Collaborative Filtering of Web Documents

Marc-Ulrich Maier

12th January 1999

Abstract

The aim of this project was to build a demostratable collaborative filtering system for the World Wide Web. Collaborative filtering systems coordinates social interaction between its user so that they can exploit the experience of others in the search for interesting information and give guidance themselves. The information overload and therefore the need for such systems is nowhere more apparent than on the internet. The implemented filtering system extends Web browsers to give users the opportunity to rate a Internet pages and to exploit the ratings issued by others for the purpose of information search.

This paper provides a short introduction to the theory of collaborative filtering, discusses to what extent simple random sampling can be applied to statistically evaluate rating information and explains the architecture and implementation of the system.

Contents

1 Introduction

Information can be valuable for a variety of different reasons. Of course information in itself can be valuable to a reader because it is information about a specific subject or general information the reader is interested in, for example, newsgroups and journals, that focus on a particular theme, or newspapers. The value of information largely depends on the person. A marketing director will find a scientific article about the discovery of a new stable element less useful than a physicist who researches in that area. Information, though, can also have a social value since it provides the consumers with "something to talk about" (television, newspapers, magazines, cinemas), a base for communication between people that otherwise would have nothing in common or the feeling of belonging to a certain community. Collaborative, or social filtering is a new technique for dealing with the huge amounts of data that modern information media like the Internet provide by introducing a kind of social interaction. This interaction serves the purpose of rating the value of information and directing the reader to valuable information.

In the real world there are mainly two factors that attract peoples' interest. Recommendations or ratings issued by trusted persons or a group of persons, e.g. organisations, and the number of (trusted) peoples who consume this information, e.g. the number of copies of a book. The goal of this project was to collect rating information from the web user and supply him with ratings of other people and therefore introduce a kind of social awareness into the Internet on a small basis. Ratings can be divided into two categories: implicit and explicit ratings. Explicit ratings require the user's effort to undertake a rating action, for example entering a number in a dialog box that is associated with a particular web page. Implicit ratings, however, are derived from the browsing behaviour of the user. The most basic evidence of the quality of a web page is the fact that it has been accessed. Other evidence includes the amount of time the user dedicates to it. Both categories will were tackled in this project.

In order to provide the user of a web browser with the possibility to issue rating information, the W3C Jigsaw Java proxy was modified so that an Java applet is injected into every web document that is accessed through this proxy. The Java applet opens a dialog box where the user can enter an explicit rating for the current document. The applet also collects information about the browsing behaviour of the user. All rating information is stored and evaluated on a single computer and made available for further analysis.

2 Theory

2.1 Collaborative Filtering

"Social, or collaborative filtering is an emerging technique for dealing with overload in information environments" [1]. Most social filtering techniques are based on user ratings, statements of quality that are used either as guidance for other readers or to predict the value of new information.

A major difficulty with this technique is that it requires the reader's effort to issue a rating. The laziness of human beings cannot be overestimated. The slightest effort reduces the number of ratings dramatically, especially if the benefits of rating are low or not clear to them. In real life there are numerous possibilities to issue a statement about the quality of information which require different degrees of effort. This ranges from the survey that is carry about by a big department store about costumer confidence, which takes a little time to answer some questions or to fill in a form, to the letter to the editor requiring one or two hours and careful thought.

Another problem is how people can judge the value of other people's ratings for them, i.e. how someone becomes trusted as a rater. Normally organisation or companies employ people that have knowledge and/or a reputation of competence in a specific area. For example, newspapers chose a journalist to write a critique about a movie or play. The only choice the reader has, is which newspaper to read. Generally readers trust the newspaper to choose a competent journalist and therefore their critiques as well. But readers also are more likely to trust recommendations from people with similar interests, especially if the reason for a statement of quality seems plausible. However, even within a relatively small interest group or community there are usually considerable differences in competence and ability. Within communities people learn to trust persons that have developed a reputation of competence.

These problems were addressed by the GroupLens project of the Department of Computer Science and Engineering at the University of Minnesota [2]. The purpose of the project was to provide a collaborative filtering mechanism for Usenet newsgroups. Common newsgroup browsers were extended to provide the possibility of rating articles to the users. Additionally, implicit ratings were acquired by measuring the time spent reading an article. The data collected from newsgroup readers showed that there is a high correlation between the time spent reading an article and the rating for it [2]. Implicit and user ratings are combined into a single utility that is visible to all other readers of the newsgroup.

Readers are guided to valuable information on three levels:

- The first level is the newsgroup itself. A newsgroup forms a virtual community - an interest group - that provides a limited degree of social interaction via topic specific "blackboards". So users will find articles they are interested in more quickly because the information in the groups is topic bound.

- The second level of aid is provided by the ratings that were acquired from other readers. Each article is labelled with the average rating [3]. Articles with a high average rating are more likely to contain high quality information.

- For some newsgroups a third level of guidance called "Personalized Predictions" based on the automated collaborative filtering technique [4] was supplied [3]. Profiles of users are built on the collected ratings to find people with similar interests. This was used to calculate a predictive utility for an article so that the ratings fit the individual tendency better, i.e. trustworthy raters are found automatically.

2.1.1 Social Filtering and the Web

Derived from the concept of a library, the design of the World Wide Web does not leave room for social interaction. Examples of exceptions are web based chats and internet games like Yahoo chess. The Web user demands information and does not leave tracks of which information he/she consumed. Virtually nobody can monitor what others are doing. In terms of privacy this is a big advantage, in terms of filtering however an immense disadvantage. In real life a lot of guidance to information, places and events of interest is provided, for example by the number of people that are attracted and by information providers such as newspapers or bullet boards.

The GroupLens project has successfully proven that social filtering can be introduced to a given structure, the Usenet, for the benefit of its users. But how can the advantages of social filtering also be made available to Web users? It is important to be aware of the differences between Usenet and WWW:

- The Web, unlike the Usenet, is not designed to support virtual communities. The Web is an universal information resource provider. The only groups that are known to the Web are domains that usually represent working units like companies and organisations rather than interest groups.

- Newgroups are meant to be discussion groups and provide a limited degree of interaction for that. Participants can read and post messages without a lot of effort. Most users that consume information, will also make a contribution. In the Web, however, the ratio of those who design web pages to those who browse them is very low. In addition, the amount of time to produce a Web page is significantly higher than the time it takes to type a message. Therefore therefore the value of information published is normally intended to be much more durable. There are a few exceptions. Some Web site, especially those of newspapers, offer the latest news or stock rates, weather forecasts and similar services. However the value of the page does not depend so much on the current content but on the general quality of the service.

- The equivalent to the message in the newsgroup is the Web page. However, the structure of a page can be very complex since frames were introduced. Frames serve the purpose of structuring a page better and lower download times by loading only the variable parts of the page and keeping the constant ones such as navigation bars. This extention of the HTML language has a lot of advantages for the users, but raises a lot of problems in terms of how to rate and evaluate frame pages.

- As the content of a web page can be composed of a variety of information (e.g. links, images, sounds, movies, forms ...), the reason why the user likes or does not like a page can be as multifaceted. To some users the visual appearance is very important, to others the navigational value of the page (as a start point for the search for good pages) is a major concern and other are more interested in the content and its actuality. For posting to a newsgroup usually the content of the message is the only value criterion.

- As web pages are hosted on computers and maintained by web masters and not posted to the information consumers, the content of a web page can change at any time and with the content, the quality of this page.

All these points show that the experience and results of the GroupLens project can not simply be transplanted into the domain of the World Wide Web and raise a number of questions that will be (at least in part) be tackled in the following chapter.

There are other filtering approaches that are based on personal bookmark lists. The Group Asynchronous Browsing (GAB) project, for example, was concerned with collection and merging of personal bookmarks within a group [6]. The result was a common bookmark list web page that was browseable by the group. The advantage is that it is based upon already existent, high quality information. However, studies show that under 50% of the web pages that users find interesting, actually have a bookmark [7] in the user's browser. Furthermore the joined bookmark page can not provide a reliable quality statement about the links which are desirable especially in larger bookmark collections.

2.1.2 Implicit and Explicit Ratings

Explicit ratings are intentional utterance of quality that require the users effort. Therefore it is adviseable to keep the time and complexity that is connected with submitting a rating information as low as possible. The quality of explicit ratings is normally very high, because the user has to think about the verdict for a page at least for a short time. But the consequence is that a lot of users can not be bothered to rate a page and therefore collected rating information is less reliable to reflect the general opinion. It has been proposed to reward users for rating. "While the economic consequences of this solution are interesting, we wonder whether compensation would be necessary if ratings could be captured without any effort on the part of the user." [2]

Explicit ratings have the problem that they are lacking in context, i.e. the circumstances of the utterance are unknown [8]. Implicit ratings can provide many hints about the context, but it is difficult to evaluate them in a way that is useful to others. We will take a look at three dimensions that contribute to the context of an implicit rating.

People Which person provides a rating is obviously important for the relevance and the significance of a rating. Ideally rating information collected from an expert in the domain of interest would provide a decent guide to high-value information. "One of the most effective ways to find information is to find the right person or people who are likely to know about what we are looking for." [6] But it is difficult to find this person and it is questionable whether he would be pleased that every step he takes in the Web is monitored by hundreds of thousands of disciples. Apart from that, it is not feasible to collect browsing statistics of each person in a daily growing community like the internet.

The question is whether rating information should be made public on a group level rather than on individual. A group in the web formed by a domain (or sub domain) which represent mostly local and aim-bound communities like companies, organisations, universities ... "One reason that groups defined by organisational affiliation have a locus of interest, is that it pays to be informed about one anothers' activities." [8] There are two advantages of collecting data on a group level. First, the amount of data that has to be stored, analysed and made publically available is much lower. Second, issues of privacy become less relevant as the retrieved information is averaged and therefore identification of the data with a particular user is not possible.

Documents The unit of implicit rating is the single HTML page. A study in connection with the GroupLens project showed that there is a linear relationship between the time spent reading a posting and the explicit rating issued [2]. It is unsure whether this holds for web pages for the following reasons:

- The time that a user spends reading the page is hard to define. Often the beginning of the web page is visible although it has not been downloaded completely due to high data traffic on the Internet connection. In pages containing so called anchor tags the opposite may be the case: The beginning of the document is already loaded, but not the requested place within the page.

- Web users may leave their computer while a page is displayed or work on something else while the page is downloading, but fail to recognise when it is downloaded. Therefore it is sensible to restrict the measure of time a page has been viewed to a certain threshold.

- Users that like a certain page will not only take more time read it, but also access the page more often. If ratings are collected on a group level there is no way to determine whether a page is viewed for a long period of time by a lot of users but just occasionally visited or whether the page is frequently browsed by a minority that are interested in the content. It is not trivial to estimate the error that is caused by this fact.

- Pages serve different purposes. Some pages are valueable because of their content, other pages because they contain a lot of useful links and serve as an originator for interesting pages [9] , so have a navigational value. Implicit ratings should take this fact into account. In the case of a navigational page, the access frequency is a realistic indicator [8] which also takes into account the fact that the navigational value can change over time.

Unfortunately, there has been no sophisticated study with the aim of investigating the relationship between the time a web page is viewed and the value a user would assign to it, as far as I know.

There are also other criteria that can be applied to determine the quality. It follows a list of measures that has been proposed by Proctor and McKinlay [8]. These measures are derived from discourse analysis which is an established technique for studying the conversational relationships between speakers' utterances.

- nominal rating: the aggregate page viewing time

- frequency: the number of times a page is requested

- sequential accountability: the number of objects and links within a page

- distributional accountability: common sequences of accesses within the community
- sources: The number of times the page is identified as the beginning of a distributionally accountable sequence
- topical coherence as measured by the inter-document text similarity
- temporal coherence: the temporal distribution of page accesses

As mentioned above, web documents, so called frame set pages, can consist of a collection of single HTML pages. It makes sense to acquire implicit rating for each single frame page for two reasons. First, each frame page can appear in a different context as a single page or be a frame set page as well. The fact that the page appears within a set of frames is not a property of the page, but the way it is presented at the time. Second, the time each frame is displayed can be independent of the other frames. A navigational frame may stay while the content frame is changing.

Time The value of a Web document for group can change over time for two reason [10]:

- The information of the page can be dependent. Pages dedicated to current events will lose or gain popularity after the event has happened. Or the content of a page can change or be deleted for numerous reasons.
- The group may change its interest or, more generally, the attitude to a page. For example, if a project is finished within a company, the Web resources for that project will be visited less frequently.

Thus it is important that all implicit data is *not* accumulative, but reflects the importance of a page for certain time window. A frequency measure has this property.

2.1.3 Privacy

If the browsing behaviour is made accessable to other people, it is appropriate to consider issues of privacy. We have already discussed above that privacy for data evaluation on a group level is basically given. Indeed, companies often try to build up a profile of their current or potential costumers by asking questions indicating that the data is kept private and is only used for statistical purposes. However, privacy is only guaranteed if the number of members in the group is respectively high. But even then user may wish to decide whether their browsing behaviour is tracked automatically, i.e. implicit ratings are collected. So the implicit rating acquisition should be controllable by the user.

Privacy is not only a matter of information acquisition, but also of presentation and usage. It is important that the collected information is only made available in a way that is acknowledged by the user. The feeling that he/she or the group is the main usufructuary and not somebody unknown, provides the ground for a positive attitude towards information collection tools.

2.2 Survey Sampling

In order to evaluate collected data, it is necessary to take a look at the mathematical models and techniques. This chapter is a summary of the relevant parts of the chapters 7 from the book "Mathematical Statistics and Data Analysis"[5] . Here equations are given without proof, but the interested reader is encouraged to look them up.

The aim of survey sampling is to obtain information about a large population (large number of values) by examining only a small fraction of it. A sample consists of randomly chosen values. So the sample is a random variable.

2.2.1 Population Parameters

We will now define some numerical characteristics of the population , called parameters, that are to be estimated from the sample. The population size will be denoted as N and and the values as x_1, x_2, ..., x_N.

The population mean or average is defined as

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i$$

The population variance is the average squared distance from the mean

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2$$

The standard deviation σ is the square root of the variance of the population. It is a measure of how spread out the value are. A high value indicates a very dispersed population, a low value that the population is closely gathered around the average.

2.2.2 Simple Random Sampling

Simple random sampling is based upon the urn model, where a sample of n balls is drawn from a total of N balls without replacement. Each ball has the same probability of being drawn and a sample contains a particular ball not more than once. As the sample is chosen randomly the mean of the sample is random as well.

In the following equations n denotes the sample size and X_1, X_2, ..., X_n the values of the sample. The expectation value of the sample is

$$E(X) = \overline{X} = \frac{1}{n} \sum_{i=1}^{n} X_i$$

It estimates the mean of the population.

The variance of the sample is

$$Var(X) = \sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (X_i - E(X))^2$$

As explained above the mean of the samples is a random variable. The variance of the mean is

$$Var\left(\overline{X}\right) = \left(1 - \frac{n-1}{N-1}\right)\frac{\sigma^2}{n}$$

and the standard error of the mean is

$$\sigma_{\overline{X}} = \sqrt{Var\left(\overline{X}\right)} = \sqrt{\left(1 - \frac{n-1}{N-1}\right)}\frac{\sigma}{\sqrt{n}}$$

The term $1 - \frac{n-1}{N-1}$ is called the *finite population correction*. Usually the sample size n is very small compared to the population size N, so that the finite population correction is almost 1. Then the standard error of the mean can be estimated as

$$\sigma_{\overline{X}} \approx \frac{\sigma}{\sqrt{n}}$$

The error of this estimation is less then 1% if the ratio $\frac{n-1}{N-1}$ is less than 2% because

$$0.99 < \sqrt{\left(1 - \frac{n-1}{N-1}\right)} \Leftrightarrow 1 - 0.99^2 > \frac{n-1}{N-1} \Leftrightarrow 0.0199 > \frac{n-1}{N-1}$$

The standard error of the mean is an appropriate measure of the accuracy of the mean. If σ is small then the values are not very dispersed and small sample size will provide enough accuracy. However, if the values are widely dispersed, a much larger sample size is required to attain the same accuracy.

3 Conception of the System

3.1 Groups

For reasons discussed in chapter 2.1.2 and 2.1.3 the unit of rating collection and publishing is the group. In this project a group is formed by a domain for the following reasons:

- The domain is an organisational unit already present in the internet, so that no group membership registration for the rating system is necessary.
- Domains usually pool people with common or similar aims such as organisations, research departments and companies.
- It is very hard to pretend to be in a different domain, so that no additional security measurements are required to ensure that somebody is actually a member of the group, he claims to be part of.

The rating system allows members of a group (domain) to submit implicit and explicit ratings for their own group but not for other groups. But each member can read published rating-derived information of any group that participates as if they were members of that group. The group that the user is a member of, is referred to as *home group* in this project and the group the user is observing, as *monitored group*. The table below summarises this important concept.

	home group	monitored group
ratings	read/write	read
group change	only if member of several domain	possible at any time

3.2 Ratings

To implement the group concept outlined in the previous chapter, rating information is collected separately for each group, i.e. domain. Within a group rating information is stored separately for each URL since a single HTML page is a sensible unit of evaluation (see chapter 2.1.2).

Both implicit and explicit rating of a page is supported by the system. Explicit ratings can be submitted by the user on a scale from 1 to 5 which is converted into a value between 0 and 1. Implicit ratings are collected in the form of the time that the user views a page, i.e. the *time viewed*, and the fact that it has been accessed. For privacy, efficiency and resource reasons only the average of the time viewed and the explicit ratings and the sum of the accesses is stored.

The system also takes in to account the fact that the quality of a page can change over time. It approximates the average of the time viewed and the explicit ratings - the measures that contribute to a combined quality index - for a certain time window, i.e. in principle only data that arrived within the time window contributes to the calculation of the averages.

The choice of the time window was restricted by two aspects: If the time window is to small, too few ratings might be available to make a reliable prediction about the quality. If the time window is too big, the reaction to changes in quality is very slow. A time window of three days was chosen. The content of some information service pages changes much faster than others, but the ratings then reflect the general quality of the service and not the grade of the page content at the very moment.

3.3 Evaluation

3.3.1 Quality Index

An index that combines the available rating information into a single value, the *quality index*, would be desirable. It should represent the overall utility of this page for the home group. For reasons that will be discussed further below, only the time viewed and the explicit ratings contribute to the quality index and not the number of accesses.

The construction of the quality index is based on simple random sampling (see chapter 2.2.2) of two populations:

- One population is formed by the explicit ratings within the time window that the members of a group would submit for a specific page, if they would access and rate the page. Assuming that the user's decision to submit a rating does not depend on the quality they would submit, the collected explicit ratings can be regarded as a random sample. We also assume that users only issue a rating once within the time window so that selection without replacement applies.

- The second population ideally would consist of the *averaged* times that each users in the group would spend viewing a specific page within the time window, if they would access the page. The random sample would be the average time viewed, averaged for each user, so selection without replacement would apply. This, however, would require a record for every user that accesses this page in order to calculate their individual average of the time viewed which is not done for time and space resource reasons and because it is difficult to identify a user with a rating. Instead the sample will consist of many values that originate from the same user, because he/she accesses the page more than once in the time window. In fact, there is obviously a monotonous relationship between the number of time viewed measurements and the average of these values. According to the statistic of the GroupLens project examining the dependency between the time spent reading a newsgroup article and the user's opinion on it. The longer a user dedicates to a message, the higher its rating. Because users are more likely to return to a page they appreciate within a short time than one they despise, the average time viewed will be normally above the real value. Unfortunately it is not simple to compensate for that fact without suitable statistical data. In this project the error was ignored.

A third population could be formed by the average accesses of each user in the group within a time window, so that the navigational value of pages could be taken into account. But without registering the access for every user and every page, simple random sampling can not be justified and there is no way to improve the data. The number of accesses were not considered as this would render the quality index useless.

The constrution of the quality index consisted of the following steps:

1. In order to combine two measures, they must have the same units. Explicit ratings are percentage values and have no unit whereas the time viewed is an absolute measure of unit seconds. It would be useful if the time viewed could be expressed as value between 0 and 1 so that the quality index is a percentage value. In order to convert the time viewed, a table to show the relationship between the time viewed and the explicit rating is necessary. Unfortunately, no study could be found on this topic. The only data that was available was the graphic from the GroupLens project that shows that there is an almost linear correlation between the time spent reading a news message and the rating that the users submitted . The average time spent reading the messages that were rated best, was approximately 100 seconds. The Web offers much more complex information presentation than a simple newgroup article, but in default of better statistics, this was taken as the basis.

   Thus the relative time viewed $r(t)$ was defined in dependence of the absolute time viewed t as:

   $$r(t) = \frac{min\left(100, \frac{t}{seconds}\right)}{100}$$

   All time measurements over 100 seconds are reduced to 100. This protects the data from being disrupted to much by protracted time viewed measurement that can be produced if users leave the browser open while doing something else.

2. The average is an easy-to-compute measure that reflects the general tendancy within a number of values, is the average value. Please note that the average value is not necessarily close to most common vaules. In fact, for controversial pages the average could be much closer to the least common values {different formulation and graphic}.

   An initial approach to the quality index $q(\overline{v}, \overline{r})$ could be the average of the average explicit ratings $\overline{v}$ and the average relative time viewed $\overline{r}$:

   $$q(\overline{v}, \overline{r}) = \frac{\overline{v} + \overline{r}}{2}$$

   $\overline{v}$ and $\overline{r}$ are random variables, because the composition of the sample is random.

3. However, this does not take into account the fact that $\overline{v}$ and $\overline{r}$ can have different reliability. An average explicit rating that is based on 2 values is not as reliable, i.e. it is more likely to change, than one that is based on 1000. As discussed in chapter 2.2.2, the standard error of the average is a reciprocal measure of reliability for random sampling. Thus a weighting according to the reliability is appropriate:

   $$q(\overline{v}, \overline{r}) = \frac{\frac{1}{\sigma_{\overline{v}}}\overline{v} + \frac{1}{\sigma_{\overline{r}}}\overline{r}}{\frac{1}{\sigma_{\overline{v}}} + \frac{1}{\sigma_{\overline{r}}}}$$

   whereby $\sigma_{\overline{v}}$ is the standard error of the average of the explicit ratings and $\sigma_{\overline{r}}$ the standard error of the average of the relative time viewed. We assume that the size of the sample is small relative to the size of the group, so that

   $$\sigma_{\overline{v}} \approx \frac{\sigma_v}{\sqrt{n_v}}, \ \sigma_{\overline{r}} \approx \frac{\sigma_r}{\sqrt{n_r}}$$

   n_v is the number of explicit ratings and n_r the number of time measures within the time window.

4. In the view of the fact that the explicit ratings are reflecting the user's actual opinion of the page and the time viewed is only approximating it, the contribution of the explicit ratings to the quality index is doubled:

   $$q(\overline{v}, \overline{r}) = a\overline{v} + b\overline{r}$$

   with $a = \frac{\frac{2\sqrt{n_v}}{\sigma_v}}{\frac{2\sqrt{n_v}}{\sigma_v} + \frac{\sqrt{n_r}}{\sigma_r}}$ and $b = \frac{\frac{\sqrt{n_r}}{\sigma_r}}{\frac{2\sqrt{n_v}}{\sigma_v} + \frac{\sqrt{n_r}}{\sigma_r}}$.

A number of assumptions were made during the construction of the quality index. A summary of them is given inthe table below. The stars indicate the probability that the assumptions are met.

assumption	explicit rating	time viewed
simple random sampling applies	***	*
the sample size is small relative to the population	**	**

The quality index is a random variable because it is the sum of two random variables. In order to determine the reliability of the quality index, the standard error has to be calculated:

$$\sigma_q = \sqrt{Var\left(q\right)} = \sqrt{Var\left(a\overline{v} + b\overline{r}\right)}$$

$\overline{v}$and $\overline{r}$ are not independent random variables because v and r are not independent. Thus:

$$\sigma_q = \sqrt{a^2 Var\left(\overline{v}\right) + b^2 Var\left(\overline{r}\right) + abCov\left(\overline{v}, \overline{r}\right)}$$

3.3.2 Data Analysis

For every group and every URL the system provides a number of statistical parameters that can be used for analysis purposes. It is crucial, however, to take a careful look at the method of data collection (see 5.2) in order to be able to interpret them. The table below shows all available parameters.

measure	average explicit rating	average time viewed	accesses	quality index
parameters	variance standard deviation average variance of average standard deviation of avg.	variance standard deviation average variance of average standard deviation of avg.	-	variance standard error

4 Architecture of the System

4.1 Design Goals

The goal of this project was to implement a demonstrate-able system that supports collaborative filtering for the World Wide Web and is easy to use. The idea was to extend a proxy server in order to inject an applet and JavaScript code that collects implicit ratings, provides the users with the opportunity to rate web pages according to their quality and supplies them with guidance in their search for high quality information by evaluating the acquired data in an useful way.

This goal splits into the following sub goals:

- modularity - the system should be modal, so that major parts can be substituted independently.
- inconspicuousness on the client side - the content of a web page should not be changed perceiveably and should behave in the same way as if the browser was connect to a normal proxy. In addition, the user interface should not disturb or even annoy the user of a browser.
- multi user ability - obviously the system must be able to cope with a large number of users although for demonstration purposes it is sufficient if the system can handle tens of users rather than millions.

- platform independence - as some browsers, at least the Netspace browsers, are available for a number of platforms, the client side should be platform independent concerning appearance and functionality. On the server side platform independence is desireable so that it can be easily installed in any environment for demonstration purposes.
- ease of use - the participation in the system should not require a big amount of user effort, such as installation of a new browser, and not demand private information or registration. The user interface should be self explainatory and clear.
- flexibility - the system should cope with the problems that are caused by HTML extension such as CGI, frames, scripts, ...
- security - it should not be easy to submit ratings for a different group or to rate more than once while viewing a particular page.
- privacy - the rating information should be stored anonymously, i.e. ratings should be averaged for each group, and not all stored information should be made public (see chapter 2.1.3)

4.2 Overview

The system consists of four major components:

- The *proxy* parses every demanded HTML document and injects the applet and the JavaScript code at the appropriate positions in the passing document.
- The *client* consists of the JavaScript code, the applet and a window that provides the user interface.
- The *rating server* collects the implicit and explicit ratings from the client, evaluates them and sends rating-derived data on a client's requested.
- The *web server* contains the Java classes of the client. The web server was not part of this project because any would suffice.

For the communication between the client and the rating server a light version of the Remote Method Invocation (RMI) was developed.

4.3 Proxy Extension

An internet proxy is a cache programme. Basically it receives requests for internet data, e.g. for a HTML page, from internet clients such as Web browsers. It first checks whether the requested data is in the cache and not out of date. If that is the case it reads from the cache and sends the information back to the client. If not the request is forwarded to another proxy or directly to the internet and the result is possibly placed in the cache. Proxies play an important role in reducing traffic on the internet and providing quicker internet access to the clients. Usually bigger organisations or companies have such a proxy server.

The aim of the proxy extension was to modify every Web page that is requested over the proxy so that implicit and explicit rating information can be acquired on the client and then be sent to the rating server. The acquisition of the rating data on the client side is done by a combination of JavaScript code and a Java applet. Both have to be inserted (injected) in every web document. Two injections must be placed. The first contains a short Java script that starts tomeasure the time (the time since the beginning of the web page was displayed). The second includes the applet and a longer Java script that communicates with the Java applet. However, not every position in a HTML document is suitable. There are a number of constraints on the positions:

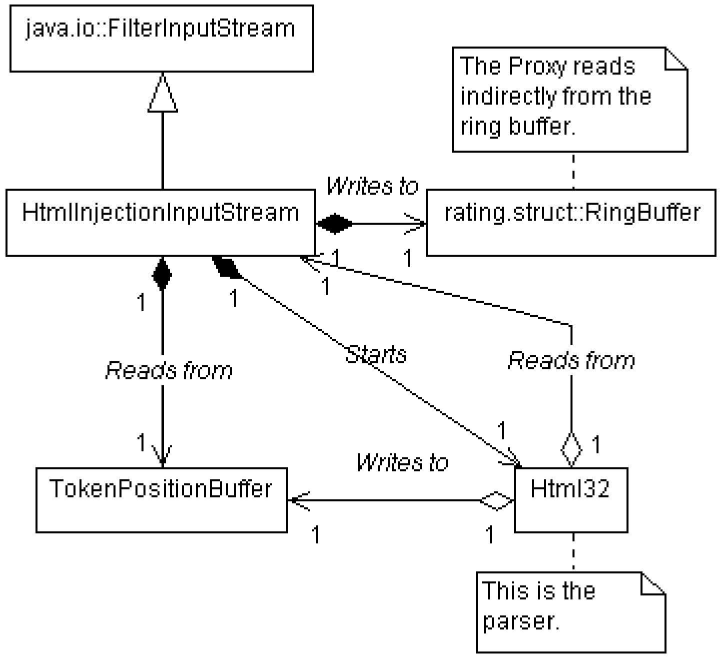
- The injection must be invisible and not destroy any functionality of the web page (see inconspicuousness design goal, chapter 4.1). This implies that injection points can not lie within a HTML tag or within JavaScript code.
- The positions must be unambiguous. Therefore HTML tags that can occur more than once within a document, can not be used as markers.
- The appropriate positions must be discoverable in every web document that is to be rated.
- The injection must be executeable by the browser which means that injection points can lie within comments or the head of the document.
- The injection point for the applet must be at the end of the page because as soon as the browser encounters an applet tag, the Java classes that are required to start the applet are downloaded. This would cause an annoying delay before or while the beginning of the Web page is displayed and would be in opposition to the inconspicuousness design goal.

Positions that satisfy (almost) all contraints are after the `<body>` tag for the first injection and before the `</body>`-end tag for the second one. Only frame set page, i.e. container pages for frame pages, do not have to contain the `<body>` tag, but should be rated as well. For a solution of this problem, please refer to chapter 5.2.4.

In order to understand the structure of the proxy extention, it is important to know that a HTML document is a stream of data, i.e. the HTML document can normally not be read as a whole but step by step in small packages. For the user of a web browser this is visible every time, the page builds up slowly. Streams are an important concept that is supported by most programming languages.

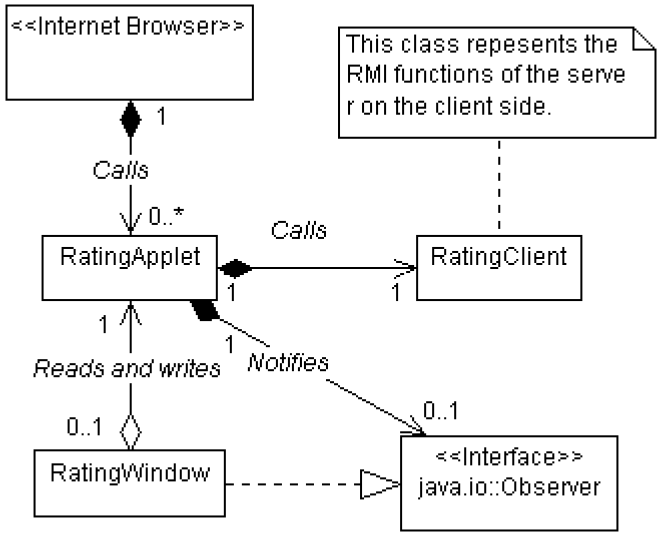
The proxy was modified so that only parsed parts of HTML documents can be read by the browser. When HTML data arrives in the proxy (either from the internet or from its cache) the parser (class `Html32`) reads the data and places a copy of this data in an internal buffer (class `HtmlInjectionInputStream`). The parser notifies the buffer of every recognised part - called token - and marks it in the buffer. The buffer than releases this token to be read by the browser. If the parser finds a injection position, the injection is placed in the buffer. For a more detailed description, please refer to chapter 5.1.

The UML class diagram shows the structure of the proxy extention:



4.4 Client

The task of the client side of the system is to supply the user with the possibility to rate web documents, collect implicit ratings and to dispatch the ratings to the rating server. It also presents information about the current page and provides navigation aids. The UML diagram shows the dependencies among the clients classes:



The client is divided into three layers:

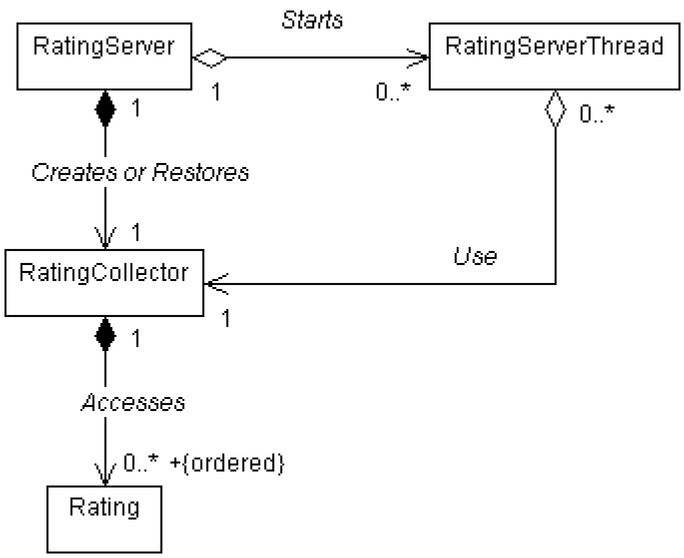
- The data source and storage layer. This layer is represented by the Java script. The Java script collects information about the web page it has been injected in and uploads it to the rating applet. If a page is unloaded, e.g. because the user clicks on a link, the browser stops all applets in the that page. The rating applet has to save its state via JavaScript before it is destroyed, so that it can be restored again. The state information that is written this way, is mainly information about the user interface, for example, the window position and size. The JavaScript also controls the start behaviour of the rating applet. In frame set documents several rating applets can be started. The JavaScript is responsible for allowing exactly one rating applet to display its window.

- The data proxy layer. It is represented by the rating applet (class `RatingApplet`). The applet communicates with the data source and storage layer, the representation layer and the rating server over the rating client (class `RatingClient`). It sends the implicit and explicit ratings to the server and retrieves the data that is demanded by the user interface. It is also responsible for handling this communication efficiently so that unnecessary data traffic is not caused by duplicate data queries. This layer provides the data for the representation layer. If the data has changed that is displayed by the representation layer, it is notified to update itself.

- The representation layer. It is formed by the rating window (class `RatingWindow`). Its purpose is to display the data of the rating applet in a user-friendly way and to allow the user to demand data from it. If the rating applet notifies the window of a change in its data, the window is responsible for updating its display accordingly.

4.5 Server

The rating server's purpose is to collect the implicit and explicit ratings sent by the clients, evaluate them and satisfy the clients' request for rating-derived information, e.g. the average rating of the current page. The structure of the server is based on an example from Sun's Java Tutorial [19]. The UML diagram below shows its class structure.



The server is organised into three major components:

- The main programme (class `RatingServer`) that listens on a port for clients that want to establish a connection. If a client requests a connection, the server creates a thread (class `RatingServerThread`) that deals with this client. The rating server also enables control by the user over itself by starting a user input thread (class `UserInputThread`).
- The rating server threads communicate with the clients. They are responsible for supplying demanded rating information from the rating collector (class `RatingCollector`) and to update it with the new ratings that are submitted by the clients. It is also their duty to ensure an efficient communication with the collector and not to waste computing power by duplicate requests.
- The rating collector accumulates all rating information. It maintains index trees for efficient access. The collector deals with the rating server threads that query it and update the rating records it keeps.

4.6 Remote Method Invocation

The remote method invocation (RMI) enables the communication between client and server. An object on the server side can make its public methods available by instantiating an executer (class `Executer`) for it. The client can call these methods by employing a caller (class `Caller`) to send calls to a RMI server over the network.

In this system a RMI call involves the following steps in the style of the OSI network layers:

1. Application layer (client side). The rating applet calls a method of the rating client (class `RatingClient`). For the applet that is a normal method call without knowing that it in fact calls a remote method over the network.

2. Transformation layer (client side). The rating client transforms the call into a more general form that is suitable for the caller (class `Caller`) and forwards the call to the caller.

3. Presentation layer (client side). The caller encodes the call into a string and sends it away over the network using a package writer (class `PackageWriter`).

4. Transport layer (client side). The package writer associates the call string with a channel and calls network function to send it away.

5. Transport layer (server side). The package reader (class `PackageReader`) listens on the network for incoming calls. When a complete package is received, it forwards the package to the executer (class `Executer`).

6. Presentation/transformation layer (server side). The executer parses the call and uses the Java reflection classes to invoke the corresponding public method of the associated rating server thread (class `RatingServerThread`).

7. Application layer (server side). The server thread executes the method that is invoked by the executer and returns the result. For the server thread this is a normal call. It is unaware of being called by a remote client.

8. Presentation/transformation layer (server side). The executer transforms the result into a string and sends it away using the package writer. If an exception was thrown by the server thread method, the exception is converted and sent. If the rating client indicated that it does not expect to receive a result, the result is discarded.

9. Transport layer (server side). The package writer associates the call string with a channel and calls a network function to send it away.

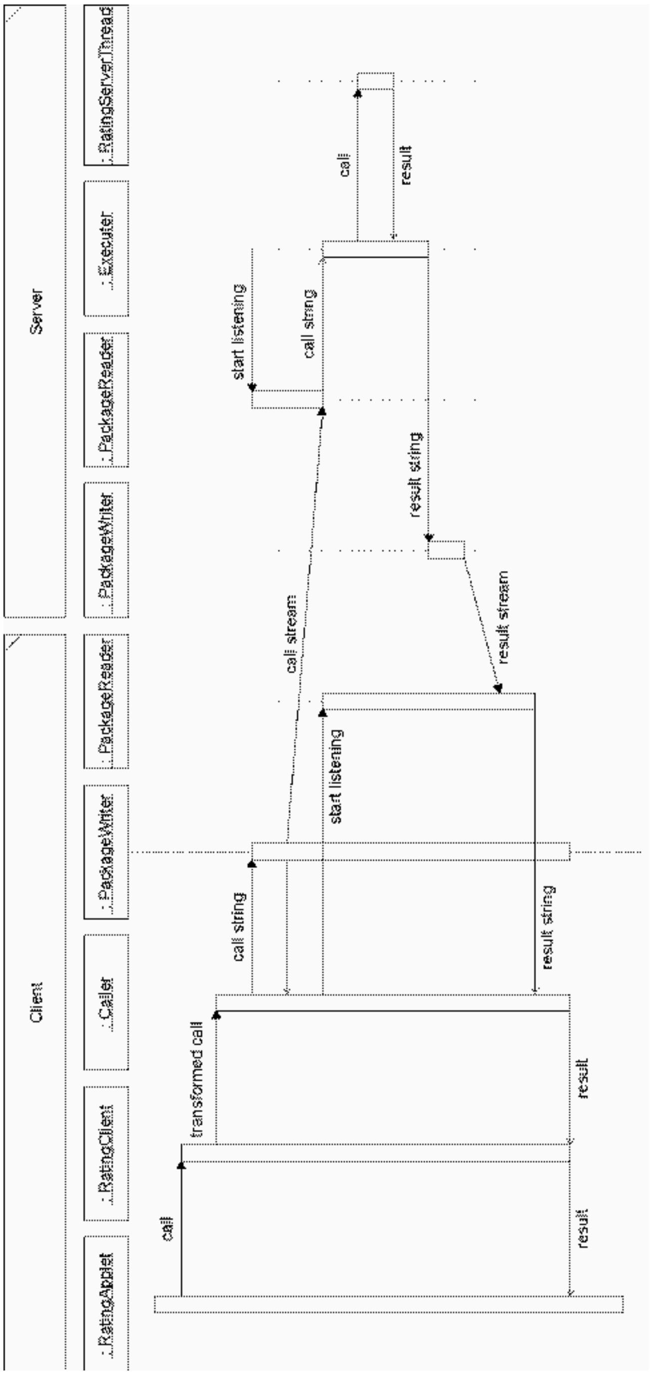
10. Transport layer (client side). The package reader listens on the network for incoming results. When a complete package is received, it forwards the package to the caller.

11. Presentation layer (client side). The caller parses the result string and returns the result object to the rating client. If the string is an exception, it is parsed and the corresponding exception is thrown.

12. Transformation layer (client side). The rating client handles exceptions and forwards the result to the rating applet.

13. Application layer (client side). The rating applet receives the result object and continues the execution.

The following UML sequence diagram shows a typical remote method invocation call that also returns a result.



5 Implementation of the System

As the implementation language for all system parts, Java was chosen. Java is one of the most modern general-purpose programming languages supporting the concepts of concurrency, exception handling, garbage collection, platform independence, object-orientation and interfaces. Java is related to C++ and C, but is strongly typed and not intended for system programming. Java source code is compiled into a machine-independent byte-code representation that can be executed by a Java virtual machine. Browsers like Netscape Navigator and Communicator and the Microsoft Internet Explorer support Java execution in the form of Java applets, that can be embedded in HTML documents. However, the functionality of Java is restricted for security reasons, e.g. reading files from the clients' disks is disabled by default.

Since applets have to be written in Java and it is not possible to embed programmes of a different language, Java was the only choice. To keep communication with the server simple and because of portability reasons (see platform independence design goal, chapter 4.1), the RMI and the server were coded in Java as well. Java is slightly slower than system programming languages like C++, but it is much simpler to write portable code. In addition, it is not necessary to recompile the system when being transferred to a different platform.

5.1 Proxy Extension

The choice of the proxy, that was extended, was limited by the following constraints:

- platform independence - the proxy should be easily installable on any common platform.
- free - the proxy should be public domain, i.e. free for noncommercial use, as no additional money was available for this project.
- availability of source code and documentation - the complete source code and documentation should be available in order to make the necessary modifications to the source code.
- up-to-date - the proxy should be able to cope with all modern HTML version and extentions, so that the usage of the Web is not pruned by the proxies limitations (see inconspicuousness design goal, chapter 4.1).

A proxy that satisfied all constaints was the Jigsaw server that was produced as part of W3C's Architecture domain. The W3C's (World Wide Web Consortium) aim is to support the evolution of the Internet and to ensure interoperability by establishing common standards like the HTTP protocol and the hypertext markup language (HTML). It also supports internet developers by providing specifications and experimental programmes like the Jigsaw server. The Jigsaw server was not designed as a proxy server but as a Web server. However, the new version 2.0 includes a proxy extention. The Jigsaw proxy is entirely programmed in Java and a good deal of documentation is available over the W3C Web page [11]. Unfortunately, only the Jigsaw 2.0 beta3 was available before the end of the project.

5.1.1 Design Goals

The goal was to extend the proxy so that the Java applet and script is inserted in every requested HTML page. The extension was designed to satisfy the following criteria:

- performance - the process of parsing and injecting should not lower the performance of the proxy significantly.
- flexibility - the parser should be able to cope with all common HTML formats and not be sensitive to incorrect HTML code.
- inconspicuousness - the extension should behave like the unmodified proxy concerning data requests. If a HTML page can not be parsed, it should be passed on unmodified (see inconspicuousness design goal, chapter 4.1).

5.1.2 HTML Injection Input Stream

The HTML injection input stream (class `HtmlInjectionInputStream`) is the main class of the proxy extension. The proxy uses the Java Core API stream classes (and descendants) to control the data flow from the HTML source to the browser. The browser basically reads from the data stream provided by the proxy step by step over the network. The injections of the applet and script can only be done if the browser reads from the HTML injection input stream and the injection input stream reads from the original proxy stream. Thus the injection input stream acts like an intermediate piece. As the browser uses basically the functions of the API-class `InputStream` class, the injection input stream is indirectly derived from that class.

Its main components are:

- The ring buffer (class `RingBuffer`, field `ringBuffer`). Its purpose is to store the data that has been read from the original proxy stream but not by the browser yet and to implement its manipulation in a efficient way.
- The token position buffer (class `TokenPositionBuffer`, field `tokenPositionBuffer`). When the parser finds a token, it delivers the position of the token end relative to the beginning of the proxy stream to the token position buffer. At most one token position can be stored in this buffer. The injection stream reads token positions from the buffer to release parts of the ring buffer for reading by the browser. The token position buffer enables synchronised unidirectional communication from the parser to the injection stream. It is an application of the well known producer/consumer problem.

When the HTML injection stream is created, it starts the parser in a separate thread. The parser immediately begins to read from the proxy stream until it finds the end of the first token, and places the position of the token end into the token position buffer (class `TokenPositionBuffer`). When the browser wants to read n bytes from the injection stream by calling its `read` method,the injection input stream tries to get the data using three different levels of sources:

1. It reads released data from the ring buffer, i.e. maximal up to the position of the current token's end. If the available data is less than n bytes, it will try to get the next token position from the token position buffer.
2. If a token position is available, the corresponding part in the token buffer is released for reading and it is continued with level 1. If no token position is available at the time, the injection input stream waits until the parser stores a new item in the token position buffer. The following kinds of token positions receive special treatment:

- injection points (constants `startInjectionTokenKind` and `endInjectionTokenKind`): The respective injection is placed at the beginning of the ring buffer (shifting all other data back), released for reading and it is continued with level 1.
- end of the stream (constant `endTokenKind`): It is indicated to the browser that everything has been read by returning `-1`.
- error while parsing (constant `errorTokenKind`): The data is read directly from the proxy input stream.

3. Data is read from the proxy input stream. If all data has been read from it, `-1` is returned.

The injection input stream also ensures that exceptions thrown while the parser reads from the proxy input stream are caught, delayed and raised again, when the browser tries to read the corresponding data from the injection input stream.

5.1.3 Parser

The purpose of the parser is to find the correct injection points in an HTML page, i.e. the `<body>` and `</body>` tags. The parser was generated using a beta version of the Java compiler compiler (JavaCC) by Sun Microsystems [12]. The grammar of this parser can be found in the file `rating/html/html-3.2.jj`. It is a strongly simplified and corrected version of the HTML 3.2 grammar [13] included as an example for the JavaCC and capable of parsing most HTML 4.0 [14] pages. Unfortunately it was necessary to make some modifications to the generated Java code in order that it could be used for the injection input stream. These changes have to be done manually each time after the parser has been generated and before the Java file is compiled.

When the injection input stream is instantiated, the parser immediately starts reading from the proxy input stream and writes the first token position item to the shared token position buffer. If no error occurred it continues the parsing process and tries to write the next token postion item. If the injection input stream has not read the previous one yet, the parser must wait, otherwise the previous token position would be overwritten. When the injection stream eventually reads the token position, the parser can write the new one and continue parsing and so on.

The parser usually writes normal token positions items, except in these cases:

- If the parser encounters the `<body>` or the `</body>` tag, it indicates the injection point by producing a token position item of the type `startInjectionTokenKind` or `endInjectionTokenKind`.
- If the complete stream has been parsed, the parser delivers a token position item of type `endInjectionTokenKind`.
- If an error occurs, the parser produces a position item of type `errorInjectionTokenKind`.

A detailed description of the parser classes, the grammar and online tutorials are available on the JavaCC Web page [12].

5.1.4 Problems

Generally, the parser and the injection stream will function reliably. Most pages are parseable, even if they contain HTML mistakes that are tolerated by the browser. However, there are some known problems:

- Generally Java scripts that are embedded in a HTML page are a potential problem if the source code of the script is not included in a HTML comment. In that case the parser may get confused as it is not capable of parsing Java script code.
- Occationally there are problems with pages that contain tables. Columns might appear twice disrupting the layout of the page, but not the content. This seems to be a problem of the Jigsaw proxy 2.0 beta3 and hopefully will be corrected in the final release.
- If pages are requested that are not in the proxy, it demands the pages directly from the internet - which may take a relatively long period of time - rather than from a different proxy .

5.1.5 Improvement Suggestions

The improvements below are strongly recommended for commercial use, but are not so relevant for demonstrations.

- The last two problems could be solved by building a virtual proxy that does not cache pages. The virtual proxy could simply connect to an existent proxy. Its only task would be the injection of the applet and the scripts. This would have three further advantages:
  - It is cheaper. Proxies need an considerable amount of disk space for decent hit/fail ratio. In addition, powerful computers are required depending on the amount of users so that the time to look a page up in the cache does not exceed the time it takes to get it directly from the internet. For commercial use, no proxy server has to be licensed for modification.
  - It is quicker. It can takesome time until a proxy server has a representative collection of pages in its cache. So if the rating system is installed in a new environment, the proxy has toobtain most pages directly from the internet at the beginning which causes annoying delays for users.
  - It is easier. The existing proxy can be used without being replaced by a possibly less powerful new proxy.
- A more sophisticated parser that is also capable of parsing JavaScript would solve the problem with embedded Java scripts that are not hidden by a HTML comment.

5.2 Client

As the test environment for the client implementation, the Netscape browsers were used. The Netscape develops the most rife, cross-platform browsers. They are available free for private use.

The implementation of the client, apart from the user interface, was mainly determined by the capabilities, restrictions and lack of documentation of the Netscape browsers.

5.2.1 Security aspects in Java and JavaScript

The security in web browser has been discussed and improved enormously over the last years. It serves mainly two purposes. First, as programmes in the form of Java applet and Java scripts can be embedded in HTML pages and downloaded, it is important that no harm to the users system is done, for example by deleting or overwriting files. Second, information on the user's system is confidential and must be treated as that.

Netscape introduced a number of concepts that enforce these points. In this report, we just want to examine at the project-relevant aspects and summarise them briefly. A detailed description of the security concepts in Netscape browser can be found under [16] and [17].

Java programmes can be embedded in HTML pages using the `<applet>` tag. These Java applets are executed on the Java Virtual Machine of the browser which enforces the following restrictions:

- File access for applets that are loaded via a network is denied completely by default. Not even files can be created on the user's file system.
- The usage of network functions is restricted by the so called *same origin policy*. This policy allows an applet only to connect to the domain it has been downloaded from. HTML documents can be easily modified on the way from the source to the user. The same origin policy tries to ensures that even if the HTML document or its applets are manipulated, information can only be sent back to the source of the applet.
- Applets can not communicate with other applets that are of different origin or on a different page.

A Java script and a Java applet are also subject to the same origin check if the applet is allowed to exploit the functionality of JavaScript by the LiveConnect technology developed by Netscape. Again this check serves the purpose that hacker can not insert applets in an HTML page, read confidential information like history lists using JavaScript and send them to their server.

However, that is exactly what the rating client is supposed to do. Information about an HTML page is to be collected and send to the rating server. Therefore LiveConnect could not be used in this case. After some trials and readings, Idiscovered that public functions of an applet can be called by JavaScript without the same origin check being applied. Thus a communication could be established with the limitation that a Java script has to poll all information from the applet instead of the applet accessing JavaScript. Unfortunately no paper could be found about the kind of attack that should be covered by the same origin policy. So it is unclear whether the client exploits a security leak or not.

5.2.2 Design Goals

The main goal was to provide the user with the opportunity to issue ratings, to collect implicit information and to supply the user with the benefits of a collaborative filtering system. The following properties were consider to be aims in the design process:

- inconspicuousness - the client should not change the working environment of the user drastically (see inconspicuousness design goal, chapter 4.1). It also should not influence the browsing and download behaviour in a way that disturbs or even annoys the user.
- platform independence - the user interface should not take advantage of any platform dependent features and should look at least approximately the same on every platform.
- parsimony - the client should not need extensive computing power or memory and therefore should not be dependent on up-to-date computer systems.
- ease of use - the user interface should be clear, self-explanatory and immediately usable by someone who is familiar with the purpose of the system (see ease of use design goal, chapter 4.1).
- multiple window support - the client should be able to cope with multiple browser windows.
- efficiency - the client has to connect to the server to obtain data, but it should not burden the server with unnecessary duplicate requests.
- slimness - the classes required to be downloaded should be kept as low as possible because the browser has to establish one connection to the proxy for every class. The time required for these connections can add up to a considerable amount.

5.2.3 Java script

“JavaScript is Netscape’s cross-platform, object-based scripting language for client and server applications.” [16] It is a simplified and interpretable version of Java that was developed to extend the functionality of Web documents. A Java script can access a considerable amount of page specific information. In fact, the whole structure of the current document and some browser informationsuch as the history is reflected by JavaScript objects. The JavaScript documentation is available on the Netscape Web site [15].

As indicated above (see chapter 4.4), the injected Java script serves the following purposes:

- coordination of applets - in frame documents more than one rating applet can be started. But the user would be very irritated if a rating window would appear for every single frame page. Instead it is desireable to have one window that displays the average rating of all frame pages and that the user can rate them all together.

When the first applet is loaded, its reference is assigned to the `top.document.ratingApplet` variable that is accessable from all pages. This applet obtains the permission to display its window. In the following chapters, this applet will be called the *principal applet*. The Java scripts in other pages check if the `top.document.ratingApplet` variable is assigned and forbid their applets to show their windows.

- retrieval of information - applets do not have access to page specific information apart from the URL. However, data like the links in the page and the URL of other pages in the document is essential for the applet. A Java script can read these information and upload them to the applet.

  As mentioned above, only one window is displayed for a frame document that acts on behalf of all single frame pages. In a frame document, the single frames are basically loaded at the same time, but due to network delays some pages can be entirely loaded while others are not even displayed in part. The principal applet is the first loaded applet. It is likely that information about other frame pages that is uploaded by its Java script, is incomplete. Therefore, when the other applets are finally loaded, they have to upload the current state of information about the web document, i.e. the URLs of all pages in the document and their links, to the principal applet.

- storage of the applet's state - applets can not store information beyond their life time. When the browser encounters an applet within an HTML page, it loads and starts it immediately and the applet is stopped and destroyed when the page is unloaded. Some user settings, such as the window position and privacy, should be saved at least for the duration of a session. As applets cannot communicate with temporally separated applets, the Java script is employed to store the settings. Fields can be added to JavaScript objects that save data for the live time of the browser. The following fragment of the Java script taken from the injection file `end.html` shows how this is done:

```
if(navigator.windows == null) {
navigator.windows = new Array();
navigator.appletX = new Array();
navigator.appletY = new Array();
navigator.appletWidth = new Array();
navigator.appletHeight = new Array();
navigator.appletCardName = new Array();
navigator.closed = new Array();
navigator.privacy = new Array();
navigator.group = new Array();
}
...
function getData() {
with(navigator) {
appletX[_index] = applet.getX();
appletY[_index] = applet.getY();
```

```
appletWidth[_index] = applet.getWidth();
appletHeight[_index] = applet.getHeight();
appletCardName[_index] = applet.getCardName();
closed[_index] = applet.getClosed();
privacy[_index] = applet.getPrivacy();
group[_index] = applet.getGroup();
}

setTimeout("getData()", 1000);
}
```

Array are used in order to store different settings for different browser windows (see multiple window support design goal, chapter 5.2.2). For the reason explained in the previous chapter, the principal's Java script has to poll the current setting every second.

5.2.4 Information Exchange between Script and Applet

Java was not designed to communicate with JavaScript and vice versa. This is not surprising considering that the two languages have been developed by different companies for different purposes. As the popularity of the internet increased and more and more Web-based applications were programmed, it became apparent that information exchange between the two languages is sometimes necessary. In order to overcome this and other restrictions Netscape developed LiveConnect. But LiveConnect could not be used in our case because of problems with the same origin policy (see chapter 5.2.1). However, JavaScript allows to call the public methods of an applet that is embedded in the same page to be called. As Java cannot send data to JavaScript directly, a script has to poll information by calling the according methods of the applet in regular intervals.

After the principal applet is started, the script starts to poll the state of this applet using the JavaScript method `setTimeout("getData()", 1000)`. It sets a timer that invokes the method `getData` after one second. `getData` calls the functions of the principal applet that return the current state and saves it to browser variables. Finally the timer is set again. The source code of the `getData` function is included in the previous chapter.

5.2.5 Applet

An applet is a Java programme embedded in an HTML page using the `<applet>` tag. The Java class that is stated as the main class in the `class` attribute of the `<applet>` tag has to be derived from the API-class `Applet`. The browser controls the execution of the applet by calling the appropriate methods of the `Applet` class.

The `RatingApplet` class implements the applet and is the main component of the client. In a frame set page, several applets can be started at the same time. All have a minimal task to fulfil, one of them, the principal applet, has extended duties.

All Applets

Each applet receives at least the following data from calls by its script:

- URL (field `url`)- the URL of the page that the applet is embedded in.
- top URL (field `topURL`) - the URL of the top most page in the current web document. If the `url` and `topURL` are the same, the page is not part of a set of frame pages.
- title (field `title`) - the title of its page.
- session identifier (field `sessionID`)- a random number generated by the browser intended to identify the current session to the rating server in order to avoid duplicate ratings. It is not used at the moment.
- privacy-flag (field `privacy`)- a flag indicating whether implicit information is to be sent.
- offset time - the lenth of time between the beginning of the page being loaded and the applet being started.
- ready-flag (field `ready`) - a flag indicating whether all previous information have been uploaded by the script. The applet waits until this flag is set. If for some reason it has not been set after a certain time, the applet reject further uploading and does nothing.

Every applet is responsible for preparing and sending the implicit rating for its own page to the server. The implicit information consists of two parts:

- access - the fact that a page has been accessed. Every time the browser calls the applet's `start` method, it send the title and the URL of the page it is embedded in to the server by calling the `setAccess` method of the server using RMI.
- time viewed - the time in seconds between the loading of the page's beginning and its unloading. It is sent invoking the server's `setTimeViewed` method when the browser indicates the unloading of the page by calling the applet's `stop` method.

Implicit ratings are only send away if the privacy-flag is not set and the ready-flag indicates that the script has completed the uploading of the data.

Principal Applet

The principal applet is the applet that is granted permission to display the rating window. It has the responsibility of managing the data that is necessary to act on behalf of all pages. It provides a cross-page, combined view on the information available from the rating server for each single page and allows one to rate the current content of the browser window as a whole. There is only one principal for every browser window. The script determines the principal by setting the show-flag (field `show`).

Apart from the information that is uploaded to every applet, the principal receives this additional data:

- URLs (field `urls`) - the URLs of all the pages that are loaded or have started to load in the browser window. These URLs are used to retrieve the corresponding quality indexes for all pages and to combine them into a single one. If the user submits an explicit rating, this rating is sent to the server for each URL .
- links (field `links`) - all links in all pages in the browser window. These links are sent to the server to determine the 10 best links, the so called Hot links.

Caching

The applet is the only class that calls server methods. It ensures that data that has been retrieved from the server is not demanded again by caching it (see efficiency design goal, chapter 5.2.2). Each field whose content can be filled with server data, like `hotLinksUrls`, has a corresponding flag, for `hotLinksUrls` this is `hotLinksUrlsCached`, indicating whether the content of the field has been downloaded from the server yet. The caching mechanism assumes that the retrieved data does not change significantly over the life cycle of the applet. There is only one exception. If the user changes the group he/she wants to monitor, the data is likely to be completely wrong. In that case, the cached data is marked invalid by reseting all flag so that the correct data can be loaded.

When the user advances to the next page, new applets are started that do not have access to information stored in the previous applets and therefore get the current data from the server.

Notification

As the rating window mirrors some data of the applet, the rating window has to be notified of any change to update its display. But the rating window also displays information about the status of the applet, e.g. if it is loading, so it should be notified in that case as well.

A standardised solution for updating problems has been proposed by Erich Gamma and is supported by the Java Core API. A class that contains monitored data subclasses `Observable` and the class that monitors implements the `Observer` interface. The observer has to be added to the list of observers that is maintained by the observable. When the data in the observable is changed, the observable notifies all its observers by calling the `update` method of the `Observer` interface so that they can react to the data change.

In the client the applet is the observable and the rating window is the observer. Therefore the rating window implements the Observer interface. Ideally, the applet should contain a separate class that is responsible for the data management and is derived from the `Observable` class. But as this is contrary to the slimness design goal (see chapter 5.2.2), the applet contains a vector of observables and implements the two method `addObserver` and `notifyObservers`:

```
private Vector observers = new Vector();
...
private void addObserver(Observer observer) {
observers.addElement(observer)
}
```

```
private void notifyObservers(Object object) {
if(onlyStatusChanged <= 1) {
for(int index = 0; index < observers.size(); index++)
((Observer) observers.elementAt(index)).update(null, object);
}
}
```

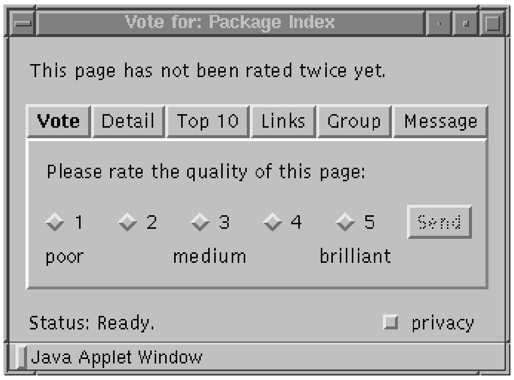
As mentioned above, the rating window has to be notified of a change in data and status. In order to avoid endless loops, the `onlyStatusChanged` counter was introduced. When the rating window is notified, it checks whether `onlyStatusChanged` is greater than `0`. A value greater than `0` indicates that not only the status but also the data has changed. If solely the status has changed, the rating window only updates controls that display status information, i.e. the status bar. `onlyStatusChanged` is a counter so that data can be loaded from and sent to the server involving several steps without the controls being updated before and after every step.

5.2.6 User Interface

The rating window (class `RatingWindow`) hosts all functionality that is relevant for the users interface. It presents the combined quality index for the page, offers the opportunity to submit an explicit rating, request a detailed report, view the 10 best links of the document and the current group, change the group that is monitored and displays messages.

Layout

To preserve a certain clarity, a card layout was chosen (see ease of use design goal, chapter 5.2.2). Card concepts in the user interface are common and should be familiar to the user. As an overall indicator of the document's quality and main information, the combined quality index is always visible. Because the current status should be visible no matter which card is currently displayed, the status bar was placed outside the cards, at bottom of the window as is the custom in most applications. The privacy checkbox is also card-independently accessible since the user should feel in control of the information that is automatically sent and always be able to verify the current security status. All other functionality and information is distributed over the cards.



In order to achieve a similar appearance and avoid disruptions and occlusions on different platforms and display settings (see platform independence design goal, chapter 5.2.2), the components positions and sizes are calculated relative to others rather than based on absolute values. The calculated layout has the additional advantage that the window can be resized with the window area being used in a sensible way. Resizing is an important feature that allows the user to adjust the user interface to meet his/her personal and immediate requirements.

The layout is controlled by the `layout`... and `minimumSize`... method families of the `ratingWindow` class.

Updating

Most aspects of the update process have been already explained in the previous chapter. Seen from the view point of the window, it accesses information of the applet. It the responsibility of the applet to supply the demanded data. If this involves a longer procedure with the possibility of errors, i.e. data exchange with the rating server, it is important to inform the user about the current state and about potential errors. In the window the current status is indicated by the status bar and in the case of an error the user is informed via the message card. The applet notifies the window every time before and after it calls a server method. In contrast to the status bar, the components that display information about the page have only to be updated in the latter case, after the data from the server is loaded. In order to distinguish which components have to be updated, the window's `update` function checks the `onlyStatusChanged` field indirectly by calling the applet's `isLoading` method.

The update process is managed by the `update`... method family of the `ratingWindow` class.

Structure

In order to meet the slimness design goal (see chapter 5.2.2) an elegant object-oriented structure was sacrificed and substituted with a flat functional one. Ideally, every card would have its own class derived from the Java Core API class Panel, each managing the layout and updating of its components. As the functionality of the Hot Links and the Top 10 cards is very similar, they could be derived from a common class that takes their mutuality into account. The diagram below shows a comparison between the ideal and actual class structure.

5.2.7 Problems

- Tests showed that sometimes the rating window is displayed at a wrong position or not at all, i.e. no principal applet could be found by the script. Because of the lack of documentation covering JavaScript execution and the starting and stopping of applets especially when they are loaded from the local browser cache, it is very difficult to analyse this problem. The current implementation assumes that scripts are executed with out interruption, i.e. scripts are not executed concurrently, which was suggested by tests.

- If a HTML includes a signed script, the injected script would not be executed and therefore no rating information could be collected for that page. Signed scripts are part of a set of security improvements in the Netscape Communicator (Version 4.x). However, there are very few pages that contain signed scripts so far. It is not impossible to overcome this security barrier, but many changes in the proxy extension would be required. Please refer to the security chapter in the JavaScript Guide [16] for further information.

- Up to now, the client is only usable under Netscape Communicator (Version 4.x). But compatibility with the Netscape Navigator (Version 3.x) was considered and it should not be difficult to overcome a small problem that still exists.

- The result parser for RMI calls is generated with Sun's JavaCC and requires at least 10 classes. A procedural parser could be intergrated in the Caller class. As the grammar for a result is quite straightforward, this should not be too difficult.

- The flat structure of the user interface makes changes difficult and error-prone. The `ratingWindow` class is big enough to lose the overview. An object-oriented approach would be more elegant, easier to maintain and expand.

- The area of the screen that is occupied by the rating window despite the card layout, is still quite big. Especially if the Top 10 or Hot Links are displayed, the space used is considerable. The size also depends heavily on the standard font size. Sometimes users choose a specific font size because they have eye problems. Therefore a smaller size cannot be chosen automatically.

- The management and display of error messages is not optimal. If an error occurs while loading data, the rating window automatically switches to the message card that contains the error message. This can be confusing for the user. A small pop-up message dialog would probably be better. Unfortunately, the display of message dialogs is not supported by the Java Core API 1.0. With reference to the slimness design goal, the question is if it is possible to implement a message dialog without producing a new class for it.

5.2.8 Improvement Suggestions

- Most problems above are caused by the fact that the functionality of the browser is extended in a way that is not intended, even unwanted by its developers. Security features required repeated work arounds that did not contribute to the clients stability and elegance. A completely different approach is possible since the source code for the Communicator was published as result of a new development concept [18]. The client could be completely integrated into the browser. While this would solve most problems, it has two immense disadvantages:

  - In order to participate in the rating system, the user has to install a new modified browser. This is in opposition to the ease of use design goal (see chapter 4.1).

- The extension has to be integrated into every new version of the browser, tested for every platform and made available for download.

Despite these disadvantages, this option should be seriously considered for a commercial use.

5.3 Server

The choice of the server's implementation language was determined by the following criteria:

- platform independence - the server should run on the most common platforms.
- ease of installation - no recompiling should be required so that it can be installed quickly for demonstration purposes on most machines.
- concurrency support - concurrency concepts such as threads and synchronised variables should be supported by the language or at least by the standard libraries.
- network support - client/server communication should be available through standard libraries.

Java fulfils all of the listed conditions. Java is platform independent, supports threads and synchronisation on a language level and does not require recompiling. The `java.net` package was included in the Java Core API for network communication.

5.3.1 Design Goals

The purpose of the server is to collect rating information sent by the clients, evaluate it and provide access to ratings and derived data. It should have the properties listed below:

- platform independence - see above.
- multiple client support - the server should be able to serve several clients at the same time.
- performance - the server should be implemented efficiently in order to provide quick replies even if the number of clients is high.

5.3.2 Rating

The elementary administrative unit is the `Rating` class that stores the measures for each group/URL pair. In terms of database nomenclature, the group and URL together would be called the primary key for the ratings. Basically the `Rating` class just collects the time a page has been viewed, i.e. time viewed, the number of accesses and the explicit ratings. As it is not possible to store each single rating individually (see performance design goal, previous chapter), the ratings are averaged or accumulated and a number of fields are required for efficient analysis.

Although the average time viewed and the average explicit ratings are approximated for a three day time window (see chapter 3.2), we are first going to ignore the time window, i.e. assume an infinitely big time window.

The table below shows the parameters that are available for each measure.

measure	average explicit rating	average time viewed	accesses	quality index
parameters	variance standard deviation average variance of average standard deviation of avg.	variance standard deviation average variance of average standard deviation of avg.	-	variance standard error

These measures are based on the submitted ratings so far. As not every single rating can be stored, the parameters have to be calculated incrementally. This is performed by the `combine` method of the `rating` class.

Given a parameter P based on n measures $X_1 \ldots X_n$, noted as $P_n(X)$ and a new measure X_{n+1}, we want to be able to calculate $P_{n+1}(X)$exclusively using $P_n$or parameters.

- average or expectation value

$$E_{n+1}(X) = \frac{\sum_{i=1}^{n+1} X_i}{n+1} =$$

$$= \frac{n\left(\sum_{i=1}^{n} X_i\right)}{n(n+1)} + \frac{X_{n+1}}{n+1} = \frac{n\,E_n(X) + X_{n+1}}{n+1}$$

Thus

$$E_{n+1}(X) = \frac{n\,E_n(X) + X_{n+1}}{n+1}$$

The new average $E_{n+1}(X)$ can be computed from the previous average $E_n(X)$, the new measure X_{n+1} and the number of measures n. The `rating` class stores the current average (e.g. field `averageTimeViewed`) and the number of measures (e.g. field `timesTimeViewed`). The combine method is then able to update the average based on the new measure and the stored values.

- variance

$$Var_{n+1}(X) = \frac{1}{n+1}\sum_{i=1}^{n+1}\left(X_i - E_{n+1}(X)\right)^2 =$$

$$= \frac{1}{n+1}\sum_{i=1}^{n+1}\left(X_i^2 - 2X_iE_{n+1}(X) + E_{n+1}(X)^2\right) =$$

$$= \frac{\sum_{i=1}^{n+1} X_i^2}{n+1} - 2\frac{\sum_{i=1}^{n+1} X_i}{n+1} E_{n+1}(X) + E_{n+1}(X)^2 =$$

$$= E_{n+1}\left(X^2\right) - E_{n+1}(X)^2 \left[= \frac{1}{n+1}\left(n\, E_n\left(X^2\right) + X_{n+1}^2\right) - E_{n+1}(X)^2\right]$$

Thus

$$Var_{n+1}(X) = E_{n+1}\left(X^2\right) - E_{n+1}(X)^2$$

This means that for the calcualation of the new variance, the new average value $E_{n+1}(X)$ and the average of the squared measures $E_{n+1}\left(X^2\right)$ is required. The previous point shows how the new average can be obtained. In order to get the average of the squares, the `rating` class stores the sum of the squares (e.g. field `squaredSumTimeViewed`). Together with the number of measures, the average of the squares is computed.

- covariance

$$Cov_{n+1}(X,Y) = E_{n+1}\left(\left(X - E_{n+1}(X)\right)\left(Y - E_{n+1}(Y)\right)\right) =$$

$$= E_{n+1}\left(XY - X E_{n+1}(Y) - Y E_{n+1}(X) + E_{n+1}(X) E_{n+1}(Y)\right) =$$

$$= E_{n+1}(XY) - E_{n+1}(X) E_{n+1}(Y) - E_{n+1}(Y) E_{n+1}(X) + E_{n+1}(X) E_{n+1}(Y) =$$

$$= E_{n+1}(XY) - E_{n+1}(X) E_{n+1}(Y) =$$

with $E_{n+1}(X) = \frac{n\, E_n(X) + X_{n+1}}{n+1}$:

$$= \frac{n\, E_n(XY) + X_{n+1} Y_{n+1}}{n+1} - E_{n+1}(X) E_{n+1}(Y) =$$

$$= \frac{n\, E_n\left(\left(X - E_n(X)\right)\left(Y - E_n(Y)\right)\right) + n\, E_n(X) + n\, E_n(Y) + X_{n+1} Y_{n+1}}{n+1} - E_{n+1}(X) E_{n+1}(Y) =$$

with $Cov_n(X,Y) = E_n\left(\left(X_i - E_n(X)\right)\left(Y_i - E_n(Y)\right)\right)$:

$$= \frac{n}{n+1} Cov_n(X,Y) + \frac{n\, E_n(X) + X_{n+1}}{n+1} + \frac{n\, E_n(Y) + Y_{n+1}}{n+1} - E_{n+1}(X) E_{n+1}(Y) =$$

with $E_{n+1}(X) = \frac{n\, E_n(X) + X_{n+1}}{n+1}$:

$$= \frac{n}{n+1} Cov_n (X,Y) + E_{n+1} (X) + E_{n+1} (Y) - E_{n+1} (X) E_{n+1} (Y)$$

Thus

$$Cov_{n+1} (X,Y) = \frac{n}{n+1} Cov_n (X,Y) + E_{n+1} (X) + E_{n+1} (Y) - E_{n+1} (X) E_{n+1} (Y)$$

In this project the covariance of the average time viewed and the average explicit ratings has to be calculated. The equation above shows that the new covariance of the two measures $Cov_{n+1} (X,Y)$ can be obtained from the new averages of both measures. Therefore the `combine` method of the `rating` class can compute the new covariance using the new averages (fields `averageAverageTimeViewed` and `averageAverageValue`) and the previous covariance (field `covariance`).

- The standard deviation or standard error is the square root of the variance.
- The computation of the average of the averages (fields `averageAverageTimeViewed` and `averageAverageValue`) is analogous to the one of the simple averages.

So far we have assumed that the time window is infinitely big. But as the information is on web pages and therefore the quality can change over time, it is desirable that the parameters are based only on the ratings submitted within the time window rather than on all ratings. As the ratings are not stored individually but accumulatively, it is impossible to determine the exact parameters. However, the parameters can be approximated with the aid of little extra information and few assumptions. Because the computation of the new parameters is directly or indirectly based on averages and the covariance, only these parameters have to be approximated.

First, we want to consider the average. Given the average for a time window $E_{n_1,t_1} (X)$of size T that contains ratings submitted from $t_1 - T$ to t_1 and a new rating X_{n_2,t_2} submitted at time t_2, we can approximate the ratings that fall outside the time window with $n_{1,2} \approx max \left(\frac{t_2 - t_1}{T}, 1 \right) n_1$ assuming that the ratings were acquired at regular intervals. Thus $n_2 \approx \left(1 - \frac{n_{1,2}}{n_1} \right) n_1 + 1$. An additional assumption has to be made, namely that the average $E_{n_1,t_1} (X)$ based on $n_1$ratings is approximately the same as the average $E_{n_2 - 1,t_2} (X)$. So

$$E_{n_2,t_2} (X) \approx \frac{(n_2 - 1) E_{n_1,t_1} (X) + X_{n_2,t_2}}{n_2}, n_2 = \left(1 - max \left(\frac{t_2 - t_1}{T}, 1 \right) \right) n_1 + 1$$

The approximation is the better the smaller $t_2 - t_1$is. Intuitively it seems to be clear that the approximation is stable, i.e. that even if the real average changes -for instance because the page was changed - the approximation always converges towards the real average. A proof for that is beyond the scope of this project.

The approximation of the covariance is basically analogous. But instead of assuming that the average does not change, we presume that the covariance $Cov_{n_1,t_1} (X,Y)$ is roughly the same as $Cov_{n_2 - 1,t_2} (X,Y)$. This is sensible because that the covariance not dependent measure on the absolute values rather than on the their distance from the average. Therefore it is likely to be more stable than the average. Thus the new covariance can be approximated

$$Cov_{n_2,t_2}(X,Y) \approx \frac{n_2}{n_2+1} Cov_{n_1,t_1}(X,Y) + E_{n_2}(X) + E_{n_2}(Y) - E_{n_2}(X) E_{n_2}(Y)$$

Again a proof and an error estimation would be appropriate but can not be given here.

5.3.3 Rating Collector

The central part of the server is the rating collector (class `RatingCollector`) that has the following responsibilities:

- efficient management of rating records - the rating collector maintains two index trees (class `IndexTree`) on the vector (field `ratingVector`) that contains the rating records. One index tree binary sorts groups on the first level and URLs on the second (field `groupIndexTree`), the other index tree (field `urlIndexTree`) vice versa. Index trees are a well-known data structure from database systems for data record access with time complexity $O(log(n))$.

- persistent storage of rating records - the `save` and `restore` method of the rating collector write and load the collector, including the rating record vector, to a file using the Java Core API classes `ObjectOutputStream` and `ObjectInputStream`. The index trees are not stored and but are rebuilt after the rating collector has been loaded in order not to waste disk space.

- supply of rating-derived information - the best 10 links of a group, the Top 10, and the 10 best links of a set of links for the monitored group, the Hot Links, can be queried from the collector. To determine the link lists, the corresponding rating records are checked first. Then the standard error of the quality index subtracted from their quality index is applied as sorting criteria punishing rating records according to their indexes' reliability.

- updating rating records - the server thread calls the `update` method of the collector to submit a rating to the collector. The collector tries to find the rating record with the same URL and group name in the rating vector. If such a record can be found, it calls the `combine` method of the `Rating` class to merge the submitted record with the existing one. Otherwise the submitted rating is added as a new record to the rating vector and the index trees are updated accordingly.

The rating collector is used by the server threads and therefore has to synchronise access, so that the threads can read and write without interference.

5.3.4 Server

The server (class `RatingServer`) contains the main programme. It restores the rating collector first, starts the user thread (class `UserThread`) to activate command input for the user and then listens on the network for incoming client requests. If a client demands a connection, the server creates and starts a new server thread (class `RatingServerThread`) and assigns the thread the responsibility of dealing with it. The server keeps a vector of running threads, so that it can shut them down properly when it is to be stopped. The server can only be shut down by the user entering the `quit` command, otherwise it will run forever. After the server has stopped listening for connection requests and has terminated all running server threads, the rating collector is saved.

5.3.5 Server Threads

The server thread implements the methods that can be called by the client. When the server starts a server thread, it assigns one client to it that is entitled to invoke its public methods using RMI. The services that are offered to the client include submitting implicit and explicit ratings (methods `setAccess`, `setTimeViewed` and `setVote`), querying a detailed report about the current page for the monitored group (method `getDetails`) and the 10 best links (method `getTop10Urls` and `getTop10Titles`).

The life cycle of the thread depends on the life cycle of the applets on the client side. A new thread is created whenever the first rating applet starts. If more than one rating applet starts in a frame set document, the calls are handled over the existing connection. When all rating applets in the current document have stopped, the network connection is closed and the thread terminates. Thus, the server thread is active maximally until the top most HTML page in the browser window is unloaded.

As all rating data is stored and managed by the rating collector, every server thread method has to call the collector to fulfil its task. The server thread's public methods are basically a high level interface for the collector. In order to encumber the collector as little as possible (see performance design goal, chapter 5.3.1), the server thread caches all relevant data.

5.3.6 Problems

The rating server proved to be very stable during a testing phase with two concurrent users. But still, there remain some minor problems:

- Sometimes the shell prompt does not appear after the server has been shut down. Control-C helps.
- The rating collector is saved to a file together with the rating vector using the Java Core API classes `ObjectOutputStream` and `ObjectInputStream`. The problem is that if the RatingCollector or Rating class are recompiled, the file is not readable any more, i.e. all rating records are lost. To solve this problem, the specification and implementation of a data format for the rating vector is necessary.
- Although all parts of the server were carefully designed for concurrent access, a test with several dozens of users is indispensable.

5.3.7 Improvement Suggestions

Despite the relatively small problems, there is enough space for major improvements:

- The solution of the last two problems would become obsolete if the data storage was done by a database. The question is whether a multi-purpose database can compete with this highly specialised system in terms of performance. The disadvantages of a database system are higher costs, higher consumption of resources and probably no portability. However, for commercial use a database system could be appropriate.

- The power of JavaScript could be used to collect more implicit information, e.g. whether the user got to a page by entering the URL manually, by clicking on a bookmark or by following a link. The problem with additional implicit ratings is that they are difficult to integrate into the quality index in a justifiable way.

- It would be helpful if the server could produce a log file with all incoming requests, the time they took to process and their result, for test and analysis purposes.

5.4 Remote Method Invocation

Although a sophisticated RMI is included in the Java Core API 1.1, it was not used in this project for the simple reason of compatibility with the Netscape Navigator 3.x that is delivered with API version 1.0.

As client and server were written in Java, the RMI was implemented in that language as well.

5.4.1 Design Goals

The purpose of the RMI is to enable server method calls by a client over a network with following properties:

- slimness on the client side - as the client side of the RMI must be loaded over the net, it should need as little classes and be as simple as possible.

- performance on the server side - the additional time cost for invoking a method on the server should be low, so that the server is not burdened by the RMI.

5.4.2 Caller

The caller (class `Caller`) is used by the client, in our case by the rating client (class `RatingClient`), and it is the counterpart of the executer (class `Executer`). Its purpose is to convert a method call into a string and send it over the network to the executer. A class can call the invoke method of the caller, passing the parameters and the name of the method that is to be executed on the server. The string representation of the call, the *call string*, is formed according to the following grammar:

```
call: ("&")? methodName "(" parameters ");#" (num)+
methodName: alpha (alphaNum)*
alpha: ("a"-"z" | "A"-"Z" | "_" | "-" | ".")
alphaNum: (alpha | num)
```

```
parameters: (parameter ("," parameter)*)?

parameter: arrayParameter | singleParameter

arrayParameter: type ":{" (value ("," value)*)? "}"

value: (char | string | nonDelimiter) singleValue: type ":"

value type: (num)+

num: ("0"-"9")

char: ((~("’" | "\\" | "\n" | "\r")) | ("\\" ( ["n" | "t" | "b"
| "r" | "f" | "\\" | "’" | "\"") | "0"-"7" ("0"-"7")?| "0"-"3" "0"-"7"
"0"-"7"))) "’"

string: ((~("\"" | "\\" | "\n" | "\r")) | ("\\"(("n" | "t" | "b"
| "r" | "f" | "\\" | "’" | "\"") | "0"-"7" ("0"-"7")? | "0"-"3"
"0"-"7" "0"-"7")))* "\""

nonDelimiter: (~("," | "{" | "}" | ")"))*
```

(For simplicity whitespaces are ignored.)

The complete parser specification can be found in the file `rating/rmi/call/call.jjt`. For example, `askUser(9:"Do you not want to format your hard drive? (yes/no)");#1` would call the `askUser` method of a server object. The call string can have a `&` prefix indicating that the caller does not wait for a result. In this case the executer discards the result of the call to save bandwidth, but the caller cannot be sure that the call actually has been executed. The number at the end of the call string identifies the call. A result returned by the server contains this number so that the caller can map the results to the calls. Up to now the call number is not used because calls are strictly sequential. The `packParameter` method is used to convert every parameter into a string. Currently only the primitive type classes `Boolean`, `Integer`, `Long`, `Float`, `Double`, `Character`, `String` and arrays of these types are supported. Each parameter is preceded by a number that corresponds with the type of the parameter. It is used to recover the object in the executer. The caller sends the call string over the network using the package writer and blocks until the result can be read as a string from the package reader. The caller only blocks if the client expects a result. The result string has the following syntax:

```
result: (arrayResult | singleResult | exception) "#" (num)+

arrayResult: type ":{" (value ("," value)*)? "}"

singleResult: type ":" value

exception: "!" exceptionClass ":" message

exceptionClass: alpha (alphaNum)*

message: string
```

(For simplicity, whitespaces are ignored.)

An example would be: `1:true#1`. The caller uses the result parser that has been generated from the file `rating/rmi/result/result.jjt` to obtain the parse tree. If the call was successful, i.e. the result is not an exception, a result object is instantiated by the `getObject` method and returned to the client as the result of the remote call. Otherwise a caller exception (class `CallerException`) is created that contains the returned exception object and then is thrown. The conversion from the string to the exception object is performed by the `getException` method. Exceptions are also thrown by the caller to report errors in the communication with the server and unsupported parameter types. After the client received the result or caught the exception, it proceeds with the execution.

5.4.3 Executer

The executer belongs to the server side. When an executer is instantiated, the constructor demands the object that is to be made available for remote invocation, i.e. the *remote server object*. In this project the server threads (class `RatingServerThread`) are passed on to the executer so that the client can call their methods. For security reasons, only the public methods of the remote server object can be called over the network. The executer waits for a call to be read from the package reader (class `PackageReader`) and then determines the parse tree of the call using the call parser (class `Call`). The grammar of the parser has been outlined in the previous chapter. The parameters of the call are converted from the string representation into the corresponding object by the `getObject` method. The `getObject` method of the executer supports the same object types resp. classes as the caller. Under usage of the Java reflection class `Method`, the respective method of the remote server object is invoked. If the method throws an exception, it is caught and converted into a string that is interpretable by the result parser. Otherwise, if the call was successful, the `packResult` method transforms the result into a parseable string. The result string is sent away using the package writer. Exception cannot only be raised within the remote server object method, but also before and after the call. For example, if the requested method does not exist or is not declared `public`, the method cannot be invoked and the Java reflection class throws an exception. If possible, exceptions that occur within the executer are send to the caller, so that it is informed about the cause of an error.

5.4.4 Problems

The RMI turned out to be reliable and relatively fast. Two problems still remain:

- The result parser was generated with Sun's JavaCC and requires several classes. So the slimness design goal could not be achieved. But as the grammar of the result parser is relatively simple, it should not cause to much trouble to write a functional parser with significantly less classes.
- During the tests, a phenomenon occurred from time to time: The call parser produced a wrong parse tree. The consequence is that the remote method cannot be invoked. As the used JavaCC parser generator is a beta version, there is the hope that this error will be corrected in the final release.

6 Conclusions and Further Work

This project showed that it is technically possible to supply users of the World Wide Web with a collaborative filtering system. A client/server architecture was built to support a group concept with the opportunity of limited social interaction in the Web environment. Collection and evaluation of implicit and explicit ratings was enabled and their benefits were made available to the users. The overall goal to build a demonstratable and testable system could be achieved despite technical difficulties and financial limitations. Especially security restrictions of modern browsers caused trouble and threaten further developments. To determine the system's abilities and helpfulness, tests with a large amount of users and over a couple of weeks are essential.

In principle, the commercial use is possible, but the described changes and adjustments would then become necessary. A marketing strategy could be based on a free distribution of the rating system and a fee that companies would have to pay to enable rating for their pages so that Web users find their way quicker to the companies home page.

The evaluation of the rating data turned out to be a surprisingly complicated problem. Implicit ratings were evaluated based on the GroupLens study examining the relationship between the time spent reading a message and the submitted rating. For reasons discussed above, it is questionable whether the GroupLens study can be applied in the domain of the World Wide Web. Therefore it is necessary to carry out an equivalent study for the Web taking into account the diversity in the structure and the content of Web pages.

For the evaluation of the explicit ratings, simple random sampling showed to be an appropriate mathematical model. However, for the accumulative evaluation of implicit ratings a modified or more sophisticated model is requisite to compensate for the facts that implicit ratings are likely to be submitted more than once within the time window and that there is a dependency between the number of submissions and the their values (see chapter 3.3.1).

Summarising, it is important to set the rating system on a firm mathematical foundation, including a model for the evaluation of implicit ratings, that is underpinned by appropriate statistical data.

7 Acknowledgements

I would like to thank especially God for giving me endurance for this project and keeping me healthy. I would like to thank Rob Procter (supervisor), whose advice, time and support were essential to this project and Prof. Martin Wirsing as well for making this project possible. I would also like to thank Samantha Osmer (user support office) for providing and maintaining a working development environment, Stephen Kitt (student) for giving me some technical hints, James Smith (statistics lecturer) for taking the time for discussion and Rachel Dear (student) for correcting this report.

8 References

References

[1] Rob Procter and Andy McKinlay, 'Social Affordances and Implicit Ratings for Social Filtering on the Web', in Kovacs, L. et al. (Eds.) Proceedings of the Fifth DELOS Workshop on Filtering and Collaborative

Filtering, Budapest, Hungary, November 10th-12th, 1997. ERCIM Press, http://www.dcs.ed.ac.uk/home/rnp/Papers/delos-97/final.ps.gz

[2] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, John Riedl, 'Applying Collaborative Filtering to Usenet News', Communictions of the ACM, March, 1997, p. 77-87

[3] http://www.cs.umn.edu/Research/GroupLens/grouplens.html

[4] Alan Wexelblat, 'Community-Based Ratings for the Net', http://wex.www.media.mit.edu/people/wex/rate-proposal-head.html

[5] John A. Rice, 'Mathematical Statistics and Data Analysis', chapter 7, International Thomson Publishing, 1995

[6] Kent Wittenburg, Duco Das, Will Hill, Larry Stead, 'Group Asynchronous Browsing on the World Wide Web', in Proceedings of the Fourth International World Wide Web Conference, Boston, MA

[7] J. Rucker, M. Polanco, 'Personalized Naviagtion for the Web', Communications of the ACM, March, 1996, p. 73-75

[8] C. Lueg, 'Social Filtering and Social Reality', in proceddings of the Delos Workshop on Collaborative Filtering, Budapest, November, 1997

[9] Linda Tauscher and Saul Greenberg, 'Revisitation Patterns in the World Wide Web Navigation' in Proceedings of the CHI'97, ACM Press, Atlanta, March, 1997

[10] Thomas Kraetschmer, 'Collaborative Browsing', University of Edinburgh, June, 1998

[11] World Wide Web Consortium, 'Jigsaw' http://www.w3.org/Jigsaw/, 1998

[12] Sun Microsystems, 'Java Compiler Compiler - The Java Parser Generator', http://www.suntest.com/JavaCC/, April, 1998

[13] World Wide Web Consortium, 'HTML 3.2 Reference Specification', http://www.w3.org/TR/REC-html32.html, January, 1997

[14] World Wide Web Consortium, 'HTML 4.0 Specification', http://www.w3.org/TR/REC-html40/, April, 1998

[15] Netspace Corporation, ' JavaScript Documentation', http://developer.netscape.com/docs/manuals/index.html?content=javascript.html, 1998

[16] Netscpae Corporation, 'JavaScript Guide', http://developer.netscape.com/docs/manuals/communicator/jsguide4/index.htm, 1998

[17] Netscape Corporation, 'Java Documentation', http://developer.netscape.com/docs/manuals/index.html?content=java.html, 1998

[18] Netscape Corporation, 'Communicator Source Code' http://developer.netscape.com/source/index.html?cp=dev09tsce, 1998

[19] Sun Microsystems, 'The Java Tutorial - A practical guide for programmers' http://java.sun.com/docs/books/tutorial/index.html, 1998

9 Attachments

- The source code documentation is available online at http://www.dcs.ed.ac.uk/~mam/project/doc/
- The source code can be found at http://www.dcs.ed.ac.uk/~mam/project/src/