StatFind

A Framework for Preference Based Searching

Olve S. Hansen

October 9, 2003

Table of Contents

1 Introduction **1**

2 The Problem Domain **3**

2.1 Nesstar - the Case Application 4

2.1.1 The Problem 5

2.1.2 Solving the Problem 7

3 Theory and Method **9**

3.1 Information Retrieval 9

3.1.1 Stopwords 10

3.1.2 Similarity Models and Global Analysis 11

3.1.3 Inverted Files 12

3.2 Case Based Reasoning 13

3.3 Collaborative Filtering and Recommender Systems 15

3.3.1 Algorithms for Collaborative Filtering 15

3.4 Methodological Approach 19

3.5 Summary 21

4 From theory to practice **23**

4.1 The StatFind Framework 23

4.1.1 Choice of Learning Method 24

4.2 The Information Retrieval System 26

4.3 The Collaborative Filtering System 28

4.3.1 Vote Prediction 29
4.4 Integrating CF and IR 31

5 Implementation issues **35**
5.1 Constructing StatFind 35
5.2 The StatFind Feeder 37
5.3 The StatFind Server 40
5.3.1 Jakarta/Lucene 44
5.4 Clients Interfacing StatFind 47
5.5 Notes on Development Tools 49

6 Evaluation and findings **51**
6.1 StatFind Evaluation 53
6.1.1 Test Set-up and Data Quality 53
6.1.2 Search Evaluation 54
6.1.3 Evaluating Search for *politics* 54
6.1.4 Evaluating Search for *politics*, Changed Usage Base 57
6.1.5 Evaluating Search for id *nsd0064e_V14* 60
6.1.6 Evaluating Search for id *nsd0005e_V83* 62
6.2 Evaluation Comments 65

7 Conclusion **67**
7.1 Further Work 68

A Extended printout of search results **75**
A.1 Search for “politics” 75
A.2 Search for id *nsd0064e_V14* 78
A.3 Search for id *nsd0005e_V83* 79

B Software documenation **81**
B.1 statfind.foreign.StatfindFeeder class source code 83
B.2 statfind.common.util.SleepyQueue class source code 88

B.3 statfind.common.CompletedOperation class source code 91
B.4 statfind.common.cf.functions.WeightFunction class source code 93
B.5 statfind.common.cf.functions.VectorSimilarity class source code 94
B.6 statfind.common.cf.functions.Correlation class source code 96
B.7 statfind.common.cf.CollaborativeFilter class source code 100
B.8 statfind.common.cf.User class source code 102
B.9 statfind.common.cf.Item interface source code 105
B.10 UserItemMatrix class source-code 105
B.11 statfind.common.StatfindClient class source code 111
B.12 statfind.common.ir.StatfindIRClient class source code 116
B.13 Property files used 120

C User to MD5 keys **123**

List of Figures

2.1 Standard search in Nesstar Explorer 5
2.2 Advanced search in Nesstar Explorer 6

3.1 The CBR Cycle 14
3.2 A multimethodological approach to IS research 20

4.1 WeightFunction UML inheritance diagram 30
4.2 Collaborative Filtering system UML inheritance diagram 32
4.3 Plot of vote distribution 33

5.1 StatFind Plugin source-code 37
5.2 StatFind plugin 38
5.3 Screenshot of a browse operation 39
5.4 Client—Server typology 41
5.5 StatFind server state diagram 42
5.6 StatFind server UML inheritance diagram 45
5.7 StatFind—Jakarta/Lucene integration, UML inheritance diagram . . . 46
5.8 StatFind VarFinder client overview 48
5.9 StatFind VarFinder component screenshot 49

6.1 Safeguard in weightfunction for comparing two idenitcal users 63

7.1 Relation between IR and CF 68

B.1 An example UML Diagram, serving as symbol legend 82

List of Tables

4.1 An example of the indexed documents 27
4.2 An excerpt of the user vote table 28

6.1 Search for *politics*, document similarity only 55
6.2 Search for *politics*, using vector similarity 56
6.3 Search for *politics*, using correlation and union 57
6.4 Search for *politics*, using correlation and intersection 58
6.5 Search for *politics*, changed usage base 59
6.6 Overview over variables viewed by more than one users. 59
6.7 Search for *nsd0064e_V14*, document similarity only 60
6.8 Search for *nsd0064e_V14*, using correlation and union 61
6.9 Search for variable *nsd0064e_V14*, using vector sim, inv user freq 62
6.10 Search for *variable nsd0005e_V83*, document similarity only. 62
6.11 Summations of votes made by each user 63
6.12 Search performed using *User A* 64
6.13 Search performed using *User B* 64
6.14 Search performed using *User C* 64
6.15 Search performed using *User E* 65
6.16 Search performed using *User G* 65

Chapter 1

Introduction

This thesis' main goal is to explore which techniques are sufficient to enhance searching for statistical material in Nesstar (Nesstar, 2002b,a).

Nesstar is a system for providing infrastructure and tools for finding and using statistical studies[1], and will be described thoroughly in section 2.1.

Motivation

Nesstar only support standard boolean searches and thematic maps of statistical studies to browse — lacking a good system assisting searching and browsing of statistical material. The developers of Nesstar have also expressed the need for solutions in this area.

Rough Thesis Overview

This thesis will look into the problems of Nesstar, and see how the search system can achieve a higher degree of sophistication by the use of new developments in IR techniques. Using Nesstar as a case, it will explore how such systems can be implemented. By developing a proof-of-concept prototype allowing for experimentation with different settings continually evaluating performance, this thesis hopes to achieve

[1] A study is a collective term which will be used for describing a collection of statistical data (the dataset), and a description (metadata) of that data structured in XML.

some insight for the requirements of such systems.

Chapter 2 describes the problem domain, and gives an introduction to the Nesstar system and how the application can be enhanced. The problems with the search system in Nesstar is discussed, and first thoughts of improvements and possibilities are given. The chapter ends with a preliminary research question.

In order to fix the stated problems and test the research question, theories of the corresponding fields must be discussed. This discussion is covered in chapter 3, *Theory and Method*, focusing mainly the fields of *information retrieval*, *collaborative filtering* and also *case-based reasoning*. The different theoretical fields are described, followed by a discussion on how they work and their advantages and disadvantages. The methodological research position of this thesis is also discussed, followed by a discussion on which of the covered research fields are most suitable for the thesis. The chapter ends by revising the research question in light of these discussions.

Chapter 4, *From theory to practice*, is devoted to describe how the theories from chapter 3 are put to use. The focus will be more technical, looking at the choices done during the implementation of the given theories. It will also discuss problems regarding to the integration of two different implemented theories used.

Much of the development in this thesis does not follow its central theories, but are nevertheless important for understanding how the system works as a whole, and what the different parts do. Chapter 5, *Implementation issues*, covers this work, and describes the architecture and development process of the system.

The performance of StatFind is covered in chapter 6, consisting of discussions about data quality, examples of search performance with discussions, and a general discussion of the findings. The chapter also covers discussions of arisen problems with the system, and other problems regarding the quality of the data gathered for this thesis.

Chapter 7 sums up all of the findings from the evaluation, and concludes with a verdict on the performance of the system as perceived during the evaluation. Further work is also discussed.

Chapter 2

The Problem Domain

The search for public information is difficult, mainly because of the large amount of information released by official institutions, and because of the increased availability of such information. Institutions that produce information for the public sphere are in a greater degree than before using the Internet as the (sole) channel of communication, making the users responsible for finding the information of their need.

Norway (and Scandinavia) has long traditions for collecting statistics for numerous areas.

Statistics Norway is a governmental institution that gathers, store and analyses this data, and has the following mission statement[1]:

> Through collection, processing and dissemination of statistics and analysis, Statistics Norway contributes to a more informed public debate, to ensure that economic and social policy management are based on the best possible factual basis and to improve the functioning of the market system.
>
> (SSB, 2002)

[1] The first Norwegian statistics act is of 1907 (SSB, 2001)

NSD is another similar institution and states that:

> Norwegian Social Science Data Services (NSD) is a national resource centre servicing the research community. NSDs objectives are to:
>
> - facilitate wider and more knowledge-based use of data
> - promote the preservation and sharing of data
> - ensure free and open access to information
>
> (NSD, 1997)

These two institutions, dealing with the production and archiving of statistic material, shows some of the traditions in Norway for providing statistics for public and political use. Enabling the use of this amount of information is both essential and a challenge since finding the right pieces of information can be like finding the often mentioned needle in the haystack.

2.1 Nesstar - the Case Application

NSD, in cooperation with UK Data Archive and Danish Data Archive (UKDA, 2002; NSD, 1997; DDA, 2003), took up this challenge by developing the Nesstar system — a set of generic tools that made it easier to:

- Locate multiple data sources (servers containing statistical data) across organisational and national boundaries.
- Browse detailed information about these data, especially the descriptive and contextual information.
- Tabulate and visualise these data quickly and easily for both naive and experienced users.
- Disseminate these data and documentation, in whole or part, in forms suitable for immediate use.

(Nesstar, 2000)

The development was sponsored by the *Telematics Application Programme — Information Engineering* (CORDIS, 1999) as well as the *Council of European Social Science Data Archives* (CESSDA, 2002).

The system saw a total of two rounds of EU funded projects, the first — NESSTAR (also the name of the system) (Nesstar, 2002a) and the second Faster-Data (Faster, 2000) for a total of four years of development time funded by the EU, leaving the system in its current state.

Overall the application (The Nesstar Explorer) has met its formal requirements stated before the project funding committee, but still some useful features are not yet implemented or unsatisfactory.

2.1.1 The Problem

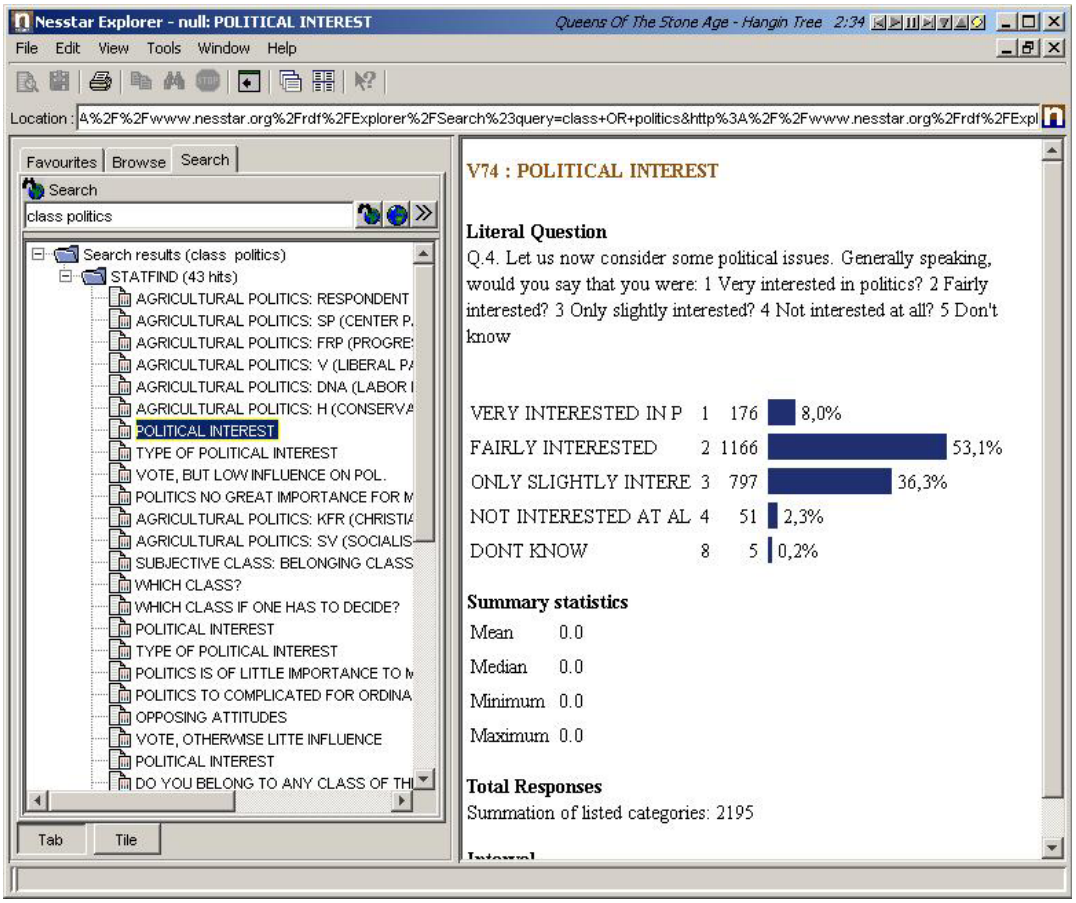

Figure 2.1: Screenshot of standard search in Nesstar Explorer

To demonstrate weaknesses with Nesstar I will do two searches, using the Nesstar Explorer, for *politics* and *class*. The first (fig. 2.1) shows the results using the standard search, the next (fig. 2.2) shows a similar search done in the advanced mode. Both searches highlight the variable *POLITICAL INTEREST* returned by the search (the content of this variable is also displayed in the documentation window). The difference between the two searches is the way the search is narrowed down in the advanced search. These two examples show the only user interface allowing for searches in the Nesstar Explorer.

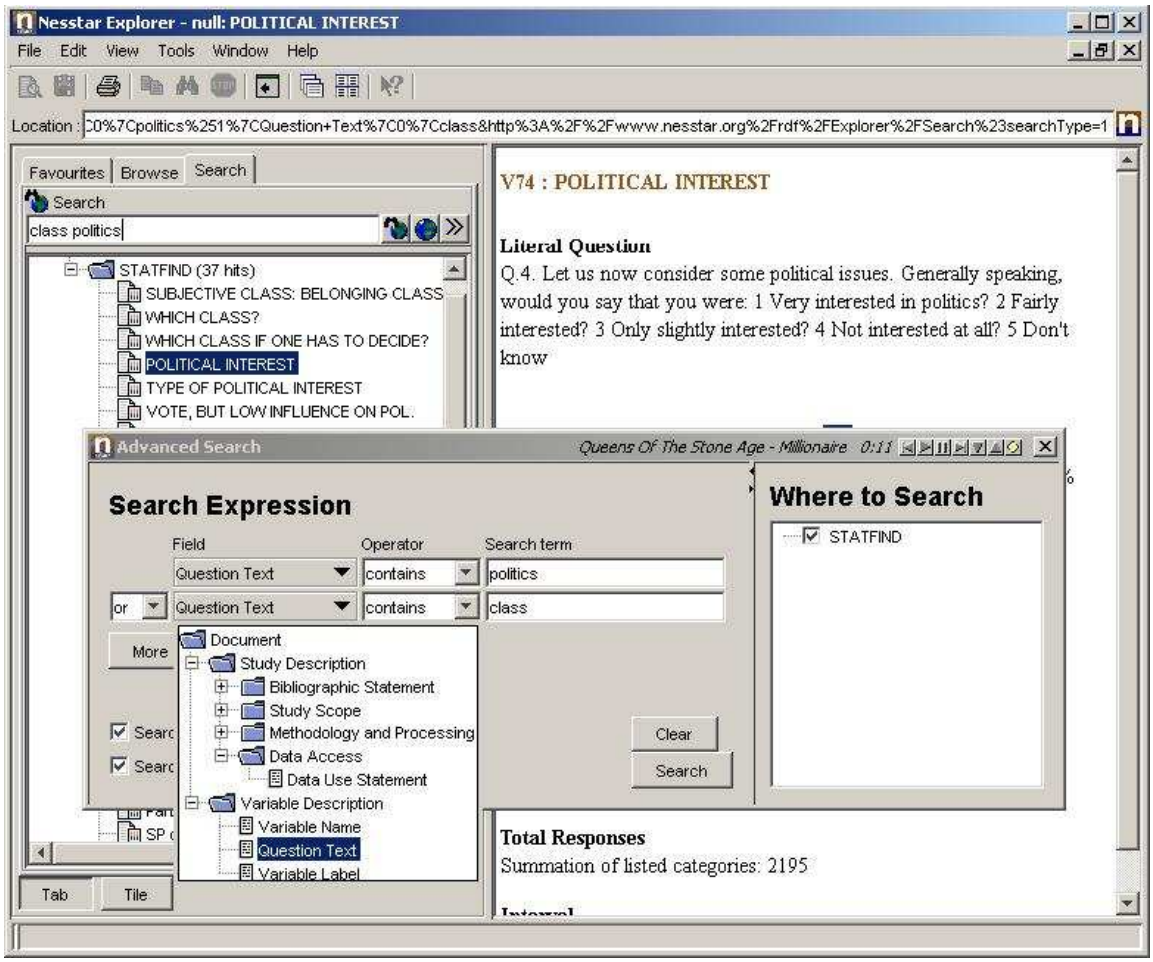

Figure 2.2: Screenshot of advanced search in Nesstar Explorer

The answer set is ordered by the underlying database — it is not ranked with regard to where the words are in the variable (e.g. if it is in the title, it could be given more weight). Finding more variables syntactically similar to the one we are looking

at (i.e. containing some of the same words), can be done by specifying specific parts in the study to search for, as the advanced search facility supports selecting in which part of the variable (*variable name, question text* and *variable label*) the words must exist (see fig. 2.2). This way a user can perform searches for variables similar to a given variable manually by using the advanced search facility.

Still, some problems appear in this approach as well, as the application does not measure how similar the elements in the answer set are to the given query[2]. Whether the ranking of the answer set is done in accordance to its similarity to the query is also undefined in the Nesstar system. If the ranking is not done in accordance to the query, a search resulting in a considerable number of hits will make the process of finding the right variables quite cumbersome. Specifying the query needed to find similar variables may become awkward, as each word wanted in the items returned may have to be specified in the query.

2.1.2 Solving the Problem

The problems outlined in the section above could have been solved by automating the process of finding similar variables in the fashion described above. By pressing a button a user could find variables similar to a current highlighted one. This tool does not need to be very complex, as most of the functionality needed to search for specific parts of statistical studies are already implemented, it is just a matter of combining them in a new way.

On the other hand, automating the steps needed to find similar variables gives us a chance to rethink the process, and achieve a greater level of sophistication than possible by combining the existing components in a new way. How this can be done will be discussed in the following chapters, *Theory and Method*, and *From theory to practice*.

This leads toward this thesis' preliminary research questions;

[2] A query is the expression of the information sought, given in the input language of the information system (Baeza-Yates and Ribeiro-Neto, 1999b)

In what way can a tool for helping users find similar statistical variables be implemented, and which methods are sufficient for measuring the similarity between variables.

The next chapter describes theory applicable to solve the problem described above.

Chapter 3

Theory and Method

This chapter will investigate the possible theories that can be used in the pursuit of a search system as specified in the preliminary research question in then end of chapter 2. The theories and methods outlined here are the ones found to be useful for making an improved search system for Nesstar. An understanding of the theories and how they can be used in the described domain is important as it will assist the implementation of the system, as well as forming the foundation for the analysis of the data gathered for evaluating the developed tool.

The main purpose of this thesis is to investigate possible improvements to searching and ranking mechanisms. In doing so it will draw on theories from Information Retrieval (IR), as well as Artificial Intelligence (AI) and Case Based Reasoning (CBR). Collaborative Filtering (CF) is a maturing technique, used in commercial and open-source software (Riordan and Sorensen, 1998), and in all of the three mentioned research fields. An overview of these areas follows.

3.1 Information Retrieval

An investigation of the research field of Information Retrieval is appropriate for improving the search abilities for Nesstar. Information Retrieval (IR) has long traditions, the first research material dating back as far as the 1930's (Mizzaro, 1997). Mainly focused on categorisation and ordering of bibliographic material.

Dramatic changes occurred in the field of IR after the advent of the digital computer and especially with the possibilities provided by the Internet, as the findings within the field could be put to use in many new areas. Baeza-Yates and Ribeiro-Neto (1999b) defines *Information Retrieval* (IR) as:

> [a] part of computer science which studies the retrieval of information (not data) from a collection of written documents. The retrieved documents aim at satisfying a *user information need* usually expressed in natural language.
>
> (Baeza-Yates and Ribeiro-Neto, 1999b, page 444)

and *entropy* as:

> [a] measure of information defined on the statistics on the characters of a text.
>
> (Baeza-Yates and Ribeiro-Neto, 1999b, page 441)

This indicates that the corpus of documents must be interpreted as containing information rather than data, and that the information can be quantified by calculating its entropy. The higher the entropy the more information is contained. These are central definitions in the research field of Information Theory, on which Information Retrieval bases much of its work. The term document is often used within IR and it is defined as;

> a unit of retrieval. It might be a paragraph, a section, a chapter, a Web page, an article, or a whole book.
>
> (Baeza-Yates and Ribeiro-Neto, 1999b, page 440)

3.1.1 Stopwords

Stopwords are words that do not carry any explicit meaning in natural language and can therefor safely be disregarded in an IR setting. According to calculations done

using Zipf's law the most frequent words in a text are stopwords (Salton and McGill, 1983). Typical stopwords are; about, for, of, onto, with, etc.

Baeza-Yates and Ribeiro-Neto defines *Stopwords* as:

> words which occur frequently in the text of a document, Examples of stopwords are articles, prepositions and conjunctions.
>
> (Baeza-Yates and Ribeiro-Neto, 1999b, page 452)

Removal of Stopwords

Removing stopwords reduces the size of an index dramatically, without reducing the texts entropy (Baeza-Yates and Ribeiro-Neto, 1999d).

Removing stopwords also makes searches more precise as a search containing e.g. *prices above or below and travel* would, when the stopwords were included, give highest rank to the document having the highest frequency of those words (Ziviani, 1999). Instead words like *and, or, not* is used as special words to build a boolean query.

Salton and McGill (1983) states that stopwords top the frequency count of a reference collection of documents. Indexing on stopwords may inhibit a search engine as the amount of data is larger than necessary.

When searching for exact phrases, the stopwords can't be disregarded. This problem is solved by utilising different types of indexes are used for phrase matching than for keyword searches (Navarro, 1999).

3.1.2 Similarity Models and Global Analysis

Global analysis is defined as;

> a reference to techniques of identifying document and term relationships through the analysis of all the documents in a collection.
>
> (Baeza-Yates and Ribeiro-Neto, 1999b, page 442)

A comparison of different documents used in a global analysis scheme must be grounded in a model. This model clarifies the boundaries for what a document can contain, as well as describing the methods used to compare the documents. A comparison should yield a number, usually between zero and one, as a figure of the similarity of the documents. Comparing documents this way is called calculating the *document distance*, and it is a *distance function*.

Calculating Document Distance

The distance can be calculated in many ways, each fitting better to some tasks than others. There is no single "best" model for calculating distance, but some guidelines exists. E.g. the method should be symmetrical ($distance(a, b) = distance(b, a)$), and should satisfy the triangle inequality ($distance(a, c) \leq distance(a, b) + distance(b, c)$) (Baeza-Yates and Ribeiro-Neto, 1999d).

To utilise the *distance function* properly, the distance for all documents against all other documents must be calculated, and stored in an appropriate way. Storing this information in e.g. a similarity matrix comparing each document against each other requires $O(n^2)$ time. This makes nearly all of the necessary calculations, thus making retrieval time short (Faloutsos and Oard, 1995).

Performance

The quadratic performance time for indexing is a major drawback, and cuts down on the areas where these methods can be used i.e. it is only feasible for collections where the number of indexed documents are fairly constant.

3.1.3 Inverted Files

An inverted file is composed of a vocabulary (the indexed words) and a list of occurrences (where the words are found), and is another approach to indexing files. There are several types of inverted files, but the most commonly used is composed of a list of sorted words (vocabulary), each having a set of pointers to the documents where

the words occur, and how many times they occur.

Several ways of building indexes of inverted files exists, as well as different ways of searching the built index. The index may be built using different types of compression, and other techniques, depending on the qualities of the text being indexed (Navarro, 1999). Details of the different techniques will not be covered in this thesis, as they fall outside the interest of this thesis.

Index Normalisation

Most indexing methods also use some normalisation method — where normalisation is done before the indexing process. The most common are stop-word removal, and stemming of the words (Navarro, 1999). Porter's stemming algorithm is the most used stemming algorithm (Ziviani, 1999) and is described fully in Baeza-Yates and Ribeiro-Neto (1999b, page 433).

3.2 Case Based Reasoning

As case-based reasoning may be viewed as operations related to similarity (Althoff and Aamodt, 1996), it may be interesting to investigate the usefulness of case based reasoning (CBR) in this domain. CBR is also appropriate for solving the problem described in 2.1.1 as CBR is within the domain of IR. The main difference or advantage is the way a CBR implementation is able to learn from use. A CBR implementation can learn which statistical variables are used together in the Nesstar system. Viewing the documentation of a statistical variable (the base case) could then trigger a presentation of the other statistical variables (the target cases) most often used together with the one currently being viewed.

The case-based reasoning process consists of these main steps:

- The RETRIEVE process retrieves the collection of cases that is most similar to the new case.

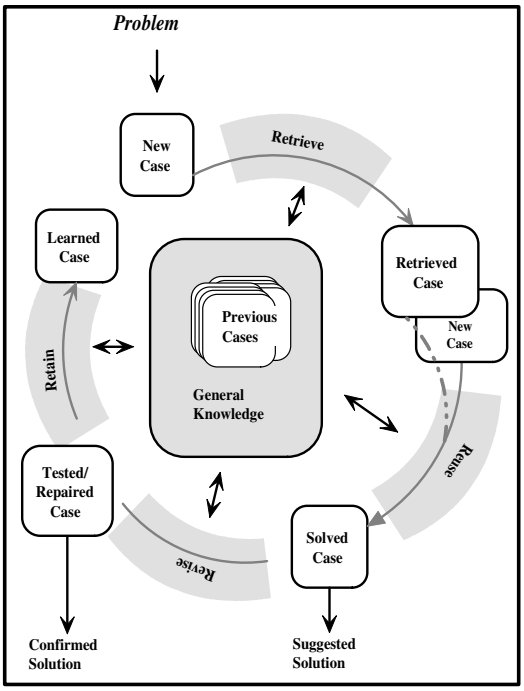

Figure 3.1: The CBR Cycle (Aamodt and Plaza, 1994)

- The retrieved case(s) are combined with the new case through the REUSE process ending up with a solved case, which is a proposed solution to the problem.

- The proposed solution is then REVISED after being tested for success. The case will be repaired after testing if needed.

- The RETAIN process extracts the useful parts of the tested (and possibly repaired) case, and puts these parts back in the case-base. This learned case is represented in the case-base as a modified existing case, or by making a new learned case.

These four stages represent the most general way in which a CBR system works (Aamodt and Plaza, 1994).

3.3 Collaborative Filtering and Recommender Systems

Recommender systems try to recommend items to a user, based on what he/she earlier has rated in a scale of some range. The rating can be a scale from 0 - 10, or simply a boolean like/dislike scale, or a metric that the system extracts during use by the aid of a heuristic method. Collaborative filtering uses the same approach, but the recommendations are based on the accumulation of ratings by several other users. This approach is similar to CBR, in that recommendations are based upon the similarity to other users' behaviour. However, recommender systems are semantically weak compared to CBR because few features are associated with each user (Burke, 2000).

Domain Applicability

How a CBR system is used depends, on the type of domain it is used in, but in general CBR systems are used in *open* and *weak theory* domains (Aamodt, 2001; Aamodt and Plaza, 1994). Recommender systems/Collaborative filtering also fall under this area (*open* and *weak theory* domains), and can be viewed as an approach that address the same problem.

As the Internet bookseller Amazon does it; instead of finding out what constitutes a good book and categorise the good books that relates to each other (e.g. according to subject, genre, language etc.), they let the users do it for them by recommending books other bought who also bought the book you are currently looking at (Schafer et al., 1999).

3.3.1 Algorithms for Collaborative Filtering

Breese et al. compares how different algorithms for predictive collaborative filtering works on different datasets[1]. The results were presented in Breese et al. (1998), and

[1]The term "dataset" is not used in the same meaning as earlier, but as a collection of structured

will serve as basis for the implementation of collaborative filtering support in this thesis. The following are the equations used in this thesis for calculating similarities between users and predicting votes for searched items.

The General Approach

Given a user database consisting of several votes $v_{i,j}$, where i is the user and j is the item which the vote is given for and If I_i is the set of items on which user i has voted, the mean vote for a user i is:

$$\bar{v}_i = \frac{1}{|I_i|} \sum_{j \epsilon I_i} v_{i,j} \tag{3.1}$$

Next, let a be the active or current user and j the item we want a prediction $p_{a,j}$ for, can be calculated as follows:

$$p_{a,j} = \bar{v}_a + \kappa \sum_{i=1}^{n} w(a,i)(v_{i,j} - \bar{v}_i) \tag{3.2}$$

where n is the number of users in the database with nonzero weights. κ is a normalising factor so that the weights sum to unity. The function $w(a,i)$ returns weights saying something about the distance between two users, their similarity, distance or other relevant measures, and is covered below, in section *Weighting Functions*.

The interesting part of the equation (3.2) is $(v_{i,j} - \bar{v}_i)$ which adjusts the weight or user distance. If the compared users' votes are below the users' mean vote, the result will be given a negative adjustment, and vice versa.

Weighting Functions

Breese et al. (1998) analyse several weighting functions and compare them to find the domain they are best suited for. The article concludes that using a Bayesian Network is generally best at predicting the votes for users, but vector similarity and correlation methods are close runners up in their comparison. According to the same article, a Bayesian Network need relatively large amounts of data in order to be trained properly. As this thesis will not have large amounts of data available, correlation and

data of any kind, not only as a statistical dataset.

vector similarity seems to fit best. These kinds of algorithms are called memory-based algorithms.

Vector Similarity According to Breese et al. (1998) vector similarity for collaborative filtering is an adoption of the formalism used by Salton and McGill (1983) for comparing documents.

The symbols used in the following equations means the same as in (3.1) and (3.2). $v_{a,j}$ points to the vote given by the active user a for item j. Item j in $j \epsilon (I_a \cap I_i)$ denotes the intersection of item votes, i.e. items voted for by both users. The summations in the denominator denotes the items in which the active and the compared user has voted for.

$$w(a,j) = \sum_{j \epsilon (I_a \cap I_i)} \frac{v_{a,j}}{\sqrt{\sum_{k \epsilon I_a} v_{a,k}^2}} \frac{v_{i,j}}{\sqrt{\sum_{k \epsilon I_i} v_{i,k}^2}} \tag{3.3}$$

Correlation For calculating correlations, the Pearson correlation coefficient is used. The collection used for the summations is the same as in (3.3) The weight function for calculating correlations is:

$$w(a,j) = \frac{\sum_{j \epsilon (I_a \cap I_i)} (v_{a,j} - \bar{v}_a)(v_{i,j} - \bar{v}_i)}{\sqrt{\sum_{j \epsilon (I_a \cap I_i)} (v_{a,j} - \bar{v}_a)^2 \sum_j (v_{i,j} - \bar{v}_i)^2}} \tag{3.4}$$

Extensions to these algorithms are default voting, inverse user frequency, and case amplification. It works as follows:

Default Voting

Vector similarity calculations provide the same result, whether $I_a \cup I_i$ or $I_a \cap I_i$is used. Using $I_a \cup I_i$ for correlation increases the number of items used in the calculations and has proven positive in domains where few votes for either a or j exist (Breese et al., 1998).

The reason this technique is called *default voting* is that when an item for which only one of the compared user has given a vote, the appropriate vote is inserted for the other user (usually 0). In an implicit voting scenario as in this thesis, a missing

vote can be counted as a vote, i.e. the value 0 is assigned as a default vote if an item is not viewed at all. This value can be changed, specifically when combined with inverse user frequency (Breese et al., 1998).

Inverse User Frequency

The *inverse user frequency* is an adoption of the inverse document frequency, normally used to reduce the weights of commonly occurring words (Baeza-Yates and Ribeiro-Neto, 1999a). Based on the assumption that heavily used words are less indicative on topic than less frequent words. *Inverse user frequency* uses the same assumption; universally preferred items are not as useful in capturing similarity as less common items (Breese et al., 1998).

Inverse user frequency works well together with default voting, as a missing vote counts as a vote. Thus items not viewed by two compared users will be treated as a similarity trait. This works by giving the vote, in a situation where no vote exist for a given user, a high value. In this way, *inverse user frequency* will give a resulting low vote for those occurrences. If n is the number of users in the database, and n_j is the number of users voted for the item j, the *inverse user frequency* will be:

$$f_j = \log \frac{n}{n_j} \tag{3.5}$$

If an item have votes from all registered users, f_j is zero.

The *inverse user frequency* changes the way the correlations and vector similarities are calculated, by inserting the f_j from (3.5) as follows:

$$\begin{aligned} U &= \sum_j f_j (\sum_j f_j v_{a,j}^2 - (\sum_j f_j v_{a,j})^2) \\ V &= \sum_j f_j (\sum_j f_j v_{i,j}^2 - (\sum_j f_j v_{i,j})^2) \\ w(a,i) &= \frac{\sum_j f_j (\sum_j f_j v_{a,j} v_{i,j} - (\sum_j f_j v_{a,j})(\sum_j f_j v_{i,j}))}{\sqrt{UV}} \end{aligned} \tag{3.6}$$

Vector similarity can be made to accept *inverse user frequency* by multiplying f_j with

the transformed vote. This leads to the equation:

$$w(a,j) = \sum_j f_j \frac{v_{a,j}}{\sqrt{\sum_{k \epsilon I_a} v_{a,k}^2}} \frac{v_{i,j}}{\sqrt{\sum_{k \epsilon I_i} v_{i,k}^2}} \tag{3.7}$$

Case Amplification

Case amplification works by transforming the calculated weights by raising the weight to a given power, so that

$$w'_{a,i} = \begin{cases} w_{a,i}^p & \text{if } w_{a,i} \geq 0 \\ -(-w_{a,i}^p) & \text{if } w_{a,i} \leq 0 \end{cases} \tag{3.8}$$

A typical value for p in Breese et al.'s experiments was 2.5.

Default voting and inverse user frequency only works for the correlation function, while case amplification works for both correlation and vector similarity. The different techniques will be implemented and tested against the user vote database, generated for the evaluation of my approach. Section 4.3 describes the implementation of these techniques.

3.4 Methodological Approach

This thesis will use research methods inspired from the multimethodological approach suggested in Nunamaker et al. (1990), as well as methods from *artificial intelligence* (AI) research.

The multimethodological approach for developing *information systems* (IS) can be described as a framework for using different methods for evaluating the developed systems. Figure 3.2 shows how different methods interplay. As Nunamaker et al. puts it;

> "Systems development is the hub of research theta interacts with other research methodologies to form an integrated and dynamic research program."

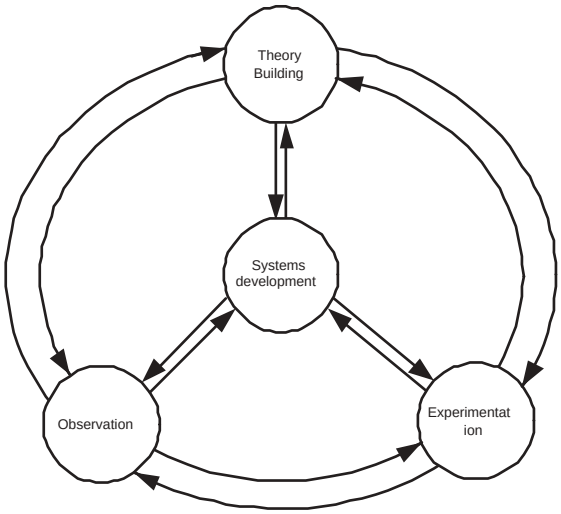

Figure 3.2: A multimethodological approach to IS research (Nunamaker et al., 1990, page 93)

(Nunamaker et al., 1990, page 95)

This way IS development is the basis that allows research in the domain of the developed component to be done. Different research methods interplay in exploring the domain in which the IS is developed.

This thesis will develop an IS and do experiments with the developed system, and thus positions itself in the *hub* and in the *experimentation* part of figure 3.2. Doing so it will employ AI as its experimental method, following the steps proposed by Simon (1995). These three points describe the method:

1. Choose a task that encompass an intelligent quality of substantial practical importance, or that is composed of properties showing complexity not earlier simulated by AI-systems.

2. Construct a system that demonstrates this intelligent property.

3. Explore the system's behaviour in different task environments and with different initial states.

And proposes that the main method within AI should be composed of building and studying systems that indicate intelligent behaviour.

This makes AI an experimental method rather than a theory about the mechanisms that enable intelligence. It also tells us that those mechanisms can best be studied through design; running and evaluating experiments with the enhancements of the system, and continued experimentation as a goal. According to Nunamaker et al. (1990) and figure 3.2, the experimentation is further used in research, as input to observational research, and in the development and refining of theories.

3.5 Summary

The theories described in this chapter are the ones most fitting for this domain. Each theory has elements in common with the others, and in this thesis the main theoretical directions will belong to IR and AI.

CBR is a sub-area of AI, as it focuses on how a system can learn from usage (see figure 3.1). Initially CBR looked quite promising for solving the preliminary research question given in section 2.1.2, but a CBR system is more often implemented as a more extensive system than the domain and scope of this thesis allows. Also, the low complexity of the information available in the thesis' domain suggests that implementing a CBR system for solving the problem might be an overkill. Other technical problems using CBR is covered in section 4.1.1. CF will hopefully suffice in bringing the qualities sought in CBR into the software developed for this thesis, since the techniques used for CF are also applied in CBR settings, although in a more sophisticated way.

AI as a method can be applied in a wide range of cases, as it in the widest sense can be viewed as a framework for using experimentation when developing *intelligent* systems. CBR and CF can in this way be viewed as a specialisation of the AI method. The research field of IR draws on traditions from the field of linguistics — as many of the theories are about operations regarding text, as well as measuring how relevant one body of text is to another (Mizzaro, 1997). Although the two main fields of AI and IR have their roots different traditions, they do not contradict each other in any way. A combination of ideas from both fields is therefore possible.

These theories represent the theoretical ground on which this thesis will be based, specifically regarding the research question below, and the research design as a whole. Methodically the thesis will be conducted as specified in section 3.4.

The question I want to address is;

Will mixing classic Information Retrieval methods with Collaborative Filtering techniques be effective and efficient in finding related statistical variables?

Chapter 4

From theory to practice

I have utilised the theories described in the previous chapter to implement a software component as a contribution to the Nesstar system. I have named the component *StatFind*. StatFind is composed of a Java server and a database, as well as an addition (plugin) to the Nesstar Explorer client.

This chapter will mainly focus on the implementational choices based on the theories from the previous chapter, while chapter 5's focus will be the development process.

StatFind is developed in the Java programming language mainly because my expertise lies in Java development and because the Nesstar System is written in Java. Making a later utilisation of StatFind, where it is added to the Nesstar system portfolio, easier. It also makes the API for StatFind less complex as inter-module communication can be done using Java method calls.

4.1 The StatFind Framework

The Nesstar server runs in a JBoss J2EE environment, using MySql as the DBMS[1]. MySql was a natural choice for a DBMS for this thesis since it eased the process of integrating the existing data needed for indexing the Nesstar document base from the Nesstar MySql database.

[1]Data Base Management System

Before discussing a choice of learning method, the unit of interest must be clarified. The statistical documentation Nesstar allows an interaction with is broken down in many parts. Documents structured in XML that lies at its centre permits fine grained control over the different parts of the documentation.

Statistical studies in Nesstar are composed of different documentation parts, following the DDI standard (*Data Documentation Initiative* (ICPSR, 2002)). The DDI permits identifying different parts of statistical documentation as e.g. abstract, sampling method, variable descriptions etc. These variables are the items containing most useful information, as they have a close relation to the statistical data they describe. Other elements from the DDI are on a more meta-level than what is contained in the data-description part of the DDI. This suggests using the documentation about the statistical variables contained in a study as the item of interest when collecting information about users behaviour in the Nesstar system.

4.1.1 Choice of Learning Method

Three theories for making an advanced search system have been proposed, the IR approach, the CBR approach, and the CF approach. An IR approach does not encompass a system for a learning search system, and a CF approach does not encompass an ordinary search system. Only CBR can be seen as a framework for combining the features from both IR and CF. At first glance, a CBR system looks most promising for making a learning search system. By treating statistical variables as cases in a CBR system, the system could track their use, and suggest other relevant statistical variables for a user using the Nesstar System. The main drawback in implementing CBR in this domain is its information need. As CBR is a system that learns from experience, the system needs to access information surrounding the decisions made by a user to be able to recreate the situation and be able to solve the problem — information that needs to be related to the problem at hand (Carrick et al., 1999).

When planning the StatFind framework, I wanted it to be non-obtrusive, so users would not need to make explicit responses to StatFind in order for the software to

learn. Since the information collected is generated at very short intervals, the user would have to answer questions quite frequently. The users may be annoyed by frequent interruptions in the for of pop-up dialogs with questions regarding how well the information they view fits to what they sought.

Another approach could be to collect this information after searches in the system, and ask the user about extra input not so often as would be necessary in the current approach. This would result in less collected data over the time period available — as the thesis scope does not allow for a longer data collection period.

Implicit voting[2] could be used in a CBR setting as well as in CF. The main problem is both technical as well as theoretical; implicit voting has difficulties in capturing some types of preferences (like where users must explain their intent). A CBR system is most often built around a user feedback system (Carrick et al., 1999). The feedback system enables the user to give a more nuanced feedback with both positive and/or negative feedback on various cases.

It is possible, but difficult, to make a non-obtrusive implicit voting system for StatFind/Nesstar which give a more nuanced feedback. The difficulties lies in the heuristics needed to draw the borders between different grades from like to dislike. If a search is done and a list of statistical variables are presented, a voting system must capture which variables from a search are used as well as which are not. This information is impossible to capture from the Nesstar Explorer without a major rewrite of the application. It is possible to know who is looking a statistical variable, but the context in which it is viewed is undefined (from the applications point of view).

The above facts suggest that the CF approach is best suited for the learning-method used in StatFind/Nesstar, resulting in the use of an IR system to do the textual comparison of statistical variables

[2]Implicit voting is an approach were users actually vote, but a user trait or behaviour is interpreted as a preference.

4.2 The Information Retrieval System

Global Analysis, the First Version

The first version of the IR system was built using a global analysis technique. Global analysis is used to analyse or compare all documents in a collection, and calculates the similarity figure for each compared document. The similarity figure is a value that describes how similar two documents are e.g. in the range of [0,1] where 0 is no similarity between them, and 1 means they are identical. In a collection of n documents, $n^2 - n$ number of comparisons must be done. This amounts to $O(n^2)$ for building the index. This thesis uses a collection containing 921 statistical variables, which amounts to 847.320 comparisons — exponential index functions are seldom efficient.

The Document Model

The process of indexing the document bases involves a comparison of each document in the database. When the words of each document is first extracted and compared to a stopword list, the words matching a stopword is removed from the list of words to compare. Next, the two documents' lists of words are sorted, and compared one by one. The similarity figure is the number of words common in the two documents divided against the number of words in the document having most words.

The table 4.1 shows 10 rows of indexed statistical variables from the database, note that no similarity figures are below .5, as it was the boundary for storing the document comparison result.

Global Analysis and its Limitations

Indexing in this manner limits searches to be done using an existing statistical variable as the query, because the index consist of the id of the two variables compared only. In order to find variables similar to the one a user may currently be viewing, retrieving the rows where one of the variables have the same id is sufficient. The comparisons between the documents are already done.

Case variable	Target variable	Similarity Figure
nsd0005e_V31	nsd0393e_V92	0.5
nsd0005e_V32	nsd0005e_V54	0.666666666666667
nsd0005e_V32	nsd0005e_V279	0.6
nsd0005e_V32	nsd0005e_V281	0.666666666666667
nsd0005e_V32	nsd0064e_V13	0.923076923076923
nsd0005e_V32	nsd0064e_V32	0.666666666666667
nsd0005e_V32	nsd0393e_V92	0.5
nsd0005e_V32	nsd0393e_V128	0.666666666666667
nsd0005e_V33	nsd0064e_V14	0.833333333333333
nsd0005e_V33	nsd0393e_V92	0.5

Table 4.1: An example of the indexed documents

In section 2.1.1 the problem with the Nesstar search system was described . The current approach (global analysis) only adds the *find similar statistical variables* method. There are a few limitations to this approach, the indexation scheme makes it impossible to use anything other than a statistical variable id as a query. E.g. a freetext search is impossible in this implementation of global analysis. Together with slow and inefficient indexing the current approach should be revised.

Improving the search functionality of the Nesstar System as suggested in 2.1.2 implies that the free text search functionality should also be improved. Especially since it is undefined whether the results in table 4.1 are in accordance with the guidelines specified in section 3.1.2, regarding *triangle inequality* and *symmetrical results*. The coherence with these guidelines could have been formally tested, but the existing document similarity model does not suggest that the guidelines hold.

It is certain that the guideline for symmetrical results does not hold. The word count used to measure the similarity always divides the number of words in the statistical variable having fewest words with the one with most words. This was done to ensure a similarity figure below or equal to one.

Addressing the Limitations of Global Analysis

The problems regarding the global analysis method of indexing led to a need for another IR component. Instead of implementing a complete new search-engine, the

effort was put into finding a component ready for use, preferably as an open-source component. Positive experiences with other Apache products lead to tests using Jakarta/Lucene (Jakarta/Lucene, 2003), an open-source project under the Apache license. Jakarta/Lucene proved to be fast, reliable and very flexible, some background research proved that it uses inverted files in what seems to be a very efficient way.

By doing this switch, the cumbersome global analysis method disappears, resulting in a far more flexible, faster, and reliable search system. This is further described in section 5.3 and in the discussion of findings in chapter 6.

4.3 The Collaborative Filtering System

As discussed earlier, StatFind will be implemented using the collaborative filtering routines outlined in section 3.3. An excerpt of the user votes are shown in table 4.2. In the database, the user and statistical variable id are combined as the primary key. This makes it possible for several users to vote for the same item. If a vote a user already has voted for is registered, only the view count field is incremented.

Statistical variable id	User	View count
nsd0393e_V14	User E	1
nsd0393e_V19	User E	1
nsd0393e_V9	User E	1
nsd0064e_V94	User F	3
nsd0064e_V95	User F	4
nsd0064e_V96	User F	2
nsd0064e_V97	User F	3
nsd0064e_V98	User F	2
nsd0064e_V101	User F	3
nsd0064e_V100	User F	3
nsd0064e_V99	User F	1
nsd0005e_V5	User C	1
nsd0005e_V206	User C	1
nsd0005e_V207	User C	2
nsd0064e_V239	User C	2
nsd0064e_V240	User C	1
nsd0005e_V209	User C	1

Table 4.2: An excerpt of the user vote table

4.3.1 Vote Prediction

The predicted vote for a given user is calculated using the other users' votes (gathered from user behaviour), the selected weighting function, and on the current and other users' mean vote. Implementation follows the equation (3.2) for the vote prediction, and equation (3.1) for calculating the mean vote. The source code for these calculations are available in appendix B.10 and B.7.

Interpreting User Behaviour

When the system is given a statistical variable–user identificator pair, e.g. nsd0064e_V95 and User C , the system responds with a prediction for the specified combination. This works by calling the `Vote predictVote(User activeUser, Item activeItem)` method in the `CollaborativeFilter` class with the active user, and the statistical variable the user want more similar variables from, as parameters to the method (see figure 4.1 for UML). User and variable correspond to a and j in equation (3.2). Next all users are iterated, and by the selected weight function and its settings, the $w(a, i)$ in equation (3.2) is calculated. Appendix B has a legend for the UML diagram (figure B.1).

The CF Method Framework

`WeightFunction` is an abstract class for providing general functionality and pluggable methods and leaves the method `double weight(final User activeUser, final User otherUser)` to be implemented by derived classes.

The class `CollaborativeFilter` (B.7) has an instance of `WeightFunction` (B.4) as a member (see fig. 4.1). The `CollaborativeFilter` class is only a wrapper class for administrating the weight functions, and used by `StatfindIRClient` to calculate the vote predictions.

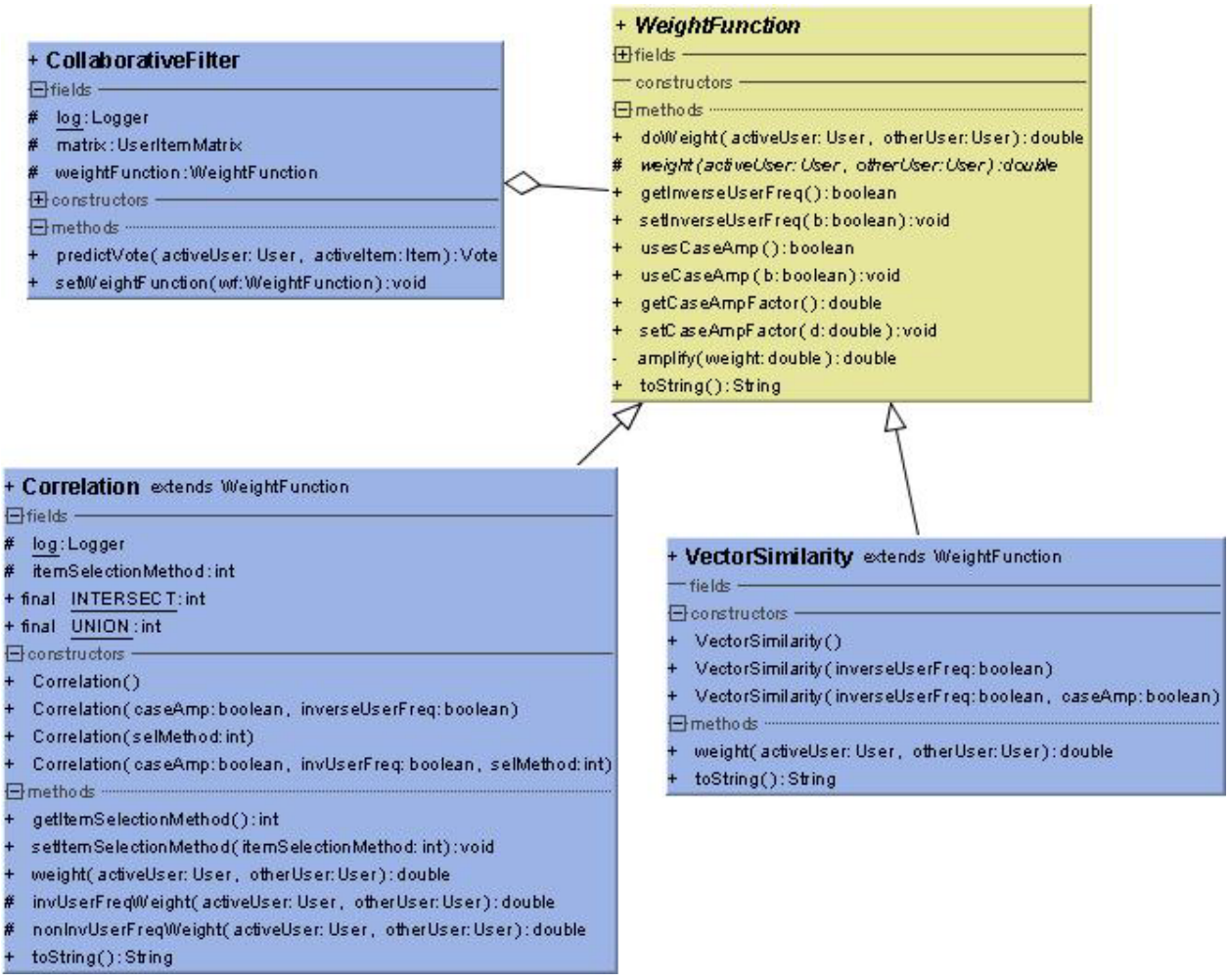

Figure 4.1: WeightFunction UML inheritance diagram

Implemented CF Functions and Techniques

StatFind implements two derived classes; `Correlation` (B.6) and `VectorSimilarity` (B.5). The two implemented weight functions have some features in common, and some are specific for the individual weight functions. See figure 4.1 for the UML model describing this architecture (legend for the UML diagrams is available in appendix B figure B.1). The `WeightFunction` class can utilise case amplification (see equation (3.8)), and inverse user frequency (3.5), which are common for the implemented subclasses. Inverse user frequency is common in the way that the method exists for both `Correlation` and `VectorSimilarity`, but is implemented differently for each class. See equations (3.6) and (3.7), and source code in appendix B.4 (`WeightFunction`),

B.5 (`VectorSimilarity`) and B.6 (`Correlation`) for details.

One method differs between the two classes; the item selection method, which is only implemented in the `Correlation` class. Section 3.3.1, *Default Voting* explains how this function works in detail.

4.4 Integrating CF and IR

When the implementation of two separate components for IR and CF are finished, it is necessary to integrate the two components in order to make them function together. In the implementation this was done by having the class `StatfindIRClient` (B.12) reference an instance of `CollaborativeFilter`, and run the ranked statistical variables through that filter to get the predicted votes, and then re-rank the statistical variables according to the given vote. Figure 4.2 shows how the classes are connected. The classes already described in 4.1 are omitted, except for `CollaborativeFilter` which is included for showing its place in the bigger context.

The weight on each components rank and predicted vote must be equal. One way of ensuring an evenly distributed weight is by employing the two-variable function

$$f(x, y) = (x + y) - (x * y) \tag{4.1}$$

This way low ranks will not punish high predicted votes, and vice versa. Each result will take the other into account, by adjusting the second value in a degree to the first. The range for x and y must be in [0,1] for the results to be balanced.

Integration Problems

A quite serious problem emerged when making the collaborative filtering algorithms work together with the information retrieval system (this should have been given more notice earlier). The IR component gives its ranks as values from 0 to 1, while the predicted vote ranges in the degree of the range of actual votes. This resulted in predicted values almost always being more than one, and thereby giving the predicted votes precedence over the rank given by the document search. To solve the problem,

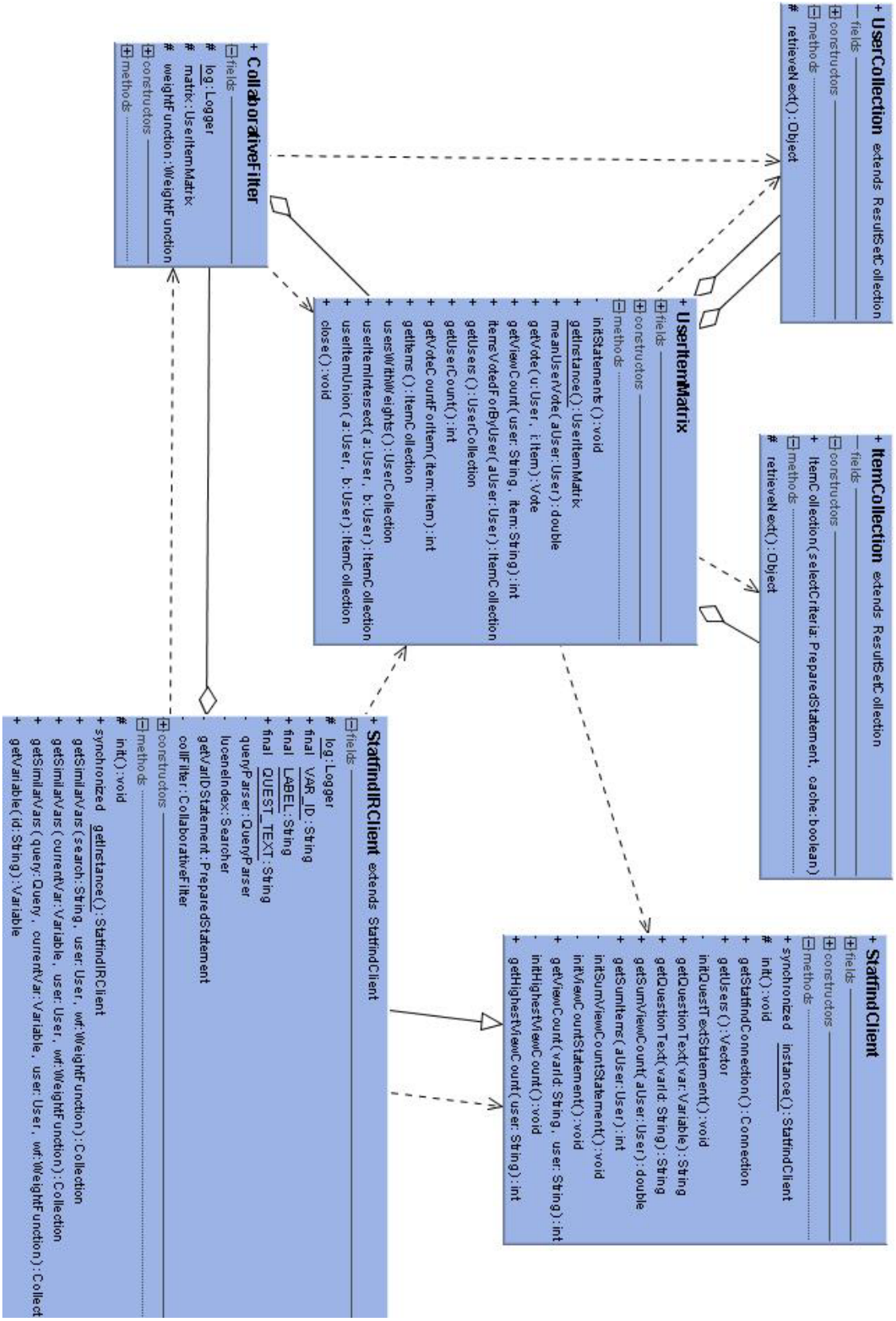

Figure 4.2: Collaborative Filtering system UML inheritance diagram

the predicted votes needs to be transposed in the range given by the IR component ([0,1]).

Transposing Predicted Votes

The data collected amounts to a total of 817 observations over 406 statistical variables, done by 7 users. The mean vote was 2.01 votes per statistical variable and the maximum vote for one is 29 — see figure 4.3. The plot maps statistical variable number (x-axis) against the number of times viewed by all the users (y-axis). If the

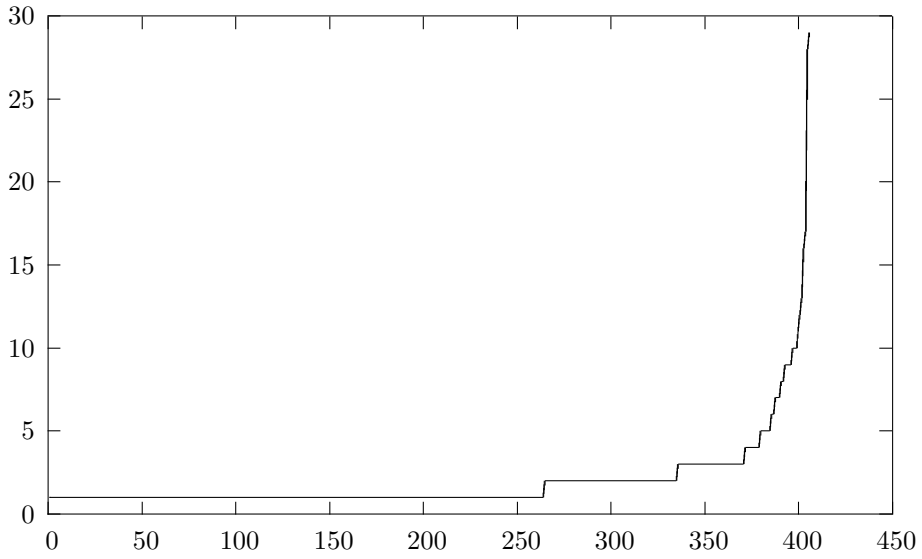

Figure 4.3: Plot of distribution of the collected votes

CF system uses the votes the way they are given, the predictions will potentially be in the same range as the votes (i.e. 0 – 29), and not compatible with equation (4.1). For the results ($f(x, y)$)to be balanced, equation (4.1) assumes the same range for x and y as well as being in the range of [0,1].

Thus some way of transposing the predicted votes into the range [0,1] is needed, and a linear function will not suffice as the current vote distribution is very uneven. This can be done by the use of a an exponential function asymptotically reaching for

1:

$$f(x) = 1 - q^{-v_{a,i}} \tag{4.2}$$

Where q is a calculated value equilibrium, in the form of:

$$q = c^{\frac{-1}{r}} \tag{4.3}$$

Where c is the value we want treat as the equilibrium, and r the number of required votes needed to reach c. This thesis used the values $c = 0.5$ and $r = 2$, which was selected with the plot in figure 4.3 in mind.

Although the last adaptations are ad hoc heuristics - they bring the system up to a working and testable condition. The values for c and r may easily be adjusted in a way that makes more sense, but as we do not know which way to adjust it, the correct adjustments is not discussed further here.

Chapter 5

Implementation issues

In order to test the research questions (section 3.5) software for this purpose had to be developed. This software includes a Java server, a plugin for an existing application, tool for analysing findings, to document indexing and retrieval tools, all these tools and components have been developed by the author.

Development Method

The components were developed in confirmation with "good" OO practice; the components are modular, well documented, and easy to extend. The documentation is described in appendix B as Javadoc embedded in the source code.

5.1 Constructing StatFind

Although the main focus of this thesis is on theoretical research questions, this chapter will look into the developed software from a technical point of view. The problem was how to retrieve information about the users' usage (explained in detail later) in a seamless non-intrusive way, with minimal changes to the existing software architecture in Nesstar.

The Ideal Solution

As briefly described in section 4.1, the Nesstar system is built around a JBoss J2EE environment. The first idea was to add functionality to the Nesstar Server, making it gather data about user requests. This approach seemed the least intrusive, and only small changes needed to be made. Problems arose after implementing this solution, as I was unaware of the caching scheme in the server and client; only the first requests in a session reached the server, because the subsequent requests where handled by the client only. Handling the information at the server could have proven best if the caching problem had been overcome, as the information could have been handled as close to its source as possible.

Later work suggests that there would have been other problems with this approach as well. When gathering data, the user context is important to take into account. When gathering data on the server, the user context is indistinct and difficult to grasp, as it only sees the objects the user request, not the situation the user is in. The user context is information about the user situation, e.g. not only what he is doing, but what led up to the current situation he is in.

The Working Solution

As adding the intended functionality to the Nesstar Server did not work as intended, other ideas had to be tested. When the information could not be generated as close to the server as possible, the other extreme looked promising; generating the information as close to the user as possible. This way, the information that deals with the user context is easier accessible than when the information is gathered at the server. All in all, the working solution is probably the best when the scope of this thesis is taken into account, when the amount of work to get the ideal solution is taken into account.

5.2 The StatFind Feeder

The only code added to the Nesstar Explorer is shown in figure 5.1. What it does (in short terms) is to collect the object (an instance of VariableNode[1]) the user has viewed, retrieves the variable identificator, and sends it off to the StatFind server, i.e. first it is passed over to the StatfindFeeder class, which in turn sends it off to the server. This class is supplied in a jar file bundled with the Nesstar Client application. This way the additions to the client code base is very small.

```
1  if(node instanceof VariableNode){
2    CompletedOperation op = new CompletedOperation();
3    VariableNode varNode = (VariableNode) node;
4    try {
5      //Retrieve the object from the server if not already done:
6      if(varNode.getVariable() == null)
7        varNode.obtainVariable();
8      op.setVarId(varNode.getVariable().getID());
9      StatfindFeeder.getInstance().processObject(op);
10   } catch (Exception e1) {
11     //don't bother
12   }
13 }
```

Figure 5.1: The source code needed to attach the plugin.

Plugin Activation

The contents of the supplied jar file does most of the work, so by executing the code in line 9, figure 5.1, all the steps in figure 5.2 are executed. This code is executed whenever a user clicks a statistical variable node in the Nesstar Client, see figure 5.3 for screenshot showing the described operation.

In order to send the details of a viewed object to the StatFind Server the component first checks if the server is started. This check results in the creation of the component[2], if the server is not previously started. The StatFind plugin reads a prop-

[1] A variable node is a node in a GUI tree containing a statistical variable.

[2] The component only starts once, and remains alive until the application is shut down.

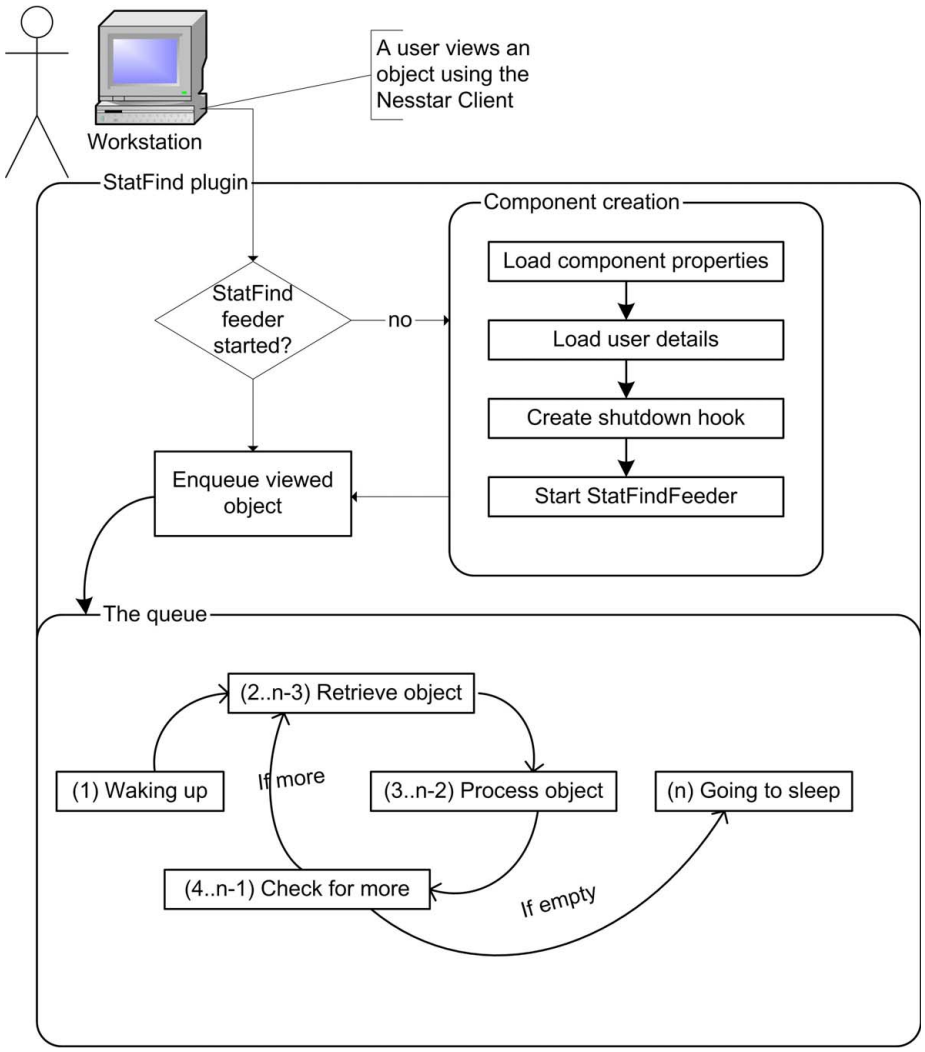

Figure 5.2: An overview of the steps execute by the StatFind client plugin.

erty file to load the connection details (see appendix B.13), and stores the current user name in the database at the specified address.

Securing User Privacy

The username is stored as a MD5 fingerprint[3]. A username such as *olvehansen* will with the MD5 hash become the unintelligible *19c493e053b98cdf42c80351b4f5a968*. This is to make sure that the system operator cannot misuse the information stored

[3]The MD5 algorithm takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input (Rivest, 1992).

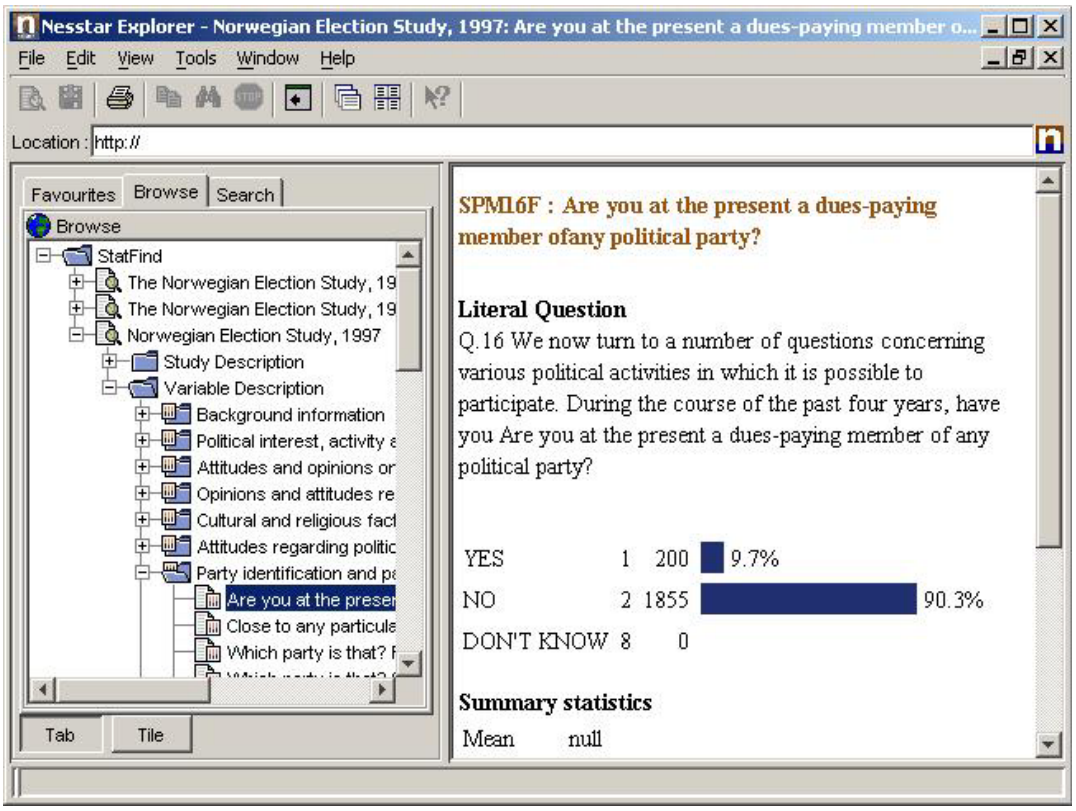

Figure 5.3: A screenshot showing a statistical variable selected using the browse tree.

in the system to trace what a user has looked at. In order to match the usage information to a specific user, the original username from the computer used to run the Nesstar Explorer is needed. All these operations happens in the *getInstance*() method in line 9, figure 5.1.

For readability purposes, the MD5 fingerprint is replaced with e.g. *User A* in examples in the text. See appendix C for MD5 keys.

Ensuring Client Responsiveness

In a client environment it is crucial that users do not experience negative side effects from using the StatFind plugin as portrayed. The queue, described in figure 5.2, has a separate thread of execution to avoid slow responses in the client user interface. This way one thread pushes an object to be sent to the server into the queue, and the queue itself takes elements from the queue at its own pace. The queue then checks

whether the object contains the information needed, and sends it to the server if the server is running (source code available in appendix B.2).

The nature of such a queue makes the system very robust, as the queue-thread only performs its operations until the queue is emptied. Then it becomes idle again — waiting for new objects to be enqueued. By caching objects to be processed, it lets the rest of the application continue unhindered by other obstacles such as a slow network connection.

5.3 The StatFind Server

The ideal solution would have been to handle the usage information in the Nesstar network communication protocol, but it proved too much work for this thesis. Instead of letting the Nesstar server handle the information, a stand-alone server had to be created for this purpose. The StatFind server is a fairly simple creation, although it has grown in complexity throughout the work on this thesis. It does not do much, but it is built to be very robust, as it must handle many asynchronous connections from many users at the same time. Figure 5.4 gives a superficial picture of how the servers are organized, the different components they consist of, and the lines of communication between the different components.

The Different States of StatFind

Starting up: When the server is started, it first performs some tests to find out if it can start. I.e. checks whether the required port is occupied, and if it is, tell the other server to shut down[4].

Next it starts the Jakarta/Lucene RMI server (more about this later) and reads property files containing some connection details and settings (see appendix B.13 for details).

Waiting for client to connect: Waiting for a client to connect involves listening to

[4]This will of course only work if the listening server is another StatFind server.

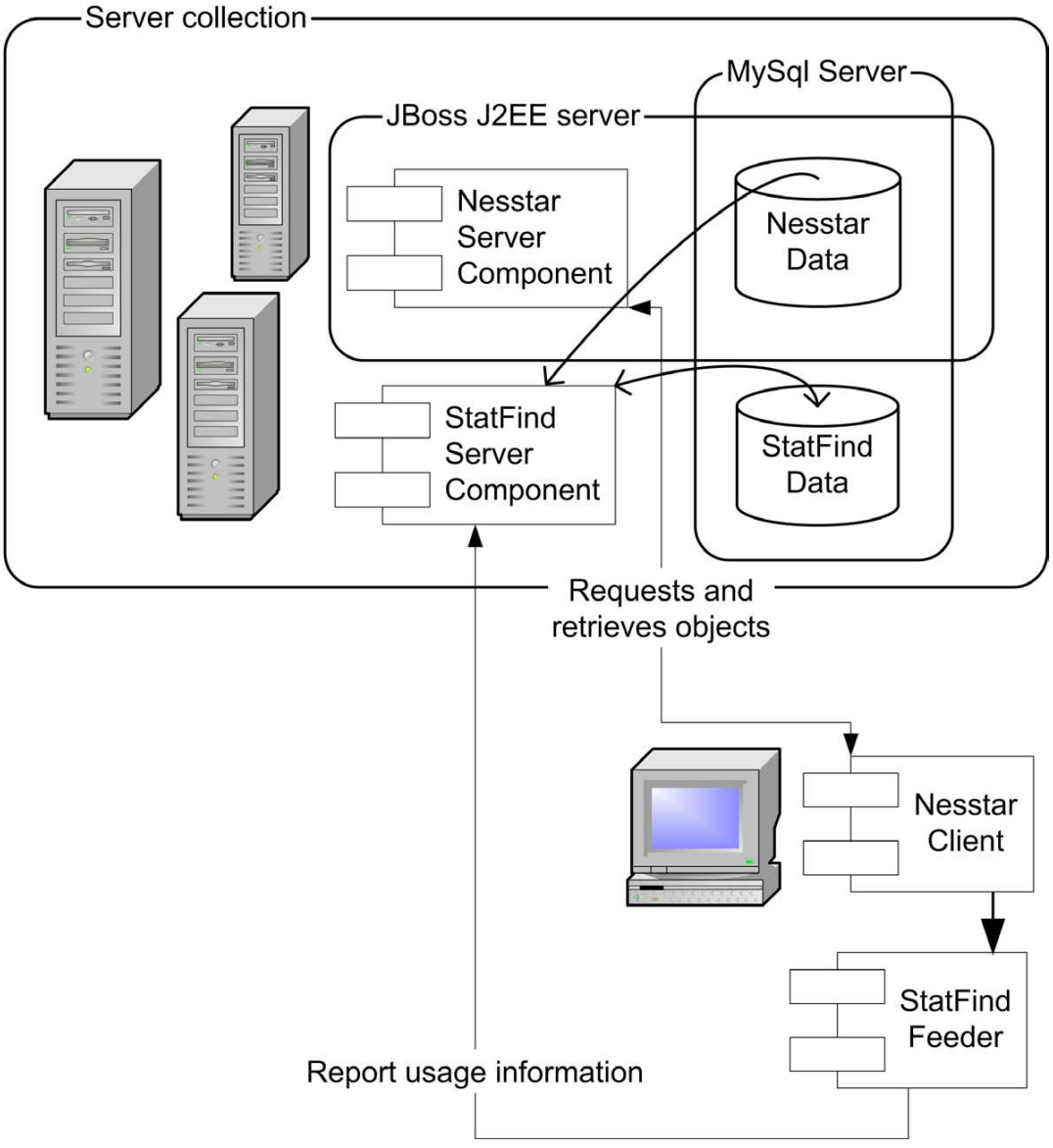

Figure 5.4: An overview of the client—server typology.

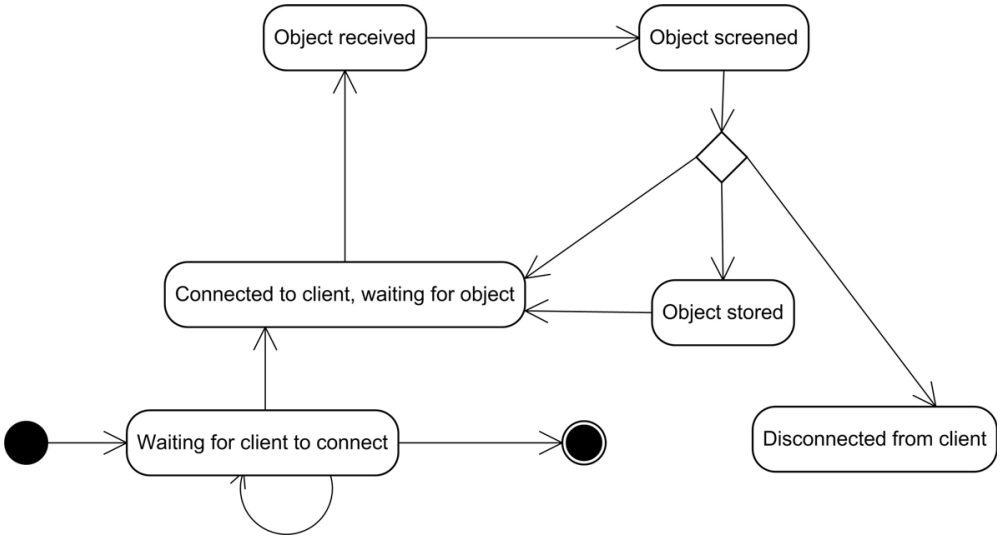

Figure 5.5: An overview of different states of a StatFind server.

a socket on a TCP/IP network connection. When a client connects to the correct socket (in this case socket 3306), the server spawns a new thread of execution and connects this thread to the database that is responsible for storing the usage information.

The server immediately goes back to waiting for new clients to connect.

Connected, waiting for object: This thread has the responsibility of handling objects that are sent from the connected client. When the client is connected to the server, the socket is open and a data stream, capable of streaming Java objects, is initiated.

Object received: Objects are received through the stream, and deserialised. The objects are checked for instance type, and discarded if the object is not known.

Object screened: The known objects are checked for usage information about statistical variables that are known to this server. If the statistical variable is unknown, the server discards the object and goes back to waiting for new objects.

Object stored: A known valid object is stored in the usage database. The users

usage information is correspondingly updated.

Disconnected from client: The object can also be an `EndOfStream` object, whose special meaning is that the client will disconnect. This is a way of shutting down the connection. Another way to disconnect is simply to close the stream. The database connections will nevertheless disconnect gracefully.

Shutting down: Before the program ends, the server first sends a disconnect message to all active threads, making them disconnect from the clients, and disconnect database connections gracefully.

Each of the above states corresponds to the states given in the diagram of figure 5.5.

StatFind Server Anatomy

Figure 5.4 shows which databases are used by which servers, and how the client connects to the different servers.

Early versions of the StatFind server only listened to a TCP port for connections, and did mainly what was described in figure 5.5. Later, several additions were made leading up to the current system portfolio:

- Robust server startup and shutdown routine.
- Statistical variable indexer, now using the Lucene indexing engine (Jakarta/Lucene, 2003).
- Usage cleaner — removal of usage information about statistical variables not used by the StatFind system.
- Component using RMI (Sun Microsystems Inc, 2003) for searching the indexed statistical variables, the two last components replace;
  - Statistical variable similarity indexer — indexing based on global analysis.
  - Search component, interfacing the statistical variable similarity database (made by variable similarity indexer).

– Stopword loader — load stopwords into a database, sorted by its category and language.

The usage cleaner was also eliminated, but as a consequence of general improvements in the StatFind server. Each received object is automatically screened so that only usage of statistical variables already contained in the Nesstar Server employed are actually stored in the StatFind database. Earlier this was not the case, and a component to clean up this database had to be implemented.

Figure 5.6 describes the `StatFindServer` class, and how it integrates with the Jakarta/Lucene server. The `Searchable` class connects to the indexed repository, while `RemoteSearchable` is the RMI server listening for client connections (legend for the UML diagrams is available in appendix B figure B.1). The classes portrayed here are only a selection for giving an overview of the Jakarata/Lucene server classes.

5.3.1 Jakarta/Lucene

The original indexing scheme was replaced for reasons given in section 4.2. From a practical point of view I think it is always feasible to use tested and available software rather than to "re-invent the wheel". This way, the developed software is made more versatile, rather than spending resources on developing specialised software to solve exactly one problem, which was the case with the *global analysis* system described in section 3.1.2.

Using third-party solutions like Jakarta/Lucene (2003) adds other useful features in addition to solving the problem at hand. These features include an implementation of Porter's stemming algorithm, allowing searches with a query of the users choice — not only statistical variable ids and adding RMI server capabilities.

Capabilities

By adding the Lucene text search engine the server is not only able to search for statistical variables similar to a given variable. It was very easy to extend the functionality to also encompass a free-text search. This way, the component also solves

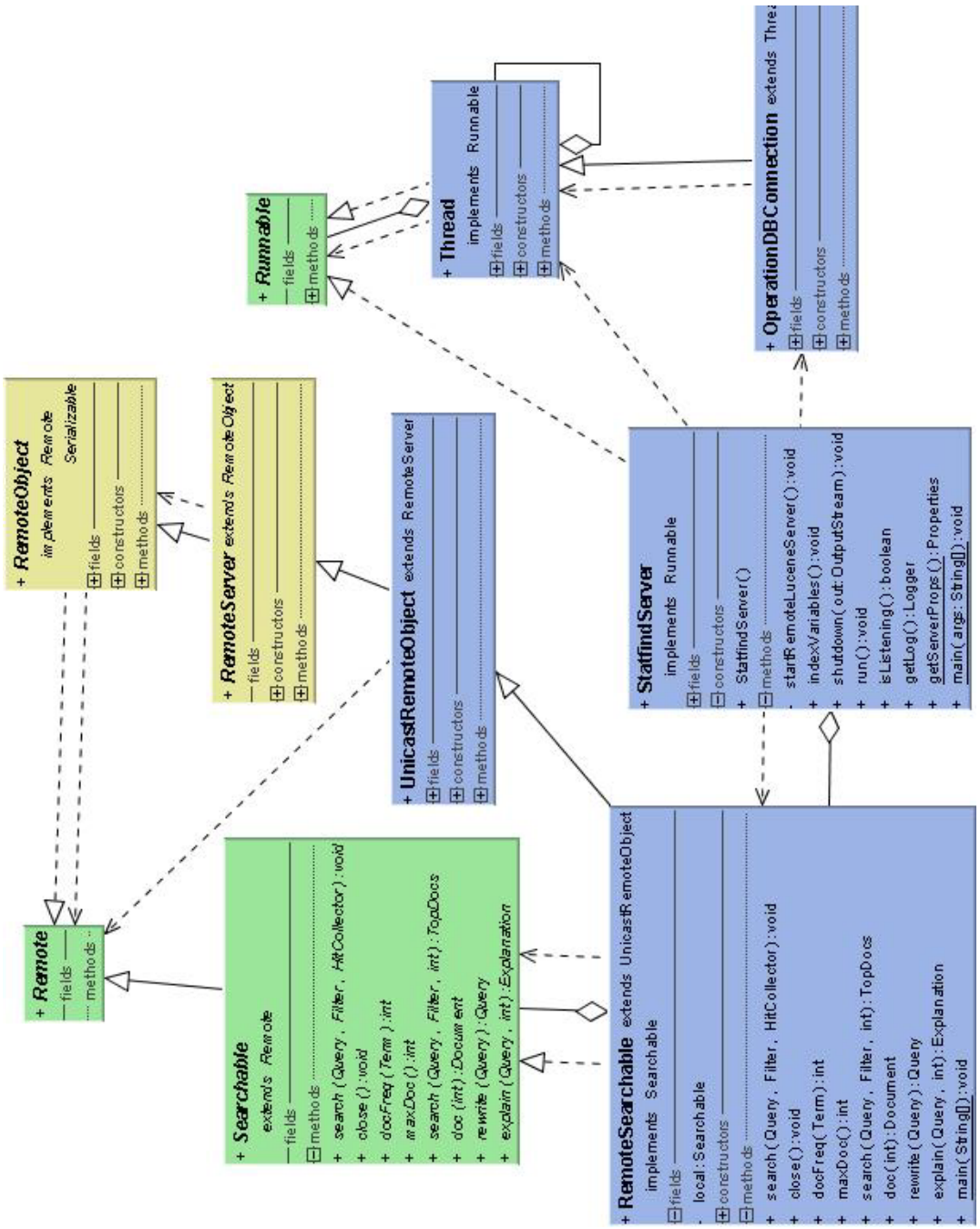

Figure 5.6: StatFind server UML inheritance diagram

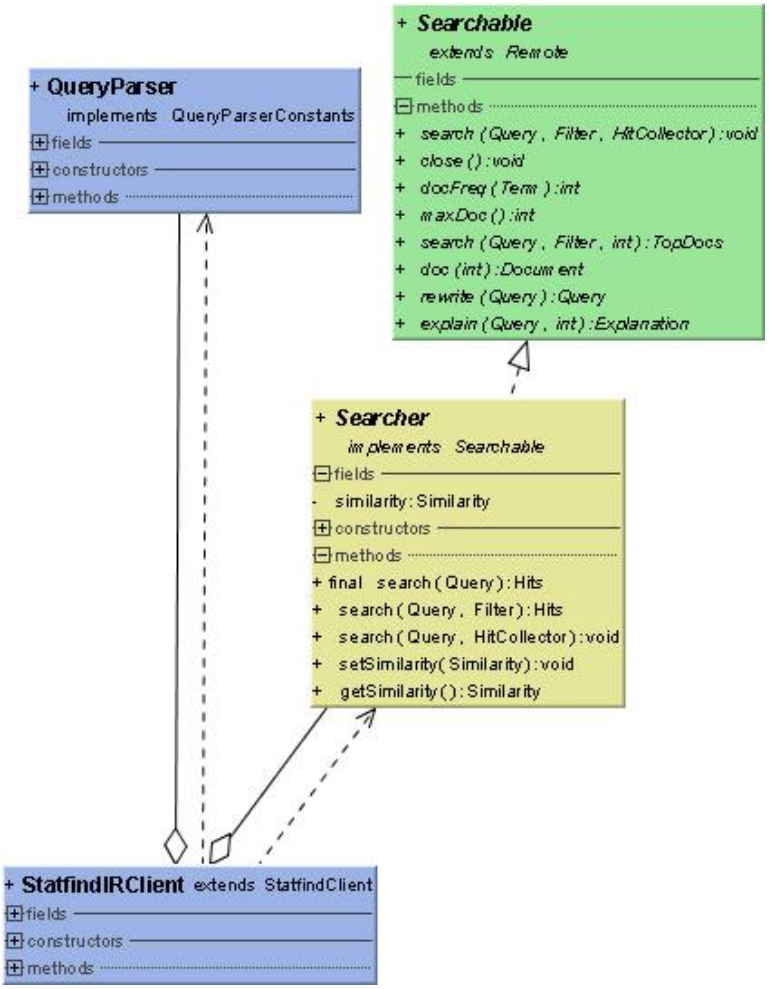

Figure 5.7: StatFind—Jakarta/Lucene integration, UML inheritance diagram

the ranking problem explained in section 2.1.1.

The Lucene search-engine also supported searching across a RMI connection, eliminating the need for making complex SQL expressions in order to perform a search. Figure 5.7 describes the integration of Lucene into StatFind.

The `QueryParser` parses text to make queries also supplies stop-words for the English language, thus eliminating that part of the original StatFind server. It also uses the Porter Stemming algorithm for reducing words to their grammatical stems. `Searcher` and `Searchable` in fig 5.7 is part of a client for connections to remote Lucene RMI servers.

Improvements

When the statistical variables' question-texts were indexed in the global analysis approach, the process lasted for approximately 10 minutes. The same process with Lucene lasts only for approximately 10 seconds. Of course the retrieval time of a search is longer using Lucene since the similarity figure is not pre-made as in the global analysis scheme, but overall the improvements are dramatic.

5.4 Clients Interfacing StatFind

Since the StatFind feeder is a central part of the StatFind system — it was described for itself in section 5.2 rather than in this section which is about the other StatFind clients.

There are actually several clients bundled in one client application. The client is called *StatFind VarFinder client*, and acts as a client for connecting to all the resources needed to present the statistical variables returned by the search engines. The client application should be regarded as a proof of concept prototype, as it does not draw a clear boundary between server and client. Also no additions to the functionality of the Nesstar Explorer was made. The StatFind Feeder does not apply, as it did not add any functionality to the Nesstar Explorer.

A large portion of the computation done in the client could ideally be performed by the server, but the scope of the thesis made the choice of developing the client as a heavy-weight client easy. Development of a thinner client involves a more complex communication protocol. StatFind, on the other hand, relies on connecting the client directly to the databases, instead of going by the way of a middle layer.

Figure 5.8 shows the different services the client connects to. A typical user session would be to enter a variable identificator or a term to search for in the text field, selecting the (encrypted) users preference base, and press *Start search* (see figure 5.9). It is also possible to fine tune the settings and which method to use for the collaborative filtering process.

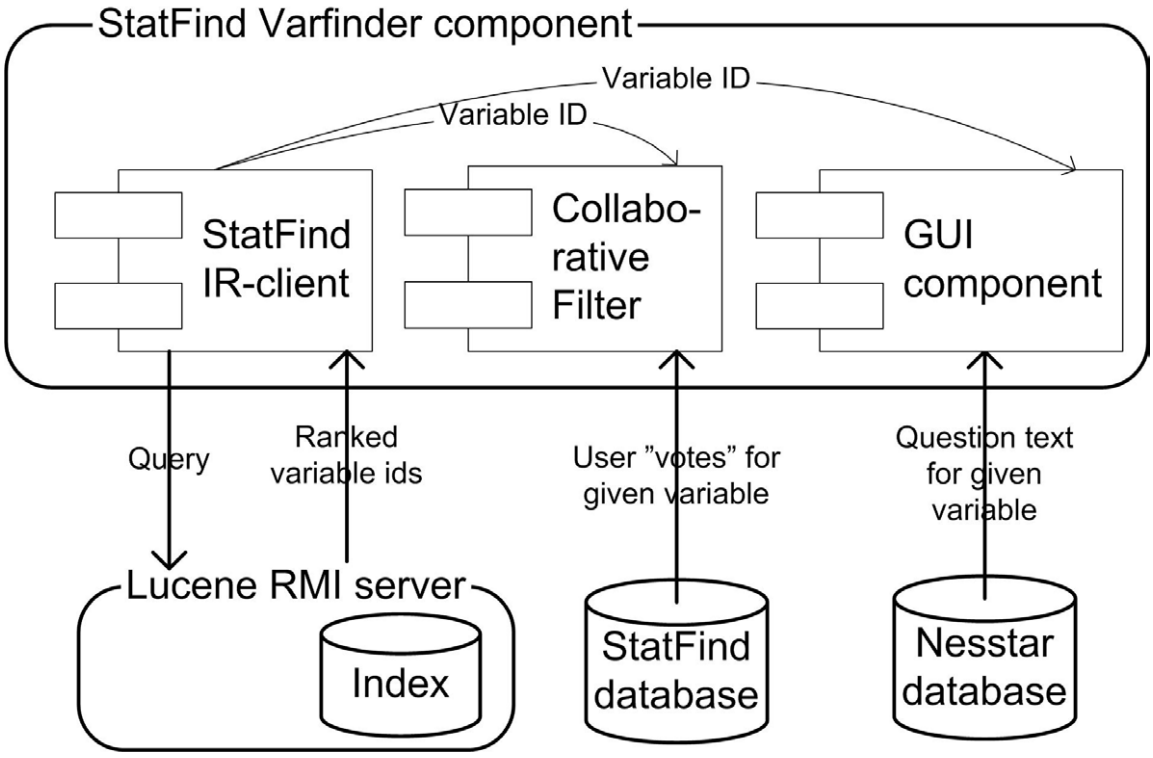

Figure 5.8: An overview of the StatFind VarFinder client

When the button is pressed, the query is made in the client and sent to the *Lucene RMI server*. This server searches the index and returns a ranked list of the statistical variable identificators; the rank being a number between 0 and 1.

For each statistical variable having a rank number over a preset threshold[5], the Collaborative Filter calculates the predicted vote for the current user, on the basis of the previous votes by other users, and the current variable identificator and user. The question texts that belong to the selected statistical variables are also retrieved from the Nesstar database and displayed in the client.

[5]Throughout this thesis the threshold was set to 0.025, as the value decreases rapidly after the first few hits. Before switching to Lucene this figure was 0.5.

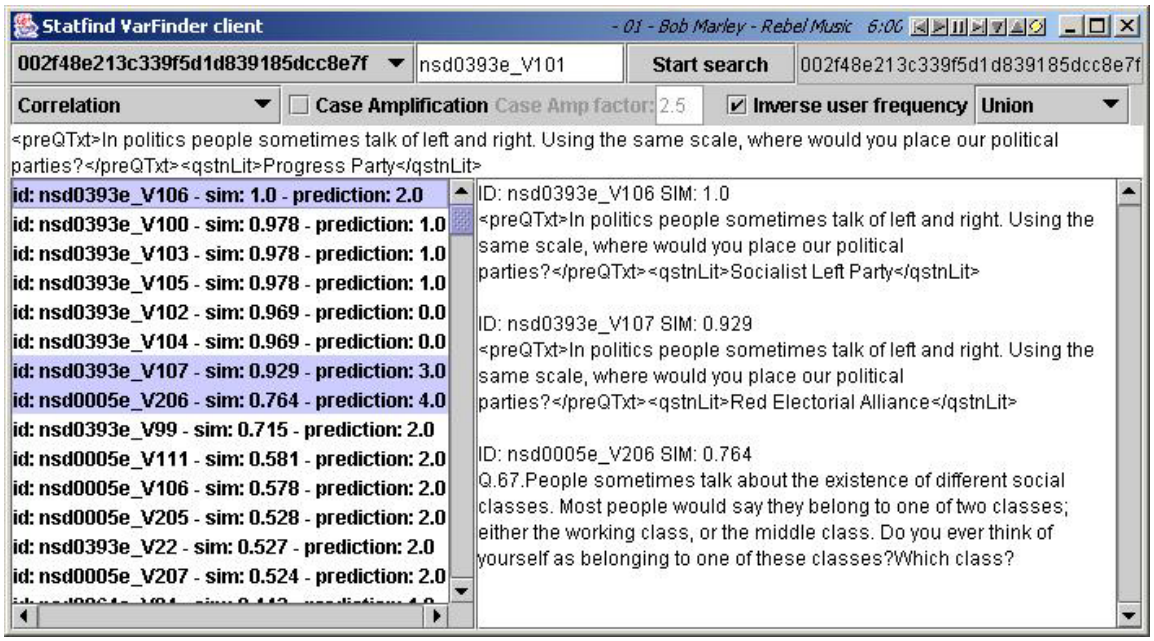

Figure 5.9: A screenshot of the StatFind VarFinder component

5.5 Notes on Development Tools

When developing software, alone or in a team with several other developers, some tools to assist your work are highly necessary. Following is a description of such tools which was used in thesis development project.

Ant

When compiling Java source code, there are numerous other things that need to be in place. It be copying of resource files, fetching latest code from CVS, starting and stopping services, etc. Ant is a tool to help build Java applications, based on XML. Ant can compile, make Java jar files, sign and many other useful actions (The Apache Software Foundation, 2003a).

CVS

When working alone or in groups with others, it is crucial to have control over different revisions of the source code. CVS is a tool which helps integrating changes done by

many people, and to keep track of all versions of the source code (CollabNet, Inc, 2003).

IntelliJ IDEA

An editor for writing Java source code should support many activities surrounding the coding activity itself. IntelliJ IDEA support integration of tools like Ant and CVS, as well as other useful techniques such as refactoring[6] (JetBrains, Inc, 2003).

log4j

All developers make the applications output debug and log information to see what happens when, and in what order. Log4j structures logging to keep the logging routines tidy and well defined (The Apache Software Foundation, 2003b).

[6] Refactoring is a technique to restructure code in a disciplined way.

Chapter 6

Evaluation and findings

The purpose of this thesis as described in 2.1.2 is;

In what way can a tool for helping users find similar statistical variables be implemented, and which methods are sufficient for measuring the similarity between variables.

In order to test this statement chapter 2 narrowed down the problem and chapter 3 discussed the theory necessary for investigating the problem domain. As well as pointing towards what needed to be developed in order to solve the problem given in chapter 2. The last two chapters have discussed and described how the theories are put to use (chapter 4) as well as given a thorough walkthrough of the software developed for this thesis (chapter 5). The work described in those two chapters form the foundation for this chapter, that analyses how the implemented system performs.

Retrieval Performance Evaluation

We are here dealing with IR systems that retrieve and rank search results based on some existing similarity model. Since similarity models are crude approximations to natural language, it must make presumptions about the text it models. The collection of documents retrieved/ranked based on this similarity model are prone/likely to make mistakes — no model can be entirely precise. This is why a central area in IR is evaluation of the retrieval system.

The evaluation process has several methods, but the two most widely used are recall and precision. Let R be the set of documents relevant to a user information request, and A the set of documents actually retrieved by an IR system — the answer set. Ra is then the intersection of these two sets. Recall is the fraction of the number of relevant documents (the set R) which has been retrieved:

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

Precision is the fraction of the number of retrieved documents (the set A) which is relevant:

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

Ideally, each document in the answer set should be relevant, and the order of the documents should be ranked in their degree of relevance. The ranking method is almost equally important as the retrieval/indexing method, because it determines the order a user normally browse a result set If the first items in this set are irrelevant for a user, then the next ones should ideally be more irrelevant. Ranking can be given by the indexing method, as it gives a figure of document similarity, but this figure can also be adjusted after the result set is computed.

In order to test if a given IR engine performs in an adequate fashion, several sets of documents must be prepared and divided in to two collections; a test collection and reference collection. Searches are performed against the test collection and compared to the reference collection for evaluation (Baeza-Yates and Ribeiro-Neto, 1999c).

Lucene and Retrieval Performance Evaluation

Since Lucene (Jakarta/Lucene, 2003) is made in accordance to established IR principles (see section 4.2 and section 5.3.1), one can argue that it has a fair possibility to achieve good scores in both precision and recall. The ultimate score is of course dependant on rigorous testing and adaptations, as well as precise continuous adjustments with regards to the results of precision and recall testing.

This thesis have no intentions in doing a precision and recall test, as it trusts the default settings of Lucene. Later work might involve conducting these tests and

conclude a more final verdict of lucene as an indexing engine, and how well it proves fitting in the current domain.

The crucial part of the system as it is built for this thesis is how the ranking given by the IR engine is re-ranked according to a figure calculated by a collaborative filter. This collaborative filter uses information about users and their behaviour to predict their interest for the re-ranking of items. Collecting user information by treating given action(s) as a "positive" operation, thus treating that action(s) as a vote for the item(s) involved. Translating to the treatment of the user operation performed to view documentation for a statistical variable as an action of preference — interpreted as a sign that the user likes or finds the documentation interesting.

6.1 StatFind Evaluation

6.1.1 Test Set-up and Data Quality

As mentioned earlier, the data gathered for use in this thesis, are not gathered in a real life setting. Nesstar is intended for users searching for statistical studies. Users might be experienced statisticians, journalists or the plain man searching for information in public registers.

During the development of of this thesis, the Nesstar System was still in a beta stage, and at the time of writing, still is. This limited the data gathering to usage done by developers and testers. Their usage tends to be quite uniform, as they are most interested in the behavour of the system, and not necessarily its contents (the statistical information). The first phase of data collection relied on usage of this kind. Early data analysis showed that the data was inadequate for use in this thesis.

After the first phase of data gathering was rejected, a new setting of users was needed. Since real life users were unavailable, fellow students from the Department of Information Science were used. A small group of students was willing to use the Nesstar Explorer and the statistical surveys prepared for use in the Nesstar Server. Three statistical studies concering the political climate in Norway was used as search

material. The students were asked to browse and search this data in a way of their liking. They knew how the StatFind system works, and this might have biased the gathered data (e.g. some variables having 20-30 observations).

By using students this way the observations became more dispersed than when relying on the test and development users, and better suited for tesing StatFind, although not of the quality originally sought.

6.1.2 Search Evaluation

The technique used for analysing the system is as following; several searches are performed first, using only the IR engine. In this way only the textual similarity is measured. This is in turn compared to several other results using the different CF algorithms and settings. Differences will be highlighted and discussed whether it is a positive effect given by the collaborative filter or if the filter does not work as intended.

6.1.3 Evaluating Search for *politics*

The first listing is a result from searching for the word *politics*. The first column gives the id of the statistical variable in the database, and the other column gives its similarity to the query. To see how the similarity relates to the textual content of the variables, look to appendix A.1.

id: nsd0005e_V33 - sim: 1.0	id: nsd0393e_V107 - sim: 0.53
id: nsd0064e_V14 - sim: 1.0	id: nsd0005e_V32 - sim: 0.442
id: nsd0393e_V98 - sim: 0.612	id: nsd0064e_V220 - sim: 0.442
id: nsd0064e_V13 - sim: 0.53	id: nsd0064e_V221 - sim: 0.442
id: nsd0064e_V217 - sim: 0.53	id: nsd0064e_V238 - sim: 0.442
id: nsd0393e_V41 - sim: 0.53	id: nsd0393e_V37 - sim: 0.442
id: nsd0393e_V99 - sim: 0.53	id: nsd0393e_V38 - sim: 0.442
id: nsd0393e_V100 - sim: 0.53	id: nsd0393e_V39 - sim: 0.442
id: nsd0393e_V101 - sim: 0.53	id: nsd0393e_V40 - sim: 0.442
id: nsd0393e_V102 - sim: 0.53	id: nsd0005e_V44 - sim: 0.354
id: nsd0393e_V103 - sim: 0.53	id: nsd0005e_V50 - sim: 0.354
id: nsd0393e_V104 - sim: 0.53	id: nsd0064e_V77 - sim: 0.354
id: nsd0393e_V105 - sim: 0.53	id: nsd0393e_V30 - sim: 0.354
id: nsd0393e_V106 - sim: 0.53	id: nsd0393e_V31 - sim: 0.354

Table 6.1: Search for *politics*, document similarity only

Table 6.1 shows a search using document similarity only, while the three next tables show searches for politics, but with different collaborative filtering methods. The first table (6.2) uses the vector similarity method, with case amplification of 2.5, as suggested in Breese et al. (1998). Inverse user frequency (see 3.3.1 for explanation) is also turned on in these examples (except one), as preliminary testing showed that it yielded the most distinct results.

User A, used for these evaluations, has viewed 117 different statistical variables, has a view count of 203, meaning that the 117 different variables have totally been viewed 203 times.

CF Setting 1

When searching for a single word, the textual comparison is very likely to return a hit with the similarity of 1, as the whole word appears in the retrieved document, possibly several times as well. This way the CF engine does not have much influence on the ranking method in order to put another document on the top of the ranking. In these cases it only has influence on the middle part of the rank. Results close to one have less possibility for change by lower values. Observe this finding by comparing

id: nsd0005e_V33 - sim: 1.0 - CF: 1.0	id: nsd0393e_V102 - sim: 0.53 - CF: 0.0
id: nsd0064e_V14 - sim: 1.0 - CF: 1.0	id: nsd0393e_V104 - sim: 0.53 - CF: 0.0
id: nsd0393e_V101 - sim: 0.834 - CF: 3.0	id: nsd0005e_V32 - sim: 0.442 - CF: 0.0
id: nsd0393e_V107 - sim: 0.834 - CF: 3.0	id: nsd0064e_V221 - sim: 0.442 - CF: 0.0
id: nsd0393e_V99 - sim: 0.765 - CF: 2.0	id: nsd0064e_V238 - sim: 0.442 - CF: 0.0
id: nsd0393e_V106 - sim: 0.765 - CF: 2.0	id: nsd0393e_V37 - sim: 0.442 - CF: 0.0
id: nsd0064e_V13 - sim: 0.668 - CF: 1.0	id: nsd0393e_V38 - sim: 0.442 - CF: 0.0
id: nsd0393e_V100 - sim: 0.668 - CF: 1.0	id: nsd0393e_V39 - sim: 0.442 - CF: 0.0
id: nsd0393e_V103 - sim: 0.668 - CF: 1.0	id: nsd0393e_V40 - sim: 0.442 - CF: 0.0
id: nsd0393e_V105 - sim: 0.668 - CF: 1.0	id: nsd0005e_V44 - sim: 0.354 - CF: 0.0
id: nsd0393e_V98 - sim: 0.612 - CF: 0.0	id: nsd0005e_V50 - sim: 0.354 - CF: 0.0
id: nsd0064e_V220 - sim: 0.605 - CF: 1.0	id: nsd0064e_V77 - sim: 0.354 - CF: 0.0
id: nsd0064e_V217 - sim: 0.53 - CF: 0.0	id: nsd0393e_V30 - sim: 0.354 - CF: 0.0
id: nsd0393e_V41 - sim: 0.53 - CF: 0.0	id: nsd0393e_V31 - sim: 0.354 - CF: 0.0

Table 6.2: Search for *politics*, using vector similarity, inverse user frequency and case amplification factor of 2.5

the results in 6.2 with the results in 6.1.

CF Setting 2

In this case (table 6.3) correlation performs exactly identical as with vector similarity from the previous example (table 6.2). Vector similarity needs fewer calculations, so if the results continue to be identical, vector similarity should be the algorithm of choice, as it needs less time to execute.

CF Setting 3

With the few data that are collected for these tests, the union item selection method should yield better results than with intersection (table 6.4), as it has more items to do the calculations over than for the intersect item selection method (Breese et al., 1998). Using the intersection item selection method, the predictions given by the CF engine in table 6.4 end up being very similar too each other. This may be a concequence of to few user observations, as it cuts the item base utilised dramatically (see 3.3.1 for details about union and intersection). In itself this is not necessarily

id: nsd0005e_V33 - sim: 1.0 - CF: 1.0	id: nsd0393e_V102 - sim: 0.53 - CF: 0.0
id: nsd0064e_V14 - sim: 1.0 - CF: 1.0	id: nsd0393e_V104 - sim: 0.53 - CF: 0.0
id: nsd0393e_V101 - sim: 0.834 - CF: 3.0	id: nsd0005e_V32 - sim: 0.442 - CF: 0.0
id: nsd0393e_V107 - sim: 0.834 - CF: 3.0	id: nsd0064e_V221 - sim: 0.442 - CF: 0.0
id: nsd0393e_V99 - sim: 0.765 - CF: 2.0	id: nsd0064e_V238 - sim: 0.442 - CF: 0.0
id: nsd0393e_V106 - sim: 0.765 - CF: 2.0	id: nsd0393e_V37 - sim: 0.442 - CF: 0.0
id: nsd0064e_V13 - sim: 0.668 - CF: 1.0	id: nsd0393e_V38 - sim: 0.442 - CF: 0.0
id: nsd0393e_V100 - sim: 0.668 - CF: 1.0	id: nsd0393e_V39 - sim: 0.442 - CF: 0.0
id: nsd0393e_V103 - sim: 0.668 - CF: 1.0	id: nsd0393e_V40 - sim: 0.442 - CF: 0.0
id: nsd0393e_V105 - sim: 0.668 - CF: 1.0	id: nsd0005e_V44 - sim: 0.354 - CF: 0.0
id: nsd0393e_V98 - sim: 0.612 - CF: 0.0	id: nsd0005e_V50 - sim: 0.354 - CF: 0.0
id: nsd0064e_V220 - sim: 0.605 - CF: 1.0	id: nsd0064e_V77 - sim: 0.354 - CF: 0.0
id: nsd0064e_V217 - sim: 0.53 - CF: 0.0	id: nsd0393e_V30 - sim: 0.354 - CF: 0.0
id: nsd0393e_V41 - sim: 0.53 - CF: 0.0	id: nsd0393e_V31 - sim: 0.354 - CF: 0.0

Table 6.3: Search for *politics*, using correlation, inverse user frequency and case amplification factor of 2.5, items selected by union.

unsatisfactory, but the results calculated by the CF engine does not differ very much and it is difficult to read any result from them. These scores might be in its place sometimes, as similar values gives a similar adjustment, and thus have only a small impact on the result.

6.1.4 Evaluating Search for *politics*, Changed Usage Base

This part of the evaluation of StatFind demonstrates the changes in the same search as above, but with changed usage statistics. Before this search was performed, *User A*'s viewings of statistical variables with a view count ≥ 2 and ≤ 5 was deleted. The printout in table 6.5 is from this search, using correlation, union item selection, inverse user frequency and a case amplification of 2.5.

The changes in table 6.5 from table 6.3 was provoked by deleting some of the usage base for the user. Experiments with deletion of other users usage information were performed, and it had minor impact on the calculated predictions of *User A*. Because of the poor data quality the likely explanation is that most of the votes used as basis for calculating the ranks in 6.5 are *User A*'s own votes. Table 6.1.4 shows

id: nsd0005e_V33 - sim: 1.0 - CF: 2.735	id: nsd0393e_V107 - sim: 0.818 - CF: 2.735
id: nsd0064e_V14 - sim: 1.0 - CF: 2.735	id: nsd0005e_V32 - sim: 0.784 - CF: 2.735
id: nsd0393e_V98 - sim: 0.85 - CF: 2.735	id: nsd0064e_V220 - sim: 0.784 - CF: 2.735
id: nsd0064e_V13 - sim: 0.818 - CF: 2.735	id: nsd0064e_V221 - sim: 0.784 - CF: 2.735
id: nsd0064e_V217 - sim: 0.818 - CF: 2.735	id: nsd0064e_V238 - sim: 0.784 - CF: 2.735
id: nsd0393e_V41 - sim: 0.818 - CF: 2.735	id: nsd0393e_V38 - sim: 0.784 - CF: 2.735
id: nsd0393e_V99 - sim: 0.818 - CF: 2.735	id: nsd0393e_V39 - sim: 0.784 - CF: 2.735
id: nsd0393e_V100 - sim: 0.818 - CF: 2.735	id: nsd0393e_V40 - sim: 0.784 - CF: 2.735
id: nsd0393e_V101 - sim: 0.818 - CF: 2.735	id: nsd0064e_V77 - sim: 0.75 - CF: 2.735
id: nsd0393e_V102 - sim: 0.818 - CF: 2.735	id: nsd0393e_V30 - sim: 0.75 - CF: 2.735
id: nsd0393e_V103 - sim: 0.818 - CF: 2.735	id: nsd0393e_V31 - sim: 0.75 - CF: 2.735
id: nsd0393e_V104 - sim: 0.818 - CF: 2.735	id: nsd0393e_V37 - sim: 0.567 - CF: 0.735
id: nsd0393e_V105 - sim: 0.818 - CF: 2.735	id: nsd0005e_V44 - sim: 0.499 - CF: 0.735
id: nsd0393e_V106 - sim: 0.818 - CF: 2.735	id: nsd0005e_V50 - sim: 0.499 - CF: 0.735

Table 6.4: Search for *politics*, using correlation, inverse user frequency and case amplification factor of 2.5, items selected by intersection.

the statistical variables that users have viewed more than once. This amounts to 66 variables, which is aproximately 1/4 of the total number of variables viewed by the users. In itself this is not a small fraction of the total, but the low number of actual user views may impede the vote prediction process.

id: nsd0005e_V33 - sim: 1.0 - CF: 1.0	id: nsd0393e_V106 - sim: 0.53 - CF: 0.0
id: nsd0064e_V14 - sim: 1.0 - CF: 1.0	id: nsd0393e_V107 - sim: 0.53 - CF: 0.0
id: nsd0064e_V13 - sim: 0.668 - CF: 1.0	id: nsd0005e_V32 - sim: 0.442 - CF: 0.0
id: nsd0393e_V100 - sim: 0.668 - CF: 1.0	id: nsd0064e_V221 - sim: 0.442 - CF: 0.0
id: nsd0393e_V103 - sim: 0.668 - CF: 1.0	id: nsd0064e_V238 - sim: 0.442 - CF: 0.0
id: nsd0393e_V105 - sim: 0.668 - CF: 1.0	id: nsd0393e_V37 - sim: 0.442 - CF: 0.0
id: nsd0393e_V98 - sim: 0.612 - CF: 0.0	id: nsd0393e_V38 - sim: 0.442 - CF: 0.0
id: nsd0064e_V220 - sim: 0.605 - CF: 1.0	id: nsd0393e_V39 - sim: 0.442 - CF: 0.0
id: nsd0064e_V217 - sim: 0.53 - CF: 0.0	id: nsd0393e_V40 - sim: 0.442 - CF: 0.0
id: nsd0393e_V41 - sim: 0.53 - CF: 0.0	id: nsd0005e_V44 - sim: 0.354 - CF: 0.0
id: nsd0393e_V99 - sim: 0.53 - CF: 0.0	id: nsd0005e_V50 - sim: 0.354 - CF: 0.0
id: nsd0393e_V101 - sim: 0.53 - CF: 0.0	id: nsd0064e_V77 - sim: 0.354 - CF: 0.0
id: nsd0393e_V102 - sim: 0.53 - CF: 0.0	id: nsd0393e_V30 - sim: 0.354 - CF: 0.0
id: nsd0393e_V104 - sim: 0.53 - CF: 0.0	id: nsd0393e_V31 - sim: 0.354 - CF: 0.0

Table 6.5: Search for *politics*, changed usage base

nsd0005e_V207 #users: 4	nsd0005e_V280 #users: 2	nsd0064e_V227 #users: 2
nsd0064e_V30 #users: 3	nsd0064e_V69 #users: 2	nsd0064e_V98 #users: 2
nsd0393e_V2 #users: 3	nsd0005e_V211 #users: 2	nsd0393e_V9 #users: 2
nsd0005e_V206 #users: 3	nsd0064e_V274 #users: 2	nsd0005e_V205 #users: 2
nsd0005e_V5 #users: 3	nsd0064e_V55 #users: 2	nsd0005e_V281 #users: 2
nsd0393e_V19 #users: 3	nsd0393e_V121 #users: 2	nsd0005e_V49 #users: 2
nsd0005e_V278 #users: 3	nsd0064e_V240 #users: 2	nsd0005e_V79 #users: 2
nsd0005e_V44 #users: 3	nsd0393e_V37 #users: 2	nsd0064e_V107 #users: 2
nsd0005e_V209 #users: 3	nsd0005e_V109 #users: 2	nsd0064e_V18 #users: 2
nsd0064e_V96 #users: 3	nsd0005e_V6 #users: 2	nsd0064e_V72 #users: 2
nsd0005e_V279 #users: 3	nsd0064e_V13 #users: 2	nsd0005e_V166 #users: 2
nsd0064e_V239 #users: 2	nsd0064e_V272 #users: 2	nsd0064e_V14 #users: 2
nsd0005e_V208 #users: 2	nsd0393e_V12 #users: 2	nsd0064e_V22 #users: 2
nsd0005e_V84 #users: 2	nsd0005e_V33 #users: 2	nsd0393e_V189 #users: 2
nsd0005e_V186 #users: 2	nsd0064e_V16 #users: 2	nsd0005e_V219 #users: 2
nsd0005e_V39 #users: 2	nsd0393e_V10 #users: 2	nsd0064e_V101 #users: 2
nsd0064e_V59 #users: 2	nsd0393e_V17 #users: 2	nsd0064e_V241 #users: 2
nsd0005e_V167 #users: 2	nsd0393e_V66 #users: 2	nsd0393e_V217 #users: 2
nsd0005e_V213 #users: 2	nsd0005e_V106 #users: 2	nsd0064e_V94 #users: 2
nsd0005e_V234 #users: 2	nsd0005e_V50 #users: 2	nsd0005e_V130 #users: 2
nsd0005e_V37 #users: 2	nsd0393e_V227 #users: 2	
nsd0064e_V97 #users: 2	nsd0005e_V38 #users: 2	
nsd0393e_V88 #users: 2	nsd0064e_V15 #users: 2	

Table 6.6: Overview over variables viewed by more than one users.

6.1.5 Evaluating Search for id *nsd0064e_V14*

In this evaluation a user with few registered observations is used, demonstrating how the document similarity measure takes over the similarity figure when the calculated prediciton is low. *User E* has viewed 50 different statistical variables a total of 69 times. To see how the similarity relates to the textual content of the variables look to appendix A.2.

id: nsd0005e_V33 - sim: 0.631	id: nsd0393e_V105 - sim: 0.035
id: nsd0393e_V38 - sim: 0.064	id: nsd0393e_V106 - sim: 0.035
id: nsd0393e_V98 - sim: 0.041	id: nsd0393e_V107 - sim: 0.035
id: nsd0064e_V13 - sim: 0.036	id: nsd0064e_V238 - sim: 0.03
id: nsd0064e_V217 - sim: 0.035	id: nsd0005e_V32 - sim: 0.03
id: nsd0393e_V99 - sim: 0.035	id: nsd0064e_V220 - sim: 0.029
id: nsd0393e_V100 - sim: 0.035	id: nsd0064e_V221 - sim: 0.029
id: nsd0393e_V101 - sim: 0.035	id: nsd0393e_V37 - sim: 0.029
id: nsd0393e_V102 - sim: 0.035	id: nsd0393e_V39 - sim: 0.029
id: nsd0393e_V103 - sim: 0.035	id: nsd0064e_V26 - sim: 0.026
id: nsd0393e_V104 - sim: 0.035	id: nsd0064e_V25 - sim: 0.025

Table 6.7: Search for *nsd0064e_V14*, document similarity only

On Using Statistical Variables as Query

When using a statistical variable as a query, each word in the variable documentation is used when the query is generated. The words are put into the query parser of Lucene where they are preprocessed — i.e. stemmed and having stopwords removed. Each word is appended with a boolean *or*. Notice how rapidly the similarity declines as a consequence of using many words in the query. This yields a much more nuanced rating list, as the chance in finding a variable in which the documentation is identical[1] to the query is quite small, since many words are used.

[1] They do not need to be excatly identical, only index-identical as the IR system transforms the original text by removal of stopwords and stemming of the remaining words.

Comparing Results

The two methods that looked most promising from the first evaluation part (section 6.1.3) was vector similarity and correlation with union item selection method, giving identical results in 6.2 and 6.3. Comparing table table 6.8 and 6.9 shows results slightly contradictory to the results from section 6.1.3. Here, the two methods do not give the same results. In fact vector similarity in table 6.9 gives the most varied results. Correlation gives a higher, but the same prediction for all items in table 6.8, which gives all items a slight boost on the ranking.

id: nsd0005e_V33 - sim: 0.838 - CF: 2.38	id: nsd0393e_V105 - sim: 0.577 - CF: 2.38
id: nsd0393e_V38 - sim: 0.59 - CF: 2.38	id: nsd0393e_V106 - sim: 0.577 - CF: 2.38
id: nsd0393e_V98 - sim: 0.58 - CF: 2.38	id: nsd0393e_V107 - sim: 0.577 - CF: 2.38
id: nsd0064e_V13 - sim: 0.577 - CF: 2.38	id: nsd0064e_V238 - sim: 0.575 - CF: 2.38
id: nsd0064e_V217 - sim: 0.577 - CF: 2.38	id: nsd0005e_V32 - sim: 0.575 - CF: 2.38
id: nsd0393e_V99 - sim: 0.577 - CF: 2.38	id: nsd0064e_V220 - sim: 0.574 - CF: 2.38
id: nsd0393e_V100 - sim: 0.577 - CF: 2.38	id: nsd0064e_V221 - sim: 0.574 - CF: 2.38
id: nsd0393e_V101 - sim: 0.577 - CF: 2.38	id: nsd0393e_V37 - sim: 0.574 - CF: 2.38
id: nsd0393e_V102 - sim: 0.577 - CF: 2.38	id: nsd0393e_V39 - sim: 0.574 - CF: 2.38
id: nsd0393e_V103 - sim: 0.577 - CF: 2.38	id: nsd0064e_V26 - sim: 0.573 - CF: 2.38
id: nsd0393e_V104 - sim: 0.577 - CF: 2.38	id: nsd0064e_V25 - sim: 0.573 - CF: 2.38

Table 6.8: Search for *nsd0064e_V14*, using correlation, inverse user frequency and case amplification factor of 2.5, items selected by union.

Preliminary Concluding Remarks for Evaluation

The predicted values from the vector similarity in table 6.9 are actually the same as *User E*'s values for view count of the same statistical variables. This way, the correlation actually performs better, as its predictions are more independant from the users own data than the predictions given for vector similarity. The problem of equal values for prediction and view count is discussed later.

id: nsd0005e_V33 - sim: 0.631 - CF: 0.0	id: nsd0393e_V104 - sim: 0.035 - CF: 0.0
id: nsd0393e_V37 - sim: 0.515 - CF: 2.0	id: nsd0393e_V105 - sim: 0.035 - CF: 0.0
id: nsd0393e_V38 - sim: 0.338 - CF: 1.0	id: nsd0393e_V106 - sim: 0.035 - CF: 0.0
id: nsd0393e_V98 - sim: 0.322 - CF: 1.0	id: nsd0393e_V107 - sim: 0.035 - CF: 0.0
id: nsd0064e_V13 - sim: 0.036 - CF: 0.0	id: nsd0064e_V238 - sim: 0.03 - CF: 0.0
id: nsd0064e_V217 - sim: 0.035 - CF: 0.0	id: nsd0005e_V32 - sim: 0.03 - CF: 0.0
id: nsd0393e_V99 - sim: 0.035 - CF: 0.0	id: nsd0064e_V220 - sim: 0.029 - CF: 0.0
id: nsd0393e_V100 - sim: 0.035 - CF: 0.0	id: nsd0064e_V221 - sim: 0.029 - CF: 0.0
id: nsd0393e_V101 - sim: 0.035 - CF: 0.0	id: nsd0393e_V39 - sim: 0.029 - CF: 0.0
id: nsd0393e_V102 - sim: 0.035 - CF: 0.0	id: nsd0064e_V26 - sim: 0.026 - CF: 0.0
id: nsd0393e_V103 - sim: 0.035 - CF: 0.0	id: nsd0064e_V25 - sim: 0.025 - CF: 0.0

Table 6.9: Search for variable *nsd0064e_V14*, using vector similarity, inverse user frequency and case amplification factor of 2.5.

6.1.6 Evaluating Search for id *nsd0005e_V83*

Searches using different users as target users. First, a printout of a search using the base statistical variable nsd0005e_V83:

id: nsd0393e_V186 - sim: 0.043	id: nsd0005e_V206 - sim: 0.032
id: nsd0005e_V207 - sim: 0.036	id: nsd0064e_V239 - sim: 0.032
id: nsd0005e_V236 - sim: 0.033	id: nsd0393e_V165 - sim: 0.029
id: nsd0005e_V205 - sim: 0.032	id: nsd0393e_V206 - sim: 0.026

Table 6.10: Search for *variable nsd0005e_V83*, document similarity only.

To see how the similarity relates to the textual content of the variables, see appendix A.3. Since correlation with union selection of items gave "best" results, and since Breese et al. (1998) concluded that this function was the best performing of the memory based weighting functions, it is used as the CF method from here on. All remaining printouts are used with the CF setting of correlation with union item selection method, inverse user frequency turned on, and a case amplification factor of 2.5. It also simplifies the comparison, as only the user varies.

First, table 6.11 gives summations of each user's votes, and how many statistical variables they have voted for. This might help when interpreting the results when using these users. For the following search evaluations, *User D* and *User F* are omitted,

user	count(var_id)	sum(viewCount)
User A	117	203
User B	55	62
User C	10	17
User D	6	8
User E	50	69
User F	8	21
User G	160	437
Sum	406	817

Table 6.11: Summations of votes made by each user

as they have a very low view count.

When comparing the results in the tables 6.12, 6.13, 6.14 and 6.16 with the votes given for the same variables in the StatFind database, it looks as the users' own behaviour has largest control over the predicted values. In most cases, when the CF system calculates a vote, the vote is the same as the view count of that statistical variable for the portrayed user. The exception is for *User E*, where none of the variables returned by the search is viewed by *User E*. This behaviour would most likely be the results of a software bug, where a user is compared with itself, but safeguards against this kind of behaviour is built in the system, as the sourcecode in figure 6.1 shows. The explanation for this behaviour might be that all items used

```
1 //Make sure we don't compare this user with her/himself:
2 if (!otherUser.equals(activeUser)) {
3    currentWeight =
4       weightFunction.doWeight(activeUser, otherUser) *
5       (matrix.getVote(otherUser, activeItem).getVote() -
6       matrix.meanUserVote(otherUser) );
7    absWeights += Math.abs(currentWeight);
8    sumWeights += currentWeight;
9 }
```

Figure 6.1: Safeguard in weightfunction for comparing two idenitcal users

by either the active or the compared user are used when calculating the predicted vote, and in a setting where few votes exist this happens as a special case. E.g. the statistical variable with id *nsd0005e_V206* has a viewcount of 4 for *User A*, while *User*

B and *User C* have also viewed the variable, but only one time each.

User E, on the other hand have not viewed any of the variables in the result list in table 6.15, while the system is still able to make some predictions, meaning the CF system is working to some extent.

id: nsd0005e_V206 - sim: 0.758 - CF: 4.0	id: nsd0393e_V186 - sim: 0.043 - CF: 0.0
id: nsd0005e_V207 - sim: 0.518 - CF: 2.0	id: nsd0005e_V236 - sim: 0.033 - CF: 0.0
id: nsd0005e_V205 - sim: 0.516 - CF: 2.0	id: nsd0393e_V165 - sim: 0.029 - CF: 0.0
id: nsd0064e_V239 - sim: 0.316 - CF: 1.0	id: nsd0393e_V206 - sim: 0.026 - CF: 0.0

Table 6.12: Search performed using *User A*

id: nsd0005e_V207 - sim: 0.318 - CF: 1.0	id: nsd0005e_V236 - sim: 0.033 - CF: 0.0
id: nsd0005e_V205 - sim: 0.316 - CF: 1.0	id: nsd0064e_V239 - sim: 0.032 - CF: 0.0
id: nsd0005e_V206 - sim: 0.316 - CF: 1.0	id: nsd0393e_V165 - sim: 0.029 - CF: 0.0
id: nsd0393e_V186 - sim: 0.043 - CF: 0.0	id: nsd0393e_V206 - sim: 0.026 - CF: 0.0

Table 6.13: Search performed using *User B*

id: nsd0005e_V207 - sim: 0.518 - CF: 2.0	id: nsd0005e_V236 - sim: 0.033 - CF: 0.0
id: nsd0064e_V239 - sim: 0.516 - CF: 2.0	id: nsd0005e_V205 - sim: 0.032 - CF: 0.0
id: nsd0005e_V206 - sim: 0.316 - CF: 1.0	id: nsd0393e_V165 - sim: 0.029 - CF: 0.0
id: nsd0393e_V186 - sim: 0.043 - CF: 0.0	id: nsd0393e_V206 - sim: 0.026 - CF: 0.0

Table 6.14: Search performed using *User C*

id: nsd0393e_V186 - sim: 0.581 - CF: 2.38	id: nsd0005e_V206 - sim: 0.576 - CF: 2.38
id: nsd0005e_V207 - sim: 0.577 - CF: 2.38	id: nsd0064e_V239 - sim: 0.576 - CF: 2.38
id: nsd0005e_V236 - sim: 0.576 - CF: 2.38	id: nsd0393e_V165 - sim: 0.574 - CF: 2.38
id: nsd0005e_V205 - sim: 0.576 - CF: 2.38	id: nsd0393e_V206 - sim: 0.573 - CF: 2.38

Table 6.15: Search performed using *User E*

id: nsd0005e_V207 - sim: 0.318 - CF: 1.0	id: nsd0005e_V206 - sim: 0.032 - CF: 0.0
id: nsd0005e_V236 - sim: 0.316 - CF: 1.0	id: nsd0064e_V239 - sim: 0.032 - CF: 0.0
id: nsd0393e_V186 - sim: 0.043 - CF: 0.0	id: nsd0393e_V165 - sim: 0.029 - CF: 0.0
id: nsd0005e_V205 - sim: 0.032 - CF: 0.0	id: nsd0393e_V206 - sim: 0.026 - CF: 0.0

Table 6.16: Search performed using *User G*

Preliminary Concluding Remarks for Evaluation

The CF system seems to work, but suffers from periodically lack of data. Cases where the CF system answers with only what the users in question have viewed, can be viewed as situations were the CF system cannot make a valid prediction. On the other hand, it is only natural that the CF system predicts a view count of 4 for a statistical variable the user in question has viewed 4 times. The CF system should calculate the same values for predictions for statistical variables as the view count of the actual variable, anything else would be an error.

6.2 Evaluation Comments

As mentioned, the data gathered for this thesis is not of real life quality, but sufficient to demonstrate how the CF system works. Unfortunately the CF system is unable to make good predictions in some cases.

The way the users' behaviour is recorded also bring forth some ethical concerns, as the component reporting the behaviour takes the form of *spy-ware*. Actually the component cannot be called spy-ware as the users were all aware of this system trait, but nevertheless; drawing conclusions from user behaviour is an ethical gray area. In an internetworked world, the position of privacy is challenged every day, and should

not be treated lightly.

For users to accept this trait, a finished system should have an opt-out possibility, and the user interface should, in some way, inform that information the users that their usage of the system is being reported. It is also important for users to know a little about how the system calculates the vote predictions, in order to be able to use and accept its predictions.

Chapter 7

Conclusion

This thesis has dealt with the creation and development of a component called StatFind. Development has been done according to the theories available within the fields of IR and CF. The implementation of StatFind is well documented, both with regards to the theories used, and the architecture of the system itself.

The goal was to make a proof-of-concept system for searching documents, allowing for other metrics than only document similarity from an information retrieval unit — enabling other similarity traits than only textual similarity. This thesis' development for this purpose is StatFind, a prototype incorporating both traditional IR, as well as a collaborative filtering component, able to make "predictions" based on the recorded behaviour of users of the system.

In this setting StatFind was used for finding related statistical variables, using the Nesstar System as a case. Specifically, the thesis set forth to test if combining IR and CF would be effective in enhancing the search system for Nesstar in a way that makes finding related statistical variables easier.

Combining IR and CF in this way makes a search system that is not dependant on both textual and behavioural similarities between the items involved — as illustrated in figure 7.1. This way of combining IR and CF makes each part independent of the results of the other. In cases when new items are added to the database and no usage similarity can be calculated, the document similarity takes over the similar-

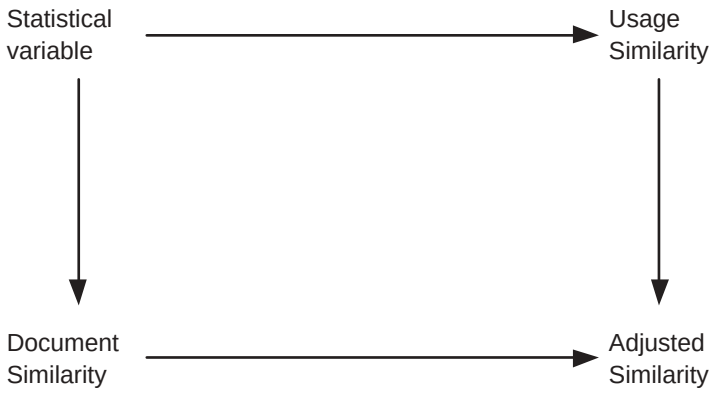

Figure 7.1: Relation between IR and CF

ity calculation, and vice versa. In cases where both exist, an adjusted similarity is calculated.

Ideally, an evaluation of this systems functionality would include running the system in a natural setting, i.e. on a server accessed by the targeted users of the Nesstar System. This way, the collected information about user behaviour would contain fewest possible cases of erroneous data. The data collected for this thesis did not stem from *real* users in a *real-life* usage situation. Nevertheless; the information collected does portray the way the users acted in the system, and can be used to indicate the usefulness of the system.

As the research done in this thesis was developing a proof-of-concept prototype, testing the feasibility of IR and CF combination, the data gathered for analysis does hold to a certain degree, as it shows the effects. However it does not give us enough information to measure the effects of the approach in a real-life setting.

7.1 Further Work

As expected in development research of this kind, there are many loose ends since this thesis might also be viewed as a starting point for later research. Chapter 3.4 discussed the methodological approach and how the figure 3.2 describes how this thesis' development work fits in the hub of the circle. In addition, this thesis has

experimented with developed system, and reached a conclusion for that work.

The meaning of the multimethodological approach is that different research methods are put together in the study of systems development. Where each method gives its results as input to the other methods, as well using the results for improving the developed system, in a way that resemblance the incremental and iterative development method used in software engineering.

This thesis started doing the experimentation work needed to evaluate and tested that the system works, as well as establishing the necessity for improved data quality for evaluation purposes. More data collection is needed in order to do a thorough evaluation of the system, by using the StatFind enabled system in a real-life setting. For the StatFind system to work on real-life data, the system also has to be properly integrated in the Nesstar Server. Data collected using real-life users are also needed for the calibration of the equilibrium calculation given in section 4.4 — for transposing user votes into the range of [0,1]. Setting this value is a question of empirical knowledge about the usage of Nesstar, as it relies on how users behave.

By using the collected data to enable StatFind and Nesstar to give improved feedback for searches, user acceptance of the system might be tested by observational methods and user interviews. Observing how the system is used, new knowledge of how the system should be developed or changed may be discovered. Combination of this knowledge and the results of user interviews can in turn be used for the further development of StatFind directly, or by the way of building new theories about how e.g. the CF or IR framework should operate, in turn influencing how StatFind is constructed.

Doing this development and research in an iterative fashion will gradually adding to StatFind what users find missing or unsatisfactory, as well as changes initiated from the research results, will eventually lead to a system mature for the ultimate test. Testing if users of the system finds the added functionality desirable.

Bibliography

Aamodt, A. (2001). Modeling the knowledge contents of cbr systems. In *ICCBR*, Lecture Notes in Computer Science, pages 32–37, Heidelberg, Germany. Springer-Verlag.

Aamodt, A. and Plaza, E. (1994). Case-based reasoning : Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–51.

Althoff, K.-D. and Aamodt, A. (1996). Relating case-based problem solving and learning methods to task and domain characteristics: Towards an analytic ramework. *AI Communications*, 9(3):109–116.

Baeza-Yates, R. and Ribeiro-Neto, B. (1999a). Modeling. In *Modern Information Retrieval*, chapter 2. ACM Press, Addison-Wesley, 1515 Broadway, 17th Floor, New York, NY 10036-5701, first edition.

Baeza-Yates, R. and Ribeiro-Neto, B. (1999b). *Modern Information Retrieval*. ACM Press, Addison-Wesley, 1515 Broadway, 17th Floor, New York, NY 10036-5701, first edition.

Baeza-Yates, R. and Ribeiro-Neto, B. (1999c). Retrieval evaluation. In *Modern Information Retrieval*, chapter 3. ACM Press, Addison-Wesley, 1515 Broadway, 17th Floor, New York, NY 10036-5701, first edition.

Baeza-Yates, R. and Ribeiro-Neto, B. (1999d). Text and multimedia languages and

properties. In *Modern Information Retrieval*, chapter 6. ACM Press, Addison-Wesley, 1515 Broadway, 17th Floor, New York, NY 10036-5701, first edition.

Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence*, pages 43–52.

Burke, R. D. (2000). A case-based reasoning approach to collaborative filtering. In *EWCBR*, Lecture Notes in Computer Science, pages 370–379, Heidelberg, Germany. Springer-Verlag.

Carrick, C., Yang, Q., Abi-Zeid, I., and Lamontagne, L. (1999). Activating CBR systems through autonomous information gathering. In *Case-Based Reasoning Research and Development: Third International Conference on Case-Based Reasoning, ICCBR-99, Seeon Monastery, Germany, July 1999. Proceedings*, volume Volume 1650 / 1999, pages 74–88, Heidelberg, Germany. Springer-Verlag.

CESSDA (2002). Cessda homepage. http://www.cessda.org (25/5-2003).

CollabNet, Inc (2003). Concurrent versioning system — the open standard for version control. http://www.cvshome.org (25/9-2003).

CORDIS (1999). Cordis - telematics applications:home. http://www.cordis.lu/telematics/home.html (25/5-2003). An Applied Research Programme of the European Commission.

DDA (2003). Dansk data arkiv. http://www.dda.dk (25/5-2003).

Faloutsos, C. and Oard, D. W. (1995). A survey of information retrieval and filtering methods. Technical Report CS-TR-3514, University of Maryland, College Park, MD 20742.

Faster (2000). Faster homepage. http://www.faster-data.org (25/5-2003). Flexible Access to Statistics, Tables and Electronic Resources.

ICPSR (2002). Data documentation initiative. http://www.icpsr.umich.edu/DDI (20/2-2003).

Jakarta/Lucene (2003). Jakarta lucene - overview - jakarta lucene. http://jakarta.apache.org/lucene (10/7-2003).

JetBrains, Inc (2003). Jetbrains intellij idea — the best java ide around. http://intellij.com (25/9-2003).

Mizzaro, S. (1997). Relevance: The whole history. *Journal of the American Society of Information Science*, 48(9):810–832.

Navarro, G. (1999). Indexing and searching. In *Modern Information Retrieval*, chapter 8. ACM Press, Addison-Wesley, 1515 Broadway, 17th Floor, New York, NY 10036-5701, first edition.

Nesstar (2000). Networked social science tools and resources - final report. Technical Report IE8028, Telematics Application Programme - Information Engeneering, European Commission, DG XIII, C/1, TELEMATICS APPLICATIONS Programme Information Desk, Rue de la Loi 200 - BU 29 - 04/05, B-1049 Brussels.

Nesstar (2002a). Nesstar homepage. http://www.nesstar.org (25/5-2003). NEtworked Social Science Tools And Resources.

Nesstar (2002b). Nesstar limited. http://www.nesstar.com (25/5-2003).

NSD (1997). About nsd. http://www.nsd.uib.no/english/about.html (25/5-2003).

Nunamaker, J. F., Chen, M., and M.Purdin, T. D. (1990). Systems development in information systems research. *Journal Of Management Information Systems / Winter*, 7(3):89–106.

Riordan, C. O. and Sorensen, H. (1998). Information filtering and retrieval: An overview.

Rivest, R. L. (1992). The md5 message-digest algorithm. In *Request for Comments*. The Internet Engineering Task Force, Massachusetts Institute of Technology, Laboratory for Computer Science, NE43-324, 545 Technology Square, Cambridge, MA, 2139-1986.

Salton, G. and McGill, M. J. (1983). *Introduction to Modernt Information Retrieval*. McGraw-Hill Book Company, New York, 1. edition.

Schafer, J. B., Konstan, J. A., and Riedi, J. (1999). Recommender systems in e-commerce. In *ACM Conference on Electronic Commerce*, pages 158–166.

Simon, H. A. (1995). Artificial intelligence: an empirical science. *Artificial Intelligence*, 77(1):95–127. TY - JOUR.

SSB (2001). Statistics norway: Statistics act. http://www.ssb.no/english/about_ssb/statlaw/statlov_en.html (25/5-2003).

SSB (2002). About statistics norway. http://www.ssb.no/english/about_ssb (25/5-2003).

Sun Microsystems Inc (2003). Java remote method invocation (rmi). http://java.sun.com/products/jdk/rmi/ (22/8-2003).

The Apache Software Foundation (2003a). Apache ant. http://ant.apache.org (25/9-2003).

The Apache Software Foundation (2003b). Jakarata apache, log4j project. http://jakarta.apache.org/log4j (25/9-2003).

UKDA (2002). Uk data archive. http://www.data-archive.ac.uk (25/5-2003).

Ziviani, N. (1999). Text operations. In *Modern Information Retrieval*, chapter 7. ACM Press, Addison-Wesley, 1515 Broadway, 17th Floor, New York, NY 10036-5701, first edition.

Appendix A

Extended printout of search results

A.1 Search for "politics"

Showing a prinout of the question texts belonging to each statistical variable returned by the search "politics".

ID: nsd0005e_V33 SIM: 1. 0

Q. 3. What concern you most: Norwegian foreign policy, local (municipal) politics, or national (domestic) politics?

ID: nsd0064e_V14 SIM: 1. 0

Q. 5. What concerns you most: Norwegian foreign policy, national (domestic) politics, or local (municipal) politics?

ID: nsd0393e_V98 SIM: 0. 612

Q. 32 When people are asked to express an opinion, do you believe most people in Norway usually say what they think about politics, or do you believe most people usually hide what the really think about politics? Using the scale on this card, (where one means that most people in Norway usually say what they really think about politics, and five means that most people usually hide what they think), where would you place yourself?

ID: nsd0064e_V13 SIM: 0. 53

Q. 4. Let us now look at some political issues. Generally speaking, would you say that you were very interested in politics, fairly interested, only slightly interested or not interested at all?

ID: nsd0064e_V217 SIM: 0. 53

Q. 76. We now come to more popular opinions concerning various political issues. Do you react to the following with complete agreement, qualified agreement, qualified disagreement, or complete disagreement?What happens in politics is rearly of any importance to me.

ID: nsd0393e_V41 SIM: 0. 53

¡preQTxt¿Then to some questions about knowledge of parties and politics. ¡/preQTxt¿¡qstnLit¿Q. 15E How many representatives are elected at the Storting? ALTERNATIVE ANSWERS SHOULD NOT BE MENTIONED, AND NO HELP SHOULD BE GIVEN TO R. ¡/qstnLit¿

ID: nsd0393e_V99 SIM: 0. 53

Q. 33A In politics people sometimes talk of left and right. Where would you place yourself on a scale from 0 to 10 where 0 means the left and 10 means the right?PERSONAL INTERVIEW: SHOW CARD 15

ID: nsd0393e_V100 SIM: 0. 53

¡preQTxt¿In politics people sometimes talk of left and right. Using the same scale, where would you place our political parties?¡/preQTxt¿¡qstnLit¿Centre Party¡/qstnLit¿

ID: nsd0393e_V101 SIM: 0. 53

¡preQTxt¿In politics people sometimes talk of left and right. Using the same scale, where would you place our political parties?¡/preQTxt¿¡qstnLit¿Progress Party¡/qstnLit¿

ID: nsd0393e_V102 SIM: 0. 53

¡preQTxt¿In politics people sometimes talk of left and right. Using the same scale, where would you place our political parties?¡/preQTxt¿¡qstnLit¿Liberal Party¡/qstnLit¿

ID: nsd0393e_V103 SIM: 0. 53

¡preQTxt¿In politics people sometimes talk of left and right. Using the same scale, where would you place our political parties?¡/preQTxt¿¡qstnLit¿Labour Party¡/qstnLit¿

ID: nsd0393e_V104 SIM: 0. 53

¡preQTxt¿In politics people sometimes talk of left and right. Using the same scale, where would you place our political parties?¡/preQTxt¿¡qstnLit¿Conservative Party¡/qstnLit¿

ID: nsd0393e_V105 SIM: 0. 53

¡preQTxt¿In politics people sometimes talk of left and right. Using the same scale, where would you place our political parties?¡/preQTxt¿¡qstnLit¿Christian Peoples Party¡/qstnLit¿

ID: nsd0393e_V106 SIM: 0. 53

¡preQTxt¿In politics people sometimes talk of left and right. Using the same scale, where would you place our political parties?¡/preQTxt¿¡qstnLit¿Socialist Left Party¡/qstnLit¿

ID: nsd0393e_V107 SIM: 0. 53

¡preQTxt¿In politics people sometimes talk of left and right. Using the same scale, where would you place our political parties?¡/preQTxt¿¡qstnLit¿Red Electorial Alliance¡/qstnLit¿

ID: nsd0005e_V32 SIM: 0. 442

Q. 4. Let us now consider some political issues. Generally speaking, would you say that you were:1 Very interested in politics?2 Fairly interested?3 Only slightly interested?4 Not interested at all?5 Don't know

ID: nsd0064e_V220 SIM: 0. 442

Q. 76. We now come to more popular opinions concerning various political issues. Do you react to the following with complete agreement, qualified agreement, qualified disagreement, or complete disagreement?Politics is often so complicated that ordinary people can't follow what it's all about.

ID: nsd0064e_V221 SIM: 0. 442

Q. 76. We now come to more popular opinions concerning various political issues. Do you reactto the following with complete agreement, qualified agreement, qualified disagreement, or complete disagreement?People like me can vote all right but there's nothing else we can do to influence politics.

ID: nsd0064e_V238 SIM: 0. 442

Q. 89. It is of course the case that some people are always interested in politics, whether or not there is an impending election; while others are less interested. How regularly would you say you follow what is going on in current affairs: would you say it was alle the time, some of the time, or only now and then?

ID: nsd0393e_V37 SIM: 0. 442

¡preQTxt¿Then to some questions about knowledge of parties and politics. ¡/preQTxt¿¡qstnLit¿Q. 15A Do you happen to know who is party leader of the Christian People's party? ALTERNATIVE ANSWERS SHOULD NOT BE MENTIONED, AND NO HELP SHOULD BE GIVEN TO R. ¡/qstnLit¿

ID: nsd0393e_V38 SIM: 0. 442

¡preQTxt¿Then to some questions about knowledge of parties and politics. ¡/preQTxt¿¡qstnLit¿Q. 15B Do you remember who's been the Minister of Local Government and Labour the year before the election? ALTERNATIVE

ANSWERS SHOULD NOT BE MENTIONED, AND NO HELP SHOULD BE GIVEN TO R.¡/qstnLit¿

ID: nsd0393e_V39 SIM: 0. 442

¡preQTxt¿Then to some questions about knowledge of parties and politics. ¡/preQTxt¿¡qstnLit¿Q. 15C Can you please tell me which of the following parties has not been represented at the Storting during the last election period? ALTERNATIVE ANSWERS SHOULD NOT BE MENTIONED, AND NO HELP SHOULD BE GIVEN TO R.¡/qstnLit¿

ID: nsd0393e_V40 SIM: 0. 442

¡preQTxt¿Then to some questions about knowledge of parties and politics. ¡/preQTxt¿¡qstnLit¿Q. 15D To which party does the President of the Storting during the last four years belong? ALTERNATIVE ANSWERS SHOULD NOT BE MENTIONED, AND NO HELP SHOULD BE GIVEN TO R. ¡/qstnLit¿

ID: nsd0005e_V44 SIM: 0. 354

Q. 12. Now we come to a number of opinions which people can sometimes be heard to express. For each of the following statements that i read out, can you tell me whether you react with complete agreement, qualified agreement, qualified disagreement, or complete disagreement?A. People like me can vote alright, but we can’t do anything else to influence politics.

ID: nsd0005e_V50 SIM: 0. 354

Q. 12. Now we come to a number of opinions which people can sometimes be heard to express. For each of the following statements that i read out, can you tell me whether you react with complete agreement, qualified agreement, qualified disagreement, or complete disagreement?G. Things that happen in politics rarely have any great importance for me

ID: nsd0064e_V77 SIM: 0. 354

Q. 52. A number of opposing attitudes or conflicts of viewpoint are constantly making themselves evident in politics. We would like to know how you regard some of these conflicts of viewpoint. FOR PERSONAL INETRVIEWS; READ CARD 3: Can you tell me which of these conflicts of viewpoint you personally belive to be the most important.FOR TELEPHONE INTERVIEW: Can you tell me which of the following conflicts of viewpoint you personally belive to be the most important. READ FROM CARD 3.

ID: nsd0393e_V30 SIM: 0. 354

¡preQTxt¿Q. 14 Let us again look at some statements of opinion. We still use the same alternative answers as earlier. For each of the statements that I read, can you tell me whether you react with complete agreement, qualified agreement, qualified disagreement, or complete disagreement?PERSONAL INTERVIEW: SHOW CARD 2 ¡/preQTxt¿¡qstnLit¿People like me can vote, but we can’t do anything else to influence politics. Complete agreement, qualified agreement, qualified disagreement or complete disagreement?¡/qstnLit¿

ID: nsd0393e_V31 SIM: 0. 354

¡preQTxt¿Q. 14 Let us again look at some statements of opinion. We still use the same alternative answers as earlier. For each of the statements that I read, can you tell me whether you react with complete agreement, qualified agreement, qualified disagreement, or complete disagreement?PERSONAL INTERVIEW: SHOW CARD 2 ¡/preQTxt¿¡qstnLit¿Politics is often so complicated that ordinary citizens cannot understand what is going on ¡/qstnLit¿

A.2 Search for id *nsd0064e_V14*

Showing a prinout of the question texts belonging to each statistical variable returned by the search for statistical variable with id *nsd0064e_V14*.

ID: nsd0005e_V33 SIM: 0. 631 Q. 3. What concern you most: Norwegian foreign policy, local (municipal) politics, or national (domestic) politics?

ID: nsd0393e_V38 SIM: 0. 064 ¡preQTxt¿Then to some questions about knowledge of parties and politics. ¡/preQTxt¿¡qstnLit¿Q. 15B Do you remember who's been the Minister of Local Government and Labour the year before the election? ALTERNATIVE ANSWERS SHOULD NOT BE MENTIONED, AND NO HELP SHOULD BE GIVEN TO R.¡/qstnLit¿

ID: nsd0393e_V98 SIM: 0. 041 Q. 32 When people are asked to express an opinion, do you believe most people in Norway usually say what they think about politics, or do you believe most people usually hide what the really think about politics? Using the scale on this card, (where one means that most people in Norway usually say what they really think about politics, and five means that most people usually hide what they think), where would you place yourself?

ID: nsd0064e_V13 SIM: 0. 036 Q. 4. Let us now look at some political issues. Generally speaking, would you say that you were very interested in politics, fairly interested, only slightly interested or not interested at all?

ID: nsd0064e_V217 SIM: 0. 035 Q. 76. We now come to more popular opinions concerning various political issues. Do you react to the following with complete agreement, qualified agreement, qualified disagreement, or complete disagreement?What happens in politics is rearly of any importance to me.

ID: nsd0393e_V99 SIM: 0. 035 Q. 33A In politics people sometimes talk of left and right. Where would you place yourself on a scale from 0 to 10 where 0 means the left and 10 means the right?PERSONAL INTERVIEW: SHOW CARD 15

ID: nsd0393e_V100 SIM: 0. 035 ¡preQTxt¿In politics people sometimes talk of left and right. Using the same scale, where would you place our political parties?¡/preQTxt¿¡qstnLit¿Centre Party¡/qstnLit¿

ID: nsd0393e_V101 SIM: 0. 035 ¡preQTxt¿In politics people sometimes talk of left and right. Using the same scale, where would you place our political parties?¡/preQTxt¿¡qstnLit¿Progress Party¡/qstnLit¿

ID: nsd0393e_V102 SIM: 0. 035 ¡preQTxt¿In politics people sometimes talk of left and right. Using the same scale, where would you place our political parties?¡/preQTxt¿¡qstnLit¿Liberal Party¡/qstnLit¿

ID: nsd0393e_V103 SIM: 0. 035 ¡preQTxt¿In politics people sometimes talk of left and right. Using the same scale, where would you place our political parties?¡/preQTxt¿¡qstnLit¿Labour Party¡/qstnLit¿

ID: nsd0393e_V104 SIM: 0. 035 ¡preQTxt¿In politics people sometimes talk of left and right. Using the same scale, where would you place our political parties?¡/preQTxt¿¡qstnLit¿Conservative Party¡/qstnLit¿

ID: nsd0393e_V105 SIM: 0. 035 ¡preQTxt¿In politics people sometimes talk of left and right. Using the same scale, where would you place our political parties?¡/preQTxt¿¡qstnLit¿Christian Peoples Party¡/qstnLit¿

ID: nsd0393e_V106 SIM: 0. 035 ¡preQTxt¿In politics people sometimes talk of left and right. Using the same scale, where would you place our political parties?¡/preQTxt¿¡qstnLit¿Socialist Left Party¡/qstnLit¿

ID: nsd0393e_V107 SIM: 0. 035 ¡preQTxt¿In politics people sometimes talk of left and right. Using the same scale, where would you place our political parties?¡/preQTxt¿¡qstnLit¿Red Electorial Alliance¡/qstnLit¿

ID: nsd0064e_V238 SIM: 0. 03 Q. 89. It is of course the case that some people are always interested in politics, whether or not there is an impending election; while others are less interested. How regularly would you say you follow what is going on in current affairs: would you say it was alle the time, some of the time, or only now and then?

ID: nsd0005e_V32 SIM: 0. 03 Q. 4. Let us now consider some political issues. Generally speaking, would you say that you were:1 Very interested in politics?2 Fairly interested?3 Only slightly interested?4 Not interested at all?5

Don't know

ID: nsd0064e_V220 SIM: 0. 029 Q. 76. We now come to more popular opinions concerning various political issues. Do you react to the following with complete agreement, qualified agreement, qualified disagreement, or complete disagreement?Politics is often so complicated that ordinary people can't follow what it's all about.

ID: nsd0064e_V221 SIM: 0. 029 Q. 76. We now come to more popular opinions concerning various political issues. Do you reactto the following with complete agreement, qualified agreement, qualified disagreement, or complete disagreement?People like me can vote all right but there's nothing else we can do to influence politics.

ID: nsd0393e_V37 SIM: 0. 029 ¡preQTxt¿Then to some questions about knowledge of parties and politics. ¡/preQTxt¿¡qstnLit¿Q. 15A Do you happen to know who is party leader of the Christian People's party? ALTERNATIVE ANSWERS SHOULD NOT BE MENTIONED, AND NO HELP SHOULD BE GIVEN TO R. ¡/qstnLit¿

ID: nsd0393e_V39 SIM: 0. 029 ¡preQTxt¿Then to some questions about knowledge of parties and politics. ¡/preQTxt¿¡qstnLit¿Q. 15C Can you please tell me which of the following parties has not been represented at the Storting during the last election period? ALTERNATIVE ANSWERS SHOULD NOT BE MENTIONED, AND NO HELP SHOULD BE GIVEN TO R.¡/qstnLit¿

ID: nsd0064e_V26 SIM: 0. 026 Q. 16. In relation to the Soviet Union, do you think that the Norwegian government during the past three or four years has been too strongly critial, not critical enough, or has been appropiately critical of Soviet foreign policy?

ID: nsd0064e_V25 SIM: 0. 025 Q. 15. The next two questions concern Norwegian government policy towards the USA and the Soviet Union. Taking the USA first, do you think that the government during the past three or four years has been too strongly critial, not critical enough, or has been appropiately critical of US foreign policy?

A.3 Search for id *nsd0005e_V83*

Showing a prinout of the question texts belonging to each statistical variable returned by the search for statistical variable with id *nsd0005e_V83*.

ID: nsd0393e_V186 SIM: 0. 043 Q. 75 Are you currently in paid employment? Paid employment also includes work as a family member without a fixed, regular wage on a farm, in a shop, or in a family business. Include all working activity of one hour or more pr. week.

ID: nsd0005e_V207 SIM: 0. 036 Q. 68. People sometimes talk about the existence of different social classes. Most people would say they belong to one of two classes; either the working class, or the middle class. Do you ever think of yourself as belonging to one of these classes?Which class?If you had to choose, would you say that you belong more to the working class or the middle class?

ID: nsd0005e_V236 SIM: 0. 033 Q. 80. Are you currently in paid employment?Paid employed also includes work as a family member without a fixed, regular wage on a farm, in a shop, or in a family business. Include all working activity of 1 hour or more per week.

ID: nsd0005e_V205 SIM: 0. 032 Q. 66. People sometimes talk about the existence of different social classes. Most people would say they belong to one of two classes; either the working class, or the middle class. Do you ever think of yourself as belonging to one of these classes?

ID: nsd0005e_V206 SIM: 0. 032 Q. 67. People sometimes talk about the existence of different social classes. Most people would say they belong to one of two classes; either the working class, or the middle class. Do you ever think of yourself as belonging to one of these classes?Which class?

ID: nsd0064e_V239 SIM: 0. 032 Q. 90. There is always a great deal of talk concerning the different social classes. Most people would say they belong to one of two classes: either the working class, or the middle class. Do you ever think of yourself as belonging to one of these classes?

ID: nsd0393e_V165 SIM: 0. 029 Q. 56 Let's imagine that we have two people, A and B, who are discussing an issue of current interest. We have formulated two different viewpoints that A and B might be heard to express. PERSONAL INTERVIEW: SHOW CARD 18A says:To avoid rising prices and rising interests, we should not use more of the oil-revenues than we do today.B says:To solve the problems of the public health services and in the care for the elderly, we can use a lot more of the oil-revenues than we do today.Which of these two viewpoints do you agree most with?

ID: nsd0393e_V206 SIM: 0. 026 SEE ANSWERS TO QUESTION 73. ONLY ASK QUESTION 92 TO 96 IF THE RESPONDENT IS MARRIED OR COHABITING. OTHERS GO TO QUESTION 97. Q. 92 Is your spouse/partner currently in paid employment? Paid employment also includes work as a family member without fixed, regular wage on a farm, in a shop, or in a family business. Include all working activities of 1 hour or more per week.

Appendix B

Software documenation

Here the source code is located/available, as well as legend for understanding symbols in UML diagrams (B.1).

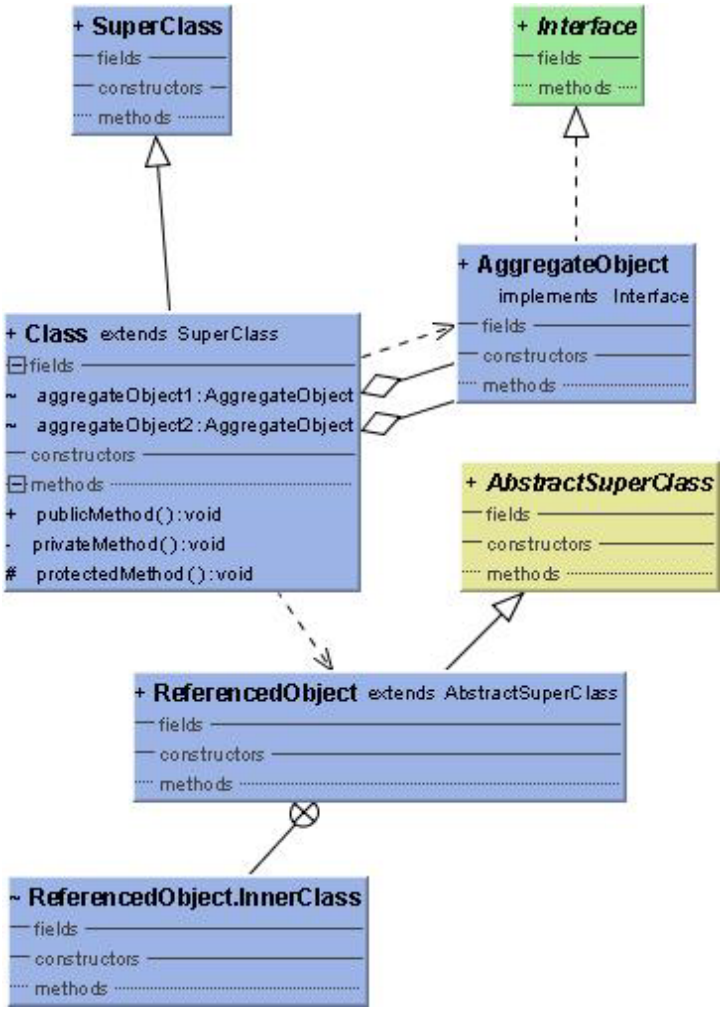

Figure B.1: An example UML Diagram, serving as symbol legend

```
1  /*
2   * User: olvesh
3   * Date: Nov 15, 2002
4   * Time: 3:07:03 PM
5   */
6  package statfind.foreign;
7
8
9  import org.apache.log4j.Level;
10 import org.apache.log4j.Logger;
11 import statfind.common.*;
12 import statfind.common.util.SleepyQueue;
13
14 import java.io.BufferedOutputStream;
15 import java.io.IOException;
16 import java.io.ObjectOutputStream;
17 import java.net.InetSocketAddress;
18 import java.net.Socket;
19 import java.util.Properties;
20
21 /**
22  * Class for serving the StatFind server with usage information,
23  * using a <code>SleepyQueue</code> to make it more robust.
24  */
25
26 public class StatfindFeeder extends SleepyQueue {
27     private static StatfindFeeder ourInstance;
28     private static Logger log = Logger.getLogger(StatfindFeeder.class);
29     private Properties props;
30     private Socket socket;
31     private ObjectOutputStream outStream;
32     private boolean isConnected = false;
33     private Thread shutdownThread;
34     private static boolean hasSaidSo = false;
35
36
37     /**
38      * This class is a Singleton, making an application connect only once to the server.
39      * @return
40      */
```

```
41  public synchronized static StatfindFeeder getInstance() {
42      if (ourInstance == null) {
43          log.setLevel(Level.INFO);
44          ourInstance = new StatfindFeeder();
45          ourInstance.start();
46      }
47      return ourInstance;
48  }
49
50  /**
51   * Connects using the supplied properties file,
52   */
53  private StatfindFeeder() {
54      if (props == null) {
55          props = new Properties(CommonProperties.getCommonDefaults());
56          try {
57              props.load(getClass().getResourceAsStream("statfind-foreign.properties"));
58          } catch (IOException e) {
59              log.fatal("Could not initialize class", e);
60          }
61      }
62      threadName = "StatfindFeeder";
63      if (UserSelector.user() == null)
64          new UserSelector(props);
65      shutdownThread = new Thread(new Stopper(), "StreamStopper");
66      Runtime.getRuntime().addShutdownHook(shutdownThread);
67  }
68
69  private void initSocketConnection() throws IOException {
70      String serverHost = props.getProperty("host");
71      int serverPort = Integer.parseInt(props.getProperty("server.port"));
72      log.debug("Connecting to host: " + serverHost + " at port: " + serverPort);
73      socket = new Socket();
74      InetSocketAddress address = new InetSocketAddress(serverHost, serverPort);
75      socket.connect(address, 0);//Wait indefinetly
76      log.info("Connected to host: " + serverHost + " at port: " + serverPort);
77      isConnected = true;
78
79  }
80
81
82  private boolean initStream() throws IOException {
83      if (!isConnected) {
84          initSocketConnection();
85      }
```

```
86
87          boolean success = false;
88
89          int maxRetry = Integer.parseInt(props.getProperty("retry"));
90          int tries = 0;
91          IOException ioe = null;
92          while (isConnected && !success && tries < maxRetry) {
93              try {
94                  outStream = new ObjectOutputStream(
95                          new BufferedOutputStream(socket.getOutputStream()));
96                  success = true;
97                  log.debug("Buffered stream initialized");
98              } catch (IOException e) {
99                  ioe = e;
100                 //No attempts to reinitialize, will only retry too
101                 //many times if the user is not connected to the net.
102             }
103         }
104         if (!success) {
105             throw ioe;
106         }
107         return success;
108     }
109
110     public void close() {
111         if (shutdownThread != null) {
112             Runtime.getRuntime().removeShutdownHook(shutdownThread);
113             shutdownThread = null;
114         }
115
116         try {
117             if (outStream != null) {
118                 outStream.flush();
119                 outStream.close();
120             }
121         } catch (IOException e) {
122             if (outStream != null) {
123                 try {
124                     outStream.close();
125                     //If the flush command threw the exception, try once more, without flushing.
126                 } catch (IOException e1) {
127                     log.warn("Problems closing buffered stream; " + e.getMessage());
128                 }
129             }
130         } finally {
```

```
131             outStream = null;
132         }
133         try {
134             socket.close();
135         } catch (IOException e) {
136             log.error("Problems closing socket: " + e.getMessage());
137         } finally {
138             socket = null;
139             isConnected = false;
140         }
141     }
142 
143 
144     public void run() {
145         log.debug("Starting thread");
146         try {
147             initStream();
148         } catch (Throwable e) {
149             if (!hasSaidSo)
150                 log.error("Could not initialize connection: " + e.getMessage());
151         }
152         try {
153             if (socket != null && socket.isConnected()) {
154                 hasSaidSo = false;
155                 super.run();
156             } else {
157                 if (!hasSaidSo)
158                     log.error("Socket not connected, stopping thread");
159                 hasSaidSo = true;
160             }
161         } catch (StatfindRuntimeException e) {
162             log.error("Thread exiting abnormally: " + e.getMessage());
163         }
164 
165         close();
166         synchronized (ourInstance) {
167             ourInstance = null;
168         }
169         log.debug("Thread finished");
170     }
171 
172 
173     private void screenObject(Object o) throws IOException {
174         if (o instanceof EndOfStream) {
175             writeObject(o);
```

```
176         } else if (o instanceof CompletedOperation) {
177             String varResTag = props.getProperty("variableResultPreTag");
178 
179             CompletedOperation op = (CompletedOperation) o;
180             log.debug("Screening object: " + op);
181             if (op.getUser() != null &&
182                     !op.getUser().equalsIgnoreCase("null") &&
183                     op.getVarId() != null &&
184                     !op.getVarId().equalsIgnoreCase("null") &&
185                     -1 != op.getVarId().lastIndexOf(varResTag)) {
186                 log.debug("Writing object to stream:\n" + o);
187 
188                 op.setVarId(op.getVarId().substring(op.getVarId()
189                         .lastIndexOf(varResTag) + varResTag.length()));
190                 writeObject(op);
191             }
192         }
193     }
194 
195 
196     protected void doTheStuff(Object o) {
197         try {
198             screenObject(o);
199         } catch (IOException e) {
200             throw new StatfindRuntimeException(e);
201         }
202     }
203 
204     private void writeObject(Object o) throws IOException {
205         if (o == null)
206             return;
207 
208         outStream.writeObject(o);
209         outStream.flush();
210         if (o instanceof EndOfStream) {
211             threadIsStopping = true;
212             //The feeder is about to be scrapped.
213         }
214     }
215 
216 
217     public synchronized void processObject(CompletedOperation msg) {
218         msg.setUser(UserSelector.user());
219         theQueue.insertLast(msg);
220         wakeUpCall();
```

```
221     }
222
223
224     public void finalize() {
225         close();
226     }
227
228     class Stopper implements Runnable {
229
230         public void run() {
231             shutdownThread = null;
232             processObject(new EndOfStream());
233         }
234     }
235 }
```

B.2 statfind.common.util.SleepyQueue class source code

```
1 /*
2  * Created by IntelliJ IDEA.
3  * User: oh
4  * Date: 13.mai.02
5  * Time: 15:54:59
6  */
7 package statfind.common.util;
8
9
10 import java.util.NoSuchElementException;
11 import java.util.logging.Logger;
12
13 public abstract class SleepyQueue implements Runnable {
14
15     protected Queue theQueue;
16     protected static boolean threadIsRunning = false;
17     protected static boolean threadIsStopping = false;
18     private boolean threadSuspended = true;
19     protected boolean errorsOccured = false;
20     private boolean isCancelled = false;
21     protected String threadName;
22     private Logger log;
```

```
23
24      public SleepyQueue() {
25          super();
26          theQueue = new VectorQueue();
27          log = Logger.getLogger(this.getClass().toString());
28      }
29
30      protected synchronized void wakeUpCall() {
31          if (threadSuspended) {
32              threadSuspended = false;
33              log.fine("waking up");
34              notify();
35          }
36      }
37
38
39      /**
40       * This method is called by the thread running this class, to start the execution of this class
41       * in the calling thread.
42       */
43
44      public void run() {
45
46          while (!threadIsStopping) {
47              try {
48                  //Check if there is more to do, if not, go into sleep.
49                  if (threadSuspended) {
50                      synchronized (this) {
51                          while (threadSuspended) {
52                              log.fine("going to sleep");
53                              wait();
54                          }
55                          initWakeup();
56                      }
57                  }
58                  //Do the parse stuff:
59                  doTheStuff(theQueue.removeFirst());
60                  if (isCancelled) {
61                      log.fine("cancelling remaining items");
62                      theQueue.removeAll();
63                      isCancelled = false;
64                  }
65
66                  if (theQueue.isEmpty()) {
67                      threadSuspended = true;
```

```
68                      cleanUpBeforeSleep();
69                  }
70
71              } catch (NoSuchElementException nsee) {
72                  log.warning(nsee.getMessage());
73                  log.warning("cause:" + nsee.getCause());
74                  //Means that the structure is empty
75                  threadSuspended = true;
76                  cleanUpBeforeSleep();
77              } catch (InterruptedException ie) {
78                  log.warning(ie.getMessage());
79                  log.warning("cause:" + ie.getCause());
80
81              }
82          }//End of while looooooop.
83      }
84
85      public void setCancelled(boolean cancelled) {
86          isCancelled = cancelled;
87      }
88
89      protected void initWakeup() {
90          //So that the vm cant exit when a thread is running.
91  //          Thread.currentThread().setDaemon(false);
92      }
93
94      protected void cleanUpBeforeSleep() {
95          //So that a thread going to sleep won't stop the vm from exiting.
96  //          Thread.currentThread().setDaemon(true);
97          //Nothing here yet...
98      }
99
100     protected abstract void doTheStuff(Object o);
101
102
103 //    public boolean threadIsStopping() {
104 //        return threadIsStopping;
105 //    }
106
107     public boolean threadIsRunning() {
108         return threadIsRunning;
109     }
110
111     /**
112      * Called to start the execution, this method will initiate a new <code>Thread</code> and
```

```
113      * assign this class to it, and then start the <code>Thread</code>, which in turn calls
114      * {@link statfind.common.util.SleepyQueue#run() run()}.
115      *
116      */
117     public void start() {
118
119         Thread t = new Thread(this, threadName);
120         t.setPriority(Thread.MIN_PRIORITY);
121         t.setPriority(Thread.MIN_PRIORITY);
122
123         t.setDaemon(true);
124         t.start();
125     }
126
127 //    public void stopThread() {
128 //        if (threadIsRunning) {
129 //            threadIsStopping = true;
130 //        }
131 //    }
132 }
```

B.3 statfind.common.CompletedOperation class source code

```
1 /*
2 * User: olvesh
3 * Date: Nov 21, 2002
4 * Time: 8:10:40 PM
5 */
6 package statfind.common;
7
8 import java.io.Serializable;
9 import java.net.URL;
10
11
12 public class CompletedOperation implements Serializable {
13
14     private String user;
15 //    protected String  method;
16     private String varId/*, hostAdr*/;
17
```

```
18
19      public String getUser() {
20          return user;
21      }
22
23
24      public String getVarId() {
25          return varId;
26      }
27
28
29      public void setUser(String name) {
30          this.user = name;
31      }
32
33  //    public void setMethod(String methodName) {
34  //        method = methodName;
35  //    }
36
37      public void setVarId(String varId) {
38          this.varId = varId;
39      }
40
41      public void setVarId(URL varUrl) {
42  //        hostAdr = theResult.getHost();
43          varId = varUrl.getPath();
44      }
45
46      public String toString() {
47          StringBuffer sb = new StringBuffer();
48          sb.append("user: ");
49          sb.append(user);
50          sb.append(" varId: ");
51          sb.append(varId);
52          return sb.toString();
53      }
54  }
```

Collaborative Filtering source code: The first class are a general function class, while the two next classes are for the different functions used in the collaborative filtering approach, vector similarity and correlation. The next class is the CollaborativeFilter, which uses a weight function. The rest of the classes are for handling the

collection of users and items.

B.4 statfind.common.cf.functions.WeightFunction class source code

```
1 /*
2  * Created by IntelliJ IDEA.
3  * User: olvesh
4  * Date: 07.sep.02
5  * Time: 18:45:17
6  */
7 package statfind.common.cf.functions;
8
9 import org.apache.log4j.Logger;
10 import statfind.common.cf.User;
11
12 import java.sql.SQLException;
13
14 public abstract class WeightFunction {
15
16     protected boolean inverseUserFreq, caseAmp;
17     protected double ampFactor = 2.5d;
18     protected static Logger log = Logger.getLogger(WeightFunction.class);
19     public static final double NO_WEIGHT = 0d;
20     public static double DEFAULT_VOTE = 0d;
21
22     public double doWeight(final User activeUser, final User otherUser) throws SQLException {
23         double weight = weight(activeUser, otherUser);
24         return Double.isNaN(weight) ?
25                 NO_WEIGHT : (caseAmp ? amplify(weight) : weight);
26     }
27
28     protected abstract double weight(final User activeUser, final User otherUser)
29             throws SQLException;
30
31     public boolean getInverseUserFreq() {
32         return inverseUserFreq;
33     }
34
35     public void setInverseUserFreq(boolean b) {
36         inverseUserFreq = b;
37     }
38
```

```
39      public boolean usesCaseAmp() {
40          return caseAmp;
41      }
42
43      public void useCaseAmp(boolean b) {
44          caseAmp = b;
45      }
46
47      public double getCaseAmpFactor() {
48          return ampFactor;
49      }
50
51      public void setCaseAmpFactor(double d) {
52          ampFactor = d;
53      }
54
55      private double amplify(double weight) {
56          double amped;
57          if (weight >= 0d) {
58              amped = Math.pow(weight, ampFactor);
59          } else {
60              //Do I have to multiply with -1 or is it sufficient to say "- 'the exression' "
61              amped = -Math.pow(Math.abs(weight), ampFactor);
62          }
63          return amped;
64      }
65
66      public String toString() {
67          String val = caseAmp ? " Case amp, factor: " + ampFactor + ", " : ", ";
68          val += "Inverse User Frequency: " + getInverseUserFreq();
69          return val;
70      }
71  }
```

B.5 statfind.common.cf.functions.VectorSimilarity class source code

```
1  /*
2  * User: olvesh
3  * Date: 07.sep.02
4  * Time: 19:48:56
5  */
```

```
6  package statfind.common.cf.functions;
7
8
9  import statfind.common.cf.Item;
10 import statfind.common.cf.ItemCollection;
11 import statfind.common.cf.User;
12 import statfind.common.cf.UserItemMatrix;
13
14 import java.sql.SQLException;
15
16 public class VectorSimilarity extends WeightFunction {
17
18     public VectorSimilarity() {
19         this(true); //Inverse user frequency is turned on by default.
20     }
21
22     public VectorSimilarity(boolean inverseUserFreq) {
23         this(inverseUserFreq, true);
24
25     }
26
27     public VectorSimilarity(boolean inverseUserFreq, boolean caseAmp) {
28         setInverseUserFreq(inverseUserFreq);
29         useCaseAmp(caseAmp);
30     }
31
32
33     public double weight(final User activeUser, final User otherUser) throws SQLException {
34         UserItemMatrix matrix = UserItemMatrix.getInstance();
35
36         ItemCollection userItemIntersect = matrix.userItemIntersect(activeUser, otherUser);
37         ItemCollection activeUserItems = matrix.itemsVotedForByUser(activeUser);
38         ItemCollection otherUserItems = matrix.itemsVotedForByUser(otherUser);
39
40         double sum = 0f, nom_1 = 0f, nom_2 = 0f, denom_1 = 0f, denom_2 = 0f;
41
42         //Do both iterators in same loop to minimize loop lengths.
43         //Very insecure if this will work, since I don't know which sets to use for
44         //activeUserItems, and otherUserItems
45         //See article Empirical Analysis of Predictive Algortihms for Collaborative Filtering by
46         //Breese et al.
47         while (activeUserItems.hasNext() || otherUserItems.hasNext()) {
48             if (activeUserItems.hasNext()) {
49                 Item actItem = (Item) activeUserItems.next();
50                 denom_1 += Math.pow(matrix.getVote(activeUser, actItem).getVote(), 2);
```

```
51              }
52
53              if (otherUserItems.hasNext()) {
54                  Item othItem = (Item) otherUserItems.next();
55                  denom_2 += Math.pow(matrix.getVote(otherUser, othItem).getVote(), 2);
56              }
57
58          }
59          denom_1 = Math.sqrt(denom_1);
60          denom_2 = Math.sqrt(denom_2);
61
62          while (userItemIntersect.hasNext()) {
63              Item item = (Item) userItemIntersect.next();
64
65              nom_1 = matrix.getVote(activeUser, item).getVote();
66              nom_2 = matrix.getVote(otherUser, item).getVote();
67
68              if (inverseUserFreq) {
69                  sum += Math.log(matrix.getUserCount() / matrix.getVoteCountForItem(item)) *
70                          (nom_1 / denom_1) * (nom_2 / denom_2);
71              } else {
72                  sum += (nom_1 / denom_1) * (nom_2 / denom_2);
73              }
74          }
75
76          userItemIntersect.close();
77          activeUserItems.close();
78          otherUserItems.close();
79
80          return sum;
81      }
82
83      public String toString() {
84          return "Vector similarity;" + super.toString();
85      }
86  }
```

B.6 statfind.common.cf.functions.Correlation class source code

```
1  /*
2   * Created by IntelliJ IDEA.
```

```
3  * User: olvesh
4  * Date: 07.sep.02
5  * Time: 19:12:21
6  * To change template for new class use
7  * Code Style | Class Templates options (Tools | IDE Options).
8  */
9 package statfind.common.cf.functions;
10
11 import org.apache.log4j.Logger;
12 import statfind.common.cf.*;
13
14 import java.sql.SQLException;
15
16 public class Correlation extends WeightFunction {
17
18     protected static Logger log = Logger.getLogger(Correlation.class);
19
20
21     /**
22      * The item selection method correspond to selection default voting regime. So setting
23      * the value to UNION, assumes that default votes are used, and that all votes that are
24      * returned are valid.
25      */
26     protected int itemSelectionMethod = INTERSECT;
27     //Inverse user frequency is turned on by default.
28     public static final int INTERSECT = 0;
29     public static final int UNION = 1;
30
31
32     public Correlation() {
33         this(true, true, INTERSECT);
34     }
35
36     public Correlation(boolean caseAmp, boolean inverseUserFreq) {
37         this(caseAmp, inverseUserFreq, INTERSECT);
38     }
39
40     public Correlation(int selMethod) {
41         this(true, true, selMethod);
42     }
43
44     public Correlation(boolean caseAmp, boolean invUserFreq, int selMethod) {
45         setInverseUserFreq(invUserFreq);
46         setItemSelectionMethod(selMethod);
47         useCaseAmp(caseAmp);
```

```
48      }
49
50
51      public int getItemSelectionMethod() {
52          return itemSelectionMethod;
53      }
54
55      public void setItemSelectionMethod(int itemSelectionMethod) throws IllegalArgumentException {
56          this.itemSelectionMethod = itemSelectionMethod;
57
58          if (!(itemSelectionMethod == INTERSECT || itemSelectionMethod == UNION))
59              throw new IllegalArgumentException("Not a valid item selection method: " +
60                      itemSelectionMethod);
61      }
62
63      public double weight(final User activeUser, final User otherUser)
64              throws SQLException {
65          return inverseUserFreq
66                  ? invUserFreqWeight(activeUser, otherUser)
67                  : nonInvUserFreqWeight(activeUser, otherUser);
68      }
69
70
71      protected double invUserFreqWeight(final User activeUser, final User otherUser)
72              throws SQLException {
73          //TODO: Add the Fi value, defined as log(n/nj). See article.
74          UserItemMatrix matrix = UserItemMatrix.getInstance();
75
76          ItemCollection selectedItems = null;
77          if (itemSelectionMethod == INTERSECT)
78              selectedItems = matrix.userItemIntersect(activeUser, otherUser);
79          else if (itemSelectionMethod == UNION)
80              selectedItems = matrix.userItemUnion(activeUser, otherUser);
81
82
83          double /*sum = 0f,*/ nom_1 = 0f, nom_2 = 0f, nom_3 = 0f,
84                  denom_1 = 0f, denom_2 = 0f, denom_3 = 0f, denom_4 = 0f,
85                  nom_A = 0f, denom_A = 0f, denom_B = 0f;
86
87
88          Vote activeUserVote, otherUserVote;
89
90          while (selectedItems.hasNext()) {
91              Item item = (Item) selectedItems.next();
92
```

```
93              double invUserFreq = Math.log(matrix.getUserCount() / matrix.getVoteCountForItem(item));
94
95              activeUserVote = matrix.getVote(activeUser, item);
96              otherUserVote = matrix.getVote(otherUser, item);
97              nom_1 += invUserFreq *
98                      activeUserVote.getVote() *
99                      otherUserVote.getVote();
100             nom_2 += invUserFreq *
101                     activeUserVote.getVote();
102             nom_3 += invUserFreq *
103                     matrix.getVote(otherUser, item).getVote();
104             denom_1 += invUserFreq * Math.pow(otherUserVote.getVote(), 2);
105             denom_2 += invUserFreq * activeUserVote.getVote();
106
107             denom_3 += invUserFreq * Math.pow(otherUserVote.getVote(), 2);
108             denom_4 += invUserFreq * otherUserVote.getVote();
109         }
110         selectedItems.close();
111         selectedItems.reset();
112
113         denom_2 = Math.pow(denom_2, 2);
114         denom_4 = Math.pow(denom_4, 2);
115
116         nom_1 = nom_1 - (nom_2 * nom_3);
117         denom_1 = denom_1 - denom_2;
118         denom_3 = denom_3 - denom_4;
119
120         while (selectedItems.hasNext()) {
121             double invUserFreq = Math.log(
122                     matrix.getUserCount() /
123                     matrix.getVoteCountForItem((Item) selectedItems.next()));
124             nom_A += invUserFreq * nom_1;
125             denom_A += invUserFreq * denom_1;
126             denom_B += invUserFreq * denom_3;
127         }
128
129         return nom_A / Math.sqrt(denom_A * denom_B);
130     }
131
132     protected double nonInvUserFreqWeight(final User activeUser, final User otherUser)
133             throws SQLException {
134         UserItemMatrix matrix = UserItemMatrix.getInstance();
135
136         ItemCollection selectedItems = null;
137         if (itemSelectionMethod == INTERSECT)
```

```
138             selectedItems = matrix.userItemIntersect(activeUser, otherUser);
139         else if (itemSelectionMethod == UNION)
140             selectedItems = matrix.userItemUnion(activeUser, otherUser);
141 
142         double activeMean = matrix.meanUserVote(activeUser),
143                 otherMean = matrix.meanUserVote(otherUser),
144                 nom = 0d, denom_1 = 0d, denom_2 = 0d, valA = 0d, valB = 0d;
145 
146         while (selectedItems.hasNext()) {
147             Item item = (Item) selectedItems.next();
148             valA = matrix.getVote(activeUser, item).getVote() - activeMean;
149             valB = matrix.getVote(otherUser, item).getVote() - otherMean;
150 
151             nom += valA * valB;
152             denom_1 += valA;
153             denom_2 += valB;
154         }
155         selectedItems.close();
156         double res = nom / Math.sqrt(Math.pow(denom_1, 2) * Math.pow(denom_2, 2));
157         log.debug("Res: " + res);
158         //If no items in intersection or union, the weight is set to null:
159         return res;
160     }
161 
162     public String toString() {
163         String itemSelMeth = itemSelectionMethod == INTERSECT ? "INTERSECTION" : "UNION";
164         String wfString = super.toString();
165 
166         return "Correlation; item selection method: " + itemSelMeth + "," + wfString;
167     }
168 
169 }
```

B.7 statfind.common.cf.CollaborativeFilter class source code

```
1 /*
2  * User: olvesh
3  * Date: 07.sep.02
4  * Time: 18:13:20
5  */
6 package statfind.common.cf;
```

```
7
8  import org.apache.log4j.Logger;
9  import statfind.common.cf.functions.WeightFunction;
10
11 import java.sql.SQLException;
12
13
14 /**
15  * Class with implementations of CollaborativeFiltering techniques.
16  * This is the main class for performing vote predictions. The function used
17  * is pluggable, according to the interface <code>WeightFunction</code>.
18  */
19 public class CollaborativeFilter {
20     protected static Logger log = Logger.getLogger(CollaborativeFilter.class);
21
22     protected UserItemMatrix matrix;
23     protected WeightFunction weightFunction;
24
25     /**
26      * The constructor, makes a CollaborativeFilter
27      */
28     public CollaborativeFilter() throws SQLException {
29         this.matrix = UserItemMatrix.getInstance();
30     }
31
32     /**
33      * Predicts the vote a user would give an item.
34      *
35      * @param activeUser The user we want the prediction for
36      * @param activeItem The item in which this prediction is done for
37      * @return the predicted vote for the given user/item combination
38      * @throws java.sql.SQLException
39      */
40     public Vote predictVote(User activeUser, Item activeItem) throws SQLException {
41         //Calculate mean vote for active user:
42         double meanForActive = matrix.meanUserVote(activeUser);
43
44         //Retrieve from database the users who have performed a vote
45         UserCollection usersWithWeights = matrix.usersWithWeights();
46
47         double sumWeights = 0d, currentWeight, absWeights = 0d;
48         while (usersWithWeights.hasNext()) {
49             User otherUser = (User) usersWithWeights.next();
50
51             //Make sure we don't compare this user with her/himself:
```

```
52              if (!otherUser.equals(activeUser)) {
53                  currentWeight = weightFunction.doWeight(activeUser, otherUser) *
54                          (matrix.getVote(otherUser, activeItem).getVote() -
55                          matrix.meanUserVote(otherUser));
56
57                  absWeights += Math.abs(currentWeight);
58                  sumWeights += currentWeight;
59              }
60          }
61          //Close the underlying database collection
62          usersWithWeights.close();
63          log.debug("Un-normalized / Normalized: " + sumWeights + " / " + (sumWeights / absWeights));
64          //Return new vote based on calculatoin above
65          return new Vote(activeUser, activeItem, meanForActive + (sumWeights / (absWeights)));
66      }
67
68
69      public void setWeightFunction(WeightFunction wf) {
70          weightFunction = wf;
71      }
72  }
```

B.8 statfind.common.cf.User class source code

```
1  /*
2   * Created by IntelliJ IDEA.
3   * User: olvesh
4   * Date: 07.sep.02
5   * Time: 18:14:27
6   * To change template for new interface use
7   * Code Style | Class Templates options (Tools | IDE Options).
8   */
9  package statfind.common.cf;
10
11 public class User implements Comparable {
12
13     private MeanVote meanVote;
14     private long lastUpdate;
15     private String id;
16
17
18     public User(String id) {
19         this(id, System.currentTimeMillis());
```

```
20      }
21
22      public User(String id, long lastUpdate) {
23          this.id = id;
24          this.lastUpdate = lastUpdate;
25          meanVote = new MeanVote();
26      }
27
28      public String getId() {
29          return id;
30      }
31
32
33      public MeanVote getMeanVote() {
34          return meanVote;
35      }
36
37      class MeanVote {
38          private double mean;
39          private long meanTimeStamp;
40
41          private MeanVote() {
42              mean = 0d;
43              meanTimeStamp = 0;
44          }
45
46
47          /**
48           * If the mean value is new, or changed, this method will update the timestamp.
49           * @param d
50           */
51          public void set(double d) {
52              meanTimeStamp = System.currentTimeMillis();
53              set(d, meanTimeStamp);
54          }
55
56          public void set(double d, long stamp) {
57              mean = d;
58              meanTimeStamp = stamp;
59          }
60
61          public double get() {
62              return mean;
63          }
64
```

```
65          /**
66           * Is valid if users last action happened before this mean vote was calculated.
67           * @return
68           */
69          public boolean valid() {
70              return lastUpdate < meanTimeStamp;
71          }
72
73          public String toString() {
74              return String.valueOf(mean);
75          }
76      }
77
78      public String toString() {
79          StringBuffer sb = new StringBuffer();
80  //          sb.append("id: ");
81          sb.append(id);
82  //          sb.append(" - meanVote: ");
83  //          sb.append(meanVote);
84          return sb.toString();
85      }
86
87      public boolean equals(Object o) {
88          if (this == o) return true;
89          if (!(o instanceof User)) return false;
90
91          final User user = (User) o;
92
93          if (!id.equals(user.id)) return false;
94
95          return true;
96      }
97
98      public int compareTo(Object o) {
99          return id.compareTo(o.toString());
100     }
101
102     public int hashCode() {
103         return id.hashCode();
104     }
105 }
106
```

B.9 statfind.common.cf.Item interface source code

```
1 /*
2  * User: olvesh
3  * Date: 07.sep.02
4  * Time: 18:14:33
5  */
6 package statfind.common.cf;
7
8 public interface Item extends Comparable {
9     String getId();
10 }
```

B.10 UserItemMatrix class source-code

```
1 /*
2  * Created by IntelliJ IDEA.
3  * User: olvesh
4  * Date: 07.sep.02
5  * Time: 18:29:17
6  * To change template for new class use
7  * Code Style | Class Templates options (Tools | IDE Options).
8  */
9 package statfind.common.cf;
10
11
12 import statfind.common.StatfindClient;
13
14 import java.sql.Connection;
15 import java.sql.PreparedStatement;
16 import java.sql.ResultSet;
17 import java.sql.SQLException;
18 import java.util.HashMap;
19
20 public class UserItemMatrix {
21
22     private static UserItemMatrix instance;
23     private static byte[] monitor = new byte[0];
24
25     private PreparedStatement allItems, allUsers, itemsByUser,
26     usersWithWeights, userItemIntersect, userItemUnion;
27
28     private PreparedStatement userCount;
29     private PreparedStatement sumVotesForItem;
```

```
30      private final HashMap voteMap;
31      private int numUsers = Integer.MIN_VALUE;
32      private final HashMap itemsByUserMap;
33      private UserCollection userCollection;
34      private final HashMap itemVoteMap;
35      private ItemCollection itemCollection;
36      private UserCollection userCollectionWithWeights;
37      private final HashMap userItemIntersectMap;
38      private final HashMap userItemUnionMap;
39
40
41      private UserItemMatrix() throws SQLException {
42          initStatements();
43          voteMap = new HashMap();
44          itemsByUserMap = new HashMap();
45          itemVoteMap = new HashMap();
46          userItemIntersectMap = new HashMap();
47          userItemUnionMap = new HashMap();
48      }
49
50      private void initStatements() throws SQLException {
51          Connection c = StatfindClient.instance().getStatfindConnection();
52          String sql;
53
54          sql = "SELECT * FROM user;";
55          allUsers = c.prepareStatement(sql,
56                  ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);
57
58          sql = "SELECT DISTINCT var_id FROM variable;";
59          allItems = c.prepareStatement(sql,
60                  ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);
61
62          sql = "SELECT var_id FROM variable WHERE user = ?;";
63          itemsByUser = c.prepareStatement(sql,
64                  ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY);
65
66          sql = "SELECT DISTINCT user.userName AS userName, MAX(var.timestamp) " +
67                  "AS lastVarUpdate, user.meanVote, user.meanVoteTS FROM variable " +
68                  "AS var, user WHERE user.userName = var.user GROUP BY user.userName;";
69          usersWithWeights = c.prepareStatement(sql,
70                  ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);
71
72          sql = "SELECT v1.var_id FROM variable as v1, variable as v2 " +
73                  "WHERE v1.var_id = v2.var_id AND v1.user = ? AND v2.user = ?;";
74          userItemIntersect = c.prepareStatement(sql,
```

```
75                 ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY);
76
77         sql = "(SELECT var_id FROM variable WHERE user = ?) " +
78                 "UNION (SELECT var_id FROM variable WHERE user = ?);";
79         userItemUnion = c.prepareStatement(sql,
80                 ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY);
81
82         sql = "SELECT COUNT(userName) AS userCount FROM user;";
83         userCount = c.prepareStatement(sql,
84                 ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY);
85
86         sql = "SELECT SUM(viewCount) as sumVotes FROM variable WHERE var_id = ?;";
87         sumVotesForItem = c.prepareStatement(sql,
88                 ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);
89     }
90
91
92     public static UserItemMatrix getInstance() throws SQLException {
93         synchronized (monitor) {
94             if (instance == null) {
95                 instance = new UserItemMatrix();
96             }
97             return instance;
98         }
99     }
100
101
102     /**
103      * Calculates the mean user vote for a selected user. If the value is already in the db,
104      * and is valid, return that one, otherwise; calculate the mean user vote by traversing
105      * all votes for the specified user, and calculate the mean vote.
106      *
107      * @param aUser the user to get the mean vote from
108      * @return the mean vote.
109      */
110     public double meanUserVote(User aUser) throws SQLException {
111         double meanVote;
112         //If the value is already calculated:
113         if (aUser.getMeanVote().valid()) {
114             return aUser.getMeanVote().get();
115         } else {
116             meanVote = StatfindClient.instance().getSumViewCount(aUser) /
117                     Math.abs(StatfindClient.instance().getSumItems(aUser));
118             if (Double.isNaN(meanVote)) {
119                 meanVote = 0d;
```

```
120             }
121             //Set the new meanVote in the used object:
122             aUser.getMeanVote().set(meanVote);
123         }
124         return meanVote;
125     }
126 
127 
128     public Vote getVote(User u, Item i) throws SQLException {
129         Vote v = null;
130         String key = u.getId() + i.getId();
131         if (voteMap.containsKey(key)) {
132             v = (Vote) voteMap.get(key);
133         } else {
134             v = new Vote(u, i);
135             voteMap.put(key, v);
136         }
137         return v;
138     }
139 
140     public int getViewCount(String user, String item) throws SQLException {
141         return StatfindClient.instance().getViewCount(item, user);
142     }
143 
144     /**
145      *Filters out the items that the supplied user has voted for.
146      */
147     public ItemCollection itemsVotedForByUser(User aUser) throws SQLException {
148         itemsByUser.setString(1, aUser.getId());
149 
150         ItemCollection retVal;
151         if (itemsByUserMap.containsKey(aUser)) {
152             retVal = (ItemCollection) itemsByUserMap.get(aUser);
153             retVal.reset();
154         } else {
155             retVal = new ItemCollection(itemsByUser, true);
156             itemsByUserMap.put(aUser, retVal);
157         }
158         return retVal;
159     }
160 
161     public UserCollection getUsers() throws SQLException {
162         if (userCollection == null)
163             userCollection = new UserCollection(allUsers, true);
164         else
```

```
165             userCollection.reset();
166         return userCollection;
167     }
168 
169     public int getUserCount() throws SQLException {
170         if (numUsers == Integer.MIN_VALUE) {
171             ResultSet rs = userCount.executeQuery();
172             rs.next();
173             numUsers = rs.getInt(1);
174         }
175         return numUsers;
176     }
177 
178     public int getVoteCountForItem(final Item item) throws SQLException {
179         int sumVotes = 0;
180         if (itemVoteMap.containsKey(item)) {
181             sumVotes = ((Integer) itemVoteMap.get(item)).intValue();
182         } else {
183             sumVotesForItem.setString(1, item.getId());
184             ResultSet rs = sumVotesForItem.executeQuery();
185             if (rs.next()) {
186                 sumVotes = rs.getInt("sumVotes");
187             }
188             itemVoteMap.put(item, new Integer(sumVotes));
189         }
190         return sumVotes;
191     }
192 
193     public ItemCollection getItems() throws SQLException {
194         if (itemCollection == null)
195             itemCollection = new ItemCollection(allItems, true);
196         else
197             itemCollection.reset();
198         return itemCollection;
199     }
200 
201     public UserCollection usersWithWeights() throws SQLException {
202         if (userCollectionWithWeights == null)
203             userCollectionWithWeights = new UserCollection(usersWithWeights, true);
204         else
205             userCollectionWithWeights.reset();
206         return userCollectionWithWeights;
207     }
208 
209     public ItemCollection userItemIntersect(User a, User b) throws SQLException {
```

```
210         String key = a.getId() + b.getId();
211         ItemCollection retVal;
212 
213         if (userItemIntersectMap.containsKey(key)) {
214             retVal = (ItemCollection) userItemIntersectMap.get(key);
215             retVal.reset();
216         } else {
217             userItemIntersect.setString(1, a.getId());
218             userItemIntersect.setString(2, b.getId());
219             retVal = new ItemCollection(userItemIntersect, true);
220             userItemIntersectMap.put(key, retVal);
221         }
222         return retVal;
223     }
224 
225     public ItemCollection userItemUnion(User a, User b) throws SQLException {
226         String key = a.getId() + b.getId();
227         ItemCollection retVal;
228         if (userItemUnionMap.containsKey(key)) {
229             retVal = (ItemCollection) userItemUnionMap.get(key);
230             retVal.reset();
231         } else {
232             userItemUnion.setString(1, a.getId());
233             userItemUnion.setString(2, b.getId());
234             retVal = new ItemCollection(userItemUnion, true);
235             userItemUnionMap.put(key, retVal);
236         }
237         return retVal;
238     }
239 
240     public void close() throws SQLException {
241         synchronized (monitor) {
242             allItems.close();
243             allUsers.close();
244             itemsByUser.close();
245             usersWithWeights.close();
246             userItemIntersect.close();
247             userItemUnion.close();
248             instance = null;
249         }
250     }
251 
252 
253 }
```

B.11 statfind.common.StatfindClient class source code

```
1  package statfind.common;
2
3  /**
4   * User: olvesh
5   * Date: Jan 26, 2003
6   * Time: 5:18:45 PM
7   */
8
9  import org.apache.log4j.Logger;
10 import statfind.common.cf.User;
11 import statfind.common.cf.functions.WeightFunction;
12
13 import java.io.IOException;
14 import java.net.MalformedURLException;
15 import java.rmi.NotBoundException;
16 import java.sql.*;
17 import java.util.Properties;
18 import java.util.Vector;
19
20 public class StatfindClient {
21     protected static Logger log = Logger.getLogger(StatfindClient.class);
22     protected Properties props;
23     private Vector users;
24     protected Connection statfindConnection;
25     protected Connection nesstarConnection;
26     private PreparedStatement questTextStatement;
27     private PreparedStatement viewCountStatement;
28     private PreparedStatement highestViewCountStatement;
29     private PreparedStatement viewSumCountStatement;
30
31     private static StatfindClient instance = null;
32
33     protected StatfindClient() throws SQLException, NotBoundException, IOException,
34             IllegalAccessException, MalformedURLException, ClassNotFoundException,
35             InstantiationException {
36         init();
37
38     }
39
40     public static synchronized StatfindClient instance() {
41         if (instance == null) {
```

```
42            try {
43                instance = new StatfindClient();
44            } catch (ClassNotFoundException e) {
45                //log.error("", e);
46                throw new StatfindRuntimeException(e);
47            } catch (InstantiationException e) {
48                //log.error("", e);
49                throw new StatfindRuntimeException(e);
50            } catch (SQLException e) {
51                //log.error("", e);
52                throw new StatfindRuntimeException(e);
53            } catch (IllegalAccessException e) {
54                //log.error("", e);
55                throw new StatfindRuntimeException(e);
56            } catch (NotBoundException e) {
57                log.error("", e);
58            } catch (MalformedURLException e) {
59                log.error("", e);
60            } catch (IOException e) {
61                log.error("", e);
62            }
63        }
64        return instance;
65    }
66
67
68    protected void init() throws InstantiationException, IllegalAccessException,
69            ClassNotFoundException, SQLException, IOException, NotBoundException,
70            MalformedURLException {
71        props = new Properties(CommonProperties.getCommonDefaults());
72        users = new UserSelector(props).getDbUsers();
73
74        String driver = props.getProperty("statfind-db.jdbcdriver");
75        String dsn = props.getProperty("statfind-db.url");
76        String usr = props.getProperty("statfind-db.user");
77        String pwd = props.getProperty("statfind-db.pass");
78
79        Class.forName(driver).newInstance();
80        statfindConnection = DriverManager.getConnection(dsn, usr, pwd);
81
82        driver = props.getProperty("nesstar-db.jdbcdriver");
83        dsn = props.getProperty("nesstar-db.url");
84        usr = props.getProperty("nesstar-db.user");
85        pwd = props.getProperty("nesstar-db.pass");
86
```

```
87          Class.forName(driver).newInstance();
88          nesstarConnection = DriverManager.getConnection(dsn, usr, pwd);
89      }
90
91      public Connection getStatfindConnection() {
92          return statfindConnection;
93      }
94
95      public Vector getUsers() {
96          return users;
97      }
98
99      private void initQuestTextStatement() throws SQLException {
100         if (questTextStatement != null)
101             return;
102
103         final String sql = "SELECT questionText FROM variableejb WHERE id = ?;";
104         //TO DO: Which fields needs to be CONCUR_UPDATABLE and which CONCUR_READ_ONLY??
105         questTextStatement = nesstarConnection.prepareStatement(sql,
106                 ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY);
107
108     }
109
110     public String getQuestionText(Variable var) throws SQLException {
111         return getQuestionText(var.getId());
112     }
113
114     public String getQuestionText(String varId) throws SQLException {
115         initQuestTextStatement();
116         String text = null;
117         questTextStatement.clearParameters();
118         questTextStatement.setString(1, varId);
119         if (questTextStatement.execute()) {
120             ResultSet rs = questTextStatement.getResultSet();
121             if (rs.next()) {
122                 text = rs.getString("questionText");
123             }
124             rs.close();
125         }
126         return text;
127     }
128
129     public double getSumViewCount(User aUser) throws SQLException {
130         initSumViewCountStatement();
131         int viewCount = 0;
```

```
132
133         viewSumCountStatement.setString(1, aUser.getId());
134
135         if (viewSumCountStatement.execute()) {
136             ResultSet rs = viewSumCountStatement.getResultSet();
137             if (rs.next()) {
138                 viewCount = rs.getInt("totViewCount");
139             }
140             rs.close();
141         }
142         return viewCount;
143     }
144
145     public int getSumItems(User aUser) throws SQLException {
146         initSumViewCountStatement();
147         int sumItems = 0;
148
149         viewSumCountStatement.setString(1, aUser.getId());
150
151         if (viewSumCountStatement.execute()) {
152             ResultSet rs = viewSumCountStatement.getResultSet();
153             if (rs.next()) {
154                 sumItems = rs.getInt("sumItems");
155             }
156             rs.close();
157         }
158         return sumItems;
159     }
160
161     private void initSumViewCountStatement() throws SQLException {
162         if (viewSumCountStatement != null)
163             return;
164         //TO DO: Which fields needs to be CONCUR_UPDATABLE and which CONCUR_READ_ONLY??
165         String sql = "SELECT SUM(viewCount) AS totViewCount, COUNT(var_id) AS sumItems " +
166                 "FROM variable WHERE user = ?;";
167         viewSumCountStatement = statfindConnection.prepareStatement(sql,
168                 ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);
169
170     }
171
172     private void initViewCountStatement() throws SQLException {
173         if (viewCountStatement != null)
174             return;
175
176         //TO DO: Which fields needs to be CONCUR_UPDATABLE and which CONCUR_READ_ONLY??
```

```
177         String sql = "SELECT viewCount FROM variable WHERE user = ? AND var_id = ?;";
178         viewCountStatement = statfindConnection.prepareStatement(sql,
179                 ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);
180     }
181
182
183     public int getViewCount(String varId, String user) throws SQLException {
184         initViewCountStatement();
185 //        log.debug("User: "+user);
186 //        log.debug("Var : "+varId);
187         int viewCount = (int) WeightFunction.DEFAULT_VOTE;
188         viewCountStatement.setString(1, user);
189         viewCountStatement.setString(2, varId);
190
191         if (viewCountStatement.execute()) {
192             ResultSet rs = viewCountStatement.getResultSet();
193             if (rs.next()) {
194                 viewCount = rs.getInt("viewCount");
195             }
196             rs.close();
197         }
198         return viewCount;
199     }
200
201     private void initHighestViewCount() throws SQLException {
202         if (highestViewCountStatement != null)
203             return;
204         //TO DO: Which fields needs to be CONCUR_UPDATABLE and which CONCUR_READ_ONLY??
205         String sql = "SELECT MAX(viewCount) AS maxViewCount FROM variable WHERE user = ?;";
206         highestViewCountStatement = statfindConnection.prepareStatement(sql,
207                 ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);
208
209     }
210
211     public int getHighestViewCount(String user) throws SQLException {
212         initHighestViewCount();
213
214         int viewCount = 0;
215         highestViewCountStatement.clearParameters();
216         highestViewCountStatement.setString(1, user);
217
218         if (highestViewCountStatement.execute()) {
219             ResultSet rs = highestViewCountStatement.getResultSet();
220             if (rs.next()) {
221                 viewCount = rs.getInt("maxViewCount");
```

```
222             }
223             rs.close();
224         }
225         return viewCount;
226     }
227
228 }
```

B.12 statfind.common.ir.StatfindIRClient class source code

```
1  package statfind.common.ir;
2
3  /**
4   * User: olvesh
5   * Date: Jan 18, 2003
6   * Time: 3:50:49 PM
7   */
8
9  import org.apache.log4j.Logger;
10 import org.apache.lucene.analysis.standard.StandardAnalyzer;
11 import org.apache.lucene.document.Document;
12 import org.apache.lucene.queryParser.ParseException;
13 import org.apache.lucene.queryParser.QueryParser;
14 import org.apache.lucene.search.*;
15 import statfind.common.StatfindClient;
16 import statfind.common.Variable;
17 import statfind.common.cf.CollaborativeFilter;
18 import statfind.common.cf.User;
19 import statfind.common.cf.UserItemMatrix;
20 import statfind.common.cf.Vote;
21 import statfind.common.cf.functions.WeightFunction;
22
23 import java.io.IOException;
24 import java.net.MalformedURLException;
25 import java.rmi.NotBoundException;
26 import java.rmi.registry.LocateRegistry;
27 import java.sql.PreparedStatement;
28 import java.sql.ResultSet;
29 import java.sql.SQLException;
30 import java.util.Collection;
31 import java.util.Iterator;
```

```
32 import java.util.SortedSet;
33 import java.util.TreeSet;
34
35
36 public class StatfindIRClient extends StatfindClient {
37     protected static Logger log = Logger.getLogger(StatfindIRClient.class);
38
39     private static StatfindIRClient instance;
40     public static final String VAR_ID = "id";
41     public static final String LABEL = "label";
42     public static final String QUEST_TEXT = "questionText";
43     private QueryParser queryParser;
44     private Searcher luceneIndex;
45     private PreparedStatement getVarIDStatement;
46     private CollaborativeFilter collFilter;
47
48     private StatfindIRClient() throws ClassNotFoundException, IllegalAccessException,
49             InstantiationException, SQLException, IOException, NotBoundException,
50             MalformedURLException {
51         init();
52         queryParser = new QueryParser(QUEST_TEXT, new StandardAnalyzer());
53         queryParser.setOperator(QueryParser.DEFAULT_OPERATOR_OR);
54         collFilter = new CollaborativeFilter();
55     }
56
57
58     protected void init() throws InstantiationException, IllegalAccessException,
59             ClassNotFoundException, SQLException, IOException, NotBoundException,
60             MalformedURLException {
61         super.init();
62
63         Searchable s[] = {(Searchable)
64                 LocateRegistry.getRegistry("olvesh.no-ip.org",
65                         Integer.parseInt(props.getProperty("rmiPort"))).
66                 lookup(props.getProperty("luceneRmiAddress"))};
67         //This forced the server to be on the same machine as the client:
68 //        Searchable s = (Searchable)
69 //                LocateRegistry.getRegistry(Integer.parseInt(props.getProperty("rmiPort"))).
70 //                lookup(props.getProperty("luceneRmiAddress"));
71 //        luceneIndex = new IndexSearcher("D:/dev/statfindRoot2/statfind/luceneIndex");
72         luceneIndex = new MultiSearcher(s);
73     }
74
75     public static synchronized StatfindIRClient getInstance()
76             throws ClassNotFoundException, InstantiationException, SQLException,
```

```
77              IllegalAccessException, IOException, NotBoundException, MalformedURLException {
78          if (instance == null) {
79              instance = new StatfindIRClient();
80          }
81          return instance;
82      }
83
84
85      public Collection getSimilarVars(String search, User user, WeightFunction wf)
86              throws SQLException, ParseException, IOException {
87          Query query = queryParser.parse(search);
88          return getSimilarVars(query, null, user, wf);
89      }
90
91      public Collection getSimilarVars(Variable currentVar, User user, WeightFunction wf)
92              throws SQLException, ParseException, IOException {
93          Query query = queryParser.parse(getQuestionText(currentVar));
94          return getSimilarVars(query, currentVar, user, wf);
95      }
96
97      /**
98       * Fetches an array list of variables similar to the one given as argument, the list is sorted
99       * descending by similarity.
100      * @param currentVar
101      * @return
102      * @throws java.sql.SQLException
103      */
104     public Collection getSimilarVars(Query query, Variable currentVar, User user, WeightFunction wf)
105             throws SQLException, ParseException, IOException {
106         log.info("");
107         if (user != null && wf != null) {
108             collFilter.setWeightFunction(wf);
109             log.info(wf);
110         }
111         SortedSet similarVars = new TreeSet();
112         Variable similarVar;
113
114         boolean sameVarFound = false;
115 //        questText = questText +" id: NOT "+varId;
116
117
118         Hits hits = luceneIndex.search(query);
119
120         if (log.isDebugEnabled()) {
121             log.debug("StatfindIRClient.getSimilarVars");
```

```
122 //            log.debug(getQuestionText(currentVar));
123             log.debug("Query: " + query);
124             log.debug("Num hits: " + hits.length());
125 
126             log.debug("Current Var: " + (currentVar != null ? currentVar.getId() : "null"));
127             if (user != null) {
128                 log.debug("User       : " + user.getId());
129                 log.debug("Mean vote  : " + UserItemMatrix.getInstance().meanUserVote(user));
130             }
131         }
132 
133         for (int i = 0; i < hits.length(); i++) {
134             if (hits.score(i) < Variable.SIGNIFICANT_VALUE) {
135                 if (log.isDebugEnabled()) {
136                     log.debug("Breaking loading of hits due to value less than SIGNIFICANT");
137                     log.debug("Listed: " + i + " remaining: " + (hits.length() - i));
138                 }
139                 break;//Now the rest of the hits are below the significant value, ignore them
140             }
141             Document doc = hits.doc(i);
142             similarVar = new Variable(doc.getField(VAR_ID).stringValue());
143             similarVar.setSimilarity(hits.score(i));
144 
145 
146             if (currentVar != null && !sameVarFound &&
147                     currentVar.getId().equalsIgnoreCase(similarVar.getId())) {
148                 sameVarFound = true;
149             } else {
150                 if (user != null && wf != null) {
151                     double oldSim = similarVar.getSimilarity();
152                     Vote predictedVote = collFilter.predictVote(user, similarVar);
153                     similarVar.updateSimilarity(predictedVote);
154                     similarVar.setPredictedValue(predictedVote.getVote());
155 
156                     if (log.isDebugEnabled()) {
157                         log.debug("var    : " + similarVar.getId());
158                         log.debug("vote   : " + predictedVote.getVote());
159                         log.debug("sim    : " + oldSim);
160                         log.debug("new sim: " + similarVar.getSimilarity());
161                     }
162 
163                 }
164                 similarVars.add(similarVar);
165             }
166         }
```

```
167
168         //Provide a printout of the variables in descending similarity:
169         Iterator i = similarVars.iterator();
170         while (i.hasNext()) {
171             log.info(i.next());
172         }
173
174         return similarVars;
175     }
176
177     public Variable getVariable(String id) throws SQLException {
178         initGetVarID();
179         getVarIDStatement.setString(1, id);
180         ResultSet rs = getVarIDStatement.executeQuery();
181         rs.next();
182         Variable v = new Variable(rs.getString(1));
183         return v;
184     }
185
186     private void initGetVarID() throws SQLException {
187         if (getVarIDStatement == null) {
188             String sql = "SELECT id FROM variableejb WHERE id = ?;";
189             getVarIDStatement = nesstarConnection.prepareStatement(sql,
190                     ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY);
191         }
192     }
193
194 }
```

B.13 Property files used

```
##############################################
#statfind-common.properties
#Default properties common to all.
server.port             = 2380
statfind-db.jdbcdriver  = org.gjt.mm.mysql.Driver
statfind-db.url         = jdbc:mysql://olvesh.no-ip.org:3306/statfind

#Use the same user/pass as in statfind-foreign.properties, as these can easily be overridden
statfind-db.user        = statfindForeign
statfind-db.pass        = findstatForeign

nesstar-db.jdbcdriver   = org.gjt.mm.mysql.Driver
```

```
nesstar-db.url          = jdbc:mysql://olvesh.no-ip.org:3306/nesstar
nesstar-db.user         = statfind
nesstar-db.pass         = findstat

luceneRmiAddress         = //olvesh.no-ip.org:1098/LuceneIndex
rmiPort                  = 1098

##############################################
#statfind-foreign.properties
host                    = olvesh.no-ip.org
retry                   = 3

variableResultPreTag    = /obj/fVariable/

#Use the same user/pass as in statfind-common.properties,
#as these can easily be overridden
statfind-db.user        = statfindForeign
statfind-db.pass        = findstatForeign

##############################################
#statfind-server.properties
shutdown.port           = 2379
retry                   = 3

statfind-db.user        = statfind
statfind-db.pass        = findstat

lucene.index            = D:/dev/statfindRoot2/statfind/luceneIndex
```

Appendix C

User to MD5 keys

Following is a list of users, and what MD5 key belong to them. This was replaced throughout the thesis for readability purposes.

User A	002f48e213c339f5d1d839185dcc8e7f
User B	128c7e758caee08b656e99149f36381d
User C	12b5a0c85b1e58c828d5c38efe7734fa
User D	20f824bc807d2b7ebbf0714e2519111f
User E	a1c8626566f2ae8e421dbfad65c46ff6
User F	c7218260ef2b966ab0454e07c55cf4e9
User G	e110b01d9000b60172fc177e634d2530