued by peacekeepers and sporadic violence, the U.N. mission to Kosovo announced new regulations Friday allowing it to remove anyone at any time if such a move is deemed in the interest of maintaining order.

An Albanian leader from Kosovo, Ibrahim Rugova, pledged to cooperate with international bodies as well as rebel leaders as he tried to end the violence between Kosovo's Albanians and the dwindling Serb population.

Marginal Notes

Selection Criteria

- UN cracks down on lawlessness; Rugova to work with rivals
- Pristina, Yugoslavia—Plagued by peacekeepers and sporadic violence, the U.N. mission to Kosovo announced new regulations Friday allowing it to remove anyone at any time if such a move is deemed in the interest of maintaining order.
- An Albanian leader from Kosovo, Ibrahim Rugova, pledged to cooperate with international bodies as well as rebel leaders as he tried to end the violence between Kosovo's Albanians and the dwindling Serb population.

Total Score: 2.23

Figure 7-2: When an article is displayed, additional information is included as marginal notes (dotted boxes added here for illustration only).
annoyance. Without understanding why a search engine produced an unwanted result, it is very
difficult to modify the query to fix the problem. MyZWrap's solution is to provide an explanation
of how any particular article was selected for presentation and to include this explanation as one of
the marginal notes (see Figure 7-2).

In developing MyZWrap, when stories about the NBA began to appear in the Middle East
section of the paper, it was immediately apparent what the problem was (Jordan the basketball
player vs. Jordan the country) and steps were taken to program the country expert to understand
the difference.

7.4 Truth in Advertising

Perhaps the most important commodity a newspaper has is its reputation. Part of that reputation
comes from both being clear about how credible each piece of information is and presenting more
than one viewpoint on a topic. Translating this into an on-line paper (which may be drawing
together articles from many different sources) is difficult but vital, if an on-line publication ever
hopes to become a "paper of record".

The first thing an on-line publication can do is to clearly indicate the source of each article, as
in from what wire service, web-site, etc., the article was drawn. This moves the credibility issue to
the source of the article, as the reader can choose to lend a different amount of credence to articles
from The New York Times than they perhaps would to ones from The Weekly World News.

Next, the context of an article can be important to understanding why it was written. For this
reason, MyZWrap always provides a link back to the original article. Also, within ZWrap it is import
that an article be displayed in a format appropriate to the source. If the presentation of a wire story
makes it appear to be from netnews, or vice-versa, the system risks misleading the reader.

Another step that an on-line system can take is to provide pointers to related articles, perhaps
from different sources. This, combined with ongoing work in bias detection, will hopefully allow on-
line papers to provide their readers with a variety of different "slants", letting readers form opinions
on topics for themselves.

7.5 Clipping Basket

To facilitate collection of articles for channel definition, sharing with others, and archival purposes,
ZWrap implements a clipping-basket facility. In the MyZWrap system itself, this is implemented via
Sunil Vemuri's Mind Meld project utilizing the WBI[3] proxy architecture. In Panorama (the IBM ZWrap implementation), this implementation is done internal to the ZWrap engine.

While straightforward to implement, an important consideration in any clipping basket is that it be as easy to "clip" an article as it is to bookmark a web-page, otherwise the tendency will be to not use this resource.

7.6 Channel Structure

Every reader of a paper has a variety of interest and stories are easier to skim if similar stories are grouped together. Traditional publications use sections such as "Sports" or "Living" to group their articles. An on-line paper can be more flexible. MyZWrap allows users to define their own sections (called channels after the Pointcast approach). Thus, users can have sections of their paper devoted to the Middle East Peace Process, the lingering impact of hurricane Mitch, or the color purple, as they see fit. ZWrap uses its channel structure to present news generated by searches as well as other methods (e.g., consulting a weather server or retrieving comics from another web site).

The arrangement of these channels on the main page is facilitated at the moment by a simple web tool allowing the user to move channels around. However, features like Fishwrap channel management, where channels that are most important to the user at the moment are floated to the top, would clearly be appropriate.

7.6.1 Tabs

One addition to a "single page" newspaper approach is the inclusion of "tabs" to help a user organize their paper. This feature, implemented in Panorama, has two helpful effects, one for the reader and one for the system. The first is to reduce the number of channels on the page, making it easier for a user to find what they are looking for. The second is to reduce the number of channels on a page so the system does not need to search every topic, but rather it can concentrate on the channels in the page being examined.

7.7 Alternative Delivery Mechanisms

A web page is not the only delivery mechanism for news. In addition to the traditional printed page, there are also pagers, email, phone calls, audio alerts, led signs, etc. MyZWrap has explored the first two of these (email and pagers) and interoperability with the Canard project[11] will hopefully lead
to easy inclusion of the others.

The important consideration with all these methods is that they are all inherently more intrusive than a web page. The “push” approach to news can be a useful one, but care must be taken that it is used only when appropriate. Receiving 50-100 pages a day quickly becomes more of a burden than a useful service.

7.8 Conclusion

This chapter touched on some of the user-interface issues considered in putting together the existing article presentation system for ZWrap. How to best to act on the augmentations to an article remains a rich area of research.

The most import observation that can be made is that the more a system knows about the news and about the user, the better potential it has to present the users with the information they need. ZWrap addresses the news understanding half of this equation, but it must await an equally rich user-understanding system before its user interface develops further.
Chapter 8

Implementation

8.1 Introduction

The MyZWrap system is implemented in approximately 50,000 lines of code in a wide variety of languages on a wide variety of platforms. Rather than exploring each piece of the system in detail, this chapter will seek to give a feeling for the issues and observations made of each component of this project.

8.2 Projects

The ideas set out in the previous chapters have been explored in two implementations as well as numerous small projects. The first implementation, MyZWrap, is a general purpose on-line news system developed at the MIT Media Lab. It obtains most of its news from the wire services (Associated Press World Stream, Associated Press State 50, Reuters, New York Times), although it does get some news from the web (The Onion, as well as various sources of weather, comics, and sports scores). MyZWrap is designed to serve as a primary news source for individuals, providing news on a variety of general topics (similar in scope to a site like CNN or USA Today).

Panorama, the second system, was designed and implemented at the IBM Almaden Research Center and is a more focused on-line news system, designed to serve the needs of an electronics design engineer. Rather than employing wire services, Panorama obtains most of its news from the web (e.g., CNN, USA Today, various company press pages), net news, and discussions internal to the site itself. Since it is a more focused application, it performs more understanding in the domain of the electronics industry, at the expense of a somewhat narrower understanding of the world at large.
Various projects which have explored pieces of this thesis include work done on The Daily Catch project, K-Lines to K-Means[24], and Dimensionality Reduction for Information Retrieval[5].

8.3 General System Flow

In all of the above projects the same basic system was implemented (see Figure 8-1). The general flow of information is as follows. News enters the system through the news streams, having been acquired from a variety of sources. These news streams reformat the news into frames and store them in a framestore. The experts examine the frames in the framestore and augment them when appropriate. Data mining (statistical examination) occurs in the background and trends that are noticed can be used in a number of ways, including augmenting the knowledge bases of the experts. Searches are performed on the framestore directly, as well as on indexes that have been developed. All of these features are exploited by a user interface to provide an augmented news presentation.

We will now examine each of these elements in detail, paying attention to the technical issues encountered, and how each piece fits into the structure developed in the previous chapters.
8.4 News Streams

News streams serve as the “interface” between the ZWrap architecture and the news sources it employs. In this respect they are the major “senses” of a ZWrap system, providing it with much of the information it uses to understand the world.

The news streams acquire the news from a variety of sources, but by the time they write the news into the framestore, it has been converted into a form that the experts can examine without consideration of the source. The consequence of this is that each news stream must be tailored to a particular source. While work has been done elsewhere in developing automated tools for extracting information from semi-structured text[16], ZWrap has yet to employ these approaches. As a result, some of the most specialized code in the system lies in the various news-stream agents.

As noted earlier, news streams acquire their news from a number of sources, including the wire services, the web, net news, internal discussions, email, and personal calendars. We will now examine each of these in turn.

8.4.1 Wire Services

Wire services account for the majority of news in the MyZWrap system. They are generally provided from satellite dishes or short-wave radios from the Associated Press, Reuters and The New York Times. Each of these sources comes with a “decoder” box that converts the news stream into a form that can be delivered over a serial line.

The code for decoding these serial streams into articles was developed in Fishwrap[10] and used with slight changes in the MyZWrap system. This code involves a “betty server” which clients can connect to over the net to obtain the streaming data.

For each news stream, a Perl script is used to break the stream into articles, keeping as many of the American Newspaper Publisher Association format tags as possible. While technically not part of the news stream, the “Structure Expert” does the last level of cleanup on these articles, extracting headline, author information, etc. from the TEXT field and finishing the normalization of the frame.

8.4.2 Web news sources

Modern programming languages make retrieving text from web pages quite easy[38]. This coupled with the increasing amount of news available on the web, make it an attractive source of articles for an on-line news system.
Technically this is straightforward to implement. The news stream employs a web library to simulate a user visiting the on-line news site and downloads the articles. Various custom filters then work to remove everything from the HTML except the raw text, which is entered into a database.

In terms of content, the web can be used for both general purpose (e.g. CNN, USA Today) news, as well as more specialized information (company web pages, sites of special interest, etc.). Panorama relies heavily on the web for much of its news.

8.4.3 Net News

While occasionally not up to the same level of journalistic excellence found in traditional sources, net news does provide opinions that might not otherwise be expressed on various products and vendors.

To this end, Panorama extracts net news from a number of different electronic-design-centered newsgroups and writes them into the frame database, preserving as many of the RFC822-style header features as it can.

8.4.4 Internal Discussion

Often the most appropriate commentaries on an article are made by one's peers. Panorama supports internal, threaded discussions of particular articles by the readership. These discussions are stored as articles, with the SOURCE being “discussion” and an additional PARENT terminal providing the linkage information needed for threading.

8.4.5 Email/Calendar

While not currently in the main MyZWrap system, Fishwrap had a “calendar following” system that would find news appropriate to your calendar entries. In MyZWrap, Ben (Max) Davis implemented an email-sniffing agent that allowed the system to provide articles appropriate to a users email for the day.

The utilization of email and calendar information brings up a point on privacy. Since the ZWrap system is distributed, additional framestores can be provided that are only accessible to local agents. By moving more of the search, selection and construction to agents running on a local machine, privacy can be increased to the point where there is no discomfort using these highly personal sources of information for article selection.
8.5 Framestores

The task of a framestore is simple; provide a mechanism by which frames may be stored, searched, and modified on request (see Chapter 3 for a discussion of all these requirements). This task is somewhat complicated by the fact that an article goes through many changes during its "lifetime". Initially, an article has a small number of large key/value pairs. These are initially broken down by structure experts into a moderate number of moderately-sized pairs. Next, agents add large numbers of very small pairs to express the information developed about an article. Finally, the article is "finished" and no further additions are done to it.

This makes the task of developing the ideal framestore difficult, as the different ways the frames are modified suggest different storage techniques for each stage in a frame's life. As a result, several different approaches have been developed over the course of the ZWrap project.

8.5.1 Flat file

The flat-file approach places all the data in a single file (see Figure 8-2). Generally, the first part of the file consists of an index into the data, consisting of a collection of offset/length values for each frame. These point to a later position in the file where each frame is stored contiguously and often with no "padding" between them. Articles are typically stored in the network-binary format (as a DType). This allows network requests to be serviced very quickly, since all that is required is to look up the frames location, seek to it, and send it over the connection. No additional processing or formatting is necessary.

While the flat-file approach results in exceptionally fast delivery of frames, it is not without its problems. One is that in order to modify a frame, it must be read into memory, modified, and written out, usually to a different place in the file. This results in a recopy every time the frame is changed and necessitates garbage collection to recycle the space used by old frames.

Additionally, since there is no indexing inherent to this approach, a separate index server be built and maintained. Keeping this index up to date is expensive, since it is advantageous to store the index in sorted order.

The last and perhaps most insidious problem with this approach is that of file size. Because a flat file is sitting on a filesystem on a physical disk, it begins to experience issues with access time when the file exceeds a certain size. For the Linux Ext2 filesystem, this was observed to be around 1 gigabyte. The need to seek around this large file more or less at random means that access time can quickly drop by a factor of as much as 10 as the framestore grows from one to two gigabytes. Even
experiments writing to disk without a filesystem encounter difficulties associated with the physical head seeking around the platter.

Still, a flat file with an index is the best way to store a static corpus and is the approach used by FramerD without difficulties when dealing with the typical fixed “standard corpora” of machine understanding.

### 8.5.2 DBM Style

The trickiest part of writing the flat file is the garbage collection. This is especially true of the index portion of the framestore. An obvious way to address this is to employ a database structure that deals with the issue of garbage collection.

DBM-style databases take a “key” and “value” and associate them in such a way that when later presented with the key, the database can return the value. This style of database tends to implement fast disk-based storage, garbage collection, and hashing.

However, they encounter difficulties similar to those of the flat-file approach. Because there is no way to predict access patterns, file size again becomes an issue. Since the garbage collection and file-space reallocation is automatic, large, constantly changing frames guarantee a fair amount of seeking and recopying.

However, an even larger problem exists. Searching the database using a sequential scan locks
the database for all write access for the duration of the scan. This is not acceptable for a real-time database, where news is constantly arriving.

Still, for small, auxiliary databases (e.g., user data, channel definitions, etc.) these databases provide an attractive, low-maintenance alternative to some of the more complicated framestore structures.

8.5.3 SQL

The last class of storage developed for ZWrap is the use of an SQL table to hold the frames. In this approach, a three-column table is used to store the data. Each key/value pair gets its own line in the database.

This approach makes adding (deleting or modifying) key/value pairs quite easy, since all these operations map to a simple SQL statement (see Section 3.3). An additional advantage to this approach is that Boolean queries on the database can be performed very quickly, using only the internal indices of the database. The difficulty is that retrieving a frame requires a SELECT, which can take some time.

The SQL database used for MyZWrap is Postgresql[35], chosen because it has excellent support for moderate-size text objects and it can handle reasonably complicated query structures (an additional feature is that is available for free). For Panorama, DB2[9] was the database of choice. While not having as many features as Postgresql, DB2 does provide for much larger tables (in terms of entry count and size) and much faster access to the data in them. Unfortunately DB2's inherent text support is somewhat weak, and many of the more complicated text searching features need to be implemented at the “API” level rather than in the database itself.

One issue with SQL is the need for all data to be typed. As a result, the value needs to be stored in a column with the most generic type, that being text. This means that parametric searches (e.g. list all frames where height is greater than six feet and age greater than 25) require a sequential scan.

Julius Quiaot in his AUP[39] investigated a hybrid approach to this problem, using a DBM-style database to hold some of the data and a fixed schema SQL table to hold other terminals of the frame. This approach allowed searching and retrieval to be performed efficiently, at the expense of some additional complexity in the framestore. A similar approach is being contemplated by many XML-type databases and it is likely that as these develop they will provide ideal platforms on which to deploy framestores.
8.5.4 Inheritance and Reference

Frames as presented are a powerful structure for data representation, but they stop short of providing all the features of more traditional frames[34], specifically those of reference and inheritance. ZWrap addresses these using the concept of the Object ID number. If an application sees a value of a terminal that is an OID, it can do one of two things with it.

In most cases this is interpreted as a reference. For example, if the value of a PERSON terminal is an OID, then it is understood that it points to a frame containing information on that person. This allows for higher-level concepts such as “William Jefferson Clinton” where articles can point directly to the knowledge base and avoid the problem of having to know that “Bill Clinton”, “President Clinton”, etc. are all the same person.

The second interpretation is if the key is ISA, in which case the framestore ifget (see Appendix B.2) function can follow the chain of inheritance in search of a particular tag. A decision has to be made as to how inheritance will be handled. Do higher-level frames “shadow” the tags of those they inherit from or should the full set of tags be returned? Is multiple inheritance allowed and if so, should depth first or breadth first searching be used to find a tag? All of these are issues for which there is not a “right” answer. While ifget provides one implementation, since the ISA tag is available to all clients, they are free to provide another.

8.6 Indexing

Performance is a key issue with any system that interacts with human users. All of the data-storage schemes outlined above eventually encounter issues relating to speed of response for basic queries. In most cases, these issues can be adequately addressed with indices.

Indices map frame features (defined with particular key/value terminals) to vectors of OIDs where they appear. For example, the system may record that (PERSON „Daniel Gruhl”) appears in frames 1, 3, 5, 7, and 203.

This type of index supports the searches discussed in Chapter 5 quite well, but has issues with positional searching (e.g. Bill within two words of Clinton). This is a tradeoff – one that could be addressed by having multiple indices, one for most searches and one for positional ones.

8.6.1 FramerD and ZWrap Style Indices

Even with a simple Boolean occurrence-index approach, there are various ways to implement them. The main difference from a usage standpoint between FramerD-style indices and ZWrap-style ones
Data File

1, 3, 5...

Figure 8-3: The ZWrap index uses a DBM style file to hold positional information, and a flat file to hold the sorted index. Note, space is left at the end of the index vector to accommodate more entries without the need to do a recopy.

are how the list of OIDs are returned when requested. With FramerD, the return DType is a DTset, which by definition is unordered. With ZWrap, the OIDs are returned as a DTvector, in numerically sorted order. While subtle, this difference has an impact on search performance, and where and when computational time is spent.

The FramerD approach allows very efficient insertion of keys to the index, since sets can have multiple occurrences of values and the order which these values are added is not an issue. Given that FramerD has built-in set-manipulation operators, this is not a bad approach for many applications. The difficulty arises when merging two different indices. They best that can be hoped for is a hash-based intersection or union, neither of which is as efficient as the merge sort operation that can be done on a sorted list. Basically, this approach trades more efficient index creation for somewhat slower index use.

The ZWrap approach spends the extra time up front, when a new key is added, to make sure that the vectors are maintained uniquely and in sorted order. The structure used is a GDBM file to hold length, allocation, and offset information, with a simple binary file of arrays to hold the actual index. This second file leaves space at the end of each array to facilitate adding new OIDs to the index, but still is required to check at the time of addition to make sure order is maintained. When returning data, however, since the OIDs can be stored as DTTypes in the file, the system merely needs to read the data off the disk and send it directly over the network, making for efficient operation and extremely fast searches.
8.7 Search Engines

The search engines for ZWrap operate under the primary constraint of speed. Each personal edition of an on-line paper may require hundreds of searches to be performed, so they need to occur quite quickly to prevent the user from having to sit and wait. This is addressed in two ways. First, the channel servers cache the results of previous searches, allowing them to examine only new information. Second, the search engines use the indices as much as possible, going to the frame database only as a last resort.

For Boolean-type searches, the system can merge each Boolean term in a single pass through the index vectors. For activation-type searches, the system creates a hash table, requiring one pass for each rule. There are issues about negating rules (e.g., "The absence of . . .") but these are dealt with by reordering the rules and working in two steps (one to add values and one to subtract). By not considering those terms that have negative values, the very large NOT sets are avoided.

Again, positional searches are supported (e.g., phrase searching and the NEAR operator) but are slow, as they require a Boolean search to identify candidates and then an examination of each frame via sequential scan to determine if it meets the rule.

8.8 Experts

The MyZWrap and Panorama systems both use a large collection of experts to develop understanding. All of these experts can connect to the Framestore, request a frame, examine what is there, and add terminals to reflect their observations. As noted earlier, these experts talk to the framestore using the DType protocol, and can be written in whatever language is convenient. Here is a list of some of the agents currently running:

- **Structure** - This expert is typically the first to run. It uses a wide variety of heuristics to identify the various "structural" elements of the story. For example, the word "by" followed by a proper noun is likely an indication of authorship if it appears in the first few lines of an article.

- **Stemmer** - The list of stemmed words is stored back into the article to facilitate the traditional word-based searches users are likely familiar with.

- **Proper Noun** - This expert uses a simple heuristic to identify what might be a proper noun.

- **Time Spotter** - This expert uses NLP to identify "time words" mentioned in the story. These are reduced to Unix-style times to allow for easy analysis and comparison. This agent
is currently being extended to incorporate the work of Doug Koen[26] that does a better job than the current system at spotting temporal events.

- **Place Spotter** - Also using NLP. Identifies places mentioned in a story.

- **Country Spotter** - Using a list of known countries drawn at runtime from the CIA World On-Line Factbook, this expert spots country names. Due to the potential inaccuracies of the Place Spotter, it runs on the headline and body text of the article.

- **Region Spotter** - Using a list of rules, this expert looks at the country feature and identifies what regions are mentioned in an article, for example Middle East or South East Asia.

- **Continent Spotter** - Using another list of rules, this expert maps country to continent.

- **People** - Using a list of known first names, the system examines all of the proper nouns and identifies people mentioned.

- **Reading Level** - The system makes use of an automated readability index to guess the “grade level” needed to make sense of an article.

- **Media Lab Professors** - This expert runs a filter on the list of people to identify when a Media Lab professor is mentioned.

- **Fishwrap Topics** - All the previously existent Fishwrap keyword-spotting topics are run and their matches written back into the frame.

- **Noun/Verb** - This expert identifies noun/verb pairs and turns them into features for inclusion in the frame. It uses FramerD’s nphrase parser and a Perl script that pulls the main noun and verb from the resultant noun and verb phrases.

This list is by no means exhaustive. Rather, it gives an idea of the span from the very general to the very specific and illustrates how agents can work with each other.

**8.9 Statistical Examination**

The system examines the frames collectively to develop context, to identify when a channel is changing, and to group articles for presentation. This is perhaps the least integrated aspect of the ZWrap system. Exposing statistically-derived information in a way that is not misleading is an art. Each application requires a fair amount of fine-tuning to get right. Much of this work has been
• Richard Butler (28 examples)
  - Iraq (89%)
  - Security Council (64%)
  - Special Commission (53%)
  - British (61%)
  - Iraqi (82%)
  - UNSCOM (54%)

Table 8.1: Associations between Richard Butler and other proper nouns over a two week period in February. This is the complete rule list generated with a support requirement of 20 articles and a confidence requirement of 50%.

done in various “test-bed” systems and is only now being folded into MyZWrap. Those that have been particularly successful will be described first. For details on specific techniques, please refer to Chapter 6.

8.9.1 Association Rules

The most interesting association-rule mining to date has been on simple Proper Noun to Proper Noun co-occurrence. It serves to create capsule descriptions of what various proper nouns have been associated with in the news. One example can be seen in Table 8.1, which lists the association rules generated from a two week period in February for Richard Butler. While this example worked particularly well, MyZWrap does a fairly good job on most major news makers (see Section 9.3 for further examples).

It is interesting to note that Panorama, with its much smaller article set, does not do anywhere near as well with developing association rules. This raises the interesting question of whether even specialized systems should obtain large amounts of news, if only to allow them to better develop their associations.

8.9.2 Occurrence Statistics

The system looks at articles that appear in a channel and computes occurrence statistics for each feature over a period of time. The user can examine these features as well as the current search criteria and make decisions as to whether the search should be modified to include a new feature, or whether old features might do to be removed.
8.9.3 Clustering, Cluster Naming

The ZWrap system can perform clustering on the feature vectors or dimensionality-reduced versions of the feature vectors. K-Means is usually used as it is fast and performs reasonably well. This clustering can be used for several different purposes, including display and presentation.

As mentioned earlier, the K-means to K-lines project[24] explored summarizing clusters with bullet points outlining the high-level similarity features. For example, it was able to identify that most of the articles in a cluster mentioned both Madeline Albright and the Middle East. This approach has yet to be integrated into the main MyZWrap system.

8.9.4 Nearest Neighbor

In addition to finding related articles, nearest neighbor can also be used to select which augmentations to include. The FramerD find-similar command used a subset of features derived from NLP to identify articles “similar” to the one being examined. This was developed (in the context of news) for the San Francisco Chronicle’s SFGate web-site[12 in the Winter of 1998, where the OIDs of related stories were written back into the frame to allow faster access.

In terms of augmentations, MyZWrap used nearest neighbor to find “similar”, hopefully appropriate quotes from Shakespeare, the UNIX Fortune File, and the collected Grateful Dead lyrics to present along with any article being examined.

8.9.5 The Difference Engine

“What’s new” is a question newspapers try to answer, and one which computers can excel at, at least in the case of simple features. One example of such a difference spotter is Word Watch, a feature which mimics a “word of the day” column.

It creates its entries by examining the words that enter the system every day and find “differences”. These new words are filtered through a copy of the Oxford English Dictionary to rule out misspellings and the resulting hits are presented with definitions linked to the articles that triggered them.

8.10 Tools to assist presentation and user interface

All of the different analysis mentioned above occurs “behind the scenes”, before the user ever sees their paper. ZWrap is defiantly a skewed project at this point, with a disproportionately small amount of effort having been dedicated to improving how articles are presented.
Still, some issues have been explored in enough detail to be worth mentioning.

### 8.10.1 Top Level Presentation

At the highest level, as discussed in Section 7.6, ZWrap presents its information in channels (see Figure 8-4). Each channel is focused on a specific topic and the channel list is, in general, shared between users. ZWrap places these channels in a 3-column format (3-columns being appropriate for the 1280 × 1024 used most often to demonstrate the system). A GUI is provided to allow simple channel selection as well as page layout management.

At the moment, the channels are provided in a simple list. A hierarchical presentation is envisioned as channel definitions become richer. Such a hierarchy can be built either by hand, or by classification of new channels using an existing hierarchy.

### 8.10.2 Searches vs. Repurposed Information

Not all channels perform searches on the newspool to generate their content. MyZWrap is migrating to an approach where a channel is implemented as a server that can generate information in any way desired. Channels such as the Word Watch channel, weather channels, and comics can acquire their news in non-search fashion and display it in a manner that is integrated with the rest of the channels.

---

Figure 8-4: The top level MyZWrap presentation is done as channels. Here are 3 channels from Cameron Marlow's paper.
8.10.3 Channel Creation

The majority of MyZWrap channels are the results of searches. Since the system does not try to anticipate all the searches that might be desired, some mechanism needs to be provided to allow for channel creation.

At the moment, the only "user friendly" creation mechanism is an AltaVista[13]-like interface in Panorama that allows a search to be turned into a named channel. This allows any query on the newspool to become a channel, and allows those familiar with popular web search engines to design channels without needing to learn a new technique. Once a channel is created, the "search explanation" and "channel analysis" feature in MyZWrap allow fine-tuning if a user is able to access and modify the low-level channel description.

Other experiments have been performed using the FramerD relevance-feedback mechanism (implemented as find-similar from a set of examples). To date, these have yet to be integrated into either of the main test systems.

8.10.4 Tools for Channel Analysis

MyZWrap provides some simple tools for channel analysis. The first is an examination of which features have recently appeared in articles that have been selected for a channel (see Figure 8-5). By examining frequency of occurrence, the channel maintainer can identify what has been active and perhaps change the channel definition to account for it.

A second tool was developed by Marko Turpeinen for the Daily Catch project[43]. It identifies the people, places, and things in an article, and provides a slider on a time-line to explore how those features occurred over time. The various features are arranged above the time-line and, as the slider moves along it, those features that were prominent for a given week appear darker, while those mentioned less appear lighter. When applied to a long running story (e.g., the Kosovo story), it provides a mechanism for examining when various players and locals come to the fore.

A third tool is the "New Feature Alert", of which WordWatch is a good example. This kind of tool brings new features to the user's attention. This may, perhaps, be a little too indiscriminant a criteria for most features (such as names, where new ones appear every day). Perhaps a better approach would be to see if a new feature is mentioned repeatedly over time, so as to avoid one-hit-wonder-type features. Still, by bringing information to the user's attention only when something new occurs, the system can minimize the amount of rechecking of channels a user needs to do to keep abreast of new events.
Analysis of topic Kosovo

- Feature selected which appear in the topic with at least 20% certainty and 3 example support.
- Most recent 77 articles will be examined (there are 77 examples of this topic in the active pool).
- First article examined dated: Wed Dec 31 19:00:00 1969
- Last article examined dated: Fri Aug 13 12:32:00 1999

<table>
<thead>
<tr>
<th>Higher Level Features</th>
<th>Low Level Features (stemmed words)</th>
</tr>
</thead>
<tbody>
<tr>
<td>format: Body Text</td>
<td>Kosovo (94% (73/77))</td>
</tr>
<tr>
<td>reference:</td>
<td>a (92% (71/77))</td>
</tr>
<tr>
<td>up:field:</td>
<td>in (85% (66/77))</td>
</tr>
<tr>
<td>filingdate:</td>
<td>ap (80% (62/77))</td>
</tr>
<tr>
<td>proper noun: Kosovo</td>
<td>the (77% (69/77))</td>
</tr>
<tr>
<td>newstream: apws</td>
<td>and (72% (56/77))</td>
</tr>
<tr>
<td>priority: r</td>
<td>of (72% (56/77))</td>
</tr>
<tr>
<td>selector: wstm-</td>
<td>to (71% (55/77))</td>
</tr>
<tr>
<td>version:</td>
<td>on (70% (54/77))</td>
</tr>
<tr>
<td>category: i</td>
<td>66% (51/77)</td>
</tr>
<tr>
<td>wordcount:</td>
<td>62% (48/77)</td>
</tr>
<tr>
<td>language: ENGLISH</td>
<td>59% (46/77)</td>
</tr>
<tr>
<td>proper noun: NATO</td>
<td>44% (34/77)</td>
</tr>
<tr>
<td>proper noun: Pristina</td>
<td>38% (30/77)</td>
</tr>
<tr>
<td>proper noun: Serbs</td>
<td>29% (23/77)</td>
</tr>
<tr>
<td>proper noun: Albanians</td>
<td>29% (23/77)</td>
</tr>
<tr>
<td>proper noun: Albanian</td>
<td>28% (22/77)</td>
</tr>
<tr>
<td>priority: u</td>
<td>28% (22/77)</td>
</tr>
<tr>
<td>proper noun: Serb</td>
<td>27% (21/77)</td>
</tr>
<tr>
<td>proper noun: June</td>
<td>24% (19/77)</td>
</tr>
<tr>
<td>proper noun: Yugoslav</td>
<td>24% (19/77)</td>
</tr>
<tr>
<td>proper noun: Russian</td>
<td>23% (18/77)</td>
</tr>
<tr>
<td>proper noun: The</td>
<td>23% (18/77)</td>
</tr>
<tr>
<td>keyword: BC-Yugoslavia-Kosovo</td>
<td>20% (16/77)</td>
</tr>
<tr>
<td>region: EURO BRIT SCAN ENGL</td>
<td>20% (16/77)</td>
</tr>
</tbody>
</table>

Figure 8-5: Feature analysis for the Kosovo topic. The higher level derived features appear on the left. On the right are the low level (stemmed word) features.
8.11 Email, Pagers, etc.

While MyZWrap is heavily web-centric, this should not imply that the web is the only medium worth considering for information delivery. MyZWrap employs email for event notification (when a new story is picked up in a channel) as well as paging (via the Canard project[11]) for delivery of certain articles of note.

These methods of delivery are somewhat more intrusive than a website, as they are both “push” delivery systems, as opposed to the “pull” approach typical of the web. Before employing either of them, a system should have a good idea that an article is important enough to be delivered.

This is where ZWrap does a better job than many conventional systems, since it allows more precise selection criteria for email or pager alerts such as a drastic change in some channel of interest, or particular events or disasters occurring, or:

BUSINESS:My_Company AND
("layoffs" OR "plant closings" OR "downsizing")

8.12 System Status

ZWrap is an architecture of several machines and dozens of programs performing their tasks more or less autonomously. Keeping them all up and running would be a Sisyphean task without some kind of monitoring tool. The monitoring tool of choice for MyZWrap is a freeware program called Big Brother[31]. This system performs periodic checks of all the ZWrap components and provides a status board (see Figure 8-6) and pages maintainers when critical components go down. By glancing periodically at the board and addressing any yellow or red lights that appear, the system maintainers can keep the system running and anticipate certain types of problems before they ever impact the user community.

Another aspect of system maintenance when there is more than one maintainer is that of communication. MyZWrap addresses this through the use of a special maintainer’s channel, to which system maintainers can post articles about what they are doing. Maintainers subscribe to this channel and read about the status of various TODO lists, changes to APIs, etc.

8.13 Conclusion

This has been a brief tour of some of the components of the ZWrap system. A few high level engineering observation:
The ZWrap approach allows for the construction of fairly complicated real-time information understanding and presentation architectures out of relatively simple pieces.

The approach allows for an almost arbitrary degree of scalability by distributing the parts of the system to arbitrary numbers of arbitrary types of machines.

The approach encourages development by allowing new components to be added without adversely impacting the existing system.

The approach encourages incremental improvement of existing components, since as long as the network interface remains unchanged, the other parts of the system don’t know or care how a component accomplishes its task.

By involving users in channel creation and providing maintenance and status tools, the number of system maintainers can be kept to a minimum.
Chapter 9

Results

The ZWrap system as a whole is an interactive information-retrieval system and, as such, it is difficult to rigorously evaluate it. Side effects such as users gaining familiarity with the task, the style of the query being made, and differences between the experience and expectations of users create a situation where useful, quantitative, repeatable results are difficult to achieve. There are a number of standard “tasks” that attempt to simulate an ongoing user interaction[46] and modifications to them that attempt to more closely simulate user interaction[7] for evaluation, but ultimately the “correct” way to evaluate these systems is to have a large number of typical users work with the system and examine their experiences and opinions of the system.

Unfortunately, such studies are expensive to run and often inconclusive. Some projects (e.g., Google[23]) take the approach of releasing their systems to the Internet and allowing “merit” to be determined by the number of hits.

Fortunately, for a thesis, the important measure of success is the successful argument of the thesis statements. For reference, the thesis statements made in Chapter 1 are:

1. The blackboard system approach allows for the easy construction of efficient, distributed machine-understanding systems for developing meaning in large, dynamically evolving corpora such as news.

2. Statistical and machine-understanding techniques can inform each other if they share a common representation that supports such sharing.

To argue the first point, we will examine the development, performance, flexibility, and scope of the existing ZWrap systems. This discussion will also address how the ZWrap approach avoids the traditional issues that arise in blackboard systems.
The second point will be argued in two steps: first an examination of how machine understanding can inform a statistical method; and second to examine an example of the reverse. The machine-understanding-to-statistical direction will be examined in the context of a nearest-neighbor classification system, where an exploration was made of performance with and without feature augmentation. The statistical to machine understanding direction will be examined in light of data mining and its applications to knowledge-base development and maintenance.

9.1 Blackboard Approach

As noted in Chapter 2, blackboard systems were initially developed to take advantage of ease of development as well as apparent opportunities for parallelism. Unfortunately this approach has fallen into a certain disfavor due to the performance limitations resulting from the serialization of experts that occurs when many of them are examining the same problem.

If ZWrap is to be a validation of the blackboard architecture, it needs to show that it can take advantage of the benefits (parallelization and ease of development) while circumventing the disadvantages (poor performance) of such a system. First, some numbers to characterize the existing system:

- MyZWrap handles on the order of 10,000 articles a day.
- This translates to approximately one gigabyte of text per month.
- At the moment, MyZWrap has been running (and gathering news) internal to the Media Lab for about 8 months. This translates to around 10 gigabytes in the total corpus of news.
- The system can keep augmentations fully integrated to within approximately 20 minutes of when news enters the system (i.e. this is the time it takes the system to fully augment an article).
- The system currently runs distributed across 5 Intel-type Unix machines.

This performance can be achieved due to ZWrap’s multi-blackboard approach. This modification to the traditional blackboard architecture keeps all of the experts working all of the time. The system maintains the ~ 20 minute performance point by spreading this work to all 5 machines.
9.1.1 Flexibility

One of the most compelling strengths of a blackboard architecture is in the ease with which new components can be added to the system and with which existing ones can be upgraded and improved upon. Three pieces of anecdotal evidence are presented here to support this observation.

First, when developing the Panorama system in the summer of 1999, a full “through” version of the system was implemented in roughly five days by one engineer (the author). This included the central blackboard, agents to post articles to the blackboard, a single augmentation agent (a Porter stemmer), the GUI, and the article selection structure. With this rather minimal (less than one week) overhead it becomes feasible to use this architecture even on small projects, since even this start up time can be circumvented by reusing components from existing systems.

The next observation is the ease with which new agents can be added to the system. While developing the technologies used by a particular agent to understand some aspect of an article may very well be a life’s work, actually grafting an existing technique into the system can be performed very quickly. Erik Mueller and Warren Sack created a “food spotter” agent that, aside from issues of getting an account on the host machine, took them an afternoon to integrate. Likewise, a “color spotter” agent was implemented at around the same time in less than an hour. This low overhead of including specialized understanding agents is heartening, as it encourages use of them whenever a particular kind of observation is needed to correctly select articles.

The last observation is to the ease at which pieces of the structure can be improved without impacting the rest of the system. For this we examine the portion of the architecture which has had the most tumultuous history, the Framestore.

When ZWrap was initiated there was the need for a database to hold news articles. This (through the Daily Catch project) led to the use of the native FramerD framestore for the first month or two of the project. When database performance issues began to dominate, the framestore was re-implemented in Perl on top of a GDBM database. When again the performance issue arose a framestore was implemented in C++ on top of a flat file.

This approach served well for around a year before it again became necessary to seek even faster database solutions. This led to the implementation (in C++ again) of a framestore tightly coupled to the underlying file system. While fast, this approach proved to be too inflexible and a transition was made to an SQL database to hold the data (with a Perl interface layer). Later transitions where made to multi-table SQL databases when table size exceeded the databases efficient-access capabilities.

Through all these changes the experts needed no modification and, in fact, did not “notice” the
Figure 9-1: ZWrap allows agents to be distributed among arbitrary numbers of machines. This is helpful up to a point. If the above system is performance bound by the stemmer (running by itself on one machine) then spreading the other agents to more machines will not result in a performance improvement.

change, since a consistent interface was continuously presented to the network. This flexibility is important in allowing groups of people to work on ZWrap without adversely impacting each others projects, as each portion of the system is free to advance at its own pace.

9.1.2 Scalability While Preserving Parallelizablity

ZWrap was designed to run on a collection of machines and ultimately the ability to add more hardware as additional computation is needed is what makes blackboard systems (or any parallelizable architecture) attractive. Since all the pieces of the system communicate over the network, the impact of changing the host with which the agents are communicating for a particular service is minimal.

Agent Performance

With the current system, adding more machines will yield improved performance only for the first half dozen or so machines. As is the case in many parallel systems, one sub-process soon dominates performance. In MyZWrap, this is most often the stemmer, and performance can only improve until it is bound by the stemmer running on a machine by itself (see Figure 9-1).

This is an artifact of how the stemmer is written and not a reflection of any particular complexity in the stemming process itself.
Figure 9-2: The multiple-blackboard architecture provides a natural way to break up the problem. Multiple instances of slower experts (e.g., the stemmer) can all be working on different parts of the framespace at the same time.

If ZWrap were a traditional blackboard system with only a single problem under consideration at a time, it could improve no more. However, since ZWrap employs a multiple-blackboard approach, more than one instance of an agent can be running at a time, as long as they are working on different blackboards. This requires just a small modification to the logging structure of Section 2.3.1 but results in many instances of an agent, each being assigned to work on different ranges of articles (see Figure 9-2).

This approach works well as long as the amount of time a single agent requires to process a single article is less than the ~ 20 minute target time. Fortunately, even with the most arduous machine-understanding methods used in MyZWrap this is currently the case. If it is ever not true, then this approach cannot help.

The next step is to look at dedicated multi-node approaches in cases where a particular agent can be internally parallelized further (e.g., an NLP engine where the initial parse of each sentence can be done in parallel). In this case, either dedicated super computers or super-computing structures (e.g. Beowulf[4]) can be used to accelerate particularly slow agents (see Figure 9-3).

Database Performance

Another potential ZWrap performance bottleneck concerns the ability of the database to handle large numbers of possibly simultaneous transactions (e.g., requests for frame contents, modifications, and additions). Extensive work has been done in the database community on how to allow databases
Figure 9-3: If one particularly slow expert is amenable to parallization itself, then a particular node can be replaced with a parallel computation structure such as a Beowulf or SP2.

to deal with large numbers of simultaneous transactions, with approaches ranging from multi-node clusters to serial-storage architectures and SMP-enabled database engines. However, with ZWrap there is an even easier approach. A simple hashing function can be applied to the frame object ID. This hash is used to identify which server holds a particular frame. This allows the framestore to be partitioned into arbitrary numbers of pieces (see Figure 9-4).

Coupled with a fast-switching network that allows machine-dedicated circuits of communication (e.g., ATM networks), there is no reason to believe that this structure could not usefully grow to the order of a hundred machines, split between experts and framestore as load balancing dictates.

But Why Bother...

Despite all these opportunities for extremely large structures, at this point the issue of scalability to these extremes is somewhat academic, since the simple five machine cluster currently running easily handles the loads discussed above. This is enough to allow for interesting research on larger dynamic real-time corpora than are traditionally contemplated with performance times attractive enough to encourage experimentation.

9.2 Machine Understanding Informing Statistics

There is a fair amount of literature on feature spotting as an adjunct to traditional information-retrieval methods[41]. ZWrap combines these methods by implementing domain-specific “spotters”
Figure 9-4: One possible scheme for distributing the load on the framestore across three machines based on a trivial hash of the frame object ID. Due to the access patterns on ZWrap, a better algorithm might be \( \lfloor oid + 128 \rfloor \mod 3 \) (supporting bulk reads and writes), but of course this is application dependent.
whenever it appears that they will offer higher-quality features for the statistical techniques.

In order to examine the impact of the ZWrap approach on these methods, experiments were performed with a traditional statistical classifier. The dataset used for these explorations is the Reuters-21578 dataset\[29\], chosen for it’s close match to the kinds of problems ZWrap encounters (i.e., news classification) as well as it’s familiarity to the information retrieval community. This test used the suggested “ModApte” split for training and testing described in the README.txt file which accompanies the corpus.

### 9.2.1 Experimental Protocol

In order to evaluate statistical classifiers in the context of the ZWrap system, the following task is proposed: To construct a typical set of selection rules for a ZWrap topic. The task will be performed with sets of training articles that are both tagged and untagged and the results compared. The goal will be to generate a set of rules for the activation-channel-selection method described in Section 5.3.

The “channels” to be created will be ones that select stories from particular continents. The CIA World Factbook\[45\] will be used to identify country names and their mapping to continents. The mention of a country in an article will be construed to be a “match” to the appropriate channel for that classification.

The identification will use the following algorithm:

1. Construct a dictionary of all terms in the training set.
2. Construct a normalized vector for the entire training corpus (This includes those articles later used for selection.).
3. Construct a normalized vector for the training set.
4. Identify the 10 terms whose presence is most indicative of the training set as compared to the corpus norm.
5. Identify the 10 terms whose presence is most indicative of being more like the corpus norm than the training norm.
6. Use a projection onto these 20 dimensions to generate a cosine distance between each test article and the “corpus-norm” point and “training-norm” point. Assign the test article to whichever point it is closest to (i.e., selected for the topic or not).
Table 9.1: Categorization results for North America.

There is some concern about the negative effect of over augmenting an article. This has been observed by the author with other experiments on this corpus, where each addition of unused augmentation resulted in a few percentage point loss on precision and recall. The approach of selecting the 20 “best” terms overcomes this issue by only examining those terms that make a difference. To illustrate this the augmented corpus has additional augmentations relating to: Stemmed Words, Proper Nouns, Geography, Metals, the Petro-Chemical industry, and Finance. All of these, except the Geography augmentation, will tend to be ignored by this classification method unless they contribute.

9.2.2 Experiment Results

The experiment was tuned for the South America (Table 9.2) topic before being applied identically to the rest of the continents. As a result, for two, Middle America (Table 9.5) and Europe (Table 9.6) the classifier could clearly be tuned to trade recall for precision.

In general, at the 400-document training levels, the precision of selection of the augmented documents is on average 3.8 times as good as for the unaugmented ones (67.3% vs. 17.3%) and the precision 3.3 times as good (92.9% vs. 27.7%).

This is not surprising since there is a “perfect” feature which the system can use for classification. Again, noting how many different features were included, there is no reason not to add spotters until “perfect” features exist for any particular classification that might be done.
<table>
<thead>
<tr>
<th>Training Examples</th>
<th>With Augmentation</th>
<th>Without Augmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
</tr>
<tr>
<td>1</td>
<td>26.5</td>
<td>44.3</td>
</tr>
<tr>
<td>2</td>
<td>27.2</td>
<td>48.5</td>
</tr>
<tr>
<td>5</td>
<td>17.5</td>
<td>66.9</td>
</tr>
<tr>
<td>10</td>
<td>11.6</td>
<td>62.6</td>
</tr>
<tr>
<td>20</td>
<td>13.3</td>
<td>77.4</td>
</tr>
<tr>
<td>50</td>
<td>69.2</td>
<td>82.3</td>
</tr>
<tr>
<td>100</td>
<td>81.8</td>
<td>63.3</td>
</tr>
<tr>
<td>200</td>
<td>85</td>
<td>68.3</td>
</tr>
<tr>
<td>400</td>
<td>85.9</td>
<td>73.2</td>
</tr>
</tbody>
</table>

Table 9.2: Categorization results for South America.

<table>
<thead>
<tr>
<th>Training Examples</th>
<th>With Augmentation</th>
<th>Without Augmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
</tr>
<tr>
<td>1</td>
<td>3.8</td>
<td>12.8</td>
</tr>
<tr>
<td>2</td>
<td>5.2</td>
<td>15.7</td>
</tr>
<tr>
<td>5</td>
<td>20.1</td>
<td>95.5</td>
</tr>
<tr>
<td>10</td>
<td>21</td>
<td>98.3</td>
</tr>
<tr>
<td>20</td>
<td>23</td>
<td>100</td>
</tr>
<tr>
<td>50</td>
<td>31.9</td>
<td>75.4</td>
</tr>
<tr>
<td>100</td>
<td>35.4</td>
<td>82.7</td>
</tr>
<tr>
<td>200</td>
<td>70.1</td>
<td>98.6</td>
</tr>
<tr>
<td>400</td>
<td>96</td>
<td>90.3</td>
</tr>
</tbody>
</table>

Table 9.3: Categorization results for Asia.

<table>
<thead>
<tr>
<th>Training Examples</th>
<th>With Augmentation</th>
<th>Without Augmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
</tr>
<tr>
<td>1</td>
<td>2.8</td>
<td>23.4</td>
</tr>
<tr>
<td>2</td>
<td>7.9</td>
<td>98.9</td>
</tr>
<tr>
<td>5</td>
<td>9.5</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>8.6</td>
<td>100</td>
</tr>
<tr>
<td>20</td>
<td>12.8</td>
<td>100</td>
</tr>
<tr>
<td>50</td>
<td>88.2</td>
<td>98.9</td>
</tr>
<tr>
<td>100</td>
<td>88.7</td>
<td>98.9</td>
</tr>
<tr>
<td>200</td>
<td>91.4</td>
<td>98.9</td>
</tr>
<tr>
<td>400</td>
<td>91.4</td>
<td>98.9</td>
</tr>
</tbody>
</table>

Table 9.4: Categorization results for Middle East.
<table>
<thead>
<tr>
<th>Training Examples</th>
<th>With Augmentation</th>
<th>Without Augmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>3.1</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>25.3</td>
<td>90.6</td>
</tr>
<tr>
<td>20</td>
<td>34.9</td>
<td>83.7</td>
</tr>
<tr>
<td>50</td>
<td>63.2</td>
<td>100</td>
</tr>
<tr>
<td>100</td>
<td>22.7</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>7.2</td>
<td>100</td>
</tr>
<tr>
<td>400</td>
<td>7.2</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 9.5: Categorization results for Middle America.

<table>
<thead>
<tr>
<th>Training Examples</th>
<th>With Augmentation</th>
<th>Without Augmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
</tr>
<tr>
<td>1</td>
<td>2.1</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
<td>21.4</td>
</tr>
<tr>
<td>5</td>
<td>21.1</td>
<td>75.3</td>
</tr>
<tr>
<td>10</td>
<td>19.6</td>
<td>95.1</td>
</tr>
<tr>
<td>20</td>
<td>16.2</td>
<td>100</td>
</tr>
<tr>
<td>50</td>
<td>18.3</td>
<td>100</td>
</tr>
<tr>
<td>100</td>
<td>17.8</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>25</td>
<td>100</td>
</tr>
<tr>
<td>400</td>
<td>37</td>
<td>94.8</td>
</tr>
</tbody>
</table>

Table 9.6: Categorization results for Europe.
9.3 Statistics Informing Machine Understanding

The approach of using statistics on observations to develop rules for a knowledge base date back to at least Drescher[17], where a synthetic being made observations about the results of its actions on the environment of a simulated world and developed rules which it could later reason with to perform tasks.

The ability to draw "rules" from observations is the goal of this aspect of ZWrap, but it is too ambitious to implement in its entirety for this thesis. The Drescher approach had the luxury of being able to formulate a hypothesis and then test it by trying to perform the operation suggested. A judgment to the validity of the hypothesis could then be made based on how effective it was at achieving the desired result.

At present, the ZWrap system cannot influence the news to test its understanding of the world, but rather must make its observations based on what happens in the real world to generate news. This greatly limits the system's ability to design experiments that fill the gaps in its understanding.

Rather than abandoning this approach all together, ZWrap seeks to identify suspected rules about the world and brings them to the attention of a person or expert who hopefully has the necessary common sense and broader understanding to "fill in the blanks" and decide whether the hypothesis is indeed valid.

The specific task explored is that of automating, to the largest extent possible, the construction of the knowledge bases used by the experts. This is done by performing association-rule data mining on the features (see Section 6.3.3) and examining those features with high occurrence rates (looking at those features with which they co-occur). For these examples, the specific association examined was proper noun to proper noun.

The system treats knowledge engineers as expensive, limited resources and thus seeks to make the best possible use of them. While certainly true of a human, this assumption likely holds true of any deep machine-understanding process, where computation expense makes it a resource best used sparingly.

By looking for implications among high-occurrence features the system seeks to focus the development on those areas that will have substantial and immediate impact to the system. By providing suggested implications, it allows the knowledge-base maintainer to work as a "critic" rather than "creator" as much as possible, lightening the load on the knowledge engineer. If a set of rules already exists, the system can test the validity of the rules in order to identify when a change may have occurred.

To demonstrate how this works in practice, simple association-data-mining techniques were ap-
plied to the proper nouns occurring in a two-week period of news in February. The association rules that were identified had a support of 20 articles and a confidence of at least 50%. We will look at three candidates generated for knowledge development and the “rules” suggested. For these examples, we look at particularly well-supported terms for examples of where additions to the knowledge base would be most helpful.

Example 1 - Biographic

The first example (see Table 9.7) is one where the rules are suggestive, but not enough to fully develop a node about Christopher Hill, who happens to be the U.S. ambassador to Macedonia and first served as the key negotiator in the Kosovo peace accords and later mediated the peace process.

For those not familiar with this kind of data-mining, a quick interpretation of support and confidence is in order. Given an implication \( X \rightarrow Y \), the number of times \( X \) appears in the corpus is called the support. This number is a reflection of how common the rule will be, and allows rules to be discarded if they appear too infrequently to be useful. Confidence is the probability that if \( X \) appears in an article that \( Y \) will too. For a further discussion of other issues related to this, the reader is referred to Section 6.3.3.

For the Christopher Hill example, in the two week period being examined, the proper noun “Christopher Hill” appeared 192 times. In these 192 articles, “Albanian” appeared 87.5% of the time, and so on through the other features. When only two features are being considered, this is just the correlation between these features, but association rule mining allows for \( X \) and \( Y \) to be sets, allowing for the discovery of more complicated relationships.

How this information would be stored in the knowledge base depends on how such a KB stores the relationships between these proper nouns. One approach would be to forge named “K-lines” between the nodes in the database suggested. When a concept (like Christopher Hill) is called to mind, various other concepts (Albania, KLA, etc.) would also be considered pertinent for whatever cognition process was using the knowledge base. If the knowledge base supports strengths for these K-lines, then the confidence and support may be used to suggest values for it (this would resemble the Inference network approach proposed by Turtle and Croft[44] in that documents would be selected by a cascade of activated concepts).

It is worth noting that the further research on Christopher Hill needed to fully develop a formal node is trivial to perform using the free-search functions built in to ZWrap.
Table 9.7: Suggested rules for 'Christopher Hill'. The exact nature of the implication would have to be developed independently of the simple association rule discovery.

<table>
<thead>
<tr>
<th>Term A</th>
<th>Implies</th>
<th>Term B</th>
<th>Support</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Christopher Hill'</td>
<td>→</td>
<td>'Albanian'</td>
<td>192</td>
<td>0.875</td>
</tr>
<tr>
<td>'Christopher Hill'</td>
<td>→</td>
<td>'Albanians'</td>
<td>192</td>
<td>0.557</td>
</tr>
<tr>
<td>'Christopher Hill'</td>
<td>→</td>
<td>'KLA'</td>
<td>192</td>
<td>0.646</td>
</tr>
<tr>
<td>'Christopher Hill'</td>
<td>→</td>
<td>'Kosovo'</td>
<td>192</td>
<td>0.938</td>
</tr>
<tr>
<td>'Christopher Hill'</td>
<td>→</td>
<td>'NATO'</td>
<td>192</td>
<td>0.510</td>
</tr>
<tr>
<td>'Christopher Hill'</td>
<td>→</td>
<td>'PRISTINA'</td>
<td>192</td>
<td>0.510</td>
</tr>
<tr>
<td>'Christopher Hill'</td>
<td>→</td>
<td>'Serb'</td>
<td>192</td>
<td>0.682</td>
</tr>
<tr>
<td>'Christopher Hill'</td>
<td>→</td>
<td>'Serbian'</td>
<td>192</td>
<td>0.542</td>
</tr>
<tr>
<td>'Christopher Hill'</td>
<td>→</td>
<td>'United States'</td>
<td>192</td>
<td>0.542</td>
</tr>
</tbody>
</table>

Table 9.8: Rules proposed for Yugoslavia.

<table>
<thead>
<tr>
<th>Term A</th>
<th>Implies</th>
<th>Term B</th>
<th>Support</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Yugoslavia'</td>
<td>→</td>
<td>'Albanian'</td>
<td>235</td>
<td>0.626</td>
</tr>
<tr>
<td>'Yugoslavia'</td>
<td>→</td>
<td>'Kosovo'</td>
<td>235</td>
<td>0.809</td>
</tr>
<tr>
<td>'Yugoslavia'</td>
<td>→</td>
<td>'NATO'</td>
<td>235</td>
<td>0.604</td>
</tr>
<tr>
<td>'Yugoslavia'</td>
<td>→</td>
<td>'United States'</td>
<td>235</td>
<td>0.672</td>
</tr>
</tbody>
</table>

Example 2 - Short duration

In this example (see Table 9.8) some short-term associations have been spotted, mapping a country to relevant events. News is a dynamic process and even a "perfect" knowledge base would quickly fall out of date. The ability to identify transient associations (and remember which ones were in force at a particular time) is an important feature for both the machine-understanding system itself, but also for the news consumer.

Example 3 - Geographic

This example of the West Bank (see Table 9.9) is an excellent one, in that it could be included as a rule almost verbatim. Unfortunately, realizing that this is the case is a difficult task. Providing the list of rules with a simple "accept" button is one way of allowing the system to maximize the time of the knowledge engineer.

93
Table 9.9: West Bank rules. Note, the statistical system does not “know” the relationship between Israel and Israeli. This is one place where a knowledge base would allow for better rule mining (perhaps built by noting the bijection ‘Israel’ ↔ ‘Israeli’ has a high confidence and support).

Summary

These examples illustrate three types of knowledge that can be identified (biographic, short duration and simple “common sense” geographic/political knowledge) when statistics are used to analyze a corpus to inform machine understanding. How these ultimately are introduced to the shared knowledge bases depends on how they are implemented.

9.3.1 Closing the Circle

One last observation to make is that after updating the knowledge bases used by the experts, the experts may want to reexamine the corpus in light of their new knowledge. If this reexamination yields changes to the corpus, then later statistical examinations will reflect this new information when the knowledge expert next updates the system. This loop can continue as new information enters the system, either as new news, new experts, or new information in the knowledge base.

9.4 Conclusion

The two thesis points have been argued by showing that large systems can be constructed with the blackboard architecture and that such an architecture can support a representation that allows both machine understanding and statistical methods for extracting meaning.
Chapter 10

Conclusions

This thesis was motivated by four observations:

- The general lack of an efficient, flexible way in which to deal with large, evolving corpora in a non-trivial (i.e., more than an inverted dictionary) manner and the general notion that larger corpora require simpler techniques.

- The perceived hard division between the machine understanding approaches to information retrieval and those developed from a purely statistical background.

- The tendency of systems that perform any understanding of their text to quickly move to representations that are opaque to human comprehension (statistical systems are notorious for this).

- The extent to which information retrieval systems (perhaps as a side effect of the above point) fail to share any of their understanding with their users and, as a result, can make little use of the end user to enhance the development of meaning.

To this end, an architecture was proposed that addressed these issues and two sample systems employing this architecture were constructed. The resulting systems were evaluated both quantitatively and qualitatively to facilitate comparison with existing techniques. In doing so, several contributions have been made.
10.1 Contributions

The contributions of this thesis can be broken down into three main areas. The first addresses providing a structure for efficiently dealing with larger corpora (the first motivation above). The second is showing a bridge between statistical and machine-understanding methods (the second motivation above). In doing so, the third point (preserving human readability) is addressed. The third contribution is in the enhanced presentation and accessibility of the system (the forth motivation).

On the point of architecture, ZWrap:

- Addresses the traditional deficiencies of the blackboard architecture and presents a model of how the approach may be used to efficiently develop meaning in large, dynamic corpora.

- In doing so, the system provides a framework in which a “society of agents” approach can be scaled up and applied to essentially arbitrary sized text understanding problems. The framework allows the agents to interact in a controlled way through the blackboard and allows them to be distributed over whatever computational resources are available.

- The architecture provides a light-weight, reusable structure that can be used in many different kinds of experiments. This ease of porting existing techniques to ZWrap will allow for a “toolbox” of experts to be developed and applied to problems as appropriate, rather than needing to be developed anew for each investigation.

- The architecture allows computationally intensive investigation of articles to be performed ahead of time, and the resultant structures stored. This allows more in-depth examination of articles at search time (using the precached features) without the need for the user to wait for the results.

On the point of representation and the link between statistical and machine-understanding systems, ZWrap:

- Demonstrates the suitability of frame-type techniques for very large (i.e. millions) collections of information.

- Provides an example of how statistical techniques (especially of the data-mining class) can assist in the creation of knowledge bases for machine-understanding techniques.

- Provides an example of how machine-understanding feature identification can assist clustering for article categorization.
• Illustrates that such a system can maintain human-accessible internal representations and still perform with sufficient speed.

On the last point, presentation, the system:

• Provides methods for sharing the information it develops with the user, opening the door for extending augmentation approaches to a wider class of articles.

• Encourages user development of topics and selection criteria. In doing so, ZWrap provides pathways for integrating back into the system information and understanding provided by the user.

10.2 Future Work

Like any project of this size, this dissertation is less an end point and more a way point. There are many directions in which this thesis could expand, and the below is just a sampling of where additional consideration may be warranted:

• Addition of more experts – The addition of more experts with focus on specific topics is always beneficial. Statistical examination can determine if a particular expert’s observations are helpful for a particular task, so a kitchen sink approach of adding any interesting feature spotting method is probably a useful direction in which to go. Specific agents that might be useful to add include time spotters, latitude and longitude computers, famous people spotters, etc. as well as domain specific “experts” such as “technology-news” spotters.

• Better integration of statistics – Mark Twain pointed out that there are “Lies, Damn Lies, and Statistics”. Even when trying not to be intentionally obtuse, raw statistics can be misleading. Ways to share such information informatively (as is done with association rules) need to be developed and integrated.

• Formalized Knowledge Bases – The current approach has no formalized knowledge bases. This makes it difficult for various agents to share this kind of observation with each other on anything but a per article basis. Providing a few formalized knowledge bases (for people, places, etc.) would allow the more referential tagging as discussed in Section 8.5.4, and thus provide even higher-level observations to the statistical methods.

• User Model – One particular knowledge base that would have tremendous impact would be a well-integrated user model. This would open the door to other statistical methods such
as automated collaborative recommendations as well as providing a method for storing user specific views on the news that might influence article selection but that might not be generally applicable (e.g., the belief that the people at the ____ lab don't know what they are talking about and I want to see no news that cites them).

- **Presentation** – The PLUM approach of a fully integrated presentation system that can take advantage of both a user model and the augmented information in the article frames would be ideal. Integrating some advanced visualization techniques such as automated maps, timelines, etc. would make the user presentation even more helpful.

- **Channel Selection** – The Daily Catch listed a large number of ways in which articles could be selected for presentation. Integrating these with easy-to-use GUIs for construction of new channels would make on-line paper even more accessible to users.

- **Expand to 1,000,000 users** – There are many purely engineering issues that need to be addressed before the system can be used simultaneously by millions of users. Many of these are database issues that can be addressed by a transition to a commercial product, but this work has not been done. Others involve the issues of caching, formatting, searching, and load balancing that are common to all large web sites.
Appendix A

Dtypes Specification

The DTYPES protocol was developed by Nathan Abramson[1] in 1988 and has been evolving ever since. By 1996 the protocol had drifted somewhat as different research groups at the Media Lab had modified it (mainly by adding new types). The following document was developed at that time to again unify the standard and allow for future extensions in a more compatible fashion.
1 Introduction

This document describes the dtypes binary network transfer specification. It is an attempt to merge the existing dtypes standards used in the machine understanding group, and in the garden, into a single new one.
2 Tag Space

Currently, tags 0x00 — 0x14 are being used in the standard dtype representation. The following method for extending this set is proposed.

2.1 Package Space

The second bit of the tag indicates whether or not a package is being used (i.e. 0x40 & tag != 0x00). A package is defined as a group of related dtypes that are useful for a given application (for example, unicode char and unicode string might have been in the "string extension" package.

Package dtypes are defined as being

Format:

<table>
<thead>
<tr>
<th>?</th>
<th>package tag</th>
<th>bytes/types</th>
<th>small/large</th>
<th>dtype in package</th>
<th>size</th>
<th>......</th>
</tr>
</thead>
</table>

For example, one might have defined:

0x40 - string-extension package
0x4000 - unicode char
0x4001 - unicode string

One important feature of the "package" class of objects is that the client need not understand the package to be able to read it in, store it, and write it out. The bit starting the second tag indicates whether the data stored in the package is bytes (0) or dtypes (1). The second bit of the tag indicates if there are few (1 byte of size code)(0) bytes or dtypes stored in the type, or many (4 bytes of size code)(1). This is the followed by a 6 bit code that specifies which type in the package is being used.

2.2 Experiment Space

It is anticipated that people will want to experiment with new dtypes from time to time. To facilitate this, it is proposed that the high order bit of the tag (i.e. 0x80 & tag != 0x00) indicate that the dtype (or dtype package) is under development or proposed.

It is encouraged that the number chosen for development is the anticipated one for implementation (e.g. the development number for the string-extension package should be 0x00, that is 0x40 with the high order bit flipped.)
3 Binary Format

All dtypes start out with a one byte tag. This is then followed by type specific data. In all cases, network byte order is used (MSB first).

**dt_invalid 0x00**
A `dt_invalid` indicates `#invalid`. It is represented by a single byte, 0x00.

**Format:**
```
tag
|0x00|
```

**dt_null 0x01**
A `dt_null` indicates `#null`. This is the representation of the empty list, and also serves as the list terminator. This is not a null pointer. This dtype is represented by a single byte, 0x01.

**Format:**
```
tag
|0x01|
```

**dt_bool 0x02**
A `dt_bool` is used to represent `#t` or `#f`. The representation is two bytes, with the first byte being the tag 0x02 and the second indicating the value of the dtype. If `value=0x00` then the dtype is expressing `#f`, otherwise it is expressing `#t`. It is appreciated if `#t` is expressed as `value=0x01`.

**Format:**
```
tag
|0x02|val|
```
```
val = 0x00 -> #f
val != 0x00 -> #t
```

**dt_int 0x03**
A `dt_int` is used to represent a fixed signed integer number. The dtype is 5 bytes, a tag of 0x03 followed by a four byte network order (MSB first) two's complement integer representing the value of the dtype.

**Format:**
```
tag
|0x03|value
```
```
value is a Network Order (MSB first) 2's complement integer
```

**dt_float 0x04**
A `dt_float` is used to represent an IEEE 754 single network order floating point number. The dtype is represented in 5 bytes, a tag 0x04 followed by a 4 byte network order (MSB first) IEEE 754 float. This is interpreted as:

\[\text{value} = 1 \times 2^{\text{exponent}} \times (1 \pm \text{fraction})\]

1 Thanks to Vadim Gerasimov for this
Chapter 3: Binary Format

Little Endian 4 bytes:
- 1 Bit (msb) - s
- 8 Bits - e
- 23 Bits (lsb) - f

\[
\begin{align*}
\text{if } 0 < e < 255 \text{ then } v &= (-1)^s \times 2^{(e-127)} \times (1.f) \\
\text{if } e = 0 \text{ and } f < 0 \text{ then } v &= (-1)^s \times 2^{(-126)} \times (0.f) \\
\text{if } e = 0 \text{ and } f = 0 \text{ then } v &= (-1)^s \times 0 \\
\text{if } e = 255 \text{ and } f < 0 \text{ then } v &= \text{NaN} \\
\text{if } e = 255 \text{ and } f = 0 \text{ then } v &= (-1)^s \times \text{Inf}
\end{align*}
\]

Format:
- type ieee float
  - 4 bytes

**dt.packet 0x05**

This is the "catch all" dtype for raw data. It is a simple stream of bytes sent over the network. The representation for this is 5 bytes plus the data. Specifically, a one byte tag (0x05) followed by a 4 byte network order unsigned integer, expressing the number of data bytes in the packet, followed by the data bytes.

Format:
- type size data
  - 4 bytes

**dt.string 0x06**

This is the dtype used to send strings of data. It very closely follows the packet layout, with a tag (0x06) followed by a 4 byte Network order unsigned integer, containing the number of characters in the string, followed by those characters.

**Note:** The trailing null on the string is optional. If it is included in the string, it must be included in the count of characters in the string.

It is preferred, however, that the null be left off. For compatibility reasons, a dtype reader should be able to understand strings with and with out nulls, but should write them without nulls when practical.

Format:
- type size data
  - 4 bytes

**dt.symbol 0x07**

Functionally equivalent to a dt.string, with the change in tag to 0x07.

**dt.pair 0x08**

This is the first of the compound dtypes. A pair is, easily enough, two other dtypes of any type. The dtype itself is just a tag (0x08) followed by two additional complete dtypes of any type.

Format:
- tag
dtype-car
dtype-cdr
Chapter 3: Binary Format

**dt_vector 0x09**
A vector is a fixed length array of other dtypes. It is transmitted as a 5 byte header, followed by the dtypes of the vector in order. The header is made up of a tag (0x09) followed by a 4 byte network order unsigned long integer containing the number of elements in the vector. These elements then follow, in order.

```
Format:
tag num elems element
|0x09| 4 bytes | elem 0 | elem 1 | elem 2 ... |
```

**dt_voidptr 0x0a**
This dtype is simply the 1 byte tag (0x0a), and it refers to some `void *` value. Since these values make no sense over the network, what this value refers to exactly may be application specific (e.g. it could be the null set).

**dt_compound 0x0b**
This is the preferred “extension type” for dtypes. It consists of a pair of objects, the first of which is a symbol to identify what the class of the dtype is, and the second of which is the data for the type. For example, one might have implemented type “complex” by the following construct:

```
<'complex . ( real . imag )>
```
Where the outside sharp braces refer to the compound.

```
Format:
tag
|0x0b|symbol dtype|data dtype|
```

**dt_error 0x0c**
This dtype signals that an error has occurred. It is followed by another dtype which elaborates on the error in some way.

```
Format:
tag
|0x0c|dtype elaborating error|
```

**dt_exception 0x0d**
This dtype signals that an exception has occurred. It is followed by another dtype which elaborates on the error in some way.

```
Format:
tag
|0x0d|dtype elaborating error|
```

**dt_oid 0x0e**
This represents a very unique identifier. OID’s are issued, and should be used to uniquely tag objects that need to remain unique worldwide. The OID pool is maintained by haase@media.mit.edu and he should be contacted if you are interested in using this type.

```
Format:
tag oid
|0x0e|8 byte identifier|
```
4 Existing Packages

The following packages currently exist, and are being maintained by the indicated parties.

4.1 Additional Character Representations

Package: Ox40
Package Name: Additional Character Representations
Package Administrator: Ken Haase <haase@media.mit.edu>

This package is for non ascii representations of characters, including unicode, gnu code, ISO standards, etc.

4.2 Advanced Numerics

Package: Ox41
Package Name: Advanced Numerics
Package Administrator: Ken Haase <haase@media.mit.edu>

This package holds additional representations for numbers, including Complex, Rational and Bignum among others.

4.3 Data Collections

Package: Ox42
Package Name: Data Collections
Package Administrator: Ken Haase <haase@media.mit.edu>

This package holds a number of ways of representing groups of data, including slot maps, sets, bags, etc.
## Type Index

<table>
<thead>
<tr>
<th>Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>3</td>
</tr>
<tr>
<td>dt_bool</td>
<td>3</td>
</tr>
<tr>
<td>dt_compound</td>
<td>5</td>
</tr>
<tr>
<td>dt_error</td>
<td>5</td>
</tr>
<tr>
<td>dt_exception</td>
<td>5</td>
</tr>
<tr>
<td>dt_float</td>
<td>5</td>
</tr>
<tr>
<td>dt_int</td>
<td>5</td>
</tr>
<tr>
<td>dt_invalid</td>
<td>5</td>
</tr>
<tr>
<td>dt_null</td>
<td>3</td>
</tr>
<tr>
<td>dt_odd</td>
<td>5</td>
</tr>
<tr>
<td>dt_pair</td>
<td>4</td>
</tr>
<tr>
<td>dt_string</td>
<td>4</td>
</tr>
<tr>
<td>dt_symbol</td>
<td>4</td>
</tr>
<tr>
<td>dt_vector</td>
<td>5</td>
</tr>
<tr>
<td>dt_voidptr</td>
<td>5</td>
</tr>
</tbody>
</table>

## Table of Contents

1. **Introduction** .............................................. 1

2. **Tag Space** ............................................. 2
   - 2.1 Package Space ....................................... 2
   - 2.2 Experiment Space .................................... 2

3. **Binary Format** .......................................... 3

4. **Existing Packages** ...................................... 6
   - 4.1 Additional Character Representations ............. 6
   - 4.2 Advanced Numerics .................................. 6
   - 4.3 Data Collections ................................... 6

Concept Index ............................................... 7

Type Index ................................................. 8
Appendix B

Framestore Protocols

B.1 FramerD Framestore Access

B.1.1 pool-data

(pool-data <session key>) => (<base oid>, <pool size>, <read only>, <desc>)

pool-data queries the framestore for some basic data. The session key is necessary as some sessions may be read/write and others read only. Returned data includes the first oid in the pool (<base oid>), the total number of oids that have been allocated to this pool (<pool size>), the read/write status of the pool (<read only>) and a short, human readable text description of this pool (<desc>).

The session key may be any valid dtype and has no defined value (although the framestore may choose to perform differently for different session keys). This data can be safely cached, as it is assumed it will not change in the lifetime of a connection for a given session key.

B.1.2 get-load

(get-load) => <number of frames currently in the pool>

get-load returns the number of oids which have been allocated in the pool (via new-oid or frame-create). These oids are allocated sequentially.

B.1.3 new-oid

(new-oid) => The next oid value.
new-oid allocates an empty frame in the next available oid and returns the oid value of that empty frame. On a read-only pool, or when the pool is full, a DTerror will be returned.

B.1.4 lock-oid

(lock-oid <oid> <session key>) => <the frame at that oid>

Allows a client to “grab” a frame. No other client may lock it or write to it until it is unlocked (via either unlock-oid or clear-lock).

B.1.5 unlock-oid

(unlock-oid <oid> <session key> <new value>) => <success>

Sets a frame to a new value, and releases the lock on it. The session key must be eq? to the one that performed the lock initially.

B.1.6 clear-lock

(clear-lock <oid> <session key>) => <success>

Removes the lock on an oid without altering the value. The session key must be eq? to the one that performed the lock initially.

B.1.7 oid-value

(oid-value <oid>) => <frame at that oid>

Simply reads the value of a given frame. Note: The system will return a value even if the oid is locked (i.e. this is a read-only operation).

B.2 ZWrap Framestore Additions

The ZWrap additions are designed to allow non-atomic access to the framestore. There is no need to do a “lock/unlock” with the entire frame sent across the network twice.

Each of these operations can, however, be considered to be atomic (i.e. they are functionally identical to a lock/operation/unlock).
B.2.1 frame-create

(frame-create) => <oid of new frame>

frame-create is functionally identical to new-oid and is included only for completeness.

B.2.2 fkeys

(fkeys <oid>) => {<key1>, <key2>, ...}

Returns the set of all keys defined in the frame.

B.2.3 fget

(fget <oid> <key>) => {<val1>, <val2>, ...} OR <val>

fget operated differently depending on the number of items on the terminal in the frame. If there is exactly one, this value is returned. If there are zero or more than one, the set of the values is returned. **Note:** for the purpose of fget all keys are defined. Most of them just have no values.

B.2.4 fadd

(fadd <oid> <key> <val>) => <success>

Adds the requested value to the key for the oid. Multiple calls with the same oid and key may create a set valued terminal.

B.2.5 fzap

(fzap <oid> <key> <val>) => <success>

Removes a particular key/value pair from a given frame.

B.2.6 fdrop

(fdrop <oid> <key>) => <success>

Removes all values associated with the key from the frame.
B.2.7 fset

(fset <oid> <key> <val>) => <success>

Is equivalent to:

(begin
  (fdrop oid key)
  (fadd oid key val))

B.2.8 ftest

(ftest <oid> <key> <val>) => #t/#f

Returns true if the noted key/value pair is defined on the frame, and false otherwise.

B.2.9 fsearch (optional)

(fsearch <key> <val>) => {<oid1>, <oid2>, ...}

For framestores that support fast internal searching, this function returns a set of all oids that contain the key/value pair specified. Not all servers need implement this feature.

B.2.10 ifget (optional)

(ifget <oid> <key>) => {<val1>, <val2>, ...} OR <val>

ifget is functionally equivalent to fget with one exception. If the result of an fget on the frame is the null set, and the ISA key is set with an oid as its value, then an ifget is recursively performed on that frame.

This allows single parent isa inheritance handled on the server (hence the i in ifget). Not all servers need implement this feature.

B.3 FramerD Index Server

The index server is a generic mapping service. The illustrations here assume that it is being used as an index for a framestore.
B.3.1 iserver-get
(iserver-get (key . val)) => {<oid1>, <oid2>, ...}

Returns the oids which contain the given key/value pair.

B.3.2 iserver-add
(iserver-add (key . val) oid) => <success>

Adds the oid to the index as one that contains the given key and value.

B.3.3 iserver-get-size
(iserver-get-size (key . val)) => <number of oids>

Returns the number of frames currently indexed to this key.

B.3.4 iserver-writable
(iserver-writable) => #t/#f

Returns false unless the server can be modified (e.g. iserver-add will work)

B.3.5 iserver-keys
(iserver-keys) => {(<key1> . <val1>), (<key2> . <val2>), (<key3> . <val3>)...}

Returns all the key/value pairs in the index.

B.4 ZWrap Index Server

The ZWrap index server operates very differently from the FramerD server (see Section 8.6.1), it employs the same API, with the following addition:

B.4.1 iserver-ordered-get
(iserver-ordered-get (key . val)) => #(<oid1>, <oid2>, ...)

Returns the oids which contain the given key/value pair in sorted, ascending, uniqued order.
Bibliography


