University of Oslo
Department of Informatics

NewsView: A Recommender System for Usenet based on FAST Data Search

Mari Wang

Cand. Scient. Thesis

16th February 2004

Abstract

This thesis combines aspects from two approaches to information access, information filtering and information retrieval, in an effort to improve the signal to noise ratio in interfaces to conversational data. These two ideas are blended into one system by augmenting a search engine indexing Usenet messages with concepts and ideas from recommender systems theory. My aim is to achieve a situation where the overall result relevance is improved by exploiting the qualities of both approaches. Important issues in this context are obtaining ratings, evaluating relevance rankings and the application of useful user profiles.

An architecture called NewsView has been designed as part of the work on this thesis. NewsView describes a framework for interfaces to Usenet with information retrieval and information filtering concepts built into it, as well as extensive navigational possibilities within the data. My aim with this framework is to provide a testbed for user interface, information filtering and information retrieval issues, and, most importantly, combinations of the three.

Preface

This thesis is submitted to the Department of Informatics at the University of Oslo as part of a candidata scientiarum degree.

Acknowledgements

First of all, I would like to thank my supervisor, Knut Omang, for being available and helpful, and for helping me out whenever I stumbled on another technical obscurity.

I would also like to thank my coworkers at Abel, abelvaktene, for helping me out by standing in for me at work in the last couple of weeks, and for being a knowledge resource around the clock.

Those who have helped me by reading, commenting, and by enduring my endless detail obsessions, in particular Petter, Tone, Siri and Bente, deserve special mention.

Finally, I would like to mention those people out there who have spent their spare time creating the work environment within which this thesis has been created – thank you all.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Today, low signal to noise ratio frequently characterises the process of fulfilling information needs using computer interfaces. This thesis studies approaches to increasing this ratio, particularly in the context of conversational data.

The two traditional approaches to information access, information retrieval and information filtering, are central to the work presented. The major idea of this work is to attempt to combine ideas from these two approaches into one system. This is done by augmenting a search engine indexing Usenet messages with concepts and ideas from recommender systems theory.

The choice of Usenet as data domain was made based on availability, and the special filtering possibilities enabled by conversational media. Conversational data contain inherent information about social structure and interaction patterns. By automatically obtaining, processing and extracting predictions from such meta-information, relevance ratings can be improved.

Thesis Objective

The objective of my thesis is to look at possible ways in which information filtering and information retrieval can be combined. This is done in context of recommender systems for Usenet built using a search engine.

My aim is to obtain a higher signal to noise ratio, through better result relevance prediction and extensive navigational possibilities within the data.

The Work with this Thesis

Several large, fairly complex subject areas are treated in this thesis, including information retrieval and information filtering as well as FAST Data Search (FDS). I have invested a substantial part of the thesis work in becoming familiar with these.

I was given the opportunity to contribute substantially to the design and directions of my thesis work. This led me to considerable research into for me uncharted territory, a process which I have very much enjoyed. I have benefited from a large number of digressions – of which far from all have made it into the final text. This roundabout route to the finished product has, however, been time consuming.

The main challenge in the work with this thesis has been twofold: I had to acquire an understanding of several large research fields and combine their core principles whilst keeping an adequate level of detail. FDS is a large, complex application. Getting my bearings within the framework it represents, and putting together a system around it using many small building blocks, has been demanding.

When building the system, I used several features of FDS which were under development at the time. As a consequence, I have used early releases of the software, with the expected potential for quirks.

For various reasons, the implementation was put in concrete terms very late in the process. As a result, ideas from recommender systems theory in particular have not been incorporated into the NewsView architecture to the desired degree. The architecture does not, however, limit the use or addition of such features.

1.1 Thesis Outline

The thesis has two main parts. Chapters concerned with theory and background can be read independently of each other and the rest of the thesis. The design and implementation chapters, however, build directly on the foundation provided by the theoretical background chapters.

Theory and Background

Usenet is the data domain on which the practical work of this thesis is based. Chapter 2 presents Usenet, a world-wide distributed discussion system carrying conversations on almost any conceivable topic.

In my thesis, Recommender systems and search engines represent the two traditional information access approaches. Recommenders systems, presented in chapter 3, exemplify information filtering systems. Search engines are commonly used applications of information retrieval, and are presented in chapter 4.

When presenting my theoretical foundation, I have focused on the information access perspective. Concepts not necessarily relevant to the architecture described have been treated nonetheless because of their importance in this context.

Design and Implementation

Chapters 5 and 6 present the design and implementation of NewsView, the architecture created as part of the work on this thesis. NewsView attempts to combine information filtering and information retrieval concepts into one interface. Results are presented by means of an interface with extensive navigational features. The focus of this work is not to present a finished interface, but to provide a testbed for user interface, information filtering and information retrieval issues, in combination or separately.

Chapter 2

Usenet

This chapter gives a broad introduction to Usenet, covering technical aspects as well as some issues related to the community emerging from it. First, an overview of what Usenet is, where it comes from, and where it is today is presented. Next, section 2.1 contains a description of the workings of Usenet, from the user perspective as well as a more technical angle. Section 2.2 gives a brief comparison of Usenet and mailing lists, whereas section 2.3 gives an introduction to Usenet community dynamics and interaction characteristics. Finally, section 2.4 presents an overview of different interfaces to Usenet. This includes applications of recent research on social aspects and dynamics of conversational media to improve user interfaces.

What is Usenet?

Usenet is a world-wide distributed discussion system. It consists of a number of hierarchies of newsgroups classified by topic. Articles can be posted to these newsgroups by anyone who has appropriate software, and are distributed to other interconnected computer systems.

Usenet is like conventional news media such as newspapers and magazines in that information is distributed to a large audience. The big difference between these traditional media and Usenet is that the readers can choose to participate in discussions on equal terms with the author. Combined, the various hierarchies carry groups covering almost any conceivable topic.

The *alt.culture.usenet* FAQ [72] presents the following alternative definitions of Usenet:

1. Usenet is the newsgroups

2. Usenet is the loosely coupled network of computers that transmit Usenet messages via various protocols

3. Usenet is the community of people who read Usenet messages

4. All of the above

For my purposes, definition number four, with a further specification as to the first point, is the most fitting. The term Usenet is employed in its broadest sense, including all groups and hierarchies.

The History of Usenet

By the end of the 1970's, ARPANET, the precursor to the Internet, was a very limited community. Joining it involved a fairly large investment in facilities and equipment, and even if you had the required resources available you might not be eligible to join. As such, it was a rather exclusive club.

In 1979, some students of Duke University in North Carolina, US, decided to explore the possibilities of creating a network using cheap modem lines. Version 7 of UNIX had just appeared, and bundled with it came a software package named UUCP ('Unix to Unix CoPy'). Among the features of this software package were file transfer between hosts and remote command execution. Despite incomplete documentation, the ability to connect hosts using cheap telephone modems was appealing and led to the creation of a program called *A News*, a distributed bulletin-board system.

The idea of message exchange was by no means new at this time. The ARPANET had carried mailing lists for a while, and these can be said to be the philosophical predecessors of Usenet. What was new and revolutionary about the Usenet approach, was the idea of a public area from which everyone could read articles instead of sending a separate copy to every participant (Spencer and Lawrence [77]).

The four students being most deeply involved in the project, Steve Bellovin, Stephen Daniel, Jim Ellis and Tom Truscott [65] believed the traffic potential of their new intention to be rather small. Their initial estimate was a few messages a day.

The number of articles posted and read grew steadily, and after Jim Ellis held a presentation of the system at the winter 1980 Usenix Conference[1] gained momentum for real. The network grew rapidly, as there were lots

[1] I have been unable to find a specific reference to this presentation, but it is mentioned in [77]

of people without access to ARPANET and modem technology was relatively cheap. Usenet got connected to the ARPANET fairly early, initially through a mailing list to Usenet gateway. Later on, when the ARPANET was evolving into the Internet, it was used as a transmission route for Usenet [77].

Usenet Today

As of today, Usenet is available to almost anyone with access to a networked computer, although possibly through a fee. Access to the Internet is neither necessary nor sufficient to access Usenet, and the set of Usenet newsgroups available to users differs as no server carries all hierarchies and groups. Many companies and organisations use local hierarchies for news exchange and discussions of importance to their activities. These hierarchies are kept on local servers, and are often unavailable to the public. Hundreds of millions of messages are posted each month and tens of thousands of servers are participating in the propagation of messages.

2.1 The Usenet Model

2.1.1 The User Perspective

Usenet users employ client programs to read, compose and reply to messages. Sometimes an article is composed from scratch, and sometimes it is a *followup* to another message. When a message is completed, it is *posted* to one or more newsgroups. If a message is posted to more than one newsgroup, it is said to be *cross-posted*.

Some newsgroups are *moderated*. In this case, a *moderator* – usually one or more persons, but sometimes automated software – decides which messages are actually posted to a newsgroup.

There is a vast variety of hierarchies available, some regional, some organisation-specific, some general. A specific subset of these have a special position, and they are known as 'The Big Eight'. These have been around for a long time, and are considered the 'official' hierarchies by many. A hierarchy external and partly parallel to these with a long history of its own, is the *alt* hierarchy. Where The hierarchies belonging to the Big Eight have a fairly rigid procedure for newsgroup creation, this process is essentially unregulated in *alt*. As a consequence of this, the latter carries groups with discussions concerning moral and ethical issues of a controversial nature in addition to less controversial subjects.

Figure 2.1 shows a schematic overview of the Usenet newsgroup hierarchy, and contains explanations of some of the key concepts (the numbers correspond to labels in the figure) :

1. Usenet has a hierarchical naming system. There are more than 150 Top level *hierarchies*.
2. Newsgroup names are hierarchically built by a series of increasingly specific terms.
3. Each group contains a number of *posts* or *articles* related to the subject of the newsgroup.
4. Posts can be *cross-posted* to multiple newsgroups.
5. Any post may be responded to, making chains of messages called *threads*.

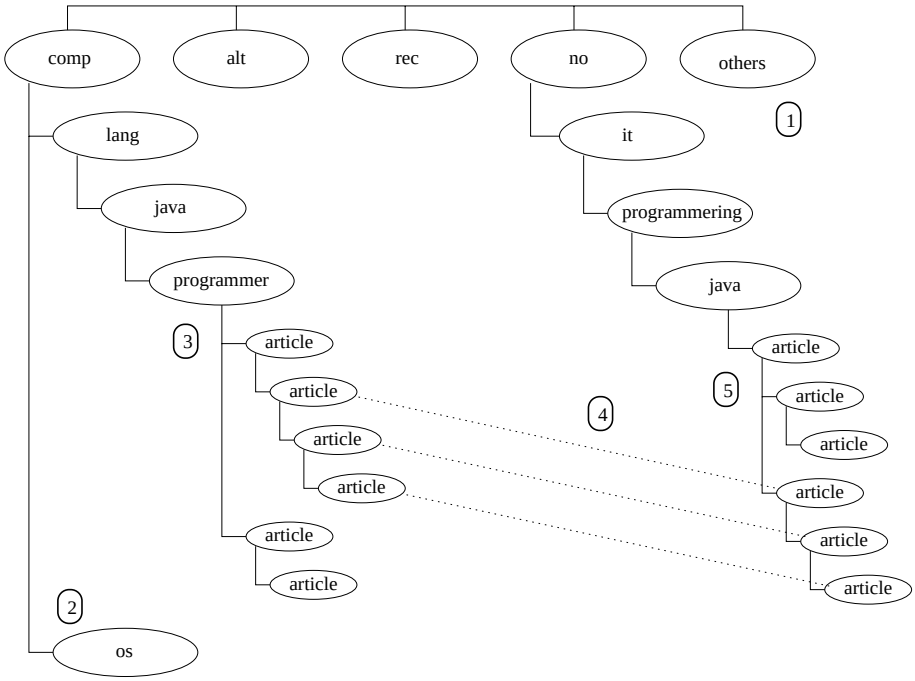


Figure 2.1: Graphical representation of the Usenet hierarchy structure. Figure layout is adapted from Smith [75]

Table 2.1 gives an overview of the mentioned hierarchies along with a short description of each:

	Name	Topics Covered
The Big Eight	Comp	Computers, computer science, and the computer industry
	Humanities	Literature and the humanities
	Misc	Everything not fitting in somewhere else
	News	Discussion of Usenet itself
	Rec	Recreational activities, including sports and music
	Sci	The physical sciences and engineering
	Soc	Socialising and social issues
	Talk	Discussions, debates and the like
	Alt	Alternative tree covering virtually everything

Table 2.1: Major Usenet hierarchies

2.1.2 The System Perspective

The three most important technical aspects of Usenet are the message format, the message distribution algorithm and the message storage mechanism. Each of these aspects is treated in the following.

Message Format

The format of Usenet messages is defined by RFC 1036 [36]. It is based on the format for Internet mail messages as defined by RFC 822 [14], but is more restrictive.

A standard Usenet message consists of some header fields, followed by a blank line, followed by the body of the message. A header field is defined to consist of a keyword, a colon, a blank and some more text. A sample message with full headers is shown in figure 2.2.

Some headers are required, and some are optional but should be recognised. All unrecognised headers of valid format are allowed, and will be passed along untouched. Table 2.2 gives an overview of required and optional headers as defined by RFC 1036 [36].

There are two types of messages. There are normal Usenet messages posted to one or more groups by any user, and there are special control messages used to perform creation and deletion of groups, cancelling of messages and other similar administrative actions. These messages can be sent by anyone, but there are usually some control mechanisms to regulate who can do what to which messages, groups and hierarchies. Control messages do not show up in regular newsgroups, but are dealt with either automatically by server software or manually by Usenet

```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Keywords: 386, preferences
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki
Lines: 20

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and
professional like gnu) for 386(486) AT clones.  This has been brewing
since april, and is starting to get ready.  I'd like any feedback on
things people like/dislike in minix, as my OS resembles it somewhat
(same physical layout of the file-system (due to practical reasons)
among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work.
This implies that I'll get something practical within a few months, and
I'd like to know what features most people would want.  Any suggestions
are welcome, but I won't promise I'll implement them :-)

                Linus (torvalds@kruuna.helsinki.fi)

PS.  Yes - it's free of any minix code, and it has a multi-threaded fs.
It is NOT protable (uses 386 task switching etc), and it probably never
will support anything other than AT-harddisks, as that's all I have :-(.
```

Figure 2.2: Sample Usenet message

administrators. The manner in which they are treated depends on the policy of the hierarchy the article is posted to and the local Usenet server policy.

The body of a Usenet message can contain anything the body of an email message can contain, for example uuencoded content and MIME encoded attachments. The parts of MIME, or Multipurpose Internet Mail Extensions, relevant to this thesis are described in RFCs 2045 [26], 2046 [27] and 2047 [60]. Some newsgroups, such as those of the alt.binaries hierarchy, are dedicated to binary content.

Message Distribution

The propagation algorithm as described below corresponds to the one given in RFC1036 [36].

Usenet is organised as a distributed network with no central control. Its network topology can be viewed as a directed graph, where each node in the graph is a host computer, and each edge is a transmission path to another host. Each edge can be thought of as labelled with a collection

Required Header Fields	
Header	**Description**
From	Contains the email address and optionally the name of the sender
Date	The date the message was originally posted to the network
Newsgroups	Specifies the newsgroup(s) the message is posted to
Subject	The title of the message, what it is about
Message-ID	A unique identifier for each message
Path	The path the message has taken through the network

Optional Header Fields	
Header	**Description**
Reply-To	Mailed replies to the message should go to this address
Sender	Alleged sender if different from 'From' address
Followup-To	Newsgroup(s) to which follow-ups should be sent
Expires	Suggested expiration date
References	Message-IDs of any messages preceding this message in a thread
Control	The message is a control message
Distribution	Limit distribution area of message
Organization	Name of senders organisation
Keywords	Keywords to describe the message
Summary	Brief summary of the message
Approved	For moderated newsgroups, contains address of moderator
Lines	Number of lines in message body
Xref	Information for local system regarding storage

Table 2.2: Usenet headers as defined by RFC 1036 [36]

of newsgroups which are forwarded along that link. Many edges are bidirectional, meaning the nodes in question forward the same collection of newsgroups to each other. Further, Usenet can be viewed as consisting of many subnetworks, where each subnetwork has a name (such as *comp* or *net*) and represents some specific subset of the hierarchies. Each of these subnets is a connected graph.

When a message is posted on one machine, it is addressed to a list of newsgroups. The local machine accepts the message, and forwards it to all neighbours having expressed an interest in at least one of the groups it has been posted to. As each of the neighbours receive the message,

they examine it to make sure they want it, accept it locally and then repeat the forwarding process for all its own neighbours. This process continues until the message has been seen by all nodes in the network.

An important part of this algorithm not covered so far is the prevention of loops in the graph. There are several solutions to this. One solution is for each node to keep a history of the messages it has seen, and throw away any incoming message it is already acquainted with. This solution prevents loops, but is not very optimised. A better version of this algorithm looks at the path header of a message before forwarding it, thereby determining whether it has been seen by the remote host. If so, that host is removed from the forward list for this message.

Currently, the most widely used method of Usenet transport is the NNTP protocol as defined in RFC977 [44]. NNTP is a protocol dealing with distribution, inquiry, retrieval, and posting of Usenet articles. It implies the use of a reliable stream client-server model, provided by for instance TCP (Transmission Control Protocol). Each server stores Usenet articles in a central database, and indexing, cross-referencing, and expiration of aged messages is provided. Conceptually, the NNTP propagation algorithm works much like the generic method described earlier, but there are some important differences. First of all, NNTP lets the two communicating servers negotiate which exact set of the available articles should be transmitted, thereby avoiding loops in the distribution scheme and saving bandwidth and processing time. Secondly, NNTP does not rely on flooding to spread articles. It defines two main ways of communication between servers: *push* and *poll*. While *push* works in much the same way as the traditional algorithm described above, *poll* differs by holding the receiving host responsible for initiating contact between servers.

A simplified view of the technical organisation of Usenet is given in figure 2.3.

Message Storage

The design of Usenet implies the notion of each server having a central database with messages. As there is no central control on Usenet, there is no such thing as a master server or database.

Each server provides Usenet access to a group of people. Some restrict their access, for example to people within a given organisation or paid customers, and some give access to anyone. The size of the user population for a given Usenet server can range from a small private business to a university or a large commercial service to which other organisations can outsource their Usenet needs.

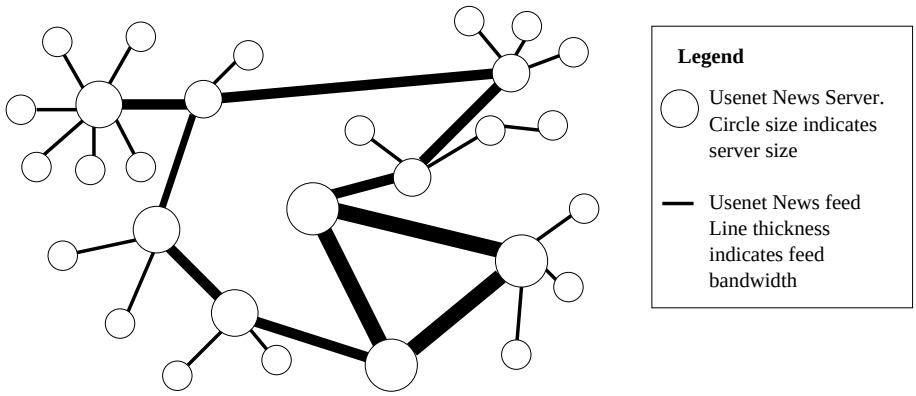


Figure 2.3: Simplified schematic view of Usenet technical organisation

The usual model of Usenet user access is by the employment of a server and client model. The user's chosen client software is configured to connect to the appropriate Usenet server, and the user may then request the articles he or she wishes to see and read them locally. The articles are, however, normally not stored locally – except possibly for some form of local caching.

2.2 Mailing Lists Versus Usenet

Usenet has its philosophical predecessors in ARPANET mailing lists created in the 1970s. Mailing lists are lists of subscribers email addresses and remailing sublists. Each such list includes all the intended recipients. A mailing list operates by remailing a copy of each message sent to the list to each subscriber. Some mailing lists are rather small and concerned with specialised topics, while others have a broad focus and contain discussions of more general issues.

Mailing lists have inherent design problems which become very obvious once they grow beyond a few dozen people. Sending a separate copy of the list submissions to each of the subscribers occupies large quantities of network bandwidth, CPU resources, and significant amounts of disk storage at the destination host. There are ways of reducing some of these problems, for instance by introducing separate remailers and message digests. Some more recent attempts at scaling mechanisms have been made, for instance by Spertus, Jeffries and Sie [78], but none of these attempted solutions address the real problem – private email from one person to another is a rather different form of communication than group discussion [44].

Maintaining the list itself presents a problem as well, as subscribers join, leave, change jobs and consequently email addresses and so on [44]. The management of subscribers is eased by recent systems for list administration which among other things allow subscribers to manage their own subscription preferences. These systems also typically have functionality for automatically expelling invalid subscriber addresses from the list.

A significant reduction of the amount of resources used can be obtained if articles are stored in a central database on the receiving host instead of in each subscribers mailbox. As described in section 2.1.2, this organisation is inherent in the Usenet system.

In addition to the obvious decrease in system resource usage, the Usenet interface to group discussions present several other advantages over mailing lists. Threading is a required part of the Usenet message exchange, and most Usenet client software supports threading. This feature means discussions are grouped, and eases the sifting of which discussions the user is interested in. If a thread is deemed uninteresting, the entire thread can be deleted at once, most often including any later correspondence in that thread. There is no administrative overhead if a user reads a particular group only periodically, or changes his or her address. Messages are automatically expired after a while, meaning users do not have to scan through lots of old messages if they have been absent for a while. Archives such as Google groups [31] are available if a user wants access to a longer backlog than provided by reachable Usenet servers.

2.3 Usenet Community Structure and Dynamics

> *The Internet is filled with junk and jerks. It is commonplace for inhabitants of the Internet to complain bitterly about the lack of cooperation, decorum, and useful information.*
>
> – Peter Kollock [47]

Usenet constitutes an huge amount of fairly unstructured information. In addition to the sheer amount of data, Usenet is an environment full of flames, rants, griping and other evidence of hostility, selfishness and simple nonsense (Figure 2.4). Despite many attempts at rules, group-specific FAQs, charters, 'words to live by', netiquette listings and the like, it continues to be a rather wild, unordered place. Consequently, the overall signal to noise ratio is low. Anarchy thrives, and any order is perceived to exist as a product of a delicate balance between individual freedom and collective good. Given a relatively anonymous interaction form, with no central authority or easily available monetary or physical

sanctions, it may very well seem surprising how many Usenet groups are well organised and productive [48, 47].

Figure 2.4: UserFriendly [25] cartoon strip of 29th of April 2003

The task of locating those groups is a daunting one, and conventional Usenet user interfaces do not scale very well in this respect. Displaying the group name, description and hierarchy position only conveys part of what goes on in a group. Other important metrics, such as the audience participating, size, knowledge level about the particular subject area and the tone of conversation may only be discovered through following the group for some length of time.

It is clear that having a way of distinguishing at least some such characteristics of a given group instantly upon viewing it for the first time would save users time and effort, and make the task of making oneself acquainted with the medium easier. The possibilities opened by such information in areas like filtering and user interface construction are obvious and vast. The question remains whether these metrics can be automatically extracted or have to be manually created and maintained. Some research on these issues is presented in section 3.2.2 as part of the treatment of 'social filtering'. For a thorough treatment of issues related to online communities, see Smith and Kollock [49].

2.4 Current Interfaces to Usenet

There are many ways in which Usenet can be accessed. The traditional Usenet interface, the text-based client program allowing a user to read and post Usenet messages subject to basic filtering, is only one of several alternatives. As a consequence of varying usage patterns, a number of interfaces meeting different needs have emerged. In the following,

a selection of these will be presented: traditional Usenet clients, web-based archives, mailing list to Usenet gateways and filtering and alerting systems. At the end of the section, recent research into user interfaces to conversational data is presented in some detail.

2.4.1 The Traditional Text-based Usenet Client

The focus of traditional text-based Usenet clients is on the display of the information received from a Usenet server, not on any particular processing of the data. The display normally includes a list of subscribed groups, and possibilities for viewing specific groups and messages. Most clients include threading functionality, and are exclusively geared towards browsing the available information. Figure 2.5 shows KNode, a sample client.

Basic filtering is normally available by means of *kill files*. Kill files enable specific posters and threads to be 'killed' based on some pattern, which usually means they are removed from the data presented visually to the user. In addition to this, some clients support *scoring*. Scoring works by matching authors against a list of scoring information created manually, and adjusting the display of postings according to the directions given in this list. The basic filtering functionality provided by these mechanisms is geared towards aiding users browsing groups already familiar to them. A user looking for new groups or topics to explore is left to a basic hierarchy browser.

2.4.2 Web-based Archives

Google groups [31] is by far the largest and most complete archive of Usenet messages available today. It offers a web-based interface suitable for browsing, searching and posting, and its archive includes more than 800 million Usenet messages, covering 20 years of human conversation.

Netscan [74] is a database storing large amounts of statistics about Usenet – number and size of postings, statistics per user, group and hierarchy and so on. The database is created primarily for research purposes, focusing on utilising social structures in Usenet user interfaces. Netscan and related research is described in more detail in section 2.4.5.

Smith and Fiore [76] argue that web-based archives with a basic search interface make users less likely to take part in the community, as their relationship with Usenet becomes one of retrieving specific pieces of information instead of one of a social connection. They argue that, when context is sparse, the likelihood of users taking part in the community is reduced.

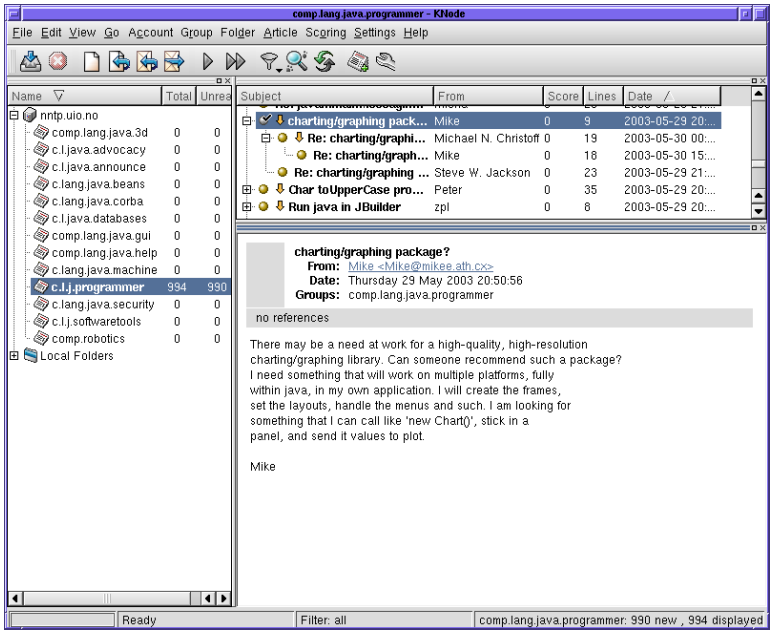


Figure 2.5: Sample text based Usenet client (KNode)

2.4.3 Mailing List to Usenet Gateways

A number of mailing list to Usenet gateways have emerged, both commercial, of which Newsadmin [55] is an example, and freely available, like gmane [38]. They offer similar services, but with some differences – for instance, gmane inserts new groups into its own separate hierarchy, *gmane*, while newsadmin inserts new newsgroups into the *alt* hierarchy. Both offer a bidirectional service (postings to the Usenet groups are forwarded to the mailing lists as well as the other way around), depending on the setup of the individual mailing lists in question. Gmane has additional features, providing among other things spam detection, cross-post handling, an encryption and forwarding service driven by the Tagged Message Delivery Agent (TMDA [18]) , a web interface and a real-time indexing search engine. It respects user requests to avoid archiving of messages (Employing the *X-No-Archive* header).

2.4.4 Triggers, Alerts and Filtering Services

Several trigger and alert services are available, some targeted exclusively at Usenet and some with a wider information base. NewsBin [61] is one such service, aimed at users of binary newsgroups. It provides an interface to binary newsgroups easing the process of downloading attach-

ments, among other things by enabling batch downloads, automatic decoding of messages and spam filtering. Other services, like binnews [7], provide monitoring functionality presenting users with overviews of binary content to reduce the amount of data eligible for download. It can provide edited as well as unedited newsgroup views.

Cyberalert [15] is a commercial monitoring service for news items posted to various sites and forums on the web, including Usenet. Users specify keywords or phrases, and receive daily reports of relevant matches.

2.4.5 User Interfaces to Conversational Data

This section presents a survey of recent developments in research directed towards user interfaces to conversational data. The approaches included here are presented because of their potential use as user interfaces in a recommender systems context.

Some such interfaces are presented in this section. They approach the task from different angles and concentrate on different aspects of it. The common ground they all share is the wish to convey more information about the interactions on Usenet, than what is the case with traditional text-oriented interfaces. They wish to do this by creating new visual interfaces or augmenting existing ones. Illustrations are included to visualise these interfaces to the reader.

The first system to be presented, the *conversation map* [70], is rooted in a linguistic perspective. It uses threading structure and language to construct 'discourse diagrams' to convey the social and semantic networks present in Usenet. Loom [17] and its successor, Loom2, are presented next. Both projects focus on using visualisation techniques to uncover social patterns of Usenet for individual users and groups.

Smith and Fiore [76] have focused on portraying the social aspects of Usenet by visually expressing the quantitative qualities of the environment such as number of people and frequency of posting. All the systems mentioned so far present interfaces with little resemblance to a traditional text-based Usenet client. The *thread arcs* [46] project has chosen a different approach, developing a thread visualisation technique intended to be applied as part of a larger interface.

The Conversation Map

Sack [71,70] has developed a Usenet newsgroup browser analysing archived Usenet messages and outputs what he terms a *conversation map*. The conversation map uses social and semantic networks. The social

networks show who is replying to or citing whose messages, whereas semantic networks are used to highlight the most frequently used terms and some of their relationships with one another. In addition to this, the conversation map creates a list of the most frequently encountered discussion themes. The overall aim of this approach, is to enable a combination of social and semantic navigation.

Figure 2.6 shows a sample conversation map interface. The upper half of the interface shows three views which are, from left to right, social networks, discussion themes, and a semantic network. The bottom half of the interface shows a graphical representation of the newsgroups messages analysed by the system.

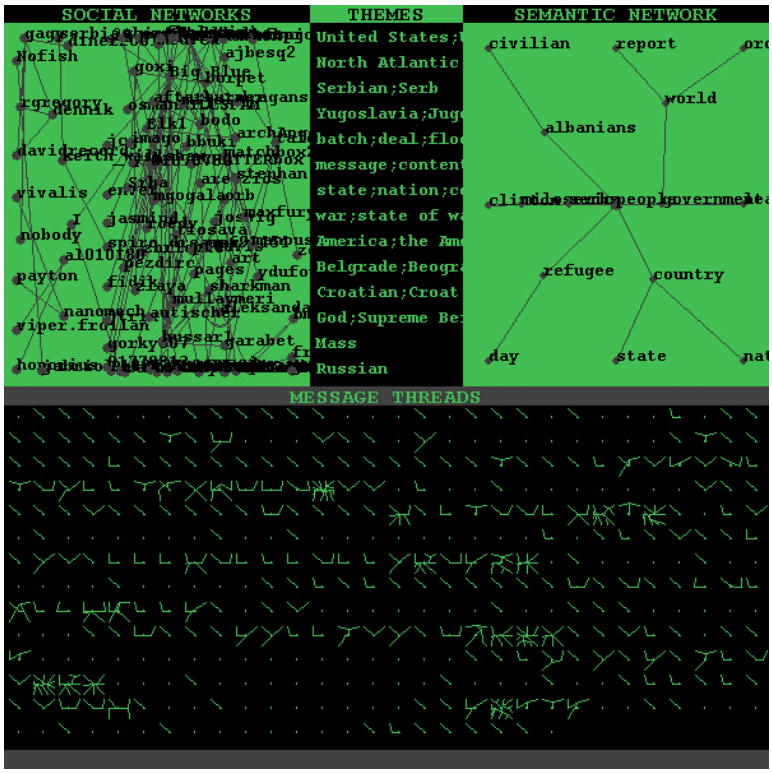


Figure 2.6: Example of conversation map interface

Loom and Loom2

The Loom project attempts to use visualisation techniques to uncover social patterns of Usenet [17]. Such patterns are typically difficult to obtain when using a textual interface unless a community is monitored over time. Work on the Loom2 project is grounded on observing text as

a communication medium – while text has a number of excellent qualities as a medium for exchanging ideas, it has severe limitations as a conversational medium. It is highly adaptable, and given a basic alphanumeric keyboard people can discuss almost any topic. It has, however, a limited capability when it comes to conveying many kinds of social information, such as conversational tone, patterns of activity and the size of the conversational group. The focus of the Loom project is to create representations highlighting social information and helping people make sense of the virtual social world. This approach is called *Social Visualisation*, which is defined as the visualisation *of* social information *for* social purposes.

Loom visualises threaded discussions. It aims at revealing patterns indicative of a person's role in the community, and the type of discussion prevalent in a particular group. It works by displaying both the information at hand without processing it further, like date, author, threading context and so on, and by categorising messages and employing this derived information to visualise other aspects (for example moods) characterising interaction.

Figure 2.7 shows an example of a group interface as presented by Loom. Users are depicted along the y axis, and time along the x axis. Each coloured line represents a thread of conversation, and the interface can be used to display the actual text of a Usenet message as shown.

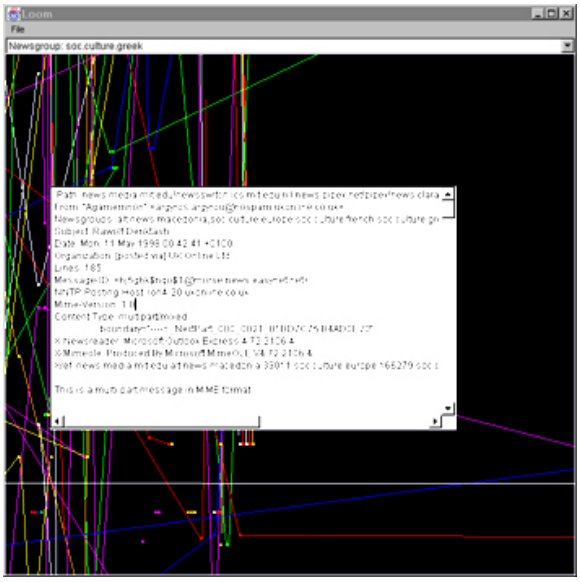


Figure 2.7: Sample Loom visualisation

The Loom2 project [8, 16] builds on Loom. It has the same basic ap-

proach to the visualisation process, that is, a goal of visualising the underlying social patterns of the Usenet communities.

The project identifies two fundamental questions – what to visualise and how to visualise it. The answers to the first question is assumed to convey the social characteristics of individuals, conversations, threads and newsgroups. The emphasis is on reflecting fundamental differences between groups instead of detailed representations of each numerical attribute. The Loom2 project has gone through several design iterations, for a detailed survey see [8].

Figure 2.8 shows a sample visualisation from Loom2, depicting the group *rec.motorcycles*. Related posts are placed in a circle. The pattern of densely populated circles implies an active conversational environment.

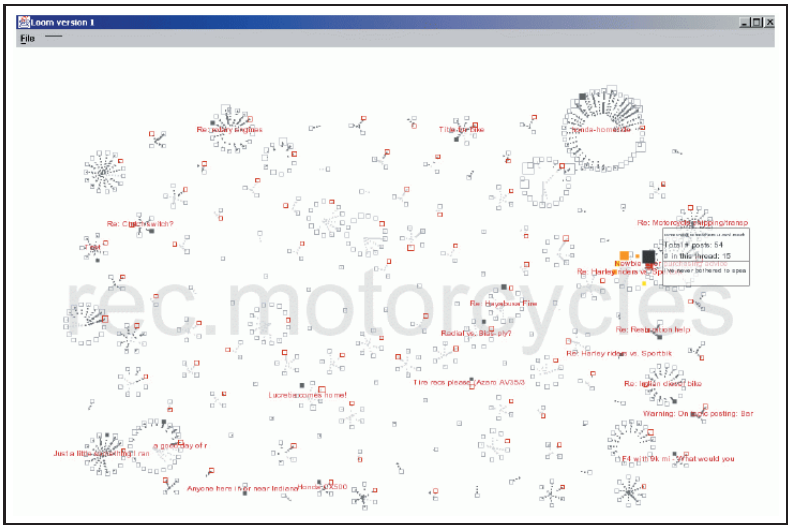


Figure 2.8: Sample Loom2 visualisation

Netscan and Tree Maps

Work on the Netscan project [74] has presented several different visualisations of social aspects of Usenet. Smith and Fiore [76] present some of these, for instance *thread trees*, the *piano roll display*, the *interpersonal connections display* and *tree maps*.

Their motivation is similar to that of the Conversation Map and the Loom projects. They build on the previously presented concept of 'social visualisation' [17], and present an interactive web based interface to Usenet composed of several visualisation components.

An almost unlimited number of facets of Usenet interactions can be visualised at various scales. The focus of Smith et. al. [76] is to visualise the chosen data in a way maximising the connection between the visualisations and the actual process of reading and contributing to newsgroups.

The authors argue that an appropriately designed interface to Usenet newsgroups could reinforce socially beneficial behaviour. They also point to how such an interface could emphasise interpersonal relationships, and as such reveal the roles and importance of individuals. This leads them to the following vision of a Usenet interface:

> *An ideal interface that would allow the user to view simultaneously the set of messages in a given time span or newsgroup, the corresponding set of people, and the interactions within and between those sets.*
>
> – Smith and Fiore [76]

Figure 2.9 shows a sample tree map [76, 22], which is a visualisation attempting to demonstrate the hierarchical structure of Usenet. The relative box sizes are based on the number of posts in a given newsgroup for a month. Colour variations indicate change in the number of messages posted compared to the month before – green indicates more posted messages, red indicates less postings. The colour intensity indicates the rate of change – the higher the intensity, the larger the change.

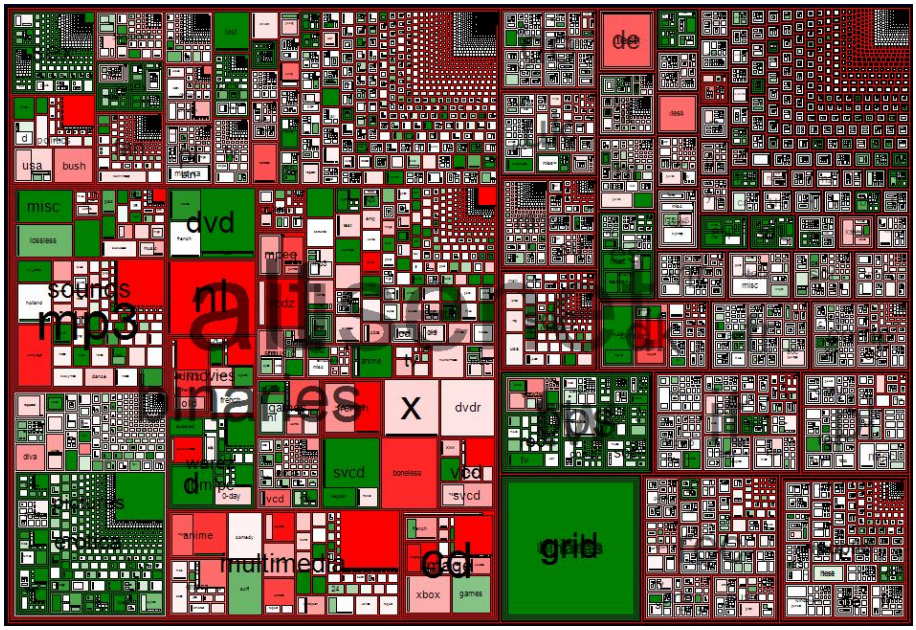


Figure 2.9: Sample tree map [63]

Thread Arcs

Thread arcs [46] is an interactive visualisation technique designed to help people use threads in email. The approach chosen is to combine the chronology of messages with the branching tree structure of a conversational thread. The approach is meant for threads of a limited size only, but many Usenet threads are small enough to benefit from the methods as described. Approaches involving sub-thread collapsing is a possible way to apply thread arcs to larger threads.

A sample application of thread arcs can be seen in figure 2.10, which is taken from Kerr [46]. The figure shows four ways to visualise the same thread. 2.10.P shows the thread highlighting one users postings, 2.10.T shows the effect of shading the thread according to time, 2.10.C colours the thread according to the different contributors and 2.10.G highlights the generations of postings in the thread.

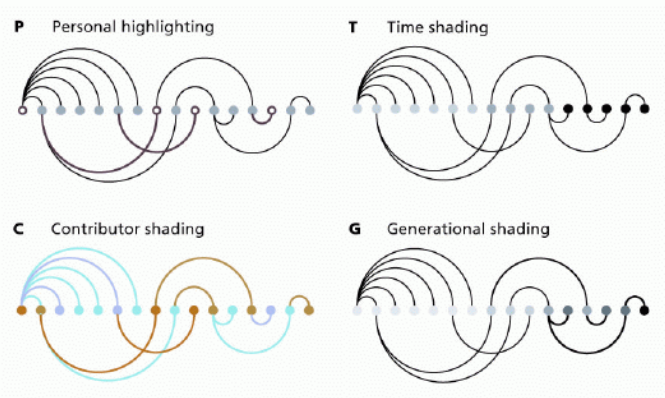


Figure 2.10: Message attribute highlighting schemes using thread arcs

Summary

This chapter has outlined Usenet from a user and systems perspective. Further, it has presented the use of social patterns as an aid for Usenet visualisation techniques.

Chapter 3

Recommender Systems

Recommender systems are proposed as an electronic equivalent to aids employed when looking for information. Such aids are word of mouth, recommendations, reviews, and surveys.

A related application of recommender systems utilised in this thesis, is to aid a user in finding interesting information in contexts where the signal to noise ratio is low. Usenet, which is the focal data domain of this thesis, is an example of such a context. The treatment of recommender systems given in this chapter is kept general as a rule, but includes some Usenet-specific details.

A typical recommender system consists of people providing recommendations, implicitly or explicitly, and a system aggregating these recommendations and somehow transferring them into information directed at appropriate recipients. The transformation can consist of merely the aggregation itself, or be a more complex operation on the data [67]. The choice of relevant information for a given user is made based on knowledge about the recipient and the data space. Some systems consider the user interface an integral part, whereas others depend on modification of preexisting interfaces or external ones. Figure 3.1 shows a graphical representation of the structure of a recommender system.

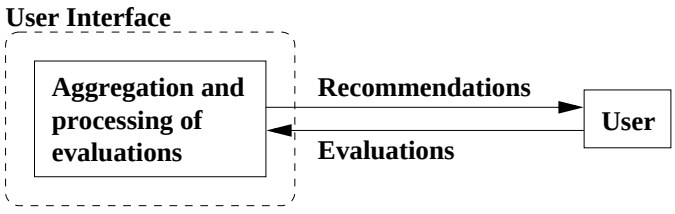


Figure 3.1: The general structure of a recommender system

This chapter attempts to place recommender systems in a larger research context, and to clarify connections to related fields. It continues with theoretical background of central concepts and ideas, and, finally, a presentation of some existing systems and approaches.

3.1 Recommender Systems in Context

The purpose of information access systems is to help users locating information relevant to them, avoiding the need to wade through large amounts of irrelevant data.

In a historical context, there have been two separate analytical approaches to information seeking, namely *information retrieval* and *information filtering*. These two approaches have been known by the common name of *information access* [28]. Figure 3.2 shows how the two approaches relate to each other.

The aim of information retrieval systems is to support users with changing interests (high information need change rate), assuming the data is fairly stable (low information source change rate), whereas information filtering systems assume quite stable user interests over time (low information need change rate), but support dynamic information sources (high information source change rate) [5]. Applications where the information source change rate and the information need change rate are both high, are the most difficult to handle, and are as such termed *the grand challenge* [62]. For an approach to taking on this challenge, see Baudisch [5].

As mentioned previously, information retrieval is similar to information filtering in several ways. Both approaches employ the same basic components, rely on the same information flows, and can be modelled using very similar architectures [6]. The difference lies in the time span of usage for a given user – requests to information retrieval systems are often made only once, and searched against the current document collections. Requests to information filtering systems, on the other hand, are often made repeatedly and against successive additions to the document collection over a period of time. As such, information filtering systems are concerned with repeated uses of the system by users with long-term goals or interests over time, whereas information retrieval systems primarily support single requests [5, 6].

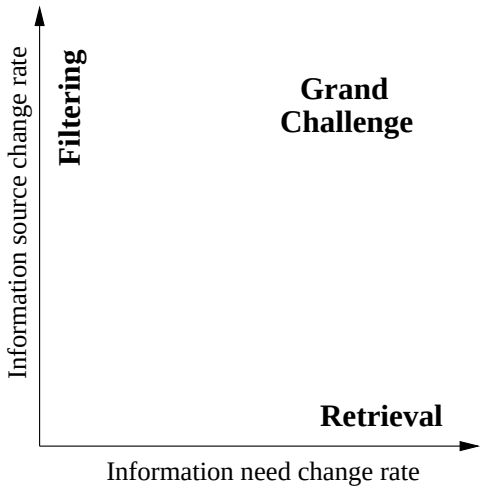


Figure 3.2: Information seeking processes, after [62]

3.2 Concepts and Theory

This section presents concepts important to recommender systems. Information needs and relevance are presented in section 3.2.1, followed by treatment of information filtering in section 3.2.2. Towards the end of this section, the theoretical framework is expanded to a point where an accurate definition of recommender systems can be given. Profile building and maintenance is treated in section 3.2.3, annotations and ratings are discussed in section 3.2.4, and section 3.2.5 treats some social implications of recommender systems. Privacy concerns and some general user interface and usability concerns are presented in sections 3.2.6 and 3.2.7, respectively.

3.2.1 Information Needs and Relevance

The purpose of an information filtering or information retrieval system is to satisfy the *information needs* or *interests* of its users. Several models defining information needs and their creation have been proposed.

Belkin and Croft [6] propose a model where a person with some goals and intentions related to a task finds the resources available to be somehow inadequate to meet those goals. This is termed an *anomalous state of knowledge (ASK)*. Mizzaro [28], on the other hand, considers four states of information needs. The initial phase, that is, the anomalous state of knowledge, is called *problem*. When the person becomes aware

of the problem, it turns into an *information need*. Expressing the information need turns it into a *request*, and formalising this request creates a *query*. These states are also referred to as *real, perceived, expressed and formalised information need* [59]. Users can, and usually will, have several distinct information needs at once.

Measuring the performance of an information access system is hard. It is inherently difficult to measure how well an information need has been satisfied by the documents chosen, and, because of this, performance is often measured by how relevant documents are judged to be by the user. *Relevance* is a very central concept in this context. For more information on this concept, see [59] for an exhaustive survey and [28] for a detailed framework of relevance notions.

The aim of information access systems is to predict the relevance of documents as the user would assign it, and deliver only those documents deemed relevant to the user. Initially, relevance predictions by information access systems were boolean. Many recent systems use gradual probabilistic models, acknowledging degrees of relevance [5]. Multidimensional relevance has been proposed [28], but as these systems do not provide a total ordering of the retrieved documents they can not produce one single, meaningfully ranked output.

The most common measure of information system performance is *precision and recall*.

Given a sample information request I and its set R of relevant documents, let $|R|$ be the number of documents in this set. Assuming a given information retrieval or filtering strategy processing request I, the result set A is generated. Let $|A|$ be the number of documents in A. Let $|Ra|$ be the number of documents in the intersection of the sets R and A, the recall and precision measures are defined as follows:

- Recall is defined as the number of relevant items delivered divided by the total number of relevant items, that is, the fraction of the relevant documents which has been retrieved.

$$Precision = \frac{|Ra|}{|A|} \tag{3.1}$$

- Precision is defined as the number of relevant items delivered divided by the total number of delivered items, that is, the fraction of the retrieved documents which is relevant.

$$Recall = \frac{|Ra|}{|R|} \tag{3.2}$$

Figure 3.3 illustrates these sets. The definition of recall and precision given here assumes all documents in the answer set A have been examined or seen, but this is not usually the case. This definition is, however sufficient for the purposes of this thesis. The figure and the definition are adapted from Baeza-Yates and Ribeiro-Neto [3, chapter 3], which has more in-depth information on these issues.

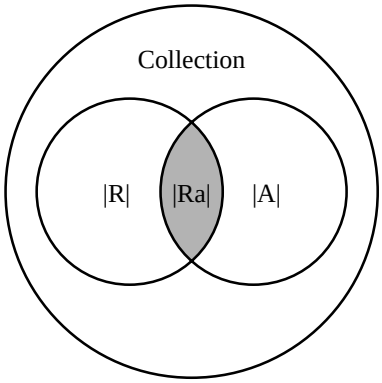


Figure 3.3: Venn diagram illustrating precision and recall

3.2.2 Information Filtering

Information filtering has been applied to many application areas.

Figure 3.4 shows the general architecture model of information filtering systems as proposed by Belkin and Croft [6]. The model consists of three blocks, consisting of four steps each. The three blocks are surrogate creation in the top left, profile creation in the top right and the filtering and refinement process at the bottom. The three smaller figures on the right show how profile maintenance fits into the model, specifically profile creation 3.4.a, interest changes 3.4.b and profile refinement 3.4.c.

To enable comparison of the surrogate and the profiles, both are usually reduced to a set of *attributes*. Attributes enable mapping of documents to a common representation by use of a distance measure, often real numbers. The following properties distinguish different approaches to information filtering:

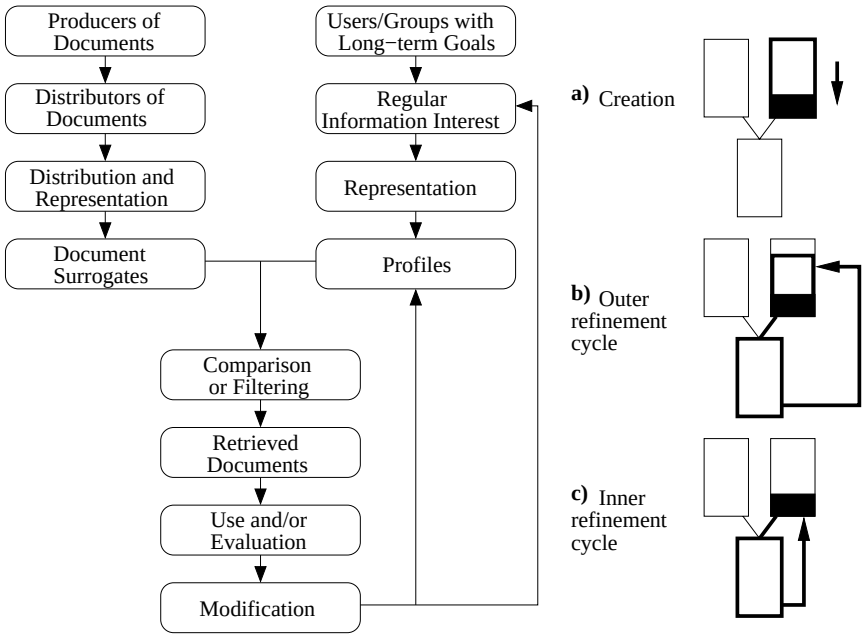


Figure 3.4: Information filtering model, after [6, 5]

1. Which attributes are used for the comparison of surrogates and profiles, and how is their similarity determined?

2. Who or what assigns the attributes to the documents (indexing)?

3. Who or what assigns the attributes to users (profile creation)?

Malone et. al. [56] presented three different approaches to automatic message filtering. Their initial proposition of cognitive, social and economic filtering has been reworked, and today the model normally consists of content-based (which is combination of cognitive and economic filtering) and social filtering.

Content-based Filtering

Content-based filtering is used to create characterisations of document contents and information profiles, which are used to intelligently match documents to recommendation receivers. Rudimentary examples of cognitive filtering include distribution lists and simple keyword matching. Content-based filtering has its roots within the information retrieval community, and utilises many techniques from this field [5].

The three most prominent retrieval models are the boolean model, the vector space model and probabilistic retrieval models [6]. The boolean model is based on the *exact match* principle, whereas the other two are based on a concept called *best match*.

The rest of this section presents these models, along with some motivation for the introduction of best match methods and a short description of query refinement though relevance feedback.

The boolean model

Boolean retrieval is based on the concept of an exact match of a query specification with one or more surrogate. The query specifications are expressed as words or phrases, and combined using standard boolean logic operators (hence the term 'Boolean' retrieval). In this retrieval model, all surrogates matching a query are retrieved and no distinction is made between any of the retrieved documents. Thus, the result of a query operation using Boolean retrieval is a partition of the available surrogates into a set of retrieved documents and a set of not-retrieved documents [6]. It is important to be aware of the distinction between using boolean queries and a boolean retrieval method – boolean queries can be and are used in combination with many different retrieval methods. A major problem with the Boolean retrieval model is its lack of relevance ranking within the retrieved document set. This leads to the conclusion that Boolean retrieval is too weak for large text collections [5].

As a response to the problems of exact match retrieval, best match methods have been proposed. These systems are built on the idea of presenting documents to a user in order of presumed relevance, and the belief that this will present more effective and usable systems. If the system has produces a good rank ordering, the density of relevant and useful documents should be greatest near the top if the list. The probability ranking principle states that if a retrieval systems' response to each request is a ranking of the documents in the order of decreasing probability of relevance, precision and recall will be maximised at any cut-off. Best-match methods give users the option of controlling the output size, and thereby improve their ability to manage large result sets [5].

The vector space model

The vector space model treats keyword representations of documents and queries as vectors in a multidimensional vector space, the dimensions of which are the keywords used. Queries and text

are compared by means of vector comparison, for instance using the cosine similarity measure. A major improvement of this model compared to exact match approaches, is the ability to weight keywords according to their presumed importance. These weights can be computed based on the statistical distribution of terms in the document base [5, 3].

Probabilistic models

Probabilistic information retrieval methods are based on the probability ranking principle. Documents are ranked according to their probability of relevance to the query, given all available evidence. Typical sources of such evidence are the statistical distribution of terms in the document base and other texts. The representations of both information needs and texts are uncertain, as well as the relevance relationship between them, and this is taken into consideration by probabilistic models [5, 3].

Query refinement through relevance feedback

Relevance feedback is a successful approach to simplification of query formulation. The basic idea of this approach is to let users give feedback on the relevance of examined documents, and let this feedback guide later document ranks. The actual method of computing the updated ranks depends on the underlying retrieval model, but common solutions include adding a subset of the terms found in documents judged relevant to the query, or boosting their term weights. Relevance feedback simplifies the query formulation process by partially replacing the task of formulating queries with the task of criticising suggestions made by the system [5].

Content-based filtering has several known weaknesses [73]. Not all items can be easily parsed in a way allowing automation of content-based filtering. Multimedia data such as sound, pictures and video are examples of this. There is no inherent facility for generating serendipitous results when using content-based filtering. The system will recommend more of what the user has already indicated a liking for, and the introduction of serendipity requires extra functionality. Content-based filtering systems lack means to describe the quality or style of an item. As an example, they are unable to distinguish a well written and a badly written article if they happen to use the same terms.

Social Filtering

A purely content-based approach to information filtering is limited by the process of content analysis. Social filtering, also known as collabor-

ative filtering, is an approach attempting to overcome this limitation by exploiting social processes.

In Malone et. al. [56], social filtering is defined as information filtering that works by supporting the personal and organisational relationships within a community. It does, in effect, rely on the users of the system knowing each other. This approach is known as *active social filtering* [5]. Hill and Terveen [34], on the other hand, claim social filtering works independently of whether the users know each other personally or not. Moreover, they propose that social filtering and personal relationships can be taken apart and put back together in interesting new ways. Social filtering performed independently of users knowing each other is referred to as *automated social filtering* [5].

Shardanand and Maes [73] define social filtering by observing how it essentially automates the process of 'word-of-mouth' recommendations. Items are recommended to a user based upon values assigned by other people with similar taste. This thesis will use the last and most general of these definitions.

In the context of Usenet, some studies of social interaction characteristics have been done [23, 88]. The main motivation behind these studies has been to investigate the interaction supported by Usenet, and to analyse the results to obtain metrics applicable in the context of predicting relevance. Whittaker et. al. [88] have analysed large amounts of Usenet messages gathered over 6 months. They provide descriptive data about newsgroup demographics, communication strategies and interactivity. In addition to this, they derive predictions from the *common ground* model of communication to test predictions about how these parameters interact.

Smith and Fiore [23] argue that systems requiring active participation of their users to assess other users may not be necessary to figure out which authors and messages are valuable. They have done extensive mining within stored Usenet messages to back this claim, as well as performing an experiment where experienced Usenet users were asked to rate authors based on some predefined metrics. As an example of their conclusions, they give the following characteristic of an author from whom users were likely to read more messages:

> *We interpret this portrait as one of a poster who participates actively and regularly in a variety of in-depth conversations, in which he or she responds to other participants but does not overwhelm the discussion or [...] participate in too many different newsgroups.*
>
> – Smith and Fiore [23]

None of the two studies mentioned have taken the content of messages into consideration, they have been concerned with behavioural metrics derived from group statistics alone. A future extension of this line of research would be to include techniques from natural language processing and information retrieval to provide additional descriptors of authors based on the material of their posts and their styles of writing.

This work is promising in the context of automated social filtering – that is, exploiting the inherent structure and social interaction of past Usenet postings to predict relevance of future messages by employing knowledge about the social structure of the domain.

Precise Definition of Recommender Systems

Recommender systems can now, for the purposes of this thesis, be defined as follows:

Definition. *Recommender System*

A recommender system is a filtering system employing social filtering, alone or along with other filtering techniques.

3.2.3 Profile Building and Maintenance

The user profile is an important aspect of a recommender system. Any such system requires a way to track a specific users' information needs to decide what content should be recommended to the user.

To describe these needs formally, a *relevance function* is employed. The relevance function is a map from a space of documents to the space of real-valued relevance factors. Denoting the space of documents D, the goal is to find a map $f : D \to \mathbb{R}$, such that the relevance function $f(x)$ corresponds to the relevance of a document x. Given such a map known for all points in D, a finite set of rank-ordered documents presented in a prioritised fashion can be guaranteed. As such, the challenge of information filtering systems can be described as that of representing a users' relevance function as a user profile, that is, a profile correctly representing the relevance of each document. Relevance functions of users are not known in advance, and often change with time [51].

One of the main challenges in the development of user profiles, is to find a good way of creating a profile when a new user comes to a system, also known as the 'cold start'-problem. How do you create an immediately useful new profile while minimising strain for the user? Even if a starter profile is present in the system, the filter will not work effectively during the training period. Several possible solutions have been proposed. One

possibility is giving new users access to previous users experiences to aid faster development of a profile reflecting the preferences of new users as accurately as possible.

A different approach to building user profiles is presented by Rashid et. al. [64]. Here, the problem is approached by observing the most immediate way to acquire the information needed – which is to directly present items to the user. This, however, confronts the system with the difficult decision of what items to present. The authors present six different strategies for choosing items, building on several different approaches to the problem. These include the use of information theory to select the items giving the most value to the recommender system, aggregate statistics to select the items the user is most likely to have an opinion about, balanced techniques seeking to maximise the expected number of bits learned per presented item, and personalised techniques predicting which items a user will have an opinion about. The article authors conclude that the choice of learning technique significantly affects the user experience. It affects the user effort required as well as the accuracy of the resulting predictions.

A related problem appearing over time in some systems, is lack of serendipity. Once a profile is established, the system will only let items fitting it through – this phenomenon may cut the user off from new material outside the scope of the existing profile. In many systems, circumventing this problem means a new user profile has to be created – which means we are facing the cold start problem yet again.

3.2.4 Annotations and Ratings

Recommender systems employing social filtering work by recording the reactions of users to documents. There are several available approaches to gathering these reactions, and thereby information about a document's relevance, classification and quality. There are two main approaches: Implicit and explicit ratings. Where explicit ratings are obtained by direct entry by the user, implicit ratings are derived from the users interactions with the system, or from processing of relevant data, for instance logs.

Explicit ratings can be submitted in the form descriptions, votes, or ratings on a given scale. They can be used for calculation of a score for each document, or as a more general evaluation to accompany the recommendation as it is displayed to its recipient. The information provided by the user in this way can be processed in various ways by the system to reveal usage patterns, general tendencies and other aspects which are not immediately available upon entry.

There are many varieties of implicit ratings [89]. One of the more popular ones is the concept of *computational wear* [35], also referred to as *use wear* and *use data wear*. The concept of computational wear builds on the observation that physical documents accumulate wear like stains, torn pages and so on as they are used. These physical characteristics may contain valuable information to a new user of the documents as to how frequently a document has been in use in the past, how it has been used, and also which parts of the document have been found more interesting by former readers.

Hill et. al. [35] present a special case of this approach, where they consider computational wear in the context of documents and the reading and editing of them. They propose to draw a parallel to wear on physical documents by defining the concepts *read wear* and *edit wear*, where various aspects of the electronic usage of documents are monitored and recorded. An example of a metric recordable this way, is total reading time. If more differentiated information is desirable, more fine-grained monitoring of the user's interaction with the document can be undertaken, to reveal for instance on which parts of the document more time is spent. In systems where it is feasible, patterns in the editing of documents can be monitored as well. This way, one can distinguish more stable parts from frequently changing ones, as well as determining who authored what.

The information gathered is then displayed graphically to the users to aid them in their work with documents. The method of graphical display chosen in [35] is called 'attribute-mapped scroll bars', and consists of displaying the wear as marks mapped onto document scroll bars in positions relative to line positions. The main advantages of this method is the ability to reuse existing screen space, and, more importantly, the information display is collocated with the navigation control points.

3.2.5 Social Implications of Recommender Systems

Recommender systems have some inherent characteristics necessitating explicit consideration of the human responses to such systems.

Most importantly, someone has to provide the recommendations. The 'free ride' problem, where users stop contributing once their profile of interests is established, is hard to circumvent. Effects of free riding are too few evaluations of each item, as well as making the evaluations potentially unrepresentative as they are provided by a subset of the user population – that is, those who enjoy the evaluation process. The 'free ride' problem cannot be solved by technical means alone, although implicit gathering of evaluations from existing sources and monitoring of

user behaviour can reduce it greatly.

Avery and Zeckhauser [2] argue that even though private bargaining could yield an optimal level and order of item evaluations, it would be unmanageable with many potential readers. They propose three centralised mechanisms able to improve the provision of evaluations.

Subscription services

This approach is based on the idea of paying customers who receive recommendations. The evaluations are provided by professional evaluators whose incentive to become good at evaluating items would be to maintain their reader base.

Transactions based compensation

The idea in this approach is to give payment in some form to those who provide early evaluations. This would yield a surplus to those evaluating the most messages, while those who evaluate the least would have to pay. An advantage of this approach is the avoidance of characterisation of users as they are added to the system – the evaluators and their skills are identified over time. This system is, however, more complicated than the subscription service system, and a precise pricing strategy may be necessary to guarantee an optimal provision of evaluations.

Exclusion

Using the threat of exclusion from the group receiving recommendations could induce evaluations. Each member of the group would be expected to provide a certain number of early evaluations to maintain membership. If the information produced is useful to the members of the group, they will be motivated to continue their contributions. The system provides incentives without an explicit payment strategy, but resources may be wasted by low quality evaluators.

They conclude the discussion by arguing that collaborative filtering systems may have a need for some centralised coordination ensuring the process produces sufficient and informative evaluations and thereby recommendations.

Another issue in need of consideration is the possibility that, if recommendations can be provided by anyone, content owners may generate enormous amounts of positive recommendations for their own items and negative recommendations for their competitors content [67]. Appropriate measures should be taken when implementing recommender systems to discourage the 'vote early and often' phenomena.

An important issue for any recommender system is that of reaching critical mass [57]. Users are much more likely to rate an article if they are getting predictions, and the more evidence they see of other people rating, the more likely they are to rate themselves [58]. Until you reach a sufficiently large number of active users of a system, it is very hard to say anything accurate about it's effect, and the users may never see enough recommendations of interesting articles to actually experience a positive effect. This problem can be eased by the inclusion of implicit ratings, but even though the threshold may be lowered, it will still be present.

3.2.6 Privacy Concerns

Recommender systems raise concerns about the privacy of their users. Recommendations will generally be more useful the more information the receiver has about them, as it enables better evaluation of the information provided – however, people may not want all information possibly relevant to their evaluations widely known. There are three ways to handle user privacy issues in a collaborative filtering system. The users may be either anonymous, they may be known to all other users, or they may use pseudonyms [58]. Neither of these approaches are complete solutions to the problem, as a blend of privacy and attributed credit for efforts might be desired. A possible solution to this might be the introduction of known practises such as blind and double-blind refereeing [67].

Whereas the evaluations people give constitute one type of potentially sensitive information, their profiles can be considered to be another. At the same time, though, they carry potentially useful information. As an example, consider granting access to parts of your profile representing special interests you may have, thus gaining contact with other people with similar interests. As argued by Malone et. al. [56], careful thought is needed as to when and how such features are desirable, but the possibilities are intriguing.

3.2.7 General User Interface and Usability Concerns

A successful recommender system is directly dependent on having a user interface where the overhead of entering new recommendations, accessing other peoples recommendations, or creating a new user profile is not excessive. The display of items and their ratings must be done in some way conveying the relevance they are assumed to have for the

user, for instance by fading colours (the brighter, the more relevant), or by utilisation of the concepts *edit wear* and *read wear* [35].

Some of the interfaces presented in section 2.4.5 are approaching recommender systems in themselves, as they are utilising information filtering to display information to a user, emphasising various aspects of interaction to aid users. They are, however, not directed at specific users, and as such lack the personalised aspect.

3.3 Known Approaches and Implementations

This section discusses some practical approaches to recommender systems.

Early recommender systems were targeted at single users and the development of personalised filters. Some such systems include Tapestry [30, 81], spynews [54], The Information Lens [56], and INFOSCOPE [24]. Later approaches have included collaborative approaches, examples of which are GroupLens [50, 58] and PHOAKS [82, 83]

Two examples of recommender systems, GroupLens and PHOAKS, are described in some detail. Table 3.1 presents a brief comparison of a wider selection of recommender systems.

3.3.1 GroupLens

GroupLens [58, 50, 66] is a recommender system for Usenet utilising collaborative filtering developed at The University of Minnesota, Minneapolis, US. It automatically selects a group of people as personal moderators for a user for a given newsgroup. These moderators are selected based on a measure of agreement as to articles previously judged. Privacy is ensured by allowing users to enter ratings under pseudonyms.

GroupLens relies on augmenting existing interfaces to display ratings. Clients connect to two servers - an NNTP server holding Usenet articles, and the GroupLens server holding ratings and generating predictions. The interface to the GroupLens server is defined using an open protocol, and client libraries written in C and Perl are available [50].

Miller [58] argues that GroupLens has demonstrated the effectiveness of collaborative filtering for the Usenet domain.

	Contents of recommendation	Explicit Entry?	Aggregation	Use of recommendations
GroupLens	**a** numeric:1–5 **b** seconds	**a** explicit **b** monitor reading time	personalised weighting based on past agreement among recommenders	display alongside articles in existing summary reviews
PHOAKS	mention of a URL	mined from Usenet postings	one person one vote (per URL)	sorted display
Java Agent	use wear data	implicit monitoring: **a** reading time **b** links followed **c** further recommendations	explicit recipient, passes along use wear data	presents use wear data alongside document
Tapestry [30, 81]	annotations	explicit	one person, one vote per document	annotations available for filtering queries
Siteseer [69]	mention of a URL	mined from existing bookmark folders	frequency of mention in overlapping folders	display
ReferralWeb [45]	mention of a person or a document	mined from public data sources	assemble referral chain to desired person	display
Fab [4]	numeric: 1–7	explicit	personalised weighting; combined with content analysis	selection / filtering

	Contents of recommendation	Explicit Entry?	Aggregation	Use of recommendations
Amazon.com [1]	numeric: 1–5, annotations	implicit by purchases, viewed items. explicit	personalised weighting based on customer taste similarity	sorted display and available annotations
Anchor text (search engines)	textual	explicit	similarity vectors or similar. Search engine specific.	search engine result ranking

Table 3.1: Example recommender systems

3.3.2 PHOAKS

PHOAKS [82, 83] (short for People Helping One Another Know Stuff) works by automatically recognising, registering and redistributing recommendations of Web resources mined from Usenet news messages. It builds on the observation that multi-confirmed recommendations appear to be significant for relevant recommendation receivers. The number of distinct recommenders of a resource is found to be an adequate measure of resource quality. This conclusion is supported by comparing mined URLs to those found in relevant FAQs.

The ideas of role specialisation and reuse are distinguishing factors of PHOAKS compared to other recommender systems. Built on the observation that a minority of users expend the effort of judging information, users are expected to form producer/consumer relationships within the system. Reuse of recommendations is an inherent quality of the system, as it mines previously recommended resources from existing Usenet messages.

The recommendations are available through a web interface, where a list of ranked URLs can be obtained for any sub-hierarchy.

Summary

Recommender systems enter the context of this thesis as one of several aids assisting users in the information access process.

I present approaches useful when obtaining rankings, evaluating relevance and building adequate user profiles. The aim is to outline design aspects for recommender systems in general, and to create a background for the design of NewsView.

Chapter 4

Search Engines

This chapter presents information retrieval theory in general, and search engine theory in particular, as another important aid in the information access process. This presentation is followed by an in-depth description of FAST Data Search as a tool in the NewsView architecture.

The term *search engine* is used to refer to a service providing users with a search interface to some indexed data. Most search engines use a centralised crawler-indexer-searcher architecture model, where the different parts of the system are responsible for gathering data, indexing it and executing queries against the index, respectively.

An important concept related to search engines is *directories*. Directories gather their listings in a manner very different from that of search engines, but they offer a similar service to users. Many search interfaces present results as a combination of entries from search engines and directories, and this type of search service is known as a hybrid search interface.

A very short presentation of data gathering techniques for the two types of search services is given below, along with a few examples of each.

Crawler-based Search Engines

Crawler-based search engines generate their listings automatically. They 'crawl' the World Wide Web (hereafter referred to as 'Web') or some other information space to obtain knowledge of content, upon which users can query this base of information. Due to this method of data gathering, crawler-based search engines suffer from poor precision (see section 3.2.1. They may, however, have high recall – depending on the size of the document base crawled.

Some well-known crawler-based search engines are:

- Google (http://www.google.com/)
- AllTheWeb (http://www.alltheweb.com/)
- Yahoo (http://www.yahoo.com/)
- MSN Search (http://search.msn.com/).

Human-Powered Directories

Human-powered directories depend on humans for listings. The process of submitting descriptions to the directory can be organised in several different ways. Whoever controls the directory may appoint editors who either write whatever descriptions are entered into the directory, or act as moderators for content submitted by other people. The submission process may also be completely open, allowing anyone to submit content directly into the directory, or community-based, where submissions need to be approved by a certain number of people before acceptance. The data gathering method used by directories lead to a situation opposite of that of search engines in a precision and recall context – they suffer from poor recall, but their precision is potentially very high.

One important aspect of human-powered directories in the context of search engines and information filtering in general, is their application as training material for automatic categorisation approaches.

Some examples of human-powered directories are:

- Open Directory (http://www.dmoz.com/)
- Zeal (http://www.zeal.com/)
- Hotrate.com (http://www.hotrate.com/)
- Kvasir (http://www.kvasir.no/).

The examples of different types of search engines are taken from Search Engine Watch [80].

4.1 Crawler-based Search Engines

Most of todays crawler-based search engines are based on a centralised architecture relying on a set of key components: crawler, indexer

and searcher. These components are presented briefly below, and Figure 4.1 shows a graphical representation of how they are interconnected. The colour and shading conventions introduced in this figure for the three key components will be used throughout the presentation of search-engine related material, as well as the design and implementation treatment of NewsView.

The structure of the presentation of general crawler-based search engine architecture as given here is taken from Risvik and Michelsen [68].

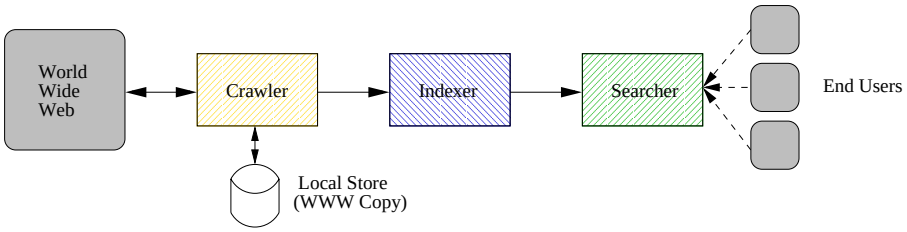


Figure 4.1: Overview of crawler-based search engine architecture

Crawler

A Crawler is a module aggregating documents to create either a local index, a local collection, or both.

Indexer

An indexer is a module taking a collection of documents and data, and creating a searchable index from it. Common approaches include inverted files, vector spaces, suffix structures and various hybrids of these [3, chapter 8].

Searcher

The searcher works on the output from the indexer. It accepts queries from a user, runs them over the index, and returns results to the query issuer.

The choice of approach when indexing or searching items depends heavily on the type of content. This thesis will deal with textual content only, and will not treat the topic of indexing of and matching against non-textual content.

4.1.1 Crawler

Crawlers are more often than not treated in the context of the Web. Search engines focused on other domains may not have crawler functionality in the 'Web crawler' sense, but use other forms of content submission. NewsView is an example of this, dealing with content submission by means of a tool to push files to the indexer. This section assumes nothing about the domain the crawler is applied to, and is equally valid for a crawler on the Web as a crawler traversing a local file system.

The terms and definitions used in the description of crawlers are used differently in the literature available on the subject – this thesis uses the terminology as defined by Cho and Garcia-Molina [11].

In general, there are two ways in which a crawler can update its index or local collection. Following the traditional approach, the crawler retrieves documents until the collection reaches a desirable size, then stops. When a refresh of the collection is necessary, the crawler repeats the process from the beginning, and replaces the old collection with the newly acquired one. This type of crawler is called a *periodic batch crawler*. An alternative mode of operation is that of an *incremental crawler*. Such a crawler works by retrieving a sufficient number of documents, and continues by re-fetching the same documents, thereby updating the collection. The collection is never deleted, and the crawler continues where it left off last time after each indexing cycle. If a previously crawled document is removed, the crawler may choose to include new documents in its collection to maintain its target size.

Theoretically, an incremental crawler can be more effective than a periodic batch crawler. As an example, given the opportunity to estimate how often documents change, the incremental crawler can update only those documents likely to have changed. An incremental crawler also has the opportunity of indexing new documents right away as they are found, whereas periodic batch crawlers can do this only after the next indexing cycle starts. As a direct consequence of this, the effectiveness of any crawler technique depends heavily on how documents change over time [11].

Two important aspects in this regard, are growth dynamics and update dynamics of the media crawled. Some research has been made in these areas in a Web context, but not much in general. For studies of Web growth dynamics, see Bray [9] and Lawrence and Giles [52, 53], for studies of Web update dynamics, see Huberman and Adamic [37], Cho and Garcia-Molina [12, 11] and Brewington and Cybenko [10].

Freshness

The freshness of a document d_i at time t is defined as the probability for that document to be up-to-date at the given time [68].

To maximise freshness, the goal is to:

Minimise time spent refreshing unchanged content	$\Longleftrightarrow$	Maximise time spent refreshing changed content

Freshness can be obtained through several measures. *Freshness through scheduling* tries to obtain freshness by combining high document retrieval capacity with a scheduling algorithm prioritising retrieval of the documents deemed most likely to be updated. To provide fresh results to the users of a search engine, short indexing cycles are needed. As an implication of this a complete refresh might not be possible between such cycles, and the scheduling algorithm is key to keeping the local store, and thereby the results, as fresh as possible.

Freshness through heterogeneity is built on the observation that documents have different freshness requirements. A solution to this is to have heterogeneity amongst the crawler machines, dedicating some crawler nodes to specific types of content.

Cooperation with providers is another approach to further improve freshness. There are several models for this cooperation. A *pull* model (figure 4.2(a)) permits the crawler to perform efficient scheduling. The crawler periodically retrieves meta-information, and uses this information in the scheduling process. Another approach is to use a *push* model (figure 4.2(b)), where content or meta-information is pushed directly to the crawler. This implies that the crawler is notified if a document is changed, added or deleted. A third approach called a *hybrid model* (figure 4.2(c)) employs aspects from both push and pull. Special proxy systems read meta-information files, act according to the content in those files, and use a push model to update the crawler (Risvik and Michelsen [68]).

NewsView uses a slightly different model to obtain freshness. The most important information is stored in a special real-time structure, implying its immediate update when new values become available. The rest of the information is updated using a batch-oriented back end. More information on this concept in the context of NewsView is given in section 6.2.4.

Regardless of the approach to freshness chosen by a given application, the indexer and searcher in a traditional search engine consider whatever

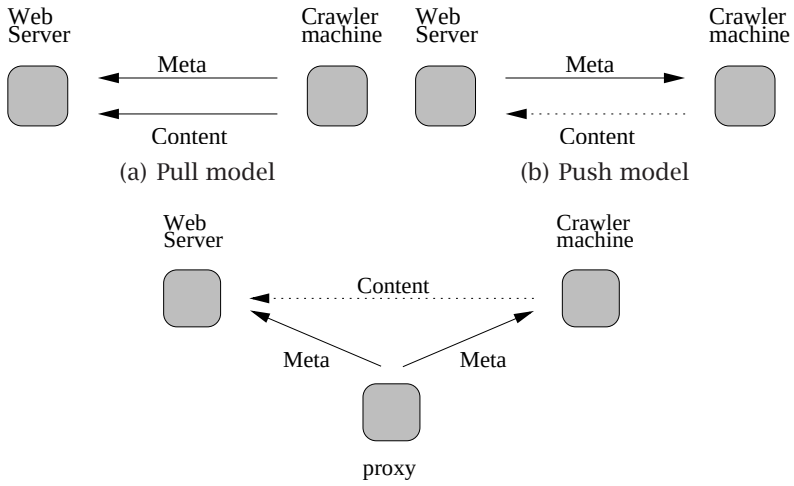


Figure 4.2: Three models for obtaining freshness by cooperation with content providers

content they have available 'current', and have no internal notion of freshness. As such, the responsibility of keeping content fresh lies solely with the crawler.

4.1.2 Indexer and Searcher

Indexing and searching, emplaced in a broad research context in section 3.1, are concepts deeply rooted in *information retrieval* theory. This section presents an introduction to some specific aspects of this field.

The motivation behind information retrieval is summarised by Baeza-Yates and Ribeiro-Neto [3]:

> *Information Retrieval (IR) deals with the representation, storage, organization of and access to information items. The representation and organization of the information items should provide the user with easy access to the information in which he is interested.*
>
> – Baeza-Yates and Ribeiro-Neto [3]

Several difficulties arise when attempting to fulfil this goal. Defining the user information need is not easy, and as described in section 3.2.1, measuring the relevance of results for a given information need presents another difficulty.

Data retrieval is a related field to information retrieval. There are some crucial differences between the two that highlight an important aspect of information retrieval systems: where a data retrieval system attempts to retrieve all items satisfying a clearly defined expression, an information retrieval system is concerned with retrieving *information* about a subject. If a data retrieval system fails to retrieve a single relevant item, there is something seriously wrong with it. Information retrieval systems may be inaccurate, and small errors are likely to go unnoticed [3].

The User Task

A user comes to an information retrieval system because of an information need, and searches for useful information executing a *retrieval task*. Users of retrieval systems may be performing two distinct types of tasks when interacting with an information retrieval system: information *retrieval* and *browsing*, as shown in figure 4.3. Classic information retrieval systems are usually concerned with the first task [3].

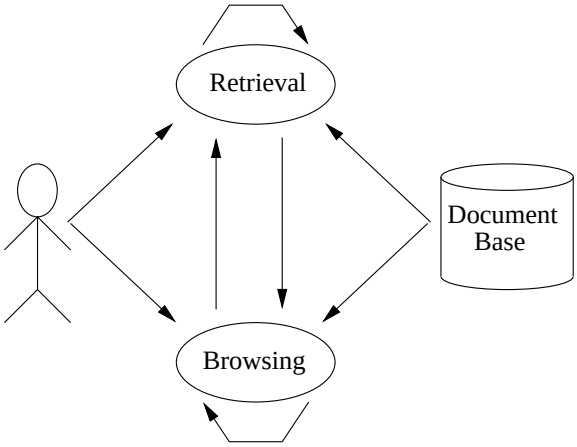


Figure 4.3: The user interaction with the retrieval system through distinct tasks, after Baeza-Yates and Ribeiro-Neto [3, page 4]

The Retrieval Process

This section will detail the retrieval process. Figure 4.4 shows a simple generic architecture used as the basis for the description.

The first thing to do when setting up an information retrieval system, is to configure the document operations and indexing details. The document base must be configured, and the set of documents to be indexed

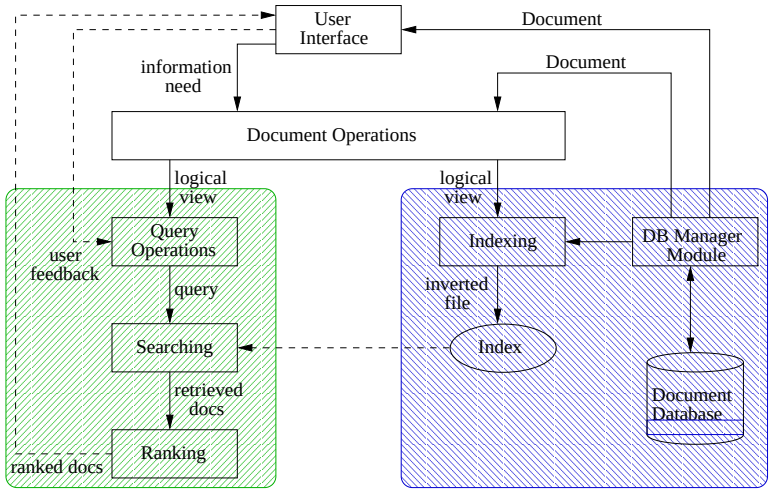


Figure 4.4: The process of retrieving information, after [3, page 10]

and the document model (that is, the document structure and what elements are available for retrieval) must be specified.

Once the *logical view* of the documents is defined, the documents can be indexed. A logical view of a document is some representation of the document, for example keywords, full text and representations created by various text operations. The full text is clearly the most complete logical view, but its use usually implies higher computational cost. An index is a data structure allowing fast searching over large volumes of data. Different index structures can be used, see Baeza-Yates and Ribeiro-Neto [3, chapter 8].

Given an indexed document base, the retrieval process can be initiated. The user expresses an *information need*, which is transformed into a logical view. This is done using the same document operations as for the documents themselves. The system might apply *query operations* to the logical view to transform it into an actual *query*, providing a system representation of the user information need. At last, the query is processed to obtain the *retrieved documents*.

Before returning results to the user, they are *ranked* according to their likelihood of being relevant. The user examines the documents he receives, and might initiate a *user feedback* cycle. In such a cycle, the documents selected by the user are used to change the query formulation, hopefully to better represent the real user information need.

Formal Characterisation of Retrieval Models

Baeza-Yates and Ribeiro-Neto [3] give the following formal definition of an information retrieval model:

Definition. *Information Retrieval Model*

An information retrieval model is a quadruple

$[\boldsymbol{D}, \boldsymbol{Q}, F, R(Q_i, D_j)]$ *where*

1. $\boldsymbol{D}$ *is a set composed of logical views (or representations) for the documents in the collection.*
2. $\boldsymbol{Q}$ *is a set composed of logical views (or representations) for the user information needs. Such representations are called queries.*
3. F *is a framework for modelling document representations, queries and their relationships.*
4. $R(q_i, d_j)$ *is a ranking function which associates a real number with a query* $q_i \in \boldsymbol{Q}$ *and a document representation* $d_j \in \boldsymbol{D}$*. Such ranking defines an ordering among the documents with regard to the query* q_i*.*

The classic approaches to realisations of this model were described in section 3.2.2, as part of the treatment of content-based filtering.

4.2 FAST Data Search

Sections 4.2 to 4.2.4 give an introduction to FAST Data Search, outlining its architectural structure. First, I present a feature and system overview, followed by a data flow overview. Next, an overview of the modules and how they interconnect is described, followed by a brief outline of the content lifecycle.

In sections 4.2.5 to 4.2.7 in-depth coverage of issues particularly relevant to this thesis is provided. Index profile and document processing issues are presented, along with categorisation information and a description of the FAST Taxonomy Toolkit.

The discussion of FAST Data Search is based on 'The FAST Data Search System Reference Guide' [20].

4.2.1 Feature and System Overview

FAST Data Search (FDS) is an integrated software application combining searching and filtering functionality. It is a distributed system, and enables information retrieval in many types of information by supporting real-time filtering, various linguistics features and several content access options.

FDS is scalable in three dimensions: volume of data, number of users, and freshness of data. It can be run on several platforms, and supports a variety of file formats.

The FAST Search Engine indexes documents and matches them against incoming search queries.

The FAST Filter Engine stores alert queries in an index. As new documents arrive at the filter engine, they are matched to the stored alert queries, and any match is submitted to the FAST Alert API.

Figure 4.5 shows a graphical representation of the architecture of FDS. Details concerning the various parts of the figure are explained in the following sections. The figure is a remake of an illustration given in 'The FAST Data Search System Reference Guide' [20], with the coloured and shaded sections following the conventions from figure 4.1 to illustrate the crawler-indexer-searcher architecture.

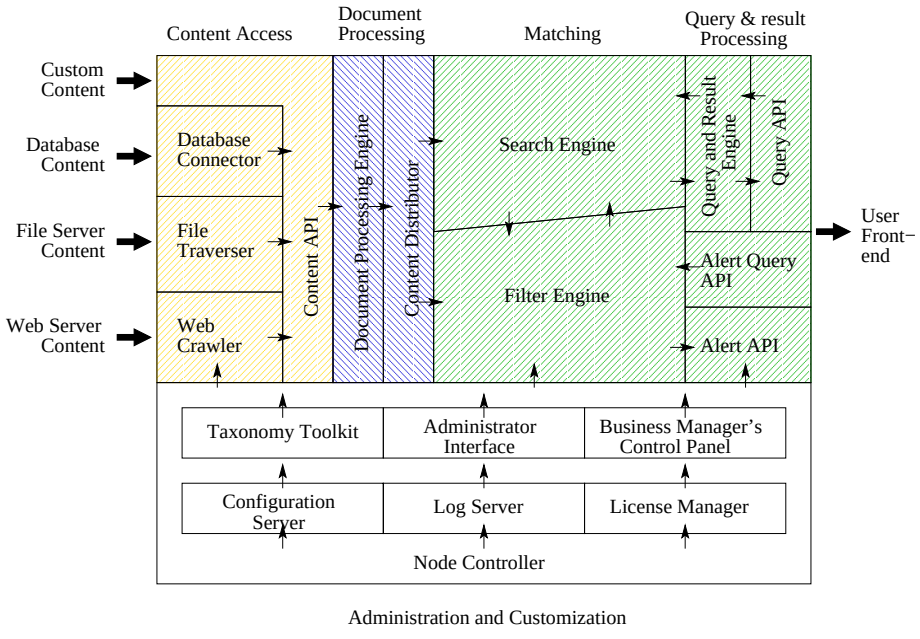


Figure 4.5: FAST Data Search system architecture

4.2.2 Data Flow Overview

The data flow through FDS consists of some basic steps:

Submitting Content

Content is submitted to FDS using one of the three predefined modules (the FAST Web Crawler, the FAST Database Connector or the FAST File Traverser) or a custom module built using the FAST Content API. This is shown in the column labelled 'content Access' in figure 4.5. This corresponds to the crawler part of the crawler-indexer-searcher architecture.

Analysing and Processing Documents

To make documents searchable and filterable, all content submitted to FDS is converted to the FDS internal document format, FastXML. To achieve this, each document goes through a pipeline of document processing stages in the FAST Document Processing Engine. The purpose of this conversion is to store all content in one consistent format, as well as extracting various meta information, like language, to improve search and filter relevancy. This is done by the 'Document Processing Engine', as depicted in the 'Document Processing' column in figure 4.5. The content of this column corresponds to the indexer part of the crawler-indexer-searcher architecture.

Matching Documents and Search Queries

When new documents are submitted to the FAST Search Engine, indices are generated from them. This makes them searchable and filterable.

Search queries are submitted through the FAST Query API. The Query API does, in turn, send the queries on to the FAST Query and Result Engine which pre-processes them to improve result relevancy. Upon pre-processing, the queries are sent to the FAST Search Engine which matches them against search indices and returns a list of resulting documents. If any further post-processing of the result list is desired, it is performed by the FAST Query and Result Engine. Finally, the result list is returned to the query issuer though the FAST Query API.

Matching Documents and Alert Queries

The FAST Filter Engine stores alert queries and keeps them in an index. These queries are submitted to the Filter Engine through the FAST Alert Query API. New documents are matched against the

stored queries as they arrive at the Filter Engine, and matches are submitted to the FAST Alert API.

The columns labelled 'Matching' and 'Query and Result Processing' in figure 4.5 show a graphical representation of the flow of operations for both search and alert query matching. These correspond to the searcher part of the crawler-indexer-searcher architecture.

4.2.3 Module Overview

FDS consists of several subsystems matching the data flow steps of the system. These subsystems are divided into one or more modules each. Table 4.1 presents these subsystems and those modules relevant to this thesis.

Subsystem	Module	Description
Content Access	FAST Web Crawler	Locates and retrieves files from Web Servers
	FAST File Traverser	Traverses and retrieves files from directories on file servers.
	FAST Database Connector	Traverses and retrieves content from databases.
	FAST Content API	Allows the standard content access modules of FDS as well as custom applications to push content to the FAST Content Distributor.
Content Distribution	FAST Content Distributor	Receives documents through the FAST Content API and distributes them to a set of FAST Document Processing Engines.
Document Processing	FAST Document Processing Engine	Performs document processing tasks for format conversion and document relevancy, such as language detection, and lemmatisation.

Subsystem	Module	Description
Matching	FAST Search Engine	Performs the searching tasks within FDS. It indexes new documents coming from the FAST Document Processing Engine, matches them against search queries submitted by the Query and Result Engine, and returns a list of resulting documents to the Query and Result Engine.
	FAST Filter Engine	Performs the filtering tasks within FDS. It stores multiple sets of alert queries submitted through the FAST Alert Query API, matches them to documents as they are received from the FAST Document Processing Engine, and returns alerts through the FAST Alert API.
Query and Result Processing	FAST Query and Result Engine	Processes search queries and search results to enable relevancy-focused searching and result presentation. It provides linguistic query processing features like proper name recognition or spell checking, and result processing features like query highlighting.
	FAST Query API	Allows external search front end systems to submit their queries and get result sets in return.
	FAST Alert Query API	Handles the alert query traffic from the filter front end to the FAST Filter Engine.
	FAST Alert API	Provides an interface from which front end systems get alerts sent from the FAST Filter Engine.

Subsystem	Module	Description
Common Services	FAST Taxonomy Toolkit	Enables creation and editing of taxonomy structures and allows you to map documents to categories.
	FDS Administrator Interface	Provides a browser-based graphical user interface allowing the system administrator to monitor and configure FDS.

Table 4.1: FAST Data Search modules. For a graphical representation of how they interact with one another, see figure 4.5.

4.2.4 The Content Lifecycle in FAST Data Search

This section describes various concepts related to the content lifecycle, as well as how content is processed to become searchable. Figure 4.6 shows a graphical representation of the content lifecycle within FDS.

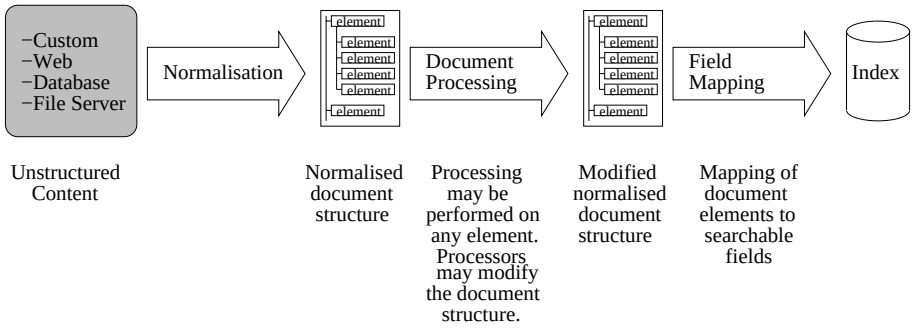


Figure 4.6: The content lifecycle within FAST Data Search

Content

Data yet to be submitted to FDS is called *content*. This contrasts with the term *document*, which designates data residing inside the system.

Collections

Content to be retrieved and made searchable, is grouped into *collections*. Collections are used to group content by source and processing needs. A collection is set up by defining the content source

and the document processing rules, and may be used to narrow down the scope of a search.

Collections provide a mechanism for differentiating the treatment of documents, by having collection-specific indexing specifications. They also provide a way to prioritise different types of content, by specifying the order in which collections are to be processed in the system.

Documents and Document Elements

When content is submitted to the FDS system, it is converted into an internal representation called *document*. The format of this internal representation is FastXML, the FDS-specific mark-up language.

Generally, there is a one-to-one relation between a content entity and a document. The content entity is represented by the document as a set of elements. The content of these elements is used during document processing. Element entries are filled in as the document flows through the system, and information stored can include parts of the original content as well as meta-information directed towards improving search and filter relevancy.

FastXML

The internal document format used by the FDS system, FastXML, names and defines document elements. Each element in a FastXML document is preserved as an entity throughout document processing, indexing and result processing.

Document Elements and Fields

Before indexing a document, each document element is mapped to a *field*. Fields are document elements defined to be searchable. These fields enable queries concerning only individual parts of a document. Fields are defined in the *index-profile*, and can be of type 'text' or 'unsigned integer'. Multiple fields can be grouped into so-called *composite fields*, allowing a query to be executed on several fields at the same time.

4.2.5 Index Profile and Document Processing

This section presents an overview of the basic concepts involved in the process of indexing documents and making them searchable.

Indexing Documents

The FAST Search Engine receives processed documents from the document processing engine (See figure 4.5), and creates search indices from the documents as they arrive.

A *search engine cluster* is a group of search engine instances sharing the same index configuration, provided by an index profile. Each search engine cluster can have a number of collections assigned to it, but a collection can belong to only one cluster. As such, there is a one-to-one relationship between an index profile and a search engine cluster, and a need for more than one index profile configuration implies a need for a corresponding number of search engine clusters and collections.

Defining How Documents are Searchable

In the process of creating a search index, FDS makes use of an *index profile*. An index profile is an XML-based configuration file defining the way documents are searchable, and how fields are to be treated by query and result processing. It specifies properties like:

- Which document processing elements become searchable fields in the index
- Definition of the field attributes, such as type (text/integer), search attributes (substring, lemmatisation, and others), rank tuning parameters and result presentation
- Grouping of fields and creation of result views
- Definition of result-processing attributes related to the fields, such as result clustering and dynamic drill-down (see section 4.2.6).
- Which document elements become fields returned as part of results.

The index profile concept is closely tied to the concepts of document processing and search engine clusters. Figure 4.7 is modelled after a figure given in [20], and shows a graphical representation of this relationship.

Document processing is performed prior to indexing. During document processing, each document is represented by a set of elements, which can be further processed and later mapped to searchable fields via the

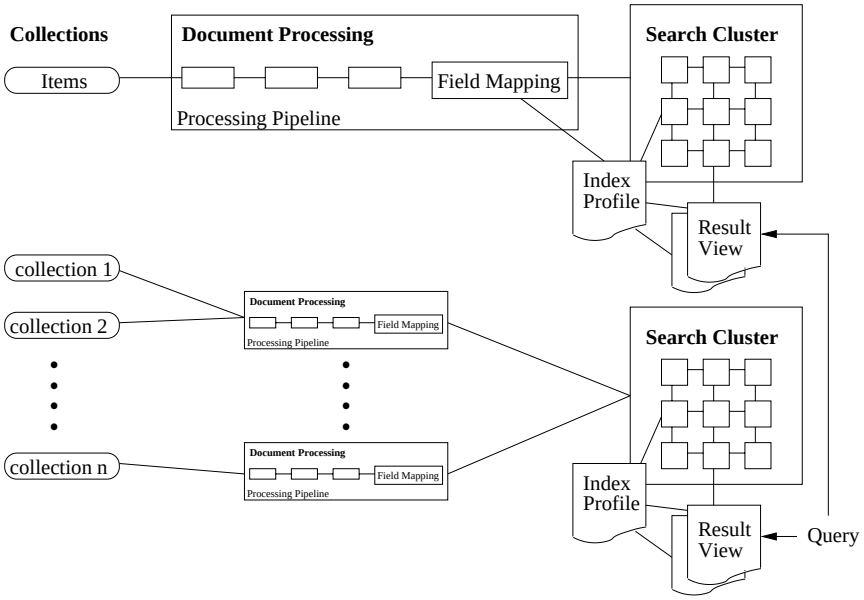


Figure 4.7: The relationship between document processing, index profiles and search engine clusters

index profile. Both elements and fields represent content parts and attributes related to the document, e.g. body, title, heading, URI, author, category etc.

The index profile defines the layout of the searchable index, and specifies how fields are to be treated by query and result processing.

The index profile also includes one or more result views. A result view defines alternative ways for a query front end to view the index with respect to queries.

The Index Profile Structure

As indicated in figure 4.8, the index profile consists of three main XML blocks:

List of Fields

The basic entity of an index profile is a *field* with its attributes. All fields are searchable and part of the default result view unless otherwise specified.

The *field-list* contains a list of all defined fields within the index, including information about mapping from the corresponding doc-

```
<?xml version="1.0"?>
<!DOCTYPE index-profile SYSTEM "index-profile-2.0.dtd">
<index-profile name="datasearch">
   <field-list>
      <field name="name" ... />
      ...
   </field-list>
   <composite-field name="name" ... >
      <field-ref name="name" ... />
      ...
      <weight-list ... />
   </composite-field>
   <result-specification>
      <result-filter ... />
      <categorization ... />
      <clustering ... />
      <result-proximity ... />
      <integer-navigator ... />
      <string-navigator ... />
      <result-view ... />
   </result-specification>
</index-profile>
```

Figure 4.8: Overview of the index profile XML format

ument processing elements, indexing and search attributes related to individual fields.

Composite Fields

Composite fields allow grouping of individual source fields into larger groups of fields by referencing the source fields. This gives the possibility of addressing groups of fields together.

The *composite-field* specifications can define combined field specifications with configured relevance score per field.

Result Specification

The *result-specification* contains a set of configuration rules for the search result handling, including field weightings.

4.2.6 Categorisation

This section presents the basic concepts and terms of categorisation in FDS. The reason for the thorough treatment of these issues, is their importance for the design and implementation of NewsView. Categorisation and clustering features are used heavily within the application, which necessitates a rather in-depth introduction to these concepts.

FDS utilises several different approaches to the concept of categorisation:

Categorisation

Categorisation is the process of assigning documents to specific categories on the basis of pre-defined rules. These categories may derive from a given hierarchy, a so-called taxonomy, or be calculated on the fly based on similarities between documents.

Clustering

Clustering means the automatic detection of groups (clusters) of documents whose content is somehow similar.

Two different types of clustering are available:

- Unsupervised clustering implies that neither cluster names nor structure is known before the clustering takes place, but are automatically computed from the document set. This way of learning is often referred to as unsupervised learning.
- Supervised clustering implies that existing knowledge of categories within a result set is used in the clustering process. Non-categorised documents will be clustered with similar documents within existing categories, based on document similarities.

Document Similarity

Categorisation as well as clustering is based on a notion of document similarity: Similarities between the content of different documents are computed, and decisions to place documents into classes or clusters are done on the basis of these similarities. Document similarities are computed using document vectors, as presented in section 3.2.2.

In FDS, clustering is performed on result sets as returned from the search engine. Document Vectors are created by the *Vectorizer* document processor.

Categorisation Feature Overview

FDS Includes support for several different categorisation techniques, which can be activated independently, or in combination with one another. These include:

Documents Tagged by Category

Documents may be tagged with categorisation information prior to submitting the content to FDS.

Automatic Categorisation

There are several different ways in which documents can be tagged with categorisation information prior to indexing. Tagging of this sort takes place during document processing.

URI based categorisation

A document can be assigned to one or more categories based on its URI.

Rule based categorisation

A document can be assigned to one or more categories based on programmatic rules specified in the Python programming language. This is configured from within the Taxonomy Toolkit.

Category Matching

Categories may be used within the matching process of FAST Data Search in several ways, including:

Search within category

Queries can be restricted to match documents belonging to a specific category alone.

Number of hits per category

If a dedicated *Taxonomy Index* is created, the number of hits per category across the entire searchable index becomes available. By using the taxonomy index, the total number of hits within all categories (as long as at least one document within the category matches the query) can be returned, including those categories not represented in the result set. This can be a very efficient tool for subsequent drill-down queries.

Categorising Results

Result Clustering

Result clustering enables a directory or taxonomy view within a result set. This can be applied in several ways, using one of

- Supervised clustering
- Similarity clustering

- Unsupervised clustering

An important distinction to keep in mind is that between category matching and result clustering. Whereas category matching is applied across the entire index, result clustering is applied to a configurable set of top ranked results.

Dynamic Drill-Down

Dynamic drill-down provides a possibility for multi-dimensional drill-down in structured data based on content properties.

An integrated part of this feature is *results-based binning*, which provides implicit ranking of dimensions (search result fields) based on relevance scores. For each dimension, the relevance is computed based on the actual drill-down capabilities of this dimension.

The advantage of this is the ability to find relevant results quickly using a combination of searching and browsing by parametric value and range.

Find Similar

Drill-down queries may be defined based on similarity to selected documents on a previously returned result set. This feature is based on the use of document similarity vectors. Specific comparisons supported are

- Find similar
- Refine similar
- Exclude similar

4.2.7 The Taxonomy Toolkit

The *FAST Taxonomy Toolkit* enables configuration and maintenance of static category mappings and taxonomies, which can be flat or hierarchical. The categorisation can be accessed either through a GUI or by editing the XML files storing the taxonomy directly. Any number of documents may be assigned to a given category, and a document may be assigned to several categories.

Documents can be mapped to categories in several ways. URI-based and rule-based mapping enable mapping based on the URI of a document and on programmatic rules specified using the Python programming language respectively. The category rank boost feature enables rank boosting towards specific queries.

The toolkit contains a set of document processors which have to be included in the processing pipeline of a collection for automatic categorisation to be performed:

ManualMapper
: Handles URI-based mapping of documents to categories.

RuleClassifier
: Handles rule-based mapping of documents to categories.

TaxonomyBoost
: Applies rank boost information to documents.

TaxonomyTagger
: Adds supplementary information to categories.

Summary

This chapter provides a link between theoretical information retrieval and the structure of a modern crawler-based search engine. It describes FDS as a tool to be used in the NewsView architecture.

Chapter 5

Design

This chapter describes the design of NewsView, an architecture created as part of the work on this thesis.

At this point, it is useful to recap my thesis objective as stated in the in the introduction:

> **Thesis Objective**
>
> The objective of my thesis is to look at possible ways in which information filtering and information retrieval can be combined. This is done in the context of recommender systems for Usenet built using search engine technology.
>
> My aim is to obtain a higher signal to noise ratio, through better result relevance prediction and extensive navigational possibilities within the data.

The NewsView architecture as presented here describes a framework for interfaces to Usenet. It has information retrieval and information filtering concepts built into it, and encourages extensive navigational possibilities within the data. The purpose of this framework is to provide a testbed for user interface, information filtering and information retrieval issues, and, most importantly, combinations of the three. The concrete system described explores the feasibility of a recommender system for Usenet built using a search engine back end.

When considering information retrieval systems in use in poorly structured information domains today, the two most prominent types are human-powered directories (hereafter referred to as directories) and crawler-based search engines (hereafter referred to as search engines). Where directories suffer from poor recall but have potentially very high

precision, the situation is reversed for search engines. Recommender Systems can be considered a parallel to directories in this regard, as ratings are gathered for documents viewed by users only. By combining the two approaches, recommender systems and search engines, the hope is to keep the best from both worlds and thereby improve both precision and recall.

By choosing a data domain of a conversational nature, certain specific filtering options are made available. Conversational data contain inherent information about social structure and interaction patterns, and can be mined for such meta-information. Given automated methods to obtain, process and extract predictions from such information, it can be utilised to improve relevance ratings. Combining predictions of this type with implicit ratings, to enhance ratings directly or to reinforce or adjust preexisting prediction methods, is an exciting possibility in the context of recommender systems. This approach can be used both as a means for easing content browsing, for rating documents before displaying them, and for improving prediction of a users relevance function when retrieving information from collections of documents.

Utilising meta-information for ranking purposes is, however, only one of many possibilities. Another option is to visualise this information, and present a user interface conveying forum, user and thread characteristics by visual means. Some examples of such visualisation approaches were given in section 2.4.5. Such visualisation methods are feasible in a browsing setting, where it can provide users with an easily available overview of the interaction patterns of a forum or other collection of messages. Such an approach can be particularly helpful for new users.

The combination of browsing and searching in an information retrieval context as described in section 4.1.2, is an emerging approach not yet very well investigated [3]. NewsView provides an easily accessible framework in which such combinations can be studied.

The user interface presented by NewsView may be used to visualise various aspects of data. I present some such options in a Usenet context in the following.

- Usenet messages can be displayed according to their hierarchy membership, as specified by a taxonomy. For instance, a hierarchical list of Usenet groups can be displayed along with the number of returned documents within each group.

- Each Usenet message can be accompanied by a link leading to a 'search for similar' query for that message.

- navigation within returned result sets based on predefined criteria such as thread membership, name or mail address of poster, message size, language or posting date can be provided through dynamic drill-downs.

Providing an almost platform-independent user interface in a widely accessible setting (Web), NewsView should be easily available on a large scale if so desired.

Technical System Sketch

In a technical context, the NewsView task divides into three main stages. Figure 5.1 shows a graphical representation of this structure, coloured and shaded according to the crawler-indexer-searcher convention introduced in chapter 4. The figure elements coloured in blue are elements either created or configured for NewsView purposes. These colouring conventions will be used throughout the design and implementation chapters. The parts of figure 5.1 corresponding to a given stage will be repeated in the section presenting that stage.

A brief introductory description of each stage is given below:

Usenet Feed

: Usenet messages are collected from an NNTP server and submitted to a search engine.

Search Engine

: A search engine takes care of storage of, and provides access to, the data. The search engine used for NewsView is FAST Data Search (FDS) [19].

Front End

: FDS includes a J2EE based toolkit called Fast Query Toolkit (FQT). This toolkit is used as the basis for the NewsView front end, which handles searching, browsing and navigating the data.

The rest of this chapter contains in-depth descriptions of the design of the three main NewsView stages as described above.

The first stage, the Usenet feed, is presented in section 5.1. The second stage is the search engine, and is presented in section 5.2. The final stage, the front end, is presented in section 5.3.

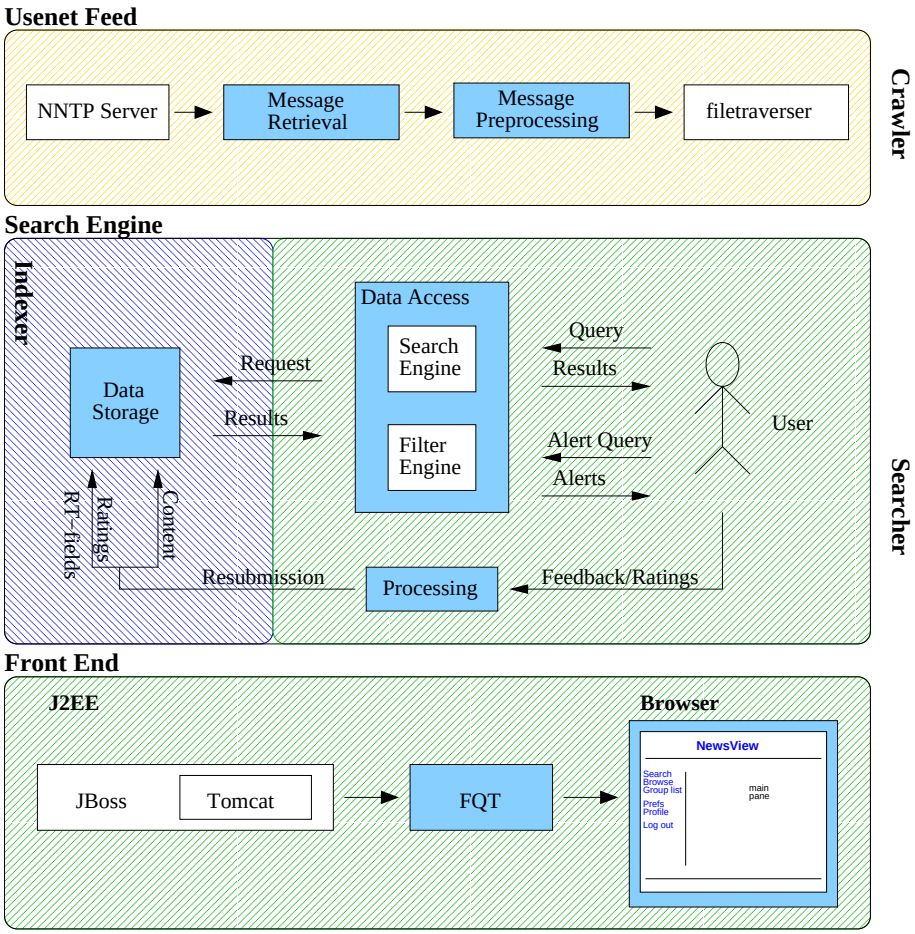


Figure 5.1: Overview of NewsView technical design

5.1 Usenet Feed

The first conceptual stage in the NewsView design, the *Usenet feed*, can be broken down into three steps as shown in figure 5.2.

First, Usenet Messages have to be retrieved from an NNTP server and saved to local disk. This is accomplished by a combination of several tools. Each article is retrieved and saved to a file named according to its message-id to ensure unique file names.

The second step is to perform any preprocessing required prior to submittal to the search engine. The task of the preprocessing step is to introduce a thread concept, and is taken care of by a script. A NewsView-specific thread id is inserted into the message as a header field – all messages belonging to the same thread get the same thread id. The pur-

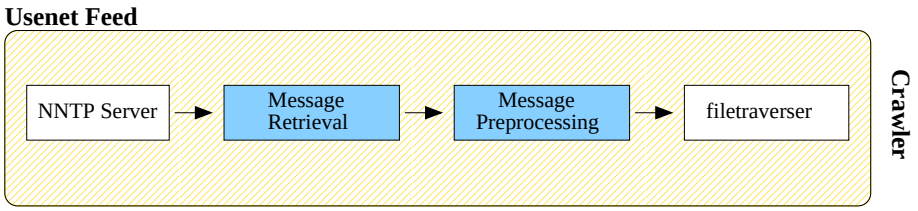


Figure 5.2: NewsView Usenet feed design

pose of this id is to enable search within threads and the ability to easily retrieve an entire thread from the search engine.

The third and final step of the creation of a Usenet feed is to submit the processed messages to the search engine. This is accomplished by means of a utility called *filetraverser*, which is bundled with FDS. Filetraverser feeds each article to the FAST Content API, upon which the search engine indexes them and makes them searchable.

5.2 Search Engine

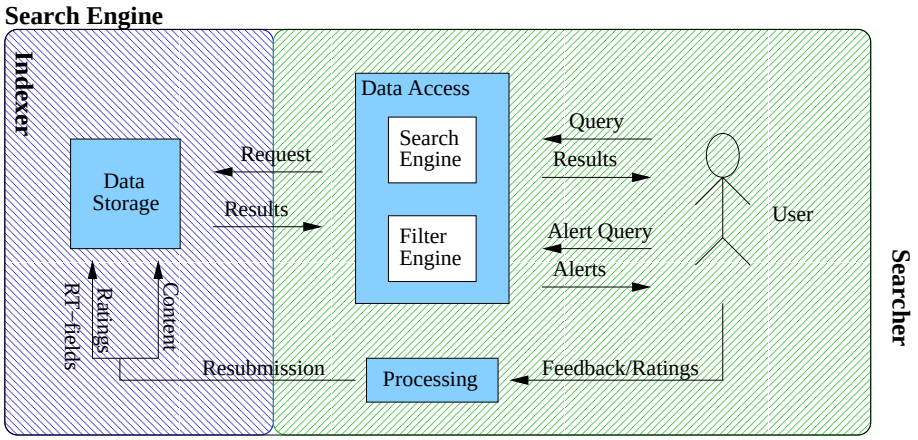


Figure 5.3: Newsview search engine design

The second conceptual stage constituting NewsView is the *search engine*. This stage serves two primary purposes, namely *data access* and *data storage*. Figure 5.3 shows a graphical representation of this stage.

The search engine used for NewsView is FAST Data Search. For a description of search engines in general as well as details concerning FDS specifics, see chapter 4.

The issues concerning data access are presented first, as the data storage design is highly motivated by the data access demands.

5.2.1 Data Access

As indicated in the introduction to this chapter, NewsView places considerable demands on the data access options available to the front end.

FDS has built-in functionality for grouping and clustering of messages, by predefined metrics as well as by advanced linguistic methods, such as automatic topic extraction and comparison. Features utilised by NewsView are described in the following. They are grouped based on similarity of scope and functionality.

Clustering and Categorisation

As described in section 4.2.6, there are two approaches to the concept of document grouping in FDS: clustering and categorisation. Both are used for different purposes in NewsView.

Categorisation by means of the taxonomy toolkit as described in section 4.2.7 is used to classify documents according to the rigid hierarchy of Usenet newsgroups. All Usenet newsgroups included in the Newsview data have to be present in the NewsView taxonomy. By making this an absolute requirement, the ability to guarantee membership in at least one category is established. Utilising the Taxonomy Toolkit for this purpose implies that any new groups included leads to deployment of a new taxonomy.

Clustering is used to obtain some sense of serendipity in the system. It is used for 'find similar' queries across and within group boundaries, and for general result clustering.

Dynamic Drill-Downs and Result Views

Dynamic drill-downs are used to create a hierarchical navigation possibility within the documents based on some given criteria. They are defined statically prior to indexing data, and are applied across a result set returned from the FAST search engine. They can be applied to strings as well as numerical values. The application of dynamic drill-downs in NewsView is to enable navigation within the returned result set based on message properties such as thread membership, name or mail address of poster, size or language among others.

Result views is a way to name specific sets of result fields. This allows special result presentations tailored for specific uses to be created in a very simple way. NewsView has three result views: The first is for message summaries, giving only a few header fields and the teaser. The second provides a basic message view giving some headers and the message body, and the third is an extended message view with all header fields included.

Ranking and Filtering

Custom values can be inserted into the ranking algorithm of FAST Data Search. Some message characteristics for which it might be interesting to include values in the ranking, are spam and flame [79] properties. Another obvious choice for such a feature, is updated message ranking based on feedback from the end users, given implicitly, explicitly or both. Section 3.2.4 gives an overview of possible types of ratings, and ways to obtain them.

5.2.2 Data Storage

The basic information items needing to be stored in the search engine, are the Usenet articles themselves and the recommendations given by end users. What needs to be done here is to decide on an information storage structure, and exactly what is to be stored.

There are two basic approaches to a solution fulfilling these requirements.

- Messages and Recommendations can be stored in two separate collections. The two collections would have different index profiles, and each would be configured to cater for its specific content needs. They would also belong to different clusters. Both collections would be indexed with respect to message id, and thread ids would be made searchable to enable searching within threads. To update recommendations and rankings for a given document, the message itself and the recommendations would both need to be resubmitted.
- Recommendations can be stored as meta information within the document containing the message. When updated or new recommendations become available, the message is resubmitted to the search engine along with the included recommendations and updated ranking information.

The first approach is chosen for NewsView. This decision was made based on the available documentation at the time, and ease of implementation. Presently, only the message collection is implemented due to limited hardware resources.

There is one remaining issue to consider before we are ready to specify in detail the information items to be stored in these collections. Conversational data, of which Usenet messages is an example, have a very different relationship with time than what is the case for search engines in general. When considering conversational data, temporal aspects may be feasible as metrics in result ranking. Search engines do not, as a rule, pay attention to such issues when queries are matched to indexed documents – the content available at any given time is considered 'current'. The crawler strategy chosen is responsible for adequate content freshness.

As a result of this, two related design aspects must be considered: How should message dates be stored and handled within the search engine to allow ranking mechanisms to take advantage of temporal aspects? And, for how long should messages be stored? The former aspect is dealt with by storing the contents of the *Date* header field, uniformly formatted, in a dedicated index profile field. The latter aspect is ignored for the purposes of this thesis, as Usenet messages are assumed to be stored indefinitely. This may not be feasible in a large-scale setting. The shorter the time-span in which content is stored, the faster recommendations and ratings need to be propagated through the system to reach other users.

Message Collection

In addition to more general document information, such as size, submission date, title, teasers, language and character set, some NewsView-specific information is stored in the index:

- Extracted Usenet message header fields, as defined in RFC 1036 [36] (see table 2.2)
- NewsView-specific header field for thread id, for filtering, display and categorisation purposes
- Message body in html-formatted form, for display purposes
- Original message headers in html-formatted form, for display purposes
- Posting date in uniform format, for filtering purposes

- NewsView-specific ranking information is stored in a message header field.
- NewsView-specific ranking information

Recommendation Collection

The information associated with each recommendation is constituted of the same general document information as for the message collection, with the addition of:

- The recommendation itself
- Information about who entered it
- A reference to the article or thread it concerns

5.2.3 Resubmitting Content to the Search Engine

When new or updated information about an article is available through user feedback, this information needs to be resubmitted to the search engine in some fashion. Recommendations are resubmitted if they are updated, or added if they are new. They are collected, prepared and submitted to the search engine at periodic intervals.

5.3 Front End

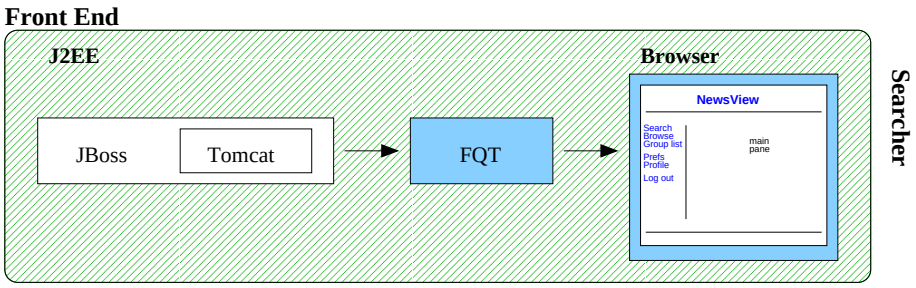


Figure 5.4: NewsView front end design

This section describes the third stage in the conceptual design of Newsview – the *front end*. It describes some of the technologies constituting

the basis for the front end, and the tools used for building it. Figure 5.4 shows a graphical representation of the front end technical structure.

An obvious starting point when selecting tools for the front end, was to look at the selection of tools bundled with FDS. The *Fast Query Toolkit (FQT)* is one of those tools. FQT is is an example of an integration of the FAST Query API with an HTTP based search application. It is geared towards demonstrating features in the area of clustering, categorisation and dynamic drill-downs, and as such implements many of the FDS features most important to a first NewsView interface. It is also easily extendible to other areas of FDS functionality, as it is implemented as a JSP tag library. FQT will be described in more detail in section 5.3.2.

My choice of FQT as a tool for NewsView implied the need for a J2EE framework implementation, specifically a Java application server with a suitable servlet container. The preferred application server for use with FQT is JBoss, and the preferred servlet container is Tomcat. As such, these were chosen for NewsView. JBoss and Tomcat will be presented in more detail in section 5.3.1, along with a description of J2EE.

Section 5.3.3 gives an overview of some issues connected to the NewsView user interface, as built on top of FQT – not only in respect to the interface design itself, but also in respect to aspects such as a user concept.

5.3.1 J2EE and JBoss with Tomcat

An Overview of J2EE

The treatment of J2EE given here is based on the presentation at the J2EE homepage [42]. All figures in this section are reworked from graphics given in the same location.

J2EE, or the Java™2 platform, Enterprise Edition, is an environment for developing and deploying multi-tiered applications. J2EE simplifies development of enterprise applications by basing them on standardised, modular components, providing a complete set of services to those components, and handling many details of application behaviour automatically. J2EE takes advantage of many features from the Java 2 platform, and adds support for more – like Enterprise JavaBeans components, the Java Servlets API and JavaServer Pages.

Figure 5.5 shows a schematic overview of the J2EE multitier architecture as described in the Sun BluePrints Design Guidelines for J2EE [32]. This architecture provides functionality for among other things transaction management, life-cycle management and resource pooling, allowing developers to concentrate on specifics such as business logic and

user interfaces. Figure 5.6 shows the J2EE Standard Enterprise Services. The areas with a cyan shaded background represent concepts used in NewsView, whereas the areas with a red shaded background represent containers (explained shortly)

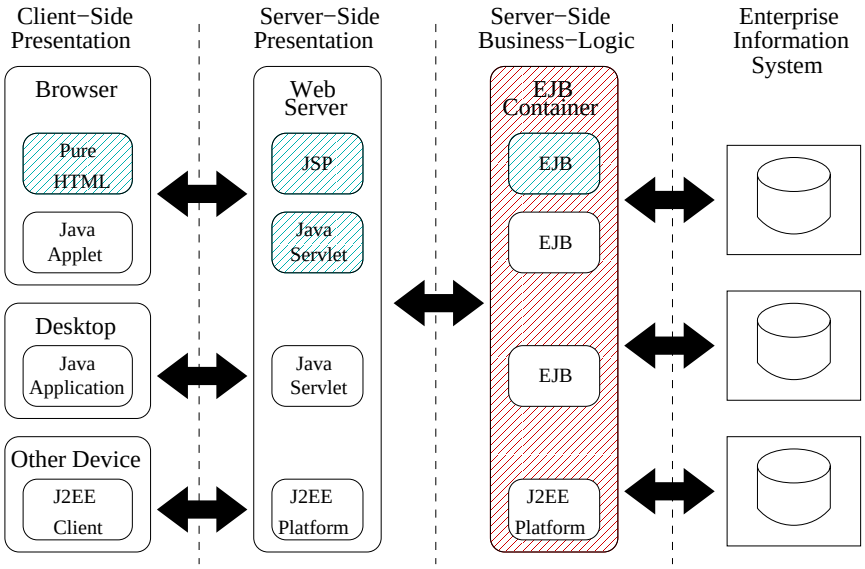


Figure 5.5: The J2EE application model

Components, Containers and Connectors

The J2EE application model splits enterprise applications into three fundamental parts: components, containers and connectors. The purpose of this structure is to hide complexity and enhance portability. Application developers focus mainly on the components, whereas the containers offer services transparently to both components and connectors. Connectors sit at the border of the J2EE platform, defining a portable API to plug into external applications. Figure 5.7 shows how they relate to each other.

Flexible User Interaction

J2EE supports a variety of choices for user interfaces, from Java applets and standard HTML via stand-alone Java application clients to Java Servlets API and JavaServer Pages technology. This allows clients to run on almost any device.

Enterprise JavaBeans

Enterprise JavaBeans (EJB) are server-side, modular, and reusable components constituting specific units of functionality. They are

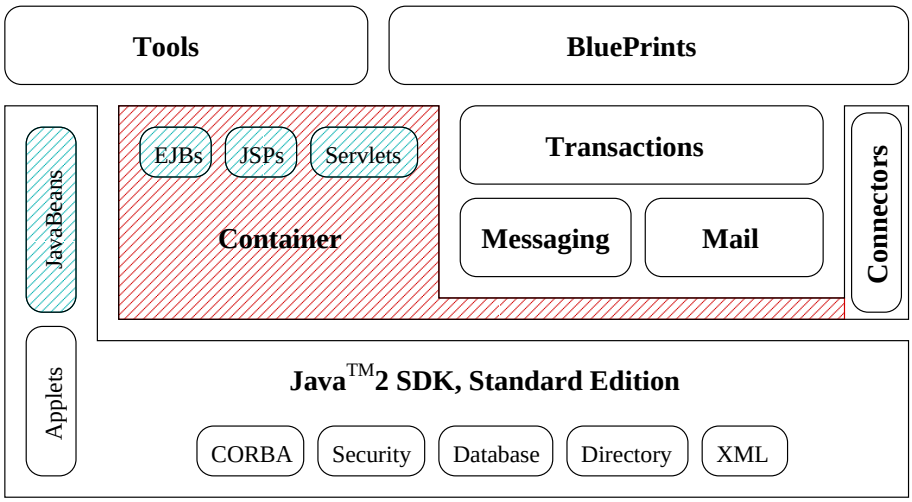


Figure 5.6: The J2EE architecture

similar to normal Java classes, but are subject to special restrictions and must provide specific interfaces for container and client use and access. In addition, they can only run properly in an EJB container, which manages and invokes specific life cycle behaviour.

There are three kinds of enterprise beans:

Session beans

: These may be either stateful or stateless, and are primarily used to encapsulate logic, carry out tasks on behalf of a client, and act as controllers or managers for other beans.

Entity beans

: Entity beans represent persistent objects or concepts existing beyond a specific application's lifetime; they are typically stored in a relational database. Entity beans can be developed using bean-managed persistence, which is implemented by the developer, or container-managed persistence, implemented by the container.

Message-driven beans

: Message-driven beans listen asynchronously for Java Message Service (JMS) messages from any client or component and are used for loosely coupled, typically batch-type, processing.

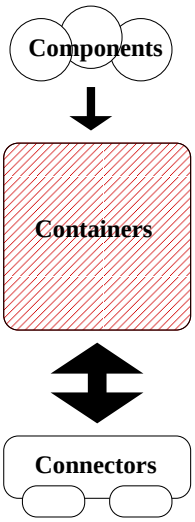


Figure 5.7: The three Cs: Components, containers and connectors

JBoss

JBoss [43] is an Open Source, J2EE standards-compliant application server implemented in Java. It implements the entire J2EE stack, and can integrate with other containers like the Tomcat Servlet/JSP Container or the Jetty Web server/servlet container. As such, it allows for mixing and matching of components to fit the needs of individual applications. The goal of the JBoss project is to provide a full J2EE stack in the Free/Open Source software world.

NewsView uses JBoss with the Tomcat servlet container.

Tomcat

Tomcat [85] is part of the Apache Jakarta project, which is a project of the Apache Software Foundation [84]. Tomcat is the servlet container used in the official reference implementation for the Java Servlet and JavaServer Pages technologies. The Java Servlet and JavaServer Pages specifications are developed by Sun Microsystems, Inc. under the Java Community Process.

5.3.2 The Fast Query Toolkit

The Fast Query Toolkit (FQT) is an example of an integration of the Java version of the FAST Query API (see sections 4.2.2 and 4.2.3 for details)

with an HTTP based search application. As such, it provides an easily accessible approach to integrating FDS functionality into an application with a web-based interface. The core technologies behind FQT are Java servlets, JavaServer Pages, the JSP Standard Tag Library (JSTL) and Java Custom Tags. Combining this foundation with the Model-View-Controller (MVC) design pattern, FQT provides a framework for the creation of web-based front ends to FDS. It is based on Java 1.3.1 or later, and is session oriented.

The following sections present the core principles and technologies on which FQT is built, and the resulting work environment.

Servlets

Servlets [39] are the Java way of doing the equivalent of Common Gateway Interface (CGI) programming. Servlets run on a Web server, acting as an extra layer between an incoming HTTP request and databases or other applications running on the HTTP server. Servlets (or CGI) programs are typically used whenever content returned to the user needs to be generated dynamically on a per request basis.

JavaServer Pages

JavaServer Pages (JSP) [41] enable the ability to mix static HTML with dynamically generated content from servlets. Many web pages built by CGI programs are primarily static, with bits of dynamic content in between. Many technologies allowing the use of dynamic content force you to generate the entire page via your program, whereas JSP lets you create the two parts separately as they can include calls to methods defined in servlets.

JavaServer Pages has spurred the development of a few related tools to ease the development of dynamic web pagers further:

The JavaServer Pages Standard Tag Library (JSTL)

: The JSTL [40] encapsulates core functionality common to many JSP applications as simple tags.

  JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalisation and locale-sensitive formatting tags, and SQL tags. It introduces a new expression language to simplify page development, and it provides an API for developers to simplify the configuration of JSTL tags and the development of custom tags that conform to JSTL conventions.

Custom Tags

Custom tags are user-defined JSP language elements encapsulating recurring tasks. Custom tags are distributed in a tag library, which defines a set of related custom tags.

General FQT Design Principles

Minimise Amount of Java Code in HTML

FQT makes use of the JSTL, which allows for richer logic in JSP pages without introducing Java code. It provides a predefined set of custom tags easing various types of logic occurring frequently when writing JSP.

Minimise Amount of HTML in Java Code

FQT tags attempt to minimise the impact of HTML printed by Java code. The tags do not print markup to the page, as this may be a maintenance problem in case of layout changes. The tags access navigators, modifiers and navigations present in the current query result, iterate over them, and expose them for the JSTL tags accessing the page context.

Another approach is to treat the body of the tag as a template, and exchange certain textual patterns with retrieved values.

MVC – or Model, View, Controller

FQT builds on the MVC design pattern as provided with Smalltalk-80 [29, chapter 1]. This section describes the general concepts and ideas of MVC, as well as how FQT fits into the MVC structure.

The MVC pattern is constituted by three main parts – model, view and controller. Each of these are presented in the following, along with a graphical representation of the pattern, figure 5.8. The figure presents the pattern, emphasising the interaction dynamics internally as well as externally. The figure depicts the different parts of MVC as circles, and shows the interactions between them as labelled arrows.

Model

The model is the core of the application. It maintains the state and data the application represents. When significant changes occur in the model, it updates all of its views.

FQT explicitly models a user's session with the SearchSession class. Each HTTP session, as defined by the servlet API, has exactly one instance of this class.

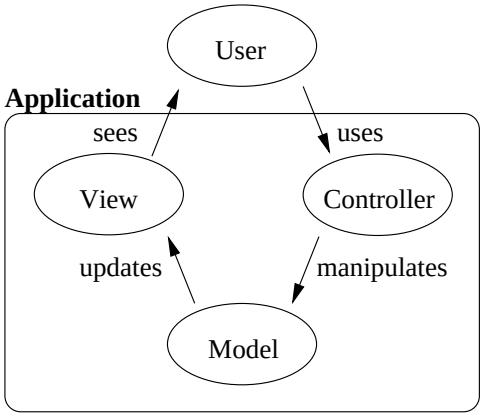


Figure 5.8: Graphical representation of the MVC design pattern

View

The view is the user interface which displays information about the model to the user. Any object needing information about the model needs to be a registered view with the model.

In the case of FQT, the view is represented by the markup produced by the server and sent to the browser for rendering.

Controller

The controller deals with the user interface to record the user's actions and allow manipulation of the application.

FQT implements the Controller as a servlet, known as SearchServlet. It maintains the state of the session object based on input given by HTML forms, which in turn are produced by the view.

For further information on design patterns in general or MVC in particular, consult Gamma et. al. [29]

5.3.3 User Interface

This section describes the design of a user interface for NewsView, giving an overview of the features and possibilities available in the architecture.

The NewsView user interface builds on the idea of a text-based Usenet client, of which the client depicted in figure 2.5 is an example, inside a web browser. In addition to the functionality offered by such traditional interfaces, NewsView includes the idea of recommendations, searching, and a user profile to store the preferences of a user – information-wise

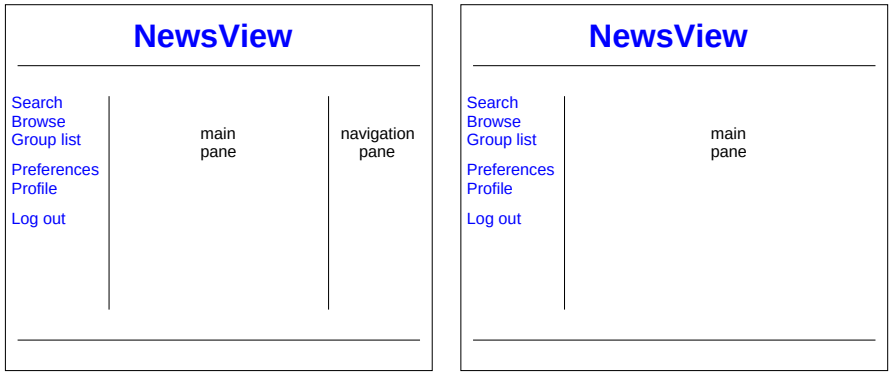
and functionality-wise. It also pays considerable attention to navigation within the data by other means than following the hierarchical Usenet structure, which is not very well supported by conventional Usenet clients.

The user interface treatment given in this section does not relate directly to the material presented in section 2.4.5 on recent interface developments to conversational data, as the main user interface focus of NewsView has been the navigational possibilities within the data, not their visualisation.

The rest of this section gives an overview of the layout of the interface, and the data available for display. It does not outline in any detail the actual views, interface communication flows and so on, as these are not the focus of this thesis.

User Interface Layout

Figure 5.9 shows the on-screen layout of the panes for browsing and searching as opposed to editing preferences and profile information.



(a) Search/browse view (b) Preferences/profile view

Figure 5.9: Sketch of the front end view types

The primary items presented in the user interface are:

Menu pane

A pane for a menu making the basic navigational possibilities available. This is where the mode of the interface is set.

Actions available from this menu:

- Log out of the system

- View and update preferences
- Create and manage the user profile

Main pane

This pane has the main information conveying responsibility. Examples of information presented here, is messages, group overviews and search results.

The main pane will have different information conveying responsibilities depending on the current mode of the interface.

Navigation pane

A pane for navigating the messages displayed – dynamic drill-downs, categorisation and clustering information, search similar, and so on.

User Interface Functionality

There are many ways in which the available data may be visualised in a NewsView user interface. The one shown in the implementation section is only one solution chosen as a prototype, and this thesis does not attempt to present a finalised design in this regard. A list of options, issues and ideas will, however, be presented in this section along with some thoughts about how they can be easily incorporated into the user interface framework as presented here.

Main pane

This pane is employed with two main tasks: presenting the basic interface for browsing and searching, and displaying results exploiting the three predefined results views.

The following navigational options are presented in the main view when applicable:

- Find similar on text query within returned result set.
- Features for excluding and refining results.
- Search within group, hierarchy level or thread id

An issue relevant to the display of result sets, is the visualisation of ratings and relevance as described in section 3.2.1. Messages are, of course, sorted according to relevance when delivered from FDS, but the idea of enabling personalised filtering settings spurs ideas for a second ranking. Messages could be faded in or out according to relevance, or relevant ranking scores could be displayed

along with the message. Colour coding is another option for rank display. These are visualisation options per message, but some options are available on the thread or group level as well. Threads or sub-threads may be collapsed or killed (not displayed at all). Metrics eligible for such display considerations include spam probability, flame probability, author rating and predicted informational value. See section 3.2.2 for more information about possible filtering techniques.

Navigation pane

The main navigational structure of NewsView is dynamic drill-downs. They are defined in the index profile, and results are binned into suitably sized categories depending on what is part of the result set and the drill-down configuration. It is possible to employ a concept called 'results-based binning' in the context of dynamic drill-downs, where all drill-downs are applied at once. This enables a multi-dimensional drill-down, with implicit ranking of dimensions.

A display option enabled by dynamic drill-down is to present a hierarchical group overview display of results for browsing in the navigation pane. Selecting a group in this mode would present the user with a listing of the matching group content grouped by thread id, with all threads collapsed. The number of messages in each thread would be displayed along with the number of hits per thread and a rating of the likelihood of that thread being interesting to the user if available.

Clustering is used for search similar queries, but can also be used to group search results into clusters depending on document similarity (see section 4.2.6). The number of categories returned when applying clustering as well as other parameters considering this feature can be set up in the index profile. When used alongside dynamic drill-downs, clustering might impose an untidy look on the interface. It is important to keep in mind that clustering must serve a practical purpose, not merely obscure the view.

5.3.4 User Concept

The user concept of NewsView allows for profiles storing information preferences as well as user interface display preferences.

Information stored for each user:

- Personalia (username and password)

- Preferences
- Group list
- User information need profile

The only aspect of the user concepts of any relevance to this thesis is user profiles. The storage layout of user information and preferences, as well as the specific details on which user information to store, is ignored for NewsView purposes.

User Profile

User profiles, as described in section 3.2.3, are an important part of any recommender system. The main idea for user profiles within NewsView is to employ alert queries to store a user's information interests. By using this built-in feature of FDS, users are ensured to be notified of any documents relevant to their profile as they become available. This part of the system has not been implemented, and is described here for reference.

Some ideas for user profile building are given below:

Forming an alert query based on a document

: Given a document viewed by the user, enable the possibility to store an alert query based on characteristics of that document. The word vector provides a list of the most important words. This list can be presented to the user, who may choose words, and if feasible, weighting for those words. Upon editing the list, it is submitted as an alert query. Section 3.2.2 gives a description of the vector model which is the basis for this approach.

Forming alert queries based on queries made by user

: Queries issued during a NewsView usage session may be presented to the user at some stage, and submitted as alert queries if deemed appropriate. An option for storing a query as an alert query may also be given in the interface presenting results.

Summary

The purpose of the NewsView framework as presented in this chapter, is to provide a testbed for user interface, information filtering and information retrieval issues, and, most importantly, combinations of the three.

Chapter 6

Implementation

This chapter describes the implementation of NewsView, following the design given in the previous chapter. Several parts of the system described in the design chapter are not implemented in the currently available prototype.

First, this section gives an overview of the development environment for NewsView. Thereafter, an outline of the system structure in an implementation context is given. The rest of the chapter presents the three main stages of NewsView, the Usenet feed in section 6.1, the search engine in section 6.2, and the front end in section 6.3.

Technical System Structure

NewsView is built from several smaller parts. As described in chapter 5, it can be broken down to three main 'stages', and these will be described briefly in this section. The rest of the chapter is devoted to an in-depth presentation of these, following the structure and conventions introduced in chapter 5.

Figure 6.1 gives a graphical representation of the implementation aspects of NewsView. It has the same overall structure and follows the same conventions as figure 5.1, but focuses on implementation details instead of general design considerations.

Usenet Feed

: Usenet articles are pulled from the NNTP server at the University of Oslo (`nntp.uio.no`) and saved to disk using a combination of *fetchnews* [21] and *procmail* [87]. A script called *newsviewpreprocess* does some message preprocessing, before *filetraverser*, a utility bundled with the search engine, passes the resulting files to the search engine for indexing and searching.

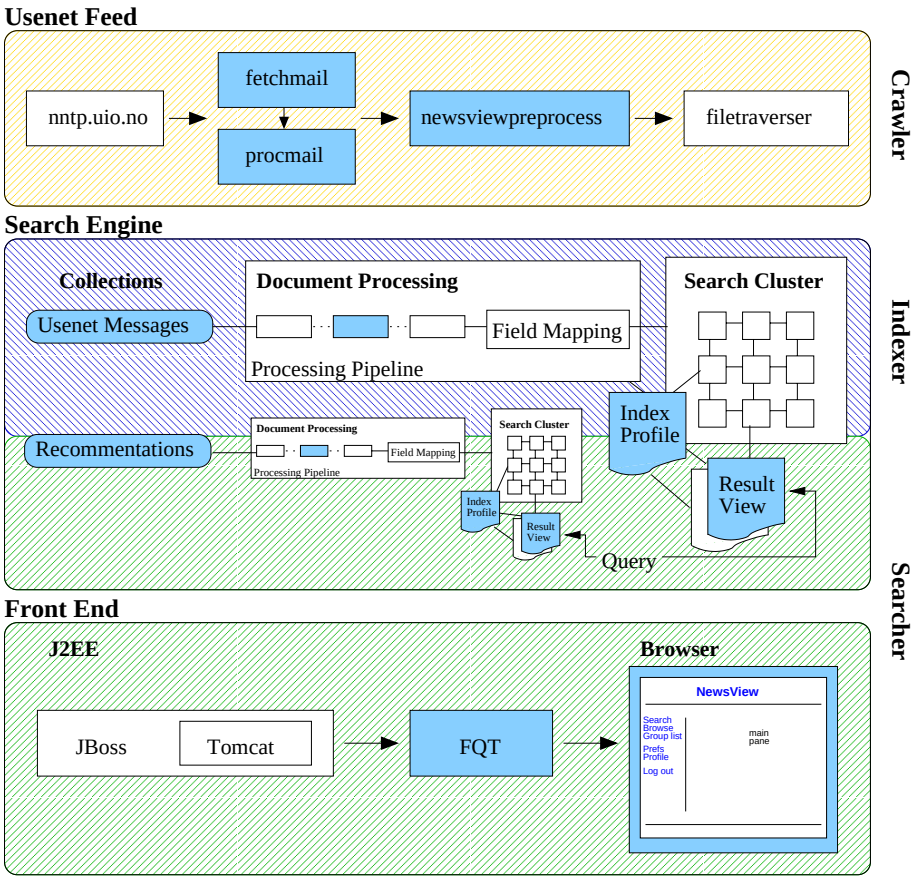


Figure 6.1: Overview of NewsView implementation

Search Engine

The search engine used with NewsView is FAST Data Search (FDS) version 3.2.2. It stores Usenet messages using a customised indexing and document processing configuration, and provides a variety of options for access to the data.

Front End

The NewsView front end provides a searchable interface to Usenet, with extensive navigation possibilities. It is built using J2EE [42], using the JBoss [43] application server, with Tomcat [85] as servlet container. The front end interface itself relies heavily on the FAST Query Toolkit (FQT), which is a development kit provided by FAST Search and Transfer [19]. FQT provides a web-based interface to FDS using Java servlets.

6.1 Usenet Feed

As described in section 5.1, the task of establishing the Usenet feed consists of three steps. Figure 6.2 shows the internal data flow of these steps, and each of them will be described in more detail in this section.

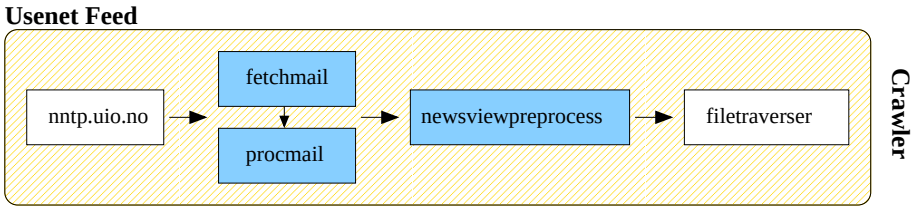


Figure 6.2: NewsView Usenet feed implementation

All operations constituting the Usenet feed are run from a cron job at regular intervals. Logs from all operations are stored in a local log directory, and are rotated using the system utility logrotate [86].

6.1.1 Retrieving Usenet Messages

The first step of creating a Usenet feed into the system is to pull messages from an NNTP server. The choice of NNTP server was made based on convenience and offered material – among the easily available servers, `nntp.uio.no` was the one offering the widest selection of groups.

A perl program called *fetchnews* [21] is used for the download phase. Fetchnews is written by Mathieu Fenniak and Aaron Trickey, and is a simple program to retrieve Usenet messages to a local machine. It uses a configuration and setup syntax well known through the use of *fetchmail*, a program for downloading email to local disk. As such, it is easy to configure, and requires a minimum of effort to reach a running state. Fetchnews does not have the full functionality of an NNTP server, but fills the requirements for a first implementation of NewsView. One of the features not supported by fetchnews in its present condition, is cancelling of messages. This issue is ignored for the purposes of this thesis, and could be resolved by employing a full-fledged NNTP server implementation.

The first approach to downloading Usenet messages to disk, was to set up a local NNTP server. Due to difficulties met when attempting to accomplish this as a non-root user using a non-standard port, this approach was rejected. Fetchnews provided a way to get around this problem, and eliminated the need to create a customised solution.

Fetchnews has support for several delivery methods, for instance SNMP (Simple Network Management Protocol) forwarding, delivery to mbox (a format for mail message storage files) files, or delivery to a Mail Delivery Agent (MDA) – of which *procmail* is an example. NewsView uses procmail to deliver each Usenet message to a file of its own in a flat directory. The files are named according to message-id, as this header field is guaranteed to be unique for each message. The choice of procmail was made because it could very easily fulfil my needs. It is a well-known tool for email sorting, well tested, easily configurable, and made for exactly what was needed in this context.

6.1.2 Preprocessing of Usenet Messages

Upon downloading Usenet messages, some preprocessing is performed before submitting them to FDS.

A perl script called *newsviewpreprocess* was written to take care of the preprocessing needs. The choice of perl as language was made based on the abundance of available perl modules, and the fact that much of the work in the script would be string manipulation, which is one of perls strongest assets.

Usenet messages may be encoded according to the MIME standard (Multipurpose Internet Mail Extensions, see RFCs 2045 [26], 2046 [27] and 2047 [60]). Content encoded according to these standards may require preprocessing to reach a readily accessible state, and newsviewpreprocess takes care of these issues.

Decoding of the message body as well as some header fields may be necessary. All necessary conversions to obtain a decoded state as specified in RFC 2045 [26] and RFC 2047 [60] are performed by newsviewpreprocess. Newsviewpreprocess uses the *MIME::Tools* perl modules to decode MIME encoded content.

Another task performed by newsviewpreprocess, is the insertion of NewsView-specific thread ids into message headers. This id is inserted to enable thread membership as a search criteria. To ensure uniqueness of thread ids, a list of ids already used along with the message-id corresponding to the root message of the thread is kept. This list is stored in a plain text file, along with the next available thread id. Newsviewpreprocess uses the specification given in RFC 1036 [36] to deduct the correct thread id for a message. As thread ids are stored between program runs, newsviewpreprocess handles resubmitting of articles.

Newsviewpreprocess also inserts a header field containing default message rating values.

Figure 6.3 shows pseudo code for the thread id insertion algorithm used. The pseudo code shows the operations as performed for each message.

```
if ( message has references field )
   # Message is part of an already existing thread
   master reference = first message id in references field
   if ( master reference exists in id list )
      # Thread has been seen before, insert known id
      thread id of message = master reference
   else
      # Thread has not been seen before, new thread id is created
      insert master reference into id list with new thread id
      thread id of message = new thread id
   endif
else
   # First message in new thread, new thread id
   if ( message has been seen before )
      # Message is resubmitted, use old thread id
      retrieve old thread id from id list
      thread id of message = old thread id
   else
      # New message, new thread id.
      insert message id into id list with new thread id
      thread id  of message = new thread id
   endif
endif
```

Figure 6.3: Pseudo code for thread id insertion algorithm

6.1.3 Submitting Usenet Messages to FAST Data Search

As described in section 4.2.2, there are several ways to submit content to FDS. For my application, the obvious choice of input method is *filetraverser*. It traverses a directory for files matching given criteria, and submits those files to FDS. Filtering is done based on filename suffix.

The documentation for filetraverser is given in the FDS System Reference Guide [20]. In the case of a system where cancelling of messages is to be supported, filetraverser would support this readily through its ability to remove documents from a named collection.

After submitting articles to FDS, files containing processed content are moved into an archive folder by means of the standard UNIX system utilities *mv* and *xargs*. Xargs is involved in the process to ensure that moving a large number of files (there is a limit on the number of arguments that can be given to a program) will work correctly.

6.2 Search Engine

This section will focus on the configuration of FDS for my purposes. The material covered in section 4.2 in general, and 4.2.5 in particular is an essential background for the treatment given in this section. The version of FDS used with NewsView is 3.2.2.

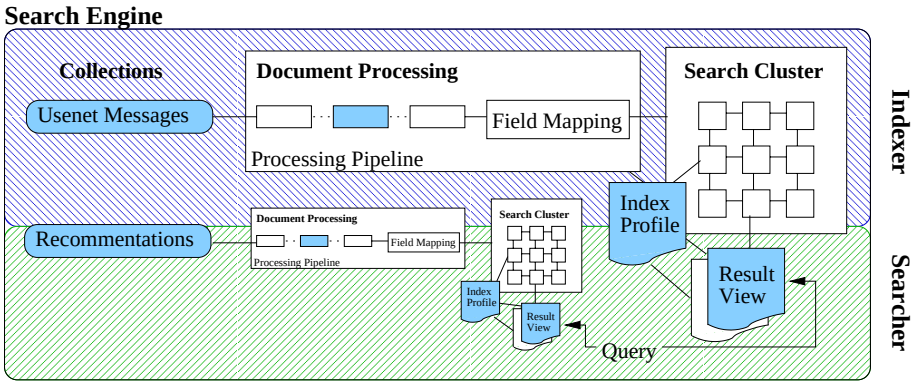


Figure 6.4: NewsView search engine implementation

As described in section 4.2.5, each collection has an index profile and a document processing pipeline assigned to it. To meet the requirements specified by the design in section 5.2, the following aspects of FDS must be configured:

- The index profile
- The document processing
- The taxonomy toolkit

Figure 6.4 shows an overview of the structure of the search engine implementation. The following sections will outline setup for each of these. The recommender collection is included in figure 6.4, but is not part of the implemented system because of hardware limitations. As such, it is not included in the description given in this chapter and any treatment is to be considered specific to the Usenet message collection unless otherwise stated.

6.2.1 Index Profile

The index profile is an XML-based configuration file defining the way documents are searchable. It defines which document elements become

searchable fields, which document elements become fields that can be returned as part of a result, and how to calculate values used for sorting and ranking. The deployed index profile is included in appendix C.1.

The index profile configuration for NewsView will be presented as a list of concepts, following the structure of the XML format as presented in figure 4.8.

Simple Information Storage Fields

Simple information storage fields are used to store document data, including extracted document parts as well as meta-information.

Lemmatisation, the act of breaking a word down to its root form, is enabled for fields containing natural language text.

The following data items are kept in simple information storage fields:

Document information

- Title, body, headings and anchor text.
- All Usenet message header fields defined in RFC 1036 [36]
- Message date in uniform format
- Body and all original headers in html formatted form

Document meta-information

Content-type, language, character set, URL, modification time, size, ranking, teaser and taxonomy membership.

Composite Fields

Composite fields are used for grouping of simple fields, to allow them to be accessed as one. Each composite field can have a ranking specific to that field.

Composite fields configured and used are:

Content

Allows access to all content, that is, body, headings, title and anchor text, as one field.

Result Specification

These fields are used to specify various aspects of the results produced by the search engine. They are listed in the order they appear in the index profile, and details are given where applicable. When fields are mentioned with no further specification attached, they are included for completeness and reference but their configuration is not particularly relevant to NewsView.

Result filter NewsView uses one result filter, which removes duplicates based on the URL field. As the URL is built using the message-id, this filter ensures no document may be present more than once in a result set.

Taxonomy, clustering and proximity boost setup

Navigators

The navigators defined in the index profile are shown in table 6.1.

Drill-down	Displayed name (unit and resolution)
size	Document Size (kB/1024)
content type	MIME Type
character set	Character set
language	Language
thread id	Thread Id
date	Date (month/#seconds in thirty days)

Table 6.1: Navigators as configured in index profile

Result views

Several result views are set up. Table 6.2 shows their names and member fields:

Name	Member Fields
Result overview	author, date, subject, teaser
Message summary	author, date, subject, newsgroups, body
Message extended	all header fields, body

Table 6.2: Views as configured in index profile

6.2.2 Document Processing Pipeline

FDS comes bundled with several predefined processing pipelines. None of these fulfilled the requirements made by NewsView in themselves, but they provided a good starting point for a NewsView-specific customised pipeline. Such a customised pipeline, called *UsenetNewsView*, was set up. It includes functionality to handle the specifics of the document format as well as other features needed.

UsenetNewsView is based primarily on two predefined pipelines called *MultiConverter* and *Taxonomy*. MultiConverter is a multi purpose pipeline supporting many file formats, accomplishing this by converting all content into an intermediate element format, which is processed as HTML. The Taxonomy pipeline is similar, but focuses on taxonomy functionality.

The *ExternalDataFilterTimeout* document processor is part of FDS and makes use of an external utility to process an attribute. This is done subject to a timeout, and the result is stored in a given field. This concept is used as the basis for all customised document processing in NewsView, except some tuning of FDS features done by customising specialised document processors bundled with the system. FDS assumes all content submitted to it within the document processing system to be encoded in UTF-8. This implies the need to convert all non-UTF-8 content before submitting it. All customised document processors included in NewsView do this.

The following list contains the customised document processing as configured for NewsView:

Extraction of Usenet header fields

> All Usenet header fields as defined by RFC 1036 [36] are extracted by means of document processors based on ExternalDataFilterTimeout. They are extracted by a perl script, and converted to UTF-8 afterwards. The *MIME::Tools* modules are used to extract headers, and the *Text::Iconv* module does character set and encoding conversion. The latter is a perl wrapper around the system call *iconv*, which is part of glibc. To avoid recreating the script for each document processor, it takes the name of the header field as an argument, and returns the appropriate text string.
>
> Quite a few such processors are needed (twenty, to be exact), and as the FDS document pipeline configuration is stored in an XML file, a perl script is used to generate the necessary XML code for these processors.

Extraction of ranking information from header field

Ranking information is extracted and stored in appropriate fields when messages are resubmitted. Customised ranking is not implemented in the current version of NewsView.

Converting the contents of the 'Date' header field to a uniform format

To allow the use of dynamic drill-downs on date information of messages, the Usenet header field extracting script mention earlier has a special case for date extraction. In addition to extracting the date field and storing it in a text field, it is converted into a uniform format, which is chosen to be the number of seconds since 1st of January 1970 (epoch), and stored in a special 'uniformdate' field.

Converting all original message headers and body into html

Two html-formatted blocks of text are stored for each message. One contains all header fields from the original message, and the other contains the message body. The body is converted using *txt2html*, whereas the headers are converted by augmenting every newline with a $< br >$ tag.

Configuration of the vectorising and taxonomy document processing facilities

The taxonomy toolkit uses a document processor to input the name of the taxonomy. The customised processor *UsenetRuleClassifier* performs this task.

The *vectorizer* processor specifies which fields to include when creating the document vector. It is not customised in the present version of NewsView, but as it provides an important possibility to tune the document comparison process, it is mentioned for reference.

Rather late in the process of writing this thesis, documentation on how to write full-fledged document processors to fit right into the pipeline became available. This would undoubtedly have been a better solution for the inclusion of NewsView-specific processing tasks.

All relevant tasks could then be collected in one processor as opposed to being spread across approximately twenty. This processor would detect the correct MIME type (text/news) and perform the appropriate processing tasks accordingly.

Table 6.3 gives an overview of the processing stages present in UsenetNewsView. Amongst other things, it contains language detection, encoding normalisation and a teaser generator.

Objective	Stage	Description
Retrieval and initialisation	DocumentRetriever	Retrieves the document.
	URLProcessor	Handles URLs.
	DocInit	Initialises document attributes without overwriting existing values.
	FormatDetector	Detects the format of the data attribute of a document.
	SimpleConverter	Converts the content of a document's data attribute to HTML.
	SearchMLConverter	Converts the content of a document's data attribute in any supported file format to HTML.
	LanguageAnd-EncodingDetector	Detects language and encoding of HTML.
	EncodingNormalizer	Ensures all HTML in data is converted to UTF-8.
	HTMLParser	Parses the HTML and extracts meta data.
	TeaserGenerator	Generates a teaser summarising the document based on the description meta data or the body.
Extraction of Usenet header fields	UsenetFrom	Extracts the *From* header from Usenet messages.
	UsenetDate	Extracts the *Date* header from Usenet messages.
	UsenetNewsgroups	Extracts the *Newsgroups* header from Usenet messages.
	UsenetSubject	Extracts the *Subject* header from Usenet messages.
	UsenetMessage-ID	Extracts the *Message-ID* header from Usenet messages.
	UsenetPath	Extracts the *Path* header from Usenet messages.
	UsenetFollowup-To	Extracts the *Followup-To* header from Usenet messages.
	UsenetExpires	Extracts the *Expires* header from Usenet messages.
	UsenetReply-To	Extracts the *Reply-To* header from Usenet messages.

Objective	Stage	Description
	UsenetSender	Extracts the *Sender* header from Usenet messages.
	UsenetReferences	Extracts the *References* header from Usenet messages.
	UsenetControl	Extracts the *Control* header from Usenet messages.
	UsenetDistribution	Extracts the *Distribution* header from Usenet messages.
	UsenetKeywords	Extracts the *Keywords* header from Usenet messages.
	UsenetSummary	Extracts the *Summary* header from Usenet messages.
	UsenetApproved	Extracts the *Approved* header from Usenet messages.
	UsenetLines	Extracts the *Lines* header from Usenet messages.
	UsenetXref	Extracts the *Xref* header from Usenet messages.
	UsenetOrganization	Extracts the *Organization* header from Usenet messages.
	UsenetX-NewsView-Threadinfo	Extracts the *X-NewsView-Threadinfo* header from Usenet messages.
Taxonomy classification	ManualMapper	Performs manual classification of documents into a predefined taxonomy.
	UsenetRuleClassifier	Rule-based classifier. Loads rules from an XML file and assigns categories for all matching rules.
	TaxonomyTagger	Tags documents with supplementary taxonomy information.
	TaxonomyBoost	Performs category boosting, boosting all documents belonging to a specific category.
Linguistic processing	Lemmatizer	Lemmatises selected attributes.
	Vectorizer	Computes the document vector from the attributes defined in the input parameter.

Objective	Stage	Description
FIXML conversion and submittal	FIXMLGenerator	Generates FIXML data, an internal document format.
	RTSOutput	Handles output to the Search Engine.

Table 6.3: The stages of the UsenetNewsView processing pipeline

6.2.3 The Taxonomy Toolkit

The taxonomy toolkit, described in section 4.2.7, is used to create a category hierarchy corresponding to the Usenet hierarchy structure. The resulting hierarchy contains those groups present in NewsView at any given time. To make sure all messages are part of at least one category, the taxonomy must be updated every time new groups are added to the repository. Implementation of search result browsing is simplified by knowing that all messages returned as part of a result will be included in at least one category. Figure 6.5 shows the taxonomy toolkit browser at work.

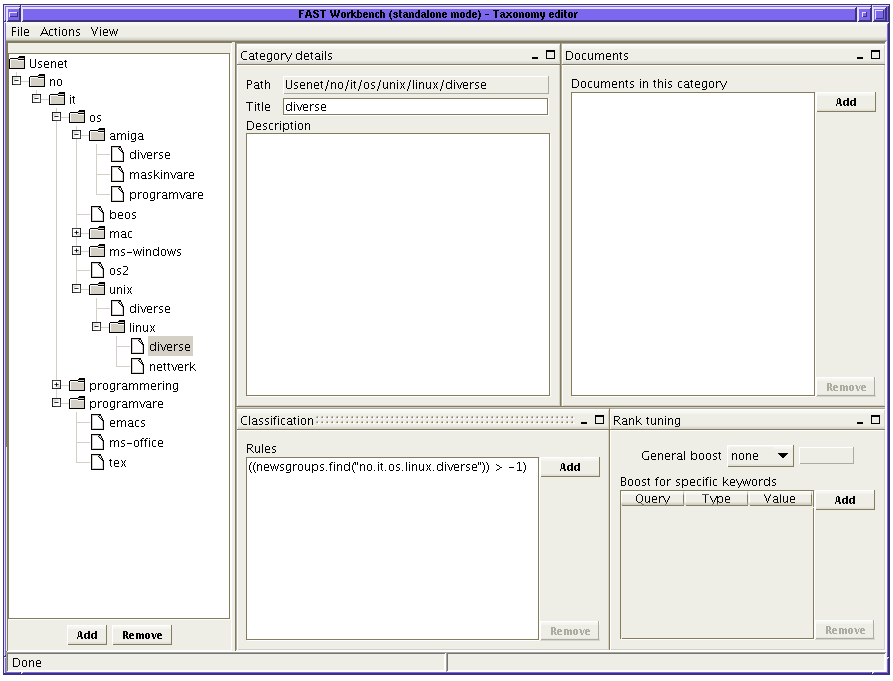


Figure 6.5: The taxonomy toolkit interface

6.2.4 Resubmitting Content to FAST Data Search

The approach described in this section is not implemented due to time constraints. A description is, however, included as an outline of how the process would be performed.

When new or updated information about a message is available, this information has to be propagated to FDS to make sure the data stored in the search engine reflects any changes. This will happen when a new recommendation is submitted, or an existing one is updated. Implicit and explicit ratings are included in this treatment.

Upon receiving new recommendations, the message concerned is retrieved from the archive. The header field containing ratings is updated, and the recommendation is formatted and prepared. The resulting files are copied to specific folders for updated content, from where they are picked up by a cron job submitting them to the search engine using filetraverser. The files are deleted from the update folder after being submitted.

The search engine within FDS is batch oriented. Incoming documents are indexed and initially inserted into a 'new' index. Batches can be of any size in number of documents, and new indexes can be generated as often as every second. They are periodically forwarded to the main index from there. This indexing cycle can be bypassed by defining fields as real-time property attributes in the index profile. An idea for the resubmitting strategy is to define the rating attribute of messages as real-time properties, and let the recommendations go though the normal indexing cycle.

6.3 Front End

NewsView provides a framework within which interfaces can be easily created, customised and tested. As such, NewsView can serve as a testbed for user interface, information filtering and information retrieval issues, and, most importantly, combinations of the three. The data access and storage functionality is made available through the search engine configuration, and the display mechanisms are provided by FQT. Cascading style sheets (CSS) are used to define the visual presentation of the interface.

The front end is, as shown in figure 6.6 and described in section 5.3, built using JBoss, Tomcat and FQT. The actual interface is viewed using a Web browser.

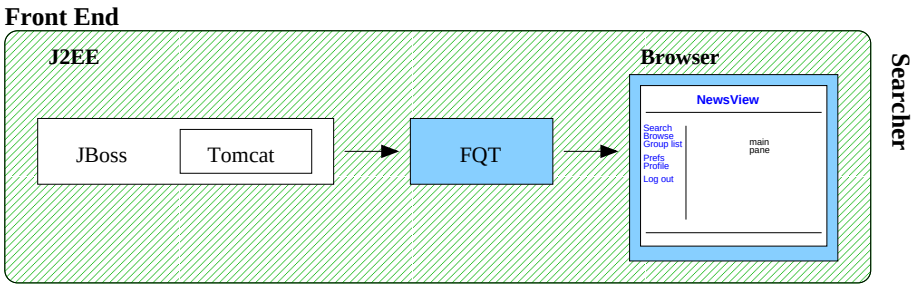


Figure 6.6: NewsView front end implementation

The 4.x releases of Tomcat, of which version 4.1.24 has been used for NewsView, implement the Java Servlet 2.3 and JavaServer Pages 1.2 specifications. The version of the Java 2 runtime environment used was 1.4.0_01.

6.3.1 FQT

FQT provides two types of tags. Simple on/off tags are used to set boolean values, whereas value selection tags are used for generic configuration strings.

There are tags for controlling input, formatting and displaying documents, displaying and utilising dynamic drill-downs and clustering, configuring various search engine features and displaying debug information.

The availability of html-formatted versions of the message body and headers eases the message display process. Appendix B contains screen shots of some possible NewsView interfaces.

6.4 The NewsView Development Environment

Several general concepts and methodologies have been kept in mind throughout the design and implementation of NewsView. Some such concepts are: Adhering to standards, portability, design patterns, object-oriented programming paradigms and technology, design for extensibility and reuse of existing tools.

NewsView was developed on a PC running Red Hat Linux version 7.3, where a local disk partition was made available for the project. The entire system, Usenet feed, search engine and front end, was developed, run and tested on this system. As such, all tools are compliant with it

– NewsView is, however, not particularly platform dependent. The tools constituting the Usenet feed would require some adjustment to run on another platform, and some of them would have to be replaced. The UNIX-specific tools used in the document processing would disappear if a proper FDS document processing module was implemented. The user interface as presented is close to platform and browser independent. It is based on HTML, and uses CSS for some formatting. All processing is done server side, and the HTML is as close to W3C compliant as possible given the starting point.

Summary

This chapter describes the implemented parts of the NewsView architecture. The discussion includes directions for some possible extensions to this implementation.

Chapter 7

Conclusion

This thesis combines aspects from two approaches to information access, information filtering and information retrieval, in an effort to improve the signal to noise ratio in interfaces to conversational data. These two ideas are blended into one system by augmenting a search engine indexing Usenet messages with concepts and ideas from recommender systems theory. The aim is to achieve a situation where the overall result relevance is improved by exploiting the qualities of both approaches. Important issues in this context are obtaining ratings, evaluating relevance rankings and the application of useful user profiles.

The NewsView architecture as presented in this thesis describes a framework for interfaces to Usenet with information retrieval and information filtering concepts built into it, as well as extensive navigational possibilities within the data. The purpose of this framework is to provide a testbed for user interface, information filtering and information retrieval issues, and, most importantly, combinations of the three.

The storage structure is adjusted to the specific structural properties of Usenet messages. Specific information filtering needs and ideas can be fitted easily into the structure by means of document processors. Various approaches to content categorisation and filtering are supported, as well as navigation within the resulting structures. Sample features provided are serendipity through the use of clustering and thread-based result binning to keep context easily available when searching discussions.

The back end draws on crawler theory, using a combination of methods to obtain freshness. When submitting articles and recommendations, the batch-oriented back end is used. Content is sent to the search engine using a crawler traversing local file systems. Resubmitted articles are indexed using the same back end, whereas the ranking information is

propagated directly to the search engine using special real-time features of the search engine.

During my thesis work, I have shifted the focus away from creating a finished interface, and towards creating a general framework. As a consequence of this, considerations regarding signal to noise ratio are not concretely discussed in a NewsView perspective. In its current form, NewsView cannot be used to facilitate evaluation of result relevance predictions. It is, however, suitable as a foundation for systems holding these qualities.

Appendix A

Future Work

In the following, I outline various possibilities for future work in the extension of this thesis. Some are concrete improvements of the architecture described, and some are general research ideas that can be realised using the NewsView architecture or by other means.

Concrete improvements of the implementation presented

FQT (or a separate) tag library can be equipped with functionality for Usenet message threading. Having a tag or set of tags perform this task would make a threaded result display easily available.

The potential for further integration of ideas from recommender systems theory is undoubtedly great. Some such ideas I would have liked to include, are implicit ratings, entry of explicit ratings and user collaboration.

Browsing and searching in an information retrieval context

According to Baeza-Yates and Ribeiro-Neto [3], not very much research has been undertaken when it comes to combining the information retrieval user tasks, that is, searching and browsing. The NewsView architecture as described provides a feasible environment in which such issues can be explored.

Usenet domain-specific filtering

Combining NewsView explicitly with the work of Whittaker et. al. [88] and Smith and Fiore [23] opens exiting possibilities for social filtering of Usenet content:

- Automatically generated ratings could be incorporated into the ranking algorithm directly, as a supplement to the ratings used in conventional recommender systems.

- Both Whittaker et. al. [88] and Smith and Fiore [23] present approaches based on mining message headers. Including processing of message bodies would be easy if these ideas were to be deployed within the NewsView architecture, as the entire message is indexed as an inherent property of the architecture.
- Combining author ratings as presented by Smith and Fiore [23] with author-based filtering in a recommender system context may prove particularly effective.

Related research ideas

As of today, no notion of relationships between users is included in the NewsView design. Employing ideas related to active collaborative filtering should provide further improvement as a supplement to the social filtering concepts emphasised by this thesis.

The work of Cosley et. al. [13] looks into how recommender interfaces may affect users in the rating and consumption process. Their work would be an important background for recommender systems interface design using NewsView.

Recent efforts from Herlocker [33] review the key decisions in evaluating collaborative filtering recommender systems, and present empirical results from the analysis of various accuracy metrics. Experiences from this work may prove valuable to recommender systems design in general, as well as providing a starting point for an evaluation toolkit that could be included into NewsView.

Appendix B

NewsView Screen Shots

This appendix contains some screen shots of possible NewsView interfaces.

The screen shot presented in figure B.1 shows a basic search interface to Usenet created using NewsView.

Figure B.1: Screen shot of a NewsView-based search interface to Usenet

The screen shot presented in figure B.2 shows a result view created with NewsView, showing the results and some navigational options.

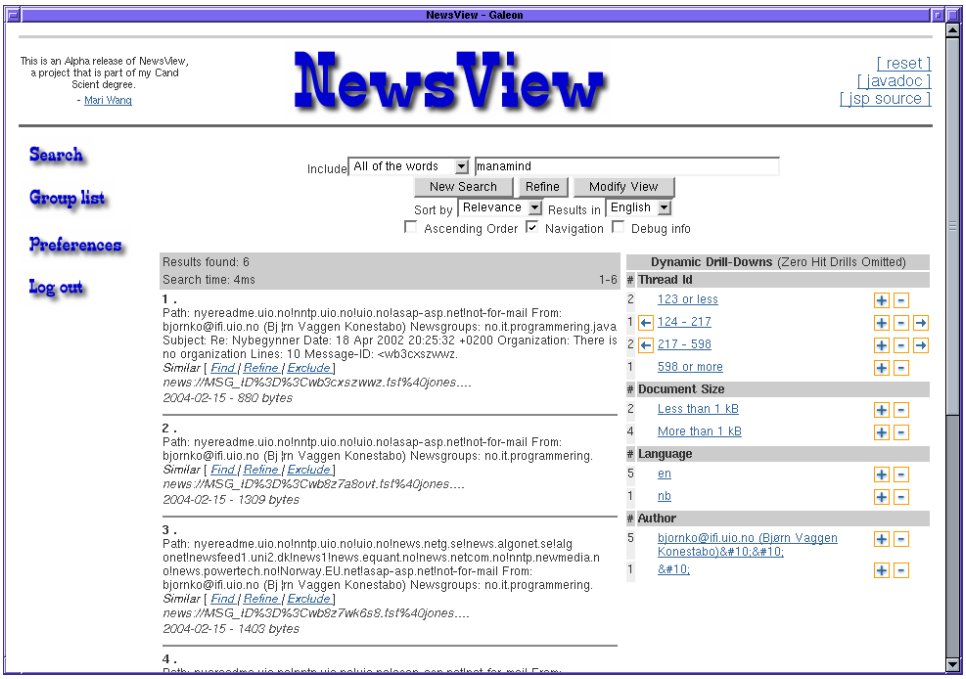


Figure B.2: Screen shot of a NewsView-based result view

Appendix C

Source Code

C.1 Index Profile

Listing C.1: NewsView Index

```
<?xml version="1.0"?>
<!DOCTYPE index-profile SYSTEM "index-profile-2.0.dtd">
<index-profile name="datasearch">
<field-list>

  <!-- left for compatibility reasons, not used -->
  <field name="description" result="no" />
  <field name="keywords" result="no" />

  <field name="title" sort="yes" element-name="toktitle"
         result-element-name="title" lemmas="yes"
         result-proximity-element-name="resulttitle" />

  <field name="body" element-name="tokbody" max-result-size="1024"
         compress="yes" result-element-name="resultbody"
         fallback-ref="teaser" lemmas="yes" result="dynamic" index="no" />

  <field name="teaser" index="no" />

  <field name="headings" element-name="tokheadings"
         result-element-name="headings" lemmas="yes" />

  <field name="anchortext" result="no"
         result-proximity-element-name="anchortext" />

  <field name="contenttype" element-name="mime"
         result-element-name="contenttype" />

  <field name="language" element-name="languages" />

  <field name="charset" />

  <!-- left for compatibility reasons, not used -->
  <field name="urls" />

  <field name="url" element-name="meta.url" index="no"/>
```

```
    <!-- left for compatibility reasons, not used -->
    <field name="domain" element-name="meta.url.domain" result="no" />
    <field name="tld" element-name="meta.url.tld" result="no" />
    <field name="path" element-name="meta.url.path" result="no" />

  <!-- Non-text fields -->
    <field name="modified" type="integer" sort="yes" />
    <field name="size" type="integer" sort="yes" />

    <!-- left for compatibility reasons, not used -->
    <field name="generic1" />
    <field name="generic2" />
    <field name="generic3" />
    <field name="generic4" result="no" />
    <field name="igeneric1" type="integer" sort="yes" />

  <!--  Taxonomy toolkit support -->
    <field name="taxids" type="string" />
    <field name="taxcatids" type="string" />

  <!--  Fields added by mariwan to accomodate for usenet postings -->

    <field name="from" />
    <field name="date" />
    <field name="newsgroups" />
    <field name="subject" lemmas="yes" />
    <field name="messageid" />
    <field name="newspath" />
    <field name="followupto" />
    <field name="expires" />
    <field name="replyto" />
    <field name="sender" />
    <field name="approved" />
    <field name="lines" type="integer" />
    <field name="xref" />
    <field name="organisation" />
    <field name="references" />
    <field name="xnewsviewthreadinfo" type="integer" />
    <field name="uniformdate" type="integer"/>
    <field name="htmlheaders" />
    <field name="htmlbody" />

  <!-- Field to store message in html format -->

    <field name="html" />

  </field-list>

  <composite-field name="content" rank="yes" default="yes" lemmas="yes">
    <field-ref name="body" level="1" />

    <field-ref name="headings" level="2" />
    <field-ref name="path" level="2" />

    <field-ref name="domain" level="3" />

    <field-ref name="title" level="5" />

    <field-ref name="anchortext" type="external" level="6" />

    <weight-list name="defaultrank" and-boost="0" or-boost="5000"
```

```
                rank-boost="5000" phrase-boost="5000" >
      <field-weight field-ref="title" value="150000" />
      <field-weight field-ref="body" value="10000" />
      <field-weight field-ref="headings" value="50000" />
      <field-weight field-ref="domain" value="150000" />
      <field-weight field-ref="path" value="50000" />
      <field-weight field-ref="anchortext" value="300000" />
  </weight-list>
</composite-field>

<result-specification>

<!-- Remove duplicates for results with equal urls -->
  <result-filter name="duplicateurlremover" type="duplicate">
    <field-ref name="url" />
  </result-filter>

  <categorization name="mytopics">
    <field-ref name="taxids" />
  </categorization>

  <clustering size="10" depth="5" threshold="0.30" />

<!-- Result proximity boosting: Set to "yes" to enable boosting per -->
<!-- default, set to "no" to generate necessary config to allow boosting -->
<!-- on a per query basis but have boosting off per default -->

  <result-proximity boost="no">
    <field-ref name="body" />
    <field-ref name="title" />
    <field-ref name="anchortext" />
  </result-proximity>

  <integer-navigator name="sizenavigator"
                     display="Document Size"
                     unit="kB"
                     divisor="1024"
                     resolution="1024">
    <field-ref name="size" />

    <range-label type="first" format="Less than %g kB" offset="1" />
    <range-label type="middle" format="%g kB - %g kB" />
    <range-label type="last" format="More than %g kB" />

    <ignore-value value="0" />
  </integer-navigator>

  <integer-navigator name="threadidnavigator"
                     display="Thread Id">
    <field-ref name="xnewsviewthreadinfo" />
  </integer-navigator>

  <integer-navigator name="datenavigator"
                     display="Date"
                     unit="month"
                     divisor="2592000"
                     resolution="2592000">
    <field-ref name="uniformdate" />
  </integer-navigator>

  <string-navigator name="contenttypenavigator" display="MIME Type">
    <field-ref name="contenttype" />
```

```
  </string-navigator>

  <string-navigator name="charsetnavigator" display="Character set">
    <field-ref name="charset" />
  </string-navigator>

  <string-navigator name="languagenavigator" display="Language">
    <field-ref name="language" />
  </string-navigator>

  <string-navigator name="fromnavigator" display="Author">
    <field-ref name="from" />
  </string-navigator>

  <result-view name="resultoverview">
    <field-ref name="from" />
    <field-ref name="date" />
    <field-ref name="subject" />
    <field-ref name="teaser" />
  </result-view>

  <result-view name="messagesummary">
    <field-ref name="from" />
    <field-ref name="date" />
    <field-ref name="subject" />
    <field-ref name="newsgroups" />
    <field-ref name="body" />
  </result-view>

  <result-view name="messageextended">
    <field-ref name="html" />
  </result-view>

</result-specification>

</index-profile>
```

Bibliography

[1] Amazon.com. http://www.amazon.com. Amazon.com, Inc.

[2] Christopher Avery and Richard Zeckhauser. Recommender systems for evaluating computer messages. *Communications of the ACM*, 40(3):88–89, March 1997.

[3] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[4] M. Balabanovic and Y. Shoham. Fab: Content-based collaborative recommendation. *Communications of the ACM*, 40(3):66–72, March 1997.

[5] Patrick Baudisch. *Dynamic Information Filtering*. PhD thesis, GMD Forschungszentrum Informationstechnik GmbH, Sankt Augustin, 2001. ISSN 1435-2699, ISBN 3-88457-399-3.

[6] Nicholas J. Belkin and W. Bruce Croft. Information filtering and information retrieval: Two sides of the same coin? *Communications of the ACM*, 35(12):29–38, 1992.

[7] binnews. http://www.binnews.com/. BINNEWS LLC.

[8] Danah Boyd, Hyun-Yeul Lee, Daniel Ramage, and Judith Donath. Developing legible visualizations for online social spaces. In *Proceedings of the Hawaii International Conference on System Sciences*, 2002.

[9] Tim Bray. Measuring the web. In *Proceedings of the fifth international World Wide Web conference on Computer networks and ISDN systems*, pages 993–1005. Elsevier Science Publishers B. V., 1996.

[10] Brian E. Brewington and George Cybenko. How dynamic is the Web? *Computer Networks (Amsterdam, Netherlands: 1999)*, 33(1–6):257–276, 2000.

[11] Junghoo Cho and Hector Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *Proceedings of the Twenty-sixth International Conference on Very Large Databases*, 2000.

[12] Junghoo Cho and Hector Garcia-Molina. Estimating frequency of change. *ACM Trans. Inter. Tech.*, 3(3):256–290, 2003.

[13] D. Cosley, S.K. Lam, I. Albert, J. Konstan, and J. Riedl. Is seeing believing? how recommender systems influence users' opinions. In *Proceedings of CHI 2003 Conference on Human Factors in Computing Systems*, pages 585–592, 2003.

[14] David H. Crocker. rfc822: Standard for the format of arpa internet text messages, 1982.

[15] Cyberalert. http://www.cyberalert.com. CyberAlert Inc.

[16] Judith Donath. A semantic approach to visualizing online conversations. *Communications of the ACM*, 45(4):45–49, 2002.

[17] Judith Donath, Karrie Karahalios, and Fernanda Viegas. Visualizing conversation. In *Proceedings of the 32nd Hawaii International Conference on System Sciences*. IEEE, 1999.

[18] Thomas Erskine and Jason R. Mastaler. Tagged message delivery agent. http://tmda.net/.

[19] Fast data search. http://www.fast.no/en/products/fast_data_search. Fast Search & Transfer Inc.

[20] Fast data search system reference guide. FAST Search and Transfer Inc.

[21] Mathieu Fenniak and Aaron Trickey. Fetchnews. http://files.moo.ca/~laotzu/fetchnews.html.

[22] Andrew Fiore and Marc Smith. Tree map visualizations of newsgroups. Technical report, MRS-TR-2001-94, 2001.

[23] Andrew T. Fiore, Scott Lee Tiernan, and Marc A. Smith. Observed behavior and perceived value of authors in usenet newsgroups: Bridging the gap. In Loren Terveen, Dennis Wixon, Elizabeth Comstock, and Angela Sasse, editors, *Proceedings of the CHI 2002 Conference on Human Factors in Computing Systems (CHI-02)*, pages 323–330, New York, April 20–25 2002. ACM Press.

[24] Gerhard Fischer and Curt Stevens. Information access in complex, poorly structured information spaces. In *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems*, Information Retrieval, pages 63–70, 1991.

[25] J.D. 'Illiad' Frazer. User friendly the comic strip. http://www.userfriendly.org/.

[26] N. Freed. rfc2045: Multipurpose internet mail extensions (mime) part one: Format of internet message bodies, 1996.

[27] N. Freed. rfc2046: Multipurpose internet mail extensions (mime) part two: Media types, 1996.

[28] S. Gabrielli and S. Mizzaro. Negotiating a multidimensional framework for relevance space, 1999.

[29] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, New York, NY, 1995.

[30] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, December 1992.

[31] Google groups. http://groups.google.com/. Google. Inc.

[32] Guidelines, patterns, and code for end-to-end java applications. http://java.sun.com/blueprints/guidelines/index.html. Sun Microsystems Inc.

[33] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004.

[34] Will Hill and Loren Terveen. Using frequency-of-mention in public conversations for social filtering. In *Proceedings of ACM CSCW'96 Conference on Computer-Supported Cooperative Work*, Filtering & Sharing, pages 106–112, 1996.

[35] William C. Hill, Jim Hollan, Dave Wroblewski, and Tim McCandless. Edit wear and read wear. In Penny Bauersfeld, John Bennett, and Gene Lynch, editors, *Proceedings of the Conference on Human Factors in Computing Systems*, pages 3–10, New York, NY, USA, May 1992. ACM Press.

[36] M. Horton and R. Adams. rfc1036: Standard for interchange of usenet messages, 1987.

[37] B. A. Huberman and L. A. Adamic. Evolutionary dynamics of the world wide web. *Nature*, 401:131, September 1999.

[38] Lars Magne Ingebrigtsen. Gmane - mail to news and back again. http://www.gmane.org/.

[39] Java servlet technology. http://java.sun.com/products/servlet/. Sun Microsystems Inc.

[40] Javaserver pages standard tag library. http://java.sun.com/products/jsp/jstl/. Sun Microsystems Inc.

[41] Javaserver pages technology. http://java.sun.com/products/jsp/. Sun Microsystems Inc.

[42] Java™2 platform, enterprise edition (j2ee). http://java.sun.com/j2ee/. Sun Microsystems Inc.

[43] Jboss application server. http://www.jboss.org/index.html. JBoss Group.

[44] Brian Kantor and Phil Lapsley. rfc977: Network news transfer protocol, 1986.

[45] Henry Kautz, Bart Selman, and Mehul Shah. Referral Web: Combining social networks and collaborative filtering. *Communications of the ACM*, 40(3):63–65, March 1997.

[46] Bernard J. Kerr. Thread arcs: An email thread visualization. Technical Report RC22850 (W0307-148), IBM Research division, IBM Research, One Rogers Street, Cambridge, MA 02142, July 2003.

[47] Peter Kollock. The economies of online cooperation: Gifts and public goods in cyberspace. In Peter Kollock and Marc A. Smith, editors, *Communities in Cyberspace*, chapter 9, pages 220–239. Routledge, 1999.

[48] Peter Kollock and Marc A. Smith. Communities in cyberspace. In Peter Kollock and Marc A. Smith, editors, *Communities in Cyberspace*, chapter 1, pages 4–25. Routledge, 1999.

[49] Peter Kollock and Marc A. Smith, editors. *Communities in cyberspace*. Routledge, 1999.

[50] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. GroupLens: Applying collaborative filtering to Usenet news. *Communications of the ACM*, 40(3):77–87, March 1997.

[51] W. Lam, S. Mukhopadhyay, J. Mostafa, and M. Palakal. Detection of shifts in user interests for personalized information filtering. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 317–325. ACM Press, 1996.

[52] Steve Lawrence and C. Lee Giles. Searching the World Wide Web. *Science*, 280(5360):98–100, 1998.

[53] Steve Lawrence and C. Lee Giles. Accessibility of information on the web. *Nature*, 400:107–109, July 1999.

[54] Christopher Lueg. Supporting situated actions in high volume conversational data situations. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI-98) : Making the Impossible Possible*, pages 472–479, New York, April 18–23 1998. ACM Press.

[55] Mail-news gateways. http://www.newsadmin.com/bit/gateway.htm. PathLink Technology Corp.

[56] Thomas W. Malone, Kenneth R. Grant, Franklyn A. Turbak, Stephen A. Brobst, and Michael D. Cohen. Intelligent information-sharing systems. *Communications of the ACM*, 30(5):390–402, May 1987.

[57] David Maltz and Kate Ehrlich. Pointing the way: Active collaborative filtering. In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 1 of *Papers: Using the Information of Others*, pages 202–209, 1995.

[58] Bradley N. Miller, John T. Riedl, and Joseph A. Konstan. Experience with GroupLens: Making Usenet useful again. In USENIX, editor, *1997 Annual Technical Conference, January 6–10, 1997. Anaheim, CA*, pages 219–233, Berkeley, CA, USA, January 1997. USENIX.

[59] Stefano Mizzaro. Relevance: The whole history. *Journal of the American Society for Information Science*, September 1997.

[60] K. Moore. rfc2047: Multipurpose internet mail extensions (mime) part three: Message header extensions for non-ascii text, 1996.

[61] Newsbin. http://www.newsbin.com. DJI Interprises.

[62] Douglas W. Oard and Gary Marchionini. A conceptual framework for text filtering process. Technical Report CS-TR-3643, University of Maryland, 1996.

[63] Pre rendered tree map views of all usenet and microsoft.public. http://netscan.research.microsoft.com/Static/treemap/. Microsoft Corporation.

[64] Al Mamunur Rashid, Istvan Albert, Dan Cosley, Shyong K. Lam, Sean M. McNee, Joseph A. Konstan, and John Riedl. Getting to know you: Learning new user preferences in recommender systems. In Yolanda Gil and David B. Leake, editors, *Proceedings of the 2002 International Conference on Intelligent User Interfaces (IUI-02)*, pages 127–134, New York, January 13–16 2002. ACM Press.

[65] Eric Raymond. The jargon file, 2003.

[66] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of ACM CSCW'94 Conference on Computer-Supported Cooperative Work*, pages 175–186, 1994.

[67] Paul Resnick and Hal R. Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, March 1997.

[68] Knut Magne Risvik and Rolf Michelsen. Search engines and web dynamics. *Computer Networks*, 39(3):289–302, July 27–31 2002.

[69] James Rucker and Marcos J. Polanco. Siteseer: Personalized navigation for the Web. *Communications of the ACM*, 40(3):73–76, March 1997.

[70] Warren Sack. Conversation map: a content-based usenet newsgroup browser. In *Proceedings of the 5th international conference on Intelligent user interfaces*, pages 233–240. ACM Press, 2000.

[71] Warren Sack. Discourse diagrams: Interface design for very large scale conversations. In *Proceedings of the Hawaii International Conference on System Sciences, Persistent Conversations Track*, pages 1–10, january 2000.

[72] Thomas Gerhard Seidenberg. alt.culture.usenet faq (frequently asked questions), 1995.

[73] Upendra Shardanand and Patti Maes. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 1 of *Papers: Using the Information of Others*, pages 210–217, 1995.

[74] Marc Smith. Netscan: A tool for measuring and mapping social cyberspaces. http://netscan.research.microsoft.com, 2001.

[75] Marc A. Smith. Invisible crowds in cyberspace: Mapping the social structure of the usenet. In Peter Kollock and Marc A. Smith, editors, *Communities in Cyberspace*, chapter 8, pages 195–219. Routledge, 1999.

[76] Marc A. Smith and Andrew T. Fiore. Visualization components for persistent conversations. In *Proceedings of ACM CHI 2001 Conference on Human Factors in Computing Systems*, Visions of Work, pages 136–143, 2001.

[77] Henry Spencer and David Lawrence. *Managing Usenet*. O'reilly & Associates, Inc., 1998.

[78] E. Spertus, R. Jeffries, and K. Sie. Dynamic sublists: Scaling unmoderated mailing lists. *Proceedings of the Fifteenth Systems Administration Conference (LISA XV) (USENIX Association: Berkeley, CA)*, page 211, 2001.

[79] Ellen Spertus. Smokey: Automatic recognition of hostile messages. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)*, pages 1058–1065, Menlo Park, July 27–31 1997. AAAI Press.

[80] Danny Sullivan and Chris Sherman. Search engine watch. http://www.searchenginewatch.com.

[81] Douglas B. Terry. A tour through tapestry. In *Proceedings ACM Conference on Organizational Computing Systems (COOCS)*, pages 21–30. ACM, November 1993.

[82] Loren Terveen, Will Hill, Brian Amento, David McDonald, and Josh Creter. PHOAKS: A system for sharing recommendations. *Communications of the ACM*, 40(3):59–62, March 1997.

[83] Loren G. Terveen, William C. Hill, Brian Amento, David McDonald, and Josh Creter. Building task-specific interfaces to high volume conversational data. In *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems*, volume 1 of *PAPERS: Collaborative Communities II*, pages 226–233, 1997.

[84] The apache software foundation. http://www.apache.org/.

[85] Tomcat. http://jakarta.apache.org/tomcat/. The Apache Jakarta Tomcat Project.

[86] Erik Troan and Preston Brown. Logrotate. 'man 8 logrotate'.

[87] Stephen R. van den Berg, Philip Guenther, et al. Procmail. http://www.procmail.org/.

[88] Steve Whittaker, Loren Terveen, Will Hill, and Lynn Cherny. The dynamics of mass interaction. In *Proceedings of ACM CSCW'98 Conference on Computer-Supported Cooperative Work*, Asynchronous Communication, pages 257–264, 1998.

[89] D. Wroblewski, T. McCandless, and W. Hill. Advertisements, proxies and wear: Three methods for feedback in interactive systems. In R. Beun, M. Baker, , and M. Reiner, editors, *Dialogue and Instruction*. Springer-Verlag, 1994.