Extending the Wikipedia Recommender System
Assessing Expertise of Recommenders

Thomas Lefevre

Kongens Lyngby 2009
IMM-MSc-2009-XX
The Wikipedia is a web-based encyclopedia, written and edited collaboratively by volunteers. Over the last couple of years the Wikipedia has surged in popularity, it is by now the de facto place on the internet to look up information on a wide array of subjects. Being as open as Wikipedia is, for new contributors, has a few pitfalls which Wikipedia also openly admits. There has been much dispute about the quality of the work these editors do, Wikipedia’s premise is that over time the articles on Wikipedia will improve and eventually become complete and stable.

The Wikipedia Recommender System was developed to help users determine the credibility of an article, to allow users to submit ratings and subsequently receive predicted ratings on articles. The WRS is a rating-based collaborative filtering system that employs trust metrics. Collaborative filtering is a filtering technique that employs the input from other users. The use of collaborative filtering has seen a surge of popularity in recent years with such services as http://delicious.com and http://digg.com. The problem with applying a collaborative filtering system on Wikipedia is the breadth of information available on it and expressing trust unilaterally in a user, without recognizing this breadth.

The objective of this thesis is to extend the Wikipedia Recommender System in such a way that it can differentiate between users based on their area of expertise. This will help produce meaningful and trustworthy predictions to the user. It is also demonstrated how the proposed trust model works intuitively and how its prediction results differ in a meaningful way from the previous Wikipedia Recommender System. Insert resume here!
This thesis was prepared at Informatics and Mathematical Modelling, at the Technical University of Denmark in partial fulfillment of the requirements for acquiring the M.Sc. degree in engineering.

The project was completed in the period from November 20\textsuperscript{th}, 2008 to May 15\textsuperscript{th}, 2009 under the supervision of Associate Professor Christian Damsgaard Jensen.

Lyngby, May 2009

Thomas Lefevre
s011531
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>i</td>
</tr>
<tr>
<td>Preface</td>
<td>iii</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.2 The Quality of Wikipedia Articles</td>
<td>2</td>
</tr>
<tr>
<td>1.3 The Wikipedia Recommender System</td>
<td>4</td>
</tr>
<tr>
<td>1.4 Extending the Wikipedia Recommender System</td>
<td>5</td>
</tr>
<tr>
<td>1.5 Structure of this thesis</td>
<td>6</td>
</tr>
<tr>
<td>1.6 Definition of terms</td>
<td>7</td>
</tr>
<tr>
<td><strong>2 State of the Art</strong></td>
<td>9</td>
</tr>
<tr>
<td>2.1 Trust and Trust Management</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Collaborative Filtering with Trust Metrics</td>
<td>18</td>
</tr>
<tr>
<td>2.3 Trust Propagation and Cold Start Problems</td>
<td>19</td>
</tr>
<tr>
<td>2.4 Determining the quality of Wikipedia articles</td>
<td>19</td>
</tr>
<tr>
<td>2.5 Summary</td>
<td>22</td>
</tr>
<tr>
<td><strong>3 Analysis</strong></td>
<td>23</td>
</tr>
<tr>
<td>3.1 Assessing Expertise of Recommenders</td>
<td>24</td>
</tr>
<tr>
<td>3.2 Categorization</td>
<td>24</td>
</tr>
<tr>
<td>3.3 Determining Support in Categories with Trust</td>
<td>32</td>
</tr>
<tr>
<td>3.4 Classification</td>
<td>33</td>
</tr>
<tr>
<td>3.5 Wikipedia as a Repository</td>
<td>36</td>
</tr>
<tr>
<td>3.6 Trust Propagation in Decentralized Recommender Systems</td>
<td>38</td>
</tr>
<tr>
<td>3.7 Summary</td>
<td>39</td>
</tr>
</tbody>
</table>
4 Design
  4.1 The Scone Proxy ........................................... 41
  4.2 The WRS Plugin ............................................ 43
  4.3 Summary .................................................. 48

5 Implementation
  5.1 Plugin Module ............................................. 49
  5.2 Page Module ............................................... 54
  5.3 Rating Module ............................................. 57
  5.4 Trust Module .............................................. 60
  5.5 Security ................................................... 63
  5.6 Summary ................................................... 64

6 Evaluation
  6.1 Accuracy of the WRS ...................................... 67
  6.2 Overhead imposed by the WRS .............................. 70
  6.3 Limitations of the WRS .................................... 72
  6.4 Functional Testing ........................................ 72
  6.5 Summary ................................................... 72

7 Conclusion
  7.1 Decentralized Recommender Systems ....................... 75
  7.2 The Wikipedia Recommender System ...................... 76
  7.3 Future Research ........................................... 77
  7.4 Future Work ............................................... 77

A Appendix
  A.1 Code ....................................................... 79
List of Tables

1.1 Number of editors listed by number of edits. ............................................. 2

2.1 Properties of the WRS’ trust evolution function ........................................ 13

2.2 Properties that the WRS does not have ..................................................... 14

3.1 The four possible interactions when adding categories as a choice. ................. 27

3.2 Interaction table with weights insterted. ..................................................... 32

3.3 A selection of Featured Articles and their size in bytes ............................... 37

4.1 The functionality of the base components of Scone ..................................... 42

4.2 The functionality of the modules of the WRS plugin ................................... 44

6.1 The most popular data sets for collaborative filtering accuracy tests .................. 68

6.2 Estimated time consumption of HTML document retrieveal by entity. .............. 71

6.3 Black box test results. ........................................................................ 73
List of Figures

2.1 Jonker and Treur’s simple trust model . . . . . . . . . . . . . . . 11
2.2 Examples of how \( n \) alters the curvature of the curve . . . . . . 15
2.3 The trust evolution function with \( n = 2 \) . . . . . . . . . . . . . . 16
2.4 The unbounded trust evolution function with \( n = 2 \) . . . . . . . 17

3.1 The distribution of categories on Wikipedia . . . . . . . . . . . . 36

4.1 The Scone framework, showing the four base components. . . . . 42
4.2 Flowchart for the HTML document in the WRS Plugin. . . . . . . 45

5.1 Class diagram of the WRS. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65

6.1 A sequence of actions and their effects demonstrated before and after . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
6.2 Page loads compared, top section loading the WRS and 100 rat- ings and the bottom section without the WRS . . . . . . . . . . . . . 70
Chapter 1

Introduction

1.1 Introduction

The Wikipedia is a web-based encyclopedia, written and edited collaboratively by volunteers[29]. Over the last couple of years the Wikipedia has surged in popularity, it is by now the de facto place on the internet to look up information on a wide array of subjects. Wikipedia prides itself on its openness to contributions from everyone. There are no default restrictions to editing articles on Wikipedia, however it is required that you sign up for an account if you wish to create new articles. Some matured articles may be locked down so that editing them requires that you register a user first. Registering a user is free and does not require a valid e-mail address. This makes Wikipedia easily editable, attracting a large user base to contribute to the contents of the Wikipedia. At present, 15th of May 2009, there are 2,871,873 English Wikipedia articles and a total of 9,793,469 users[1]. Assessing the amount of actual editors of the Wikipedia is difficult, these numbers are tracked but there is no distinction between automated bots and actual people editing the Wikipedia. An example of the statistics computed by Wikipedia is shown in table[1.1] Wikipedia’s official statistic on the number of bots active on their site puts that number at 566[2].

Table 1.1: Number of editors listed by number of edits.

<table>
<thead>
<tr>
<th>Edits</th>
<th>Editors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>321.475</td>
</tr>
<tr>
<td>200</td>
<td>55.568</td>
</tr>
<tr>
<td>500</td>
<td>36.109</td>
</tr>
<tr>
<td>1000</td>
<td>24.895</td>
</tr>
<tr>
<td>4000</td>
<td>8.533</td>
</tr>
</tbody>
</table>

Being as open as Wikipedia is, for new contributors, has a few pitfalls which Wikipedia also openly admits. There has been much dispute about the quality of the work these editors do, Wikipedia’s premise is that over time the articles on Wikipedia will improve and eventually become complete and stable. Wikipedia does insert a caveat to this in their FAQ, saying that: As anyone can edit any article, it is of course possible for biased, out of date or incorrect information to be posted[29]. Robert McHenry, a critic of Wikipedia and former editor-in-chief of Encyclopedia Britannica, describes the Wikipedia process as follows.

1. *Anyone, irrespective of expertise in or even familiarity with the topic, can submit an article and it will be published.*

2. *Anyone, irrespective of expertise in or even familiarity with the topic, can edit that article, and the modifications will stand until further modified.*

3. *Some unspecified quasi-Darwinian process will assure that those writings and editings by contributors of greatest expertise will survive; articles will eventually reach a steady state that corresponds to the highest degree of accuracy.*[16]

The second to third step is what Robert McHenry calls the faith-based step. This then becomes a question of determining the maturity of Wikipedia articles.

### 1.2 The Quality of Wikipedia Articles

There have been many studies over the years comparing Wikipedia articles to the articles of other encyclopedias, most notably Nature’s[6] 2005 study that compared Wikipedia articles to Encyclopedia Britannica. There are two kinds of reviews or comparisons, those that take a scientific approach and those that do not. It’s important to distinguish between these because of their credibility.
A scientific review will be done by an expert, either in reference materials in general or assigned experts will be asked to review articles within his expertise domain, but most importantly articles are reviewed thoroughly by fact and source checking. The sources cited here are only of the scientific kind. The conclusion in Nature was that the accuracy of articles were comparable, however the Wikipedia articles were often “poorly structured”.

In 2008, a paper was published in Reference Services Review \[22\] that concluded: “While Wikipedia provides a wealth of information and is a model for non-proprietary peer-production of reference materials, it does not fare as favorably as do other reference resources under scrutiny for accuracy, comprehensiveness and reliability. Academics may question students’ or colleagues’ use of Wikipedia as a scholarly resource.” In the articles reviewed in that study, Wikipedia has an 80% accuracy rate and a 90.7% verifiability rate. So while Wikipedia is somewhere between comparable and subpar to other reference works.

However, what’s interesting to note is the difference these two studies have versus studies carried about by PC Pro \[21\] and the Denver Post \[4\] which include reviews of Featured Articles. Featured Articles are articles on Wikipedia that have reached a certain degree of completeness, accuracy and remain neutral. These are reviewed and approved by Wikipedia editors. As such, they are some of the best Wikipedia has to offer. This is interesting because the articles that Nature and the paper in Reference Services Review reviewed were not in this category, suggesting that their choices were made more at random. Which is scientifically sound, but the review of the Featured Articles by PC Pro and the Denver Post gives us an insight into where the articles on Wikipedia are headed. PC Pro found that the article on Plate Tectonics “was good for the bare facts, but didn’t read particularly well”. The Denver Post reviewed 5 articles, 4 of which were Featured Articles and the last article, on China, was not. One of the Featured Articles had no negative remarks for poor style, despite this all of the Featured Articles were applauded with such comments as “stick to the science and avoid confusing the reader with political controversy”, “It would have been a great place for a student to begin building his or her knowledge” and “It looks like something that might have been done by a young graduate student, or assistant professor, or two or three.”.

From this we can roughly conclude that the average accuracy of Wikipedia articles is lower than other reference works, but still close enough to be comparable. At the Featured Article level, they are so good that experts in their various fields would hand them over to students for an introduction into that field. So it is obvious that there is big quality difference between an average Wikipedia article and the Featured Articles. This clearly means that there is a need for users of Wikipedia to be able to distinguish where on this spectrum of quality that the currently viewed article is at.
1.3 The Wikipedia Recommender System

The Wikipedia Recommender System\textsuperscript{[13]} (WRS) was developed to help users determine the credibility of an article, to allow users to submit ratings and subsequently receive predicted ratings on articles. The WRS is a rating-based collaborative filtering system that employs trust metrics. Collaborative filtering is a filtering technique that employs the input from other users\textsuperscript{[3]} in determining what is useful information. The use of collaborative filtering has seen a surge of popularity in recent years with such services as \url{http://delicious.com} and \url{http://digg.com}. One shortcoming in traditional collaborative filtering is that it is not personalized. That is, it will not distinguish between what you find useful and what other people find useful\textsuperscript{[7]} and it is not inherently resistant to manipulation. For this purpose, trust metrics has been introduced. It is a method that compares the user with other specific users and then calculates a trust profile. This trust profile adds true user personalization and resistance towards attacks to the collaborative filtering technique. Consider a site such as Digg, on this site recommendations are made through positive recommendations alone and the more recommendations an item get the higher it will be listed. The high position on the list is derived by weighting all recommendations equally, which is tantamount to saying that you agree equally with everyone which is an inherently flawed approach for obvious reasons. Also for these reasons it is not very resistant to attacks, since all it requires to tamper with this system is a distributed effort to “digg” articles and pump their rating. Introducing a trust metric alleviates these issues, by introducing trust each recommendation has an individual weighting allowing for a personalized representation of usefulness. Also attacks are made considerably more difficult, before a successful attack can be launched the attacker needs to do many times the work previously required. A single directed attack in a system with trust metrics first requires that the attacker knows the preferences of the target, so that he can become a trusted source by the target. After carrying out the attack, the attacker will quickly become distrusted. Because of this, trust metrics is viewed as an obvious improvement to the collaborative filtering technique.

The WRS employs trust metrics for these reasons. It is a goal of the system to require no configuration by the user for ease of use and also not to bother the user with concepts or ideas that they might not be familiar to. The system is designed in such a way that it will, based on interactions with other users, determine their trust profile. Gradually this gives a trust profile that is unique to the user, which describes to what degree the user trust others and whether to be cautious or optimistic about future interactions. As the user develops the trust profile, the system will be able to more accurately calculate a rating fitting

\textsuperscript{3}Collaborative filtering can also use other types of input, such as sensor data.
1.4 Extending the Wikipedia Recommender System

that user. When viewing an article, the system will lookup those that have rated the article and match them to the trust profile and then calculate a rating to present to the user. Upon rating an article, the system distributes the rating from the user to a central repository that is accessible by all other users. This allows the system to always get a hold of relevant ratings to update the users own trust profile with. The WRS stores these ratings in the Wikipedia itself, which means that the system does not need anything besides the MediaWiki installation already present. The interface for the WRS is injected into the website via Scone, a programmable proxy which allows for the manipulation of HTML documents before they are shown to the user. Scone is based on IBM’s Web Intermediaries (WBI) technology, now a part of their WebSphere Transcoding Publisher software.

1.4 Extending the Wikipedia Recommender System

The objective of this project is to expand upon the WRS. Currently the WRS will only calculate a single trust value for each user it encounters. Given the breadth of Wikipedia, this is woefully inadequate and implies that users that you encounter on Wikipedia possess equal abilities in all possible categories. For example, a user visits and rates a number of history articles and forms a trust profile based on these interactions. Later he visits an article on plate tectonics and is presented with a rating. This rating could potentially be very misleading, because as mentioned the system currently does not distinguish upon the category to which the article belongs.

The WRS was developed as a proof-of-concept prototype to demonstrate the feasibility of integrating a recommender system into the Wikipedia. It is therefore necessary to perform a number of unintuitive steps to get the implementation of the WRS running on an ordinary PC. Even while running it is necessary to have intimate familiarity with the WRS to properly use it, as currently it has an inordinate amount of restrictions that needs to be bypassed for it to function at all. To properly evaluate the value of trust and the trust models employed by reputation systems such as the WRS, it is essential to gather large sets of data. In one study[18] it was concluded that the size of the data set they were able to accumulate was simply not large enough. It is also noted in other research[11] that much research on trust models has been “proven” with argumentation that appeals to common sense and further notes: “The only way to assess the correctness of a trust model, which claims to describe cognitive processes related to trust, is to perform experiments with human test-subjects.” The required data to make an evaluation of such a system as the WRS and other full fledged trust
models is very large and we feel that the only way to be able to get such large data sets is to make a robust and deployable system. Thus it is the goal of this author to make the WRS usable by ordinary users, in essence deployable to a large group of uninitiated users. This will empower researchers with a usable platform through which they can gather these very large data sets and substantiate the research in trust and trust models.

1.5 Structure of this thesis

This thesis is structured as follows.

Chapter 1 (Introduction) contains an introduction to the credibility issues facing the Wikipedia, the concept behind the WRS and the reasons why trust metrics are important in current collaborative filtering systems. The chapter concludes with defining the objectives of this thesis.

Chapter 2 (State of the Art) reviews the research upon which the theoretical basis of the WRS is composed, such as the idea of trust and the specifics of modeling trust. Collaborative filtering is explained, the associated advantages and disadvantages and attempts at mitigating these disadvantages. Computational efforts in determining article quality and classifying articles on the Wikipedia is explored.

Chapter 3 (Analysis) discusses the problems with applying rating-based collaborative filtering to Wikipedia in a sensible manner and proposes a new way for collaborative filtering to precisely predict ratings. The specifics of the solution is discussed, the way it impacts the current WRS and how to handle this. Problems in the current version of WRS is also discussed and solutions to solving these issues are proposed.

Chapter 4 (Design) describes the design of the WRS and the changes made to accomodate the proposed additions from Chapter 3.

Chapter 5 (Implementation) deals with the specifics on how the additions are implemented into the existing code and also the code is made to conform with the specifications outlined in Chapter 3.

Chapter 6 (Evaluation) discusses evaluations of collaborative filtering systems and the problems faced when doing so.

Chapter 7 (Conclusion and Future Work) discusses the contributions made in
1.6 Definition of terms

In this section a set of terms are defined and used throughout the thesis.

**WRS.** Wikipedia Recommender System (WRS) is the general term for the implementation of the recommender system, that provides the recommendations to the users.

**The trustor.** The trustor is the user that uses the WRS to obtain recommendations about the articles on the Wikipedia.

**The trustee** The term "trustee", "the users" or "the other users" refers to all the other users of the Wikipedia that use the WRS, but not the trustor. The trustor user benefits from the recommendations from the other users.

**Web of Trust.** The Web of Trust (WoT) is the set of Wikipedia users from which the trustee has collected recommendations.

**Trust profile.** Each trustee, that the trustor has in the Web of Trust, has a trust profile. This trust profile holds information on how many interactions the active user has had with the other user, what kind of experience the interactions have been, if the active user trust or distrusts this user and if the active user is optimistic or cautious towards this user. This information in the trust profile calculates to a trust value. The trust profile can also be referred to as the overall collection of trust profiles.

**Trust value.** The trust value is a decimal value between $-1$ and $1$, that describes how much the trustor trusts a trustee. $1$ is complete trust and $-1$ is complete distrust.

**Article.** The term article refers to an article on the Wikipedia. An article is similar to an entry in an ordinary encyclopedia.

**Annotation.** The annotation is the composite of the score and category selection given to an article. It can also refer to the combined string uploaded to the Wikipedia.

**Rating.** The rating is the integer assigned by a trustee or trustor to an article. Can also be referred to as score and in certain contexts is interchangeable with annotation.

Interaction. The trustor and the trustees have interactions with other, based on the recommendations that the both give an article. The similarity of these recommendations define if it is a positive interaction.

Experience. The trustor has an experience with a provided annotation from the WRS. The trustor defines through feedback if the experience is positive or negative.
In this chapter we will review the concepts of trust, its dynamics and how to model it computationally. A simple review of the work by Korsgaard\[13\] is not possible, since we are not just merely drawing upon conclusions made by Korsgaard but building upon that work and thus the foundation for that must be verified.

Furthermore the current research on collaborative filtering and its incorporation of trust metrics will be discussed. We will also take a look at current research on extracting categories from Wikipedia, look at how these are used and their applicability to this thesis.

### 2.1 Trust and Trust Management

Since this project is based on the WRS, there is no need to take a look at the underlying technology which is being used to run it. We will focus on the way trust is being represented, handled and calculated.

Trust in todays internet is one of the relatively unknown revolutions, with Google championing trust via its PageRank technology. PageRank builds a
web of page links, both links from and towards a page, then uses these to compute a PageRank for a page\cite{14}. Essentially, PageRank is a trust metric that describes how much Google trusts the content on the site to be useful. However, PageRank is not very useful for a Recommender System but it does show trust has made a significant impact on how people use the internet today. As said by Jøsang, Keser and Dimitrakos\cite{12}:

\textit{The management of trust is important because if we are able to distrust an entity, then we can be protected from the harm that it might have caused us. The trust management system should be used as “a compass for guiding us safely through a world of uncertainty, risk and moral hazards”}

In the following we will describe the concepts of trust and trust management simultaneously while describing how it has been implemented into the WRS.

\subsection{The Recommender System’s Trust Model}

To represent trust computationally, the WRS uses the work by Stephen Marsh\cite{15} which defines trust as a decimal number between negative and positive one. Negative one is the point of absolute distrust and consequently, plus one is absolute trust. Jonker and Treur\cite{10} introduces concepts like initial trust, trust dynamics and trust evolution and offers a simple trust model.

\subsubsection{Initial Trust}

According to Jonker et al.\cite{10} initial trust describes how to handle initial encounters with trustees, do trustors intially trust or distrust trustees. Either choice can be to a varying degree, such that if initial trust is chosen it can be positioned somewhere between 0.0 and 1.0 and if initial distrust somewhere between -1.0 and 0.0. As previously stated, the goal for the WRS is that it does not require any user configuration before starting. We can again use the work by Marsh\cite{15} and use 0.0 as an initial trust value. Marsh indicates that trust value of 0.0 may be assigned to new trustees either because they are not known, or the trustee was previously distrusted but a positive interaction allocated the trust value to 0.0.

\footnote{Marsh\cite{15} is not cited by Korsgaard\cite{13} to support the decision, however Marsh does support 0.0 as an initial trust value.}
2.1.1.2 Trust Dynamics

Trust dynamics is the description of how trust changes over time and because of time on a general level and the interactions in the system. In early research\cite{10} it is suggested that there are six different types of dynamics. Two of them are blindly negative or positive, such that once either extreme is reached the trust will be locked to that level indefinitely. The four other alternatives are more moderate, where the differences are at what rate of change in trust a negative or positive experience have for the trust in the trustee. For example a high rate of change on positive experiences and a slow rate of change on negative experiences yields the slow negative, fast positive dynamic. This kind of trust dynamic is very rigid, it was discarded because there is no choice that is encompassing enough to avoid imposing configuration requirements upon the user.

A more dynamic approach is suggested in later research by Jonker et al.\cite{11} which is based on the just described research\cite{10} on trust dynamics. The experiment from this research suggests that the order in which negative and positive experiences are encountered is important and that the earlier interactions are less important than current ones. So the trust value is influenced less by interactions in the past. This leaves it open to definition of what the past is. The past can be any interaction that happened further back than $N$ interactions or $N$ time. For the WRS the latter approach was decided upon, such that the result of an interaction is subject to decay with time. The decay is defined such that if
an interaction is less than a month old it is counted in full, between one month and six months an interaction is worth 50\% and finally if a rating is between six months and a year old it is worth 25\%. Furthermore, in the WRS there is a limit to how old a rating can be for it to be included in updating trust values, after that limit the rating is no longer used. This limit is a relationship between how much of the article has been edited since the rating was given compared to the articles current total size, anything above 10\% is past the limit.

It also falls within trust dynamics to determine the value of each interaction. The trust dynamic model in [10] was simple enough that one interaction was enough to shift the trust value one point. In this system there are interactions that go in either a positive or negative direction that affect the trust value in their respective directions. It is interesting to note that the trust dynamic model presented is a state machine and because of that there is no need to keep track of interactions, however moving away from a state machine as a way to represent interactions means that there is a need for an absolute value to describe the sum of interactions. Trust described as being between -1.0 and 1.0 makes the WRS situation incomparable to [10], the granularity is much more fine and as such it is necessary to have a measure for the interactions with equal granularity. It is proposed that the sum of interactions is described by a decimal number in the -1.0 to 1.0 range. Thus we arrive at a cartesian coordinate system where x describes the sum of interactions and y describes the associated trust value derived through the trust evolution function.

However, the value of an interaction still needs to be determined. The choice can be adequately stated as how many positive interactions in a row should it take for the trust value to go from neutral (0.0) to the maximum (1.0). An interaction has been defined in [13] as being worth the absolute value of $\frac{1}{10}$. This gives a total of twenty steps between -1.0 and 1.0, if time decay is ignored.

2.1.1.3 Trust Evolution

The trust evolution function is the core of the trust model, it describes mathematically how trust values are derived and implements the initial trust and trust dynamics. Jonker and Treur [10] made a list of of sixteen possible properties for a trust evolution function; the WRS trust evolution function has eleven of these properties. Note that the properties are not inherently positive or negative and as such it is not a measure of success or failure how many properties a trust evolution function possess. These properties are shown in two tables, table 2.1 lists the properties that the trust evolution function has and table 2.2 the
2.1 Trust and Trust Management

Table 2.1: Properties of the WRS’ trust evolution function

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Future independence</td>
<td>The output of the function only depends on experiences in the past.</td>
</tr>
<tr>
<td>2. Monotonicity</td>
<td>The trust function is monotonic, ( x \leq y \Rightarrow f(x) \leq f(y) ).</td>
</tr>
<tr>
<td>4. Maximal initial trust</td>
<td>There is a maximum on how much trust a trustee can have initially.</td>
</tr>
<tr>
<td>5. Minimal initial trust</td>
<td>There is a minimum on how little trust a trustee can have initially.</td>
</tr>
<tr>
<td>6. Positive trust extension</td>
<td>Trust can progress positively.</td>
</tr>
<tr>
<td>7. Negative trust extension</td>
<td>Trust can progress negatively.</td>
</tr>
<tr>
<td>8. Forgetting about the past</td>
<td>The function will not include values past a certain age.</td>
</tr>
<tr>
<td>9. Degree of trust dropping</td>
<td>Describes after how many negative experiences trust will always be negative.</td>
</tr>
<tr>
<td>10. Degree of trust gaining</td>
<td>Describes after how many positive experiences trust will always be positive.</td>
</tr>
<tr>
<td>11. Positive limit approximation</td>
<td>Describes a condition under which trust will become maximal. (continous metric case)</td>
</tr>
<tr>
<td>13. Negative limit approximation</td>
<td>Describes a condition under which trust will become minimal. (continous metric case)</td>
</tr>
</tbody>
</table>

properties that it does not. Note for the reader: The section in Korsgaard\[13\] describing these properties is wrong, the reader is encouraged to consult Jonker and Treur\[10\].

Jonker and Treur assumes that 1 holds for all trust evolution functions, this is true for the trust evolution function of the WRS as well. As described in the paragraph on initial trust, all trustees start out on the point of neutrality at 0.0 so 4 and 5 holds. As described in the paragraph on trust dynamics, the WRS does not include ratings past a certain age for updating trust values and as such property 8 holds. Properties 9 and 10 hold as well, at a maximum of 10 negative/positive experiences trust will always be negative/positive. Properties not addressed here will be addressed later at an appropriate time in the paper.

As previously described, the WRS implements decay of interactions over time and thus it becomes important when an interaction occurred and property 3 does not hold. Also since trust values and interactions have been chosen to be represented by decimal values, it inherently disqualifies property 12 and 14. Properties not addressed here will be addressed later at an appropriate time in
Table 2.2: Properties that the WRS does not have

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3. Indistinguishable past</td>
<td>Only the interactions themselves count, not the point in time at which they occurred.</td>
</tr>
<tr>
<td>12. Positive limit approximation (discrete case)</td>
<td>Describes a condition under which trust will become maximal.</td>
</tr>
<tr>
<td>14. Negative limit approximation (discrete case)</td>
<td>Describes a condition under which trust will become minimal.</td>
</tr>
<tr>
<td>15. Negative trust fixation of degree n</td>
<td>After n negative experiences, trust will no longer be possible.</td>
</tr>
<tr>
<td>16. Positive trust fixation of degree n</td>
<td>After n positive experiences, distrust will no longer be possible.</td>
</tr>
</tbody>
</table>

Defining the trust evolution function, $x$ is the sum of interactions and $y$ is the calculated trust value. So whether or not the trustor is in a state of trust or distrust with a trustee comes down to the sum of interactions, where distrust is achieved when $-1.0 \leq x < 0.0$ and trust is achieved by $0.0 \leq x \leq 1.0$.

The trust evolution function of the WRS is based on a super ellipse, which has the formula:

\[
\left| \frac{x}{a} \right|^n + \left| \frac{y}{b} \right|^n = 1
\]

The parameters $a$ and $b$ determine the radius of the superellipse. Because the WRS incorporates the work of Stephen Marsh\[15\] the radius of the superellipse is 1, which means that $a = 1$ and $b = 1$ to reflect this. $n$ determines the curvature of the curve, such that it is possible to reflect the disposition of the trustor. The curve starts out at $n = 1$ and is updated in increments of either -0.1 or +0.1 depending on whether or not the trustor becomes more cautious or optimistic relatively. A cautious curve is defined as having $n < 1.0$ and an optimistic curve is defined as having $n > 1.0$. An example of this change in curvature is shown in Fig. 2.2(a) and 2.2(b).

Before curves such as those in 2.2 can be achieved, the formula for the trust value needs to be transformed so it conforms to the requirements set forth by the WRS. The trust evolution function may only produce coordinates in the 1st and 3rd quadrant of the coordinate system. A set of coordinates in the 2nd quadrant
2.1 Trust and Trust Management

(a) A cautious curve having its $n$ value increased.

(b) An optimistic curve having its $n$ value increased.

Figure 2.2: Examples of how $n$ alters the curvature of the curve

would mean that the trustor has an overall negative sum of interactions with a trustee, but a positive trust value. A set of coordinates in the 4th quadrant would mean that the trustor has an overall positive sum of interactions with a trustee, but a negative trust value. This is counterintuitive and should not be permitted. Four functions are needed to describe the possible scenarios in the WRS: an optimistic curve in trust (2.1), a cautious curve in trust (2.2), an optimistic curve in distrust (2.3) and a cautious curve in distrust (2.4).

$$|x - 1|^n + |y|^n = 1 \text{ for } 0.0 \leq x \leq 1.0, \ 0.0 \leq y \leq 1.0, \ n \geq 1.0 \quad (2.1)$$

$$|x|^n + |y - 1|^n = 1 \text{ for } 0.0 \leq x \leq 1.0, \ 0.0 \leq y \leq 1.0, \ n \geq 1.0 \quad (2.2)$$

$$|x|^n + |y + 1|^n = 1 \text{ for } -1.0 \leq x < 0.0, \ -1.0 \leq y < 0.0, \ n \geq 1.0 \quad (2.3)$$

$$|x + 1|^n + |y|^n = 1 \text{ for } -1.0 \leq x < 0.0, \ -1.0 \leq y < 0.0, \ n \geq 1.0 \quad (2.4)$$

These equations, with $n = 2$, give the curves as shown in figure 2.3.

With the above equations and figure 2.3, it is now possible to further determine what properties the WRS trust evolution function has. Looking at the graph we can see that the trust evolution is monotonous, $x \leq y \Rightarrow f(x) \leq f(y)$, so that property 2 holds. Furthermore, from previous descriptions of the trust dynamics and looking at the graph we can see that trust can progress both
positively and negatively and we can thus conclude that property 6 and 7 holds as well. Looking at the trust evolution function and the previous description of how the $x$ value describes the sum of the interactions, we can see that there is no way for the trust evolution function to be fixated on either trust or distrust; thus the trust evolution function does not possess property 15 and 16. To further explore property 11 and 13 it is best to quote Jonker et al.\cite{10}, below is stated the property: \textit{positive limit approximation(continuous metric case)}.

If there exists an $M$ such that for all $m > M$ it holds $e_m$ is maximal, then an $N$ exists such that $te(e, n)$ is maximal for all $n > N$, where $te$ is the trust evolution function, $e$ is an experience sequence and $n$ denotes a number from 0...$n$ such that $te(e_0...e_{n-1})$.

This basically describes that the trust evolution function has a maximum limit, that $te(e, n)$ can be no higher than a certain point. As we have already defined earlier, based on the work of Marsh\cite{15}, our trust value is defined as being in the range $[-1.0; 1.0]$. As also defined earlier, disregarding decay through time which is possible if all interactions has occurred in the last month, each interaction is worth 0.1 and the maximum value of $N$ to satisfy this is then $N = 20 + M$ and as such property 11 holds. The argumentation for \textit{negative limit approximation(continuous metric case)}, property 13, is the same and it holds as well with $N = 20 + M$.

In the work by Korsgaard\cite{13} the boundaries are not set properly which then violates the definition of monotonicity. More importantly, the failure to impose proper boundaries to $x$ means that the sum of interactions can surpass the value of 1.0. The superellipse function, with variables $a = 1.0$ and $b = 1.0$ describes a circle with radius 1.0. By imposing these boundaries and making the adjustments, as seen in 2.1,2.2,2.3 and 2.4 we effectively focus on one fourth of that circle and move it into a position so that the results produced by the
equation gives the shape and results that we are looking for. Unbounded, the trust evolution functions are four transposed half-circles as shown in figure 2.4.

![Figure 2.4: The unbounded trust evolution function with n = 2](image)

These boundaries are not observed with the cautious distrustful and optimistic trustful curve in the work by Korsgaard[13] and as such it is possible to find values of $x$ below -1.0 and above 1.0 respectively. On the cautious distrustful curve, values below $x = -1.0$ will begin to increase the trust value such that $y \rightarrow 0.0$ for $x \rightarrow -2.0$ which is beyond the defined values of the system and unintuitive. Similarly with the optimistic trustful curve, values above $x = 1.0$ will begin to decrease the trust value such that $y \rightarrow 0.0$ for $x \rightarrow 2.0$. The nature of the two curves were not examined beyond these points, since it has already been established how important it is to make the trust evolution function bounded. This observation is not merely theoretical, this problem was found in the WRS implementation as well. The mathematical functions to be inserted in the program is then for optimistic trust (2.5), cautious trust (2.6), optimistic distrust (2.7) and cautious distrust (2.8).

$$y = (-|x - 1|^n + 1)^{1/n} for 0.0 \leq x \leq 1.0, n \geq 1.0$$ (2.5)

$$y = -(|x^n| + 1)^{1/n} + 1 for 0.0 \leq x \leq 1.0, n \geq 1.0$$ (2.6)

$$y = -|x^n| + 1)^{1/n} - 1 for -1.0 \leq x < 0.0, n \geq 1.0$$ (2.7)
\[ y = -(-|(x + 1)^n| + 1)^{1/n} \text{ for } -1.0 \leq x < 0.0, n \geq 1.0 \] (2.8)

At the heart of recommender systems is the research in the algorithms that predict the ratings, such as [17, 24, 28, 5]. However, compatibility between these and the WRS is questionable and needs to be researched further. The compatibility problems arise from the fact that usual collaborative filtering systems employ the rating as the only metric to rate an article while the WRS now also employs a choice in category.

### 2.2 Collaborative Filtering with Trust Metrics

Collaborative filtering is a filtering technique based on the subjective evaluations, generally called annotations, of other readers[7]. I.e. it uses these annotations to find similar users, then uses the ratings of these similar users to predict future ratings. However, only recently [28][2] has trust metrics begun to be considered for collaborative filtering. A problem with traditional collaborative filtering techniques is that it does not distinguish between users, calculated ratings are simply an aggregate rating of all ratings with no form of weighting. By implementing a collaborative filtering system with trust metrics, one achieves such a weighting scheme. This applies the collaborative filtering technique to only a subset of the entire community, allowing for personal calculated ratings. The WRS[13], as described in 2.1.1, implements collaborative filtering with trust metrics for personalized ratings.

Another common problem with similarity-based collaborative filtering is that it requires a centralized service, in order to identify similar users. The computational burden put on this central service quickly leads to scalability problems. Because the WRS is implemented in the MediaWiki environment, it is possible to escape this by storing all the annotations inside the Wikipedia itself. However, because the annotations are stored in the Wikipedia framework and there is no centralized service to handle the calculations it means that the annotations will be available to the public much like with GroupLens[4][24] and Tapestry[7]. It is suggested that if people prefer anonymity there is nothing stopping them from using a nickname, the same applies to the Wikipedia and the WRS. Also because of this decentralized approach, the calculations will be done by the WRS client eliminating the computational scalability problems.

---

4GroupLens is a collaborative filtering system for usenet and the eponymous research lab at University of Minnesota.
2.3 Trust Propagation and Cold Start Problems

Collaborative filtering systems have something called cold start problems, which means that until a certain level of annotations have been made the calculated rating will be imprecise. Consider a collaborative filtering system in which one user creates an annotation for a new item, this is the first and only one for this item. At this point the system only has one annotation to determine a rating from, as more annotations are made on the article this rating will increase in precision. A similar case can be stated for new users in the system that have only interacted with a few users, thus leaving them with a sparsely populated trust profile, leading to the same problems with precision. This is what is considered the cold start problem, a problem that any collaborative filtering system has. It has been suggested by Victor et al. that this problem could be overcome by “guiding users through the connection process by suggesting to connect to key figures”. This however would go against the previously stated goal of imposing configuration requirements on the user and the approach suggested by Victor et al. relies on data that is not available in the WRS. Also, this research is based on collaborative filtering systems that are centralized.

Trust propagation is suggested as another way of alleviating cold start problems. The idea presented is that as users evolve trust profiles, the trustees that become trusted will propagate some of their trust values on other users back to the trustor to expand the trust profile of the trustor. This can quickly help build the trustors web of trust and do it in a reliable way by extending the trust of the trustor.

2.4 Determining the quality of Wikipedia articles

As mentioned in the introduction, a big problem for Wikipedia has been the varied quality of its articles. There has been some research in this area, some trying to computationally assess article quality and others trying to provide tools for the user to better assess the trustworthiness and quality of an article. In this section we will present a few of these approaches.
2.4.1 Computational assessment of article quality

One approach, to make all efforts like these oblivious, is that you can computationally assess the quality of Wikipedia articles. Dondio et al.\[19\] suggests evaluating articles via modeling a number of macro-areas. A full list of macro areas is not available as the work cited by Dondio is not available. However two of these macro-areas are described in the article and they are *User’s distribution* and *Stability*. User distribution describes how the total number of edits are distributed among the users that have edited the article. This information is used to determine the amount of leadership in an article, such that if a low amount of the users that have contributed to an article stands for the vast majority of edits it means that the article has a high amount of leadership. Stability is determined by examining the rate of editing done since time \( t \) and the percentage of text that differs from the current version to the version at time \( t \). However, as mentioned an article can be stable because of a simple lack of activity and thus stability is defined as active articles with good text.

It is difficult to evaluate the work by Dondio et al.\[19\], because it is based on work that is unavailable to this author and thus leaves much of what is described in \[19\] beyond the authors understanding and thus ability to evaluate. There is one shortfall though, that no computational approach can estimate the veracity of an article, unchallenged bias and other computationally evasive errors. However, the work shows promise in assessing article quality but there are certainly hurdles to be overcome before it could be considered the answer to Wikipedia’s credibility issues.

2.4.2 WikiTrust, author reputation

WikiTrust is an extension to the MediaWiki software, that assign trust values to individual words and by proxy of that to its author\[3\]. As described on the WikiTrust homepage\[6\] trust of text is computed in two steps:

- *First, we compute the reputation of each author by analyzing the author’s contributions. When an author makes a contribution that is preserved in subsequent edits, the author gains reputation. When an author makes a contribution that is undone or reverted quickly, the author loses reputation.*

- *The trust value of a new word is proportional to the reputation of its author. When subsequent authors edit the page, words that are left unchanged*
2.4 Determining the quality of Wikipedia articles

This system implicitly states that words that have been present in an article through several edits are to be trusted. As shown by the Nature study, serious mistakes and factual errors do occur in the Wikipedia and may in fact not be noticed by its editors. The system relies on rapid error correction by the Wikipedia editors to correctly reflect the trust in the authors and editors. Again, this leaves us with the conclusion that there is no real computational way to assess the quality of an article. By analyzing the editing pattern of editors, WikiTrust strikes a middle ground where it tries to analyze the human interactions with the material and derive a trust value from that interaction. However, it is still short of an actual reviewing process and the implicit assumptions made by WikiTrust can be quite misleading. The main contribution of WikiTrust is not the way in which it can assign trust in content, but mainly how it can assign distrust to that content. We are not convinced that the way it assigns trust is sufficient and sound enough, however we are convinced that the approach does suffice for assigning distrust.

Another tool, similar to WikiTrust in approach, is the WikiDashboard\[20\] that provides an overview of article authors and author edit history via a timeline at the top of the browsing window. Instead of any trust calculations, these conclusions are left solely to the user. The WikiDashboard exists only to make the information easily accessible so as to assist the user in making a judgment on the article credibility. The conclusion from that experiment was that the credibility scores by users went up significantly with the extra information at hand.

2.4.3 Determining article topic in Wikipedia

To date there has not been done much research into deriving the topic of an article on Wikipedia, Kittur et al.\[1\] is responsible for the only contribution on the field. Granted, the field is somewhat narrow but with the rate at which Wikipedia increases in size and the expected rise in quality of its articles it will only become an increasingly important endeavour.

Kittur et al.\[1\] introduce a mapping technique “that takes advantage of socially-annotated hierarchical categories while dealing with the inconsistencies and noise
inherent in the distributed way that they are generated”. The categories referred to here is the Wikipedia category structure. The nature of the MediaWiki category structure makes it difficult to map categories because of their socially-annotated nature. The result is a structure that can not be classified easily and contains many paradoxes, such as categories being their own grandparents. The problem then becomes how to traverse this hierarchical structure in order to map lower-level nodes to a specified higher-level ontology. For this approach, the semantic-relatedness for Wikipedia categories is determined via edge-counting. This method determines semantic distance by the shortest path between two nodes. When two paths have the same distance to multiple top level nodes, they are weighted equally. The product of this mapping technique is a new, true tree structure for the Wikipedia category structure that is understandable to computers.

With this method for traversing the structure, the leaf categories are associated with one or more top-level categories as described above. Articles are then assigned topics by using these associations to find an aggregate distribution of topics.

2.5 Summary

In this chapter we have detailed how to represent trust computationally, how trust can be modeled with trust dynamics and the trust evolution function. We have enforced the proposed trust evolution function by correctly identifying its bounds. The differences between centralized and decentralized collaborative filtering systems have been explained and specifically how cold start problems effect them. Ways to deal with the cold start problem has been summarized and trust propagation has been discussed to show why it is an important tool to alleviate cold start problems.

Proposed systems and approaches to evaluate article quality and article topic has been reviewed as well.
An encyclopedia usually has an editor that selects experts to write the different entries. Trust in an encyclopedia is therefore a trust in the editor’s ability to identify competent experts in the different topics covered by the encyclopedia. The Wikipedia, however, is collaboratively edited, so there is no editor to act as a trustee. To further complicate things each article has several authors and to precisely express trust in those would necessitate an analysis of the article to first determine each author’s contribution. This makes little sense as the Wikipedia structure, as is, does not lend itself to that kind of activity. Without modifying the MediaWiki software the most obvious solution is to diversify trust in trustees in such a way that allows the trustor to decide the level of expertise of the trustee.

The choice then becomes how expertise is classified. The target for the WRS is the Wikipedia, which is a repository for knowledge. Thus it stands to reason that the conclusion must be that experts should be classified according to the knowledge they possess, their expertise.
3.1 Assessing Expertise of Recommenders

There are two ways to assess recommender expertise, either through evaluation or statement. Either they can be evaluated in what they say or they can make a statement about their own expertise. The latter option is not very relevant in its most direct form, as it would imply that the trustor would implicitly trust the trustees own evaluation of their expertise which goes against the idea behind the WRS. Evaluating a statement by a trustee in order to assess their expertise is more in vein with the WRS’ approach, however by combining both ways of assessing expertise a new approach reveals itself. By allowing the trustee to make a statement about both the rating and the area of expertise within he gives the rating, we can evaluate the rating in the context of his stated area of expertise and thus determine the degree of expertise the trustee possess according to the truster.

3.2 Categorization

Currently the possible interactions in the WRS are limited to two outcomes, based on a comparison between your rating and the ratings of the other users. These can be either agreements or disagreements, which will modify the trust value accordingly. Trying to create an encompassing trust profile for users on a huge online encyclopaia that covers such a broad spectrum of knowledge, as Wikipedia does, with a single metric has some obvious flaws. The current effect is that once you trust a person, you trust that persons opinion in all areas. A persons ability to determine generalities such as an article structure, writing style and so forth does not depend on what the articles’ topic is or what the person’s knowledgeability on that topic is. However, what will change between articles is the degree to which each user can determine what is factually a good article and the ability to determine completeness. The reason for this shift is that a person is not equally well versed in all topics, or more basically, categories. Thus we suggest that to further refine the usefulness of the calculated trust value, the topic or category of the article should be considered when determining the trust value of a user. Using categories as a differentiator effectively creates a trust value for each category for each user, instead of a single one. Each trustee thus has one trust value tied to each category.

With this addition, the WRS will continue to operate in almost the same fashion. What changes is the trust value that is being used and modified, since each user would now have a trust value for each category in the system instead of just a
3.2 Categorization

single one. The category of the article determines the trust value being used and/or modified.

3.2.1 Categorizing Articles

Introducing categories as a differentiator means that the WRS needs a way to determine what the category of an article is. We have determined that there are three ways to do this, they are the following.

- Portals
- Wikipedia Categories
- User defined

In the following sections, each approach to determining article categories will be analyzed.

3.2.1.1 Portals

A portal, or Wikiportal, on Wikipedia serves as an entrypoint to Wikipedia content within a topic area. These vary from a very broad category, such as the History portal or down to a very specific topic such as the Led Zeppelin portal. They’re structured with a strictly top-down approach, such that you can enter the portal and find a selection of articles and sub-portals but you can not, in general, enter an article and find which portal it belongs to. A portal is not an enumeration of articles belonging to the category, or topic, of the portal. This severely limits the use of portals as a means to determine the category of an article, because not even in the top-down approach can you be certain to ascertain what category an article belongs to. As such using Portals to determine which category an article belong to is going to be computationally difficult, perhaps even impossible, and from an overall perspective yield incomplete results.

3.2.1.2 Wikipedia Categories

Categories on Wikipedia are, in a sense, a tree-like structure enumerating all the articles on the Wikipedia. That is, assuming that all articles on Wikipedia have categories. Wikipedia rules state that each article shold belong to at least
one category, thus the assumption is a fairly safe one to make. According to the
Wikipedia entry on categorization:

*Categorization is a feature of Wikipedia’s software, enabling pages to be placed
in categories which can then be used by readers to find sets of articles on
related topics.*

Each article has a set of categories, each of those categories leads up back
through the tree to the root category. Contrary to how portals work, these also
go the same way back down again. There is one issue with these categories
and that is the fact that they are all created and applied by the users of the
Wikipedia, or socially-annotated if you will. While the idea of the categories
were that they were aligned into a tree-like structure, the fact that they are
made by users has mutated what might once have been a tree into something
unidentifiable. For example, one leaf can have several parents and there are even
cyclical occurrences which means that some categories are their own grandparents
as previously described.

If we for a moment consider how to use this system to derive a classification for
an article, we assume that it is a tree-like structure. Thus it would be possible
to select a set of categories near the root of the tree and designate them as main
categories. Otherwise all upward searches would result in all categories arriving
back at a generic container, like Java’s Object class. From each category listing
in an article, we would then start traversing the tree downwards towards the root
and when we arrive at one of the designated main categories a counter would
be incremented for that category. Once this has been done with all categories,
an aggregate classification would present itself. This would give a distribution
between the main categories, from where it would be possible to give the article a
proper classification. Now as previously mentioned, the categories on Wikipedia
is not a tree-like structure anymore at least. Due to the fact that a category itself
can have several parents and in some cases can even be its own grandparent,
the above approach is suddenly many times more complex.

To ease up on the loadtimes, it would be possible to precompute some of this
data. Two obvious data sets to precompute would be a category distribution for
each leaf or a complete category distribution for all articles. The only reasonable
way to handle this real-time would be to precompute the data, in such a way
that the system would never have to do any actual aggregation beyond on the
article-level. However, considering that as of January 2008 there were 11 top
level categories, 276,834 subcategories, 666,537 category hierarchy assignments,
and over 20 million category-page assignments the size of that precomputed data

with the latter option would be enormous and entirely unfeasible. The savings in computation time as well makes it a bad compromise considering the fact that the precomputed data-set would need to be packaged with the WRS itself. Precomputing the category distribution for each leaf is much more reasonable in that the expected data set size is much smaller than the alternate option. Kittur et al.[1] was published April 8th, 2009 and as such was too late to impact this thesis beyond the theoretical level, however the approach used by Kittur et al. mirrors the leaf based topic distribution approach suggested. The approach was abandoned however, seeing as the implementation of algorithms to implement the stated idea would be a project unto itself. There is one caveat with this approach though, the data-set would need to be updated so as to be synced with the state of the Wikipedia.

3.2.1.3 User Defined Categories.

The third option is to allow users to select the article category themselves. This of course avoids many problems, but also creates new ones. For example, the two previous approaches would have made the determination of article category an automatic one and void of user decision. The consequence will be, as previously described, to have one trust value per category per user. However, in this case the category becomes more than a differentiator and instead it becomes a user action. As such the matter becomes much more complex, as now the category is a choice by the user and thus opens up for scrutiny. Not only can users agree or disagree on a rating, it would now also be possible to agree or disagree on the classification of an article. Adding this choice another set of possible interactions between users, which combined gives a total of four sets of interactions as shown in table 3.1. Also, since the categories are user defined and not derived from Wikipedia itself a classification scheme also needs to be determined.

<table>
<thead>
<tr>
<th>Rating</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agree</td>
<td>Disagree</td>
</tr>
<tr>
<td>Agree</td>
<td>Agree</td>
</tr>
<tr>
<td>Disagree</td>
<td>Agree</td>
</tr>
<tr>
<td>Disagree</td>
<td>Disagree</td>
</tr>
</tbody>
</table>

Table 3.1: The four possible interactions when adding categories as a choice.

The category interaction in the current system is defined to be an agreement, thus the two outcomes that has the category interaction listed as agreements are the possible ones in the current WRS. We have chosen this approach for two reasons. Firstly it is the only remaining option, because the other options are either not viable or too time consuming to implement within the scope of this
project. Second of all this approach becomes interesting as the category is no longer just a differentiator, but another choice for the user. As far as the author is aware, this is the first time an annotation in a collaborative filtering system has consisted of more than one choice.

The general operation of the WRS in regards to this new knowledge is as follows. The WRS will download the stored information on the viewed article, category selections will be viewed and balanced by trust values. What this means is that a percentage value will be found for each category, then this value will be modified by the average trust value of the users that chose the category and that should determine what category the majority has chosen. If there is a tie, the selection with the highest average trust value will be chosen, in effect the one with the least actual selections. This would mean that the selection of the few very trusted users will be trusted above the selection by many averagely trusted users. Modification should only happen on the trust value that the user picked, since we are trying to use the trust value to determine a user's expertise in that area.

3.2.2 Categories as a Choice

Introducing categories as a choice by the user extends the possibilities we as designers have with the trust model. When the category is no longer an automatically determined property of an article but a conscious choice, some inferences can be made on that choice and that opens up a lot of possibilities. However, since there are now one trust value per category per user it means that we need to determine in which category to reward or penalize trustees. Consider Alice, an annotation downloaded by her states that she thinks the article on the Wheel is a History article and has given it a score of 3. You think it is a rather good article though, but you recognize that the Wheel article is not about History but Technology & Applied Science. The question is then what category Alice should be penalized in. The goal that was set forth was to try and determine the expertise of trustees, so we regard the only reasonable thing to do as penalizing Alice on what she has said. So we penalize Alice in the History category and not in the Technology & Applied Science category. We could penalize Alice in the Technology & Applied Science category, but Alice never made a statement about Technology & Applied Science.

Another important question is whether or not users can consistently determine the category of an article. Objectively it is not important whether or not it is the correct category as long as a substantial amount of the users agree. If a large enough percentage of users can pick the same category for an article, such that it would be reasonable to suggest that there is a definite consensus on an
article, it would allow conclusions to be drawn on user interactions based on their category choice. For example, user A picks the category for an article that is the one supported by the majority. User B has picked another category, that is only supported by a minority. More than just using this as a differentiator, with a large enough majority supporting a category we can now say that user B was wrong. This is an important distinction.

3.2.2.1 Consistency in Categorization

To look into whether or not users would consistently categorize articles, a survey was conducted where people were given three different articles and asked to pick a category. The survey was carried out in an online community of online gamers which resulted in 19 replies. The three articles from the Wikipedia and their results:

1. Fermat’s Last Theorem - 15.8% History, 84.2% Mathematics
2. Power Drill - 15.8% Information & general works, 84.2% Technology & applied science
3. Perfect Competition - 26.3% Mathematics, 57.9% Social sciences, 10.5% Information & general works, 5.3% Languages

Due to concerns on the appropriateness of performing the survey among such a specific group, it was decided that another round of surveys would be done among a demographically wider choice which resulted in 13 replies. The articles in the first survey were picked to be ambiguously placed between two categories and/or belong to a category that people would perhaps have difficulty identifying. However, it seemed that most of the respondents were correctly able to identify the power drill article as an technology article. The respondents were confused on the perfect competition article, but still the majority of them were able to correctly place the economics article in the social sciences category. To determine whether or not users would correctly identify an easily identifiable article with a high degree of precision, two more articles were introduced in the second survey.

1. Patagonia - 92.3% History and geography, 7.7% Information & general works
2. Fermat’s Last Theorem - 100% Mathematics
3. Power Drill - 92.3% Technology & applied science, 7.7% Information & general works

4. Perfect Competition - 7.7% Information & general works, 7.7% Philosophy & psychology, 53.8% Social sciences, 30.7% Mathematics

5. Punic Wars - 100% History

The results from the second survey are even more convincing than the results from the first. The respondents have with high accuracy categorized articles, except in the case of perfect competition. The reasons for the survey are two-fold, first as mentioned to see if users can consistently categorize articles and secondly to see how well users can apply a categorization scheme. It appears that users are having difficulties correctly identifying perfect competition as a social sciences article, which may be because people are unfamiliar with the fact that economics is indeed a social science and not mathematics. Furthermore, respondents were not asked to fully read each article as they can be rather comprehensive but instead were asked to read each section of the article, determine their category and then aggregate those decisions into a single choice. There is some noise to the survey and even with that in mind, the results are still conclusive: users are able to consistently determine the category of an article. In a later section the categorization scheme will be analyzed.

With the introduction of the category as a choice, there are now two metrics for user accuracy in the WRS. Each metric has their own significant meaning for the interaction, the rating metric determines the perceived quality of writing and article structure while the category rating measures how the user understands the article. It is clear that not each metric has equal importance, but it is difficult to determine the exact importance, and thereof weighting, of each metric. To determine which metric to consider the primary one, a decision needs to be made on what is the most important factor for an article. That it is well written and structured or that the content in it is easily distinguishable as belonging to a certain category. It seems obvious that it is indeed most important that it is well written and structured, therefore we consider the rating metric the primary and the category rating the secondary.

In combination with the primary metric, the secondary metric becomes further complex in that there are two cases in each disagreement interaction. Either the user chose the majority supported category or one of the minority supported categories. In agreement of category this is irrelevant but in the case of disagreement on the category this becomes more complex. This complexity, the category being a choice and the consistency with which users can pick categories suggests that it would be reasonable to use different weightings to the trust calculations for each case. This will be explored in the following sections.
3.2 Categorization

3.2.2.2 Agreement on rating, disagreement on category.

In terms of article quality and writing style, both are in agreement but in terms of article understanding one person has apparently misunderstood something. In the case where the user chooses a minor category and the other user chooses the major category, there is reason to believe that the user misunderstood the article. In the reverse case, where the user chooses the major category and the other users chooses the minor category, it seems that the other user misunderstood the article. However, they both chose close primary metrics which means they are mostly in agreement but not quite as evident by the different choices in secondary metrics. Because the rating metric is primary and in this case there is agreement, the overall interaction is deemed positive but not certain enough that we can credit the other user with a full increase in trust value. As such, a value of $+\frac{1}{2}$ seems a proper weighting to reflect this reasoning.

3.2.2.3 Disagreement on rating and category.

The distinguishment here between whether the user disagrees with the majority or minority on category selection is important. In the case of the user disagreeing with the majority, it is obvious that not only do these two users disagree on the rating of the article but also the other user seems to have misunderstood the area of expertise to which the article belongs. As such, it is reasonable to further penalize the trust value since in this case there is clear evidence that these two users do not trust each other. A weighting of $-\frac{3}{2}$ seem reasonable, this is an increase in half a point from the normal trust value penalty. Should the user agree with the minority, it could be argued that the rating given by the user is done so on a wrongful assumption and might be different if the user had put the article in its proper category. Example from Fermat’s last theorem, should the user have picked history as the category and then proceeded to give it a low rating because the user understood the article in the context of his chosen category, a category that is apparently wrong. The interaction is still overall bad, but there is reasonable doubt that a misunderstanding took place and thus a $-\frac{1}{2}$ weighting is chosen. Table 3.2 shows what the updated interaction table looks like with these weights inserted.

It is important to determine what constitutes a majority, in the mathematical sense it is merely more than 50%, but that seems inadequate when distinguishing between severity of penalties. Given the previous survey done on the consistency of user categorization, it seems that 80% would be a good choice for determining when a majority is present. Both Fermat’s Last Theorem and the Power Drill had 84.2% for Mathematics and Technology & applied science respectively, while
Table 3.2: Interaction table with weights insterted.

<table>
<thead>
<tr>
<th>Weight</th>
<th>Rating</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{1}{2}$</td>
<td>Agree</td>
<td>Disagree</td>
</tr>
<tr>
<td>1</td>
<td>Agree</td>
<td>Agree</td>
</tr>
<tr>
<td>-1</td>
<td>Disagree</td>
<td>Agree</td>
</tr>
<tr>
<td>$-\frac{3}{2}$</td>
<td>Disagree</td>
<td>Disagree with majority</td>
</tr>
<tr>
<td>$-\frac{1}{2}$</td>
<td>Disagree</td>
<td>Disagree with minority</td>
</tr>
</tbody>
</table>

Perfect Competition had its highest percentage as 57.9%.

The current WRS only has the weightings 1 and -1, which means that given an equal spread of interactions, agreement and disagreement, the average trust value should stay the same. However, introducing these new weightings there is a discrepancy since there is now $\frac{1}{2}$, $-\frac{1}{2}$ & $-\frac{3}{2}$. Remember that, given an equal spread of interactions, the $\frac{1}{2}$ will be represented twice as much as the others. This could potentially lead to a destabilization of the trust values; it is difficult to hypothesize on this without actual live data sets.

### 3.3 Determining Support in Categories with Trust

Given that the WRS is a trust-metric enabled recommender system, it is reasonable to suggest that when determining the majority category the trust metric should be used. Of course, we should only ever consider the users that the trustor trusts for determining the majority. Between all the trusted users, there is a different degree of trust from the trustors side and to reflect this we propose to calculate a category’s support as a weighted mean by using the trust values as weights.

$$ c_j = \frac{1}{\sum_{i=0}^{n} t_i} \sum_{i=0}^{n} t_i c_{i,j}, t_i > 0.0 $$

(3.1)

$t_{i,i}$ denotes the trust value of user $i$ for category $j$ and $c_{i,j}$ denotes the choice of category $j$ by user $i$. Not employing trust would be unintuitive and allow scenarios where less trusted users could constitute a majority based solely on their quantity without regard for the quality of their statements.
3.4 Classification

As the user defined categorization approach has been chosen as the way to proceed, it now necessitates a look upon the current classification systems. There are currently only three classification systems in use, many other systems exist but are no longer in widespread use.

- Library of Congress Classification
- Universal Decimal Classification
- Dewey Decimal Classification

Each system will be reviewed for potential use in the WRS. A classification system in this setting has different needs than it has in a library setting. In this setting we are looking for a system that covers the entire spectrum of subjects, in easy to understand categories with little overlap between them.

3.4.1 Library of Congress Classification

The Library of Congress Classification, referred to as the LCC, is developed by one specific library, the Library of Congress. It is in widespread use among research and academic libraries and as such qualifies for consideration. The system contains 21 classes. Classes are added as needed, instead of from overall considerations. This has led to much criticism because of a lack of a sound theoretical basis. The classes are as follows.

- Class A - General Works
- Class B - Philosophy, Psychology, Religion
- Class C - Auxiliary Sciences of History (General)
- Class D - World History (except American History)
- Class E and F - American History
- Class F - Local History of the United States and British, Dutch, French, and Latin America
- Class G - Geography, Anthropology, Recreation
As is evident in this list, it is regionally specific to the US and also specific to other regions. Some unusual sciences have their own categories, such as Military and Naval sciences. At 21 classes, it is a cumbersome system and the US-centric categories means that the system needs restructuring.

### 3.4.2 Universal and Dewey Decimal Classification

The Universal Decimal Classification (UDC) is derived from the Dewey Decimal Classification (DDC), but uses a complex system of additional symbols to indicate special aspects or relationships of a subject. However, this complex system goes beyond what is needed for this and can be ignored. This leaves us with the primary class table for the Universal Decimal Classification.

- 0 - Generalities
- 1 - Philosophy and Psychology
- 2 - Religion and Theology
• 3 - Social Sciences
• 4 - Vacant
• 5 - Natural Sciences
• 6 - Technology
• 7 - The Arts
• 8 - Language, Linguistics and Literature
• 9 - Geography, Biography and History

Compared to the LCC, the UDC, and by extension the DDC, have much fewer classes and do not show any regional influence. The 4 space is vacant and it is unknown to the author why this is so, no information on it could be found. The DDC class table is as follows.

• 0 - Computer science, information, and general works
• 1 - Philosophy and psychology
• 2 - Religion
• 3 - Social sciences
• 4 - Languages
• 5 - Science and Mathematics
• 6 - Technology and applied science
• 7 - Arts and recreation
• 8 - Literature
• 9 - History and geography and biography

When looking at this class table, it is worth to notice that the UDC class table does not have any classification for the computer sciences. There has been some criticism of the DDC that computer science has just been lumped in together with information and general works, but it is nonetheless represented in the table. Also Language (and Linguistics) is in the 4 space, which in the UDC is vacant. Looking at these two tables, the DDC is clearly more descriptive and as noted previously it is deemed undesirable to make changes to these classification systems, they should be taken as is. Modifying these, or creating a
new classification, could evolve to a project in library and information sciences, where special attention could be paid to the results by Kittur et al. as shown in figure 3.1 that describes the presence of each category in the Wikipedia.

![Figure 3.1: The distribution of categories on Wikipedia](image)

Between the UDC and the DDC, the DDC classification is clearly superior. The DDC looks much the same as the UDC, but has a more well defined scope where the UDC could be considered plain lacking. Comparing the DDC to the LCC, the most important difference here is the fact that the DDC is comprised of 10 classes while the LCC is comprised of 21 classes. 21 classes is too much to keep track of, it is bloated beyond our needs by the US-centric classes and such questionable categories as Military Science and Naval Science. The Dewey Decimal Classification system seems much more appropriate in size, kept to ten classes that seems adequate in breadth. The choice then falls on the Dewey Decimal Classification system.

### 3.5 Wikipedia as a Repository

As previously described the WRS stores its annotation data in the articles themselves, by means of the Wiki markup language it was possible to hide this from display on the page itself so that it would only show up at the bottom on the edit page. However the data was still being downloaded on each page visit, just not shown. This caused some controversy among Wikipedians, because it would
impact users regardless if they opted in on the WRS or not. This prompted work to be done on identifying a new location for the annotations.

Users on the Wikipedia are allowed their own userpages, but it is stated that all content in the user pages should be relevant to the Wikipedia project. As the WRS is an effort to provide credibility to the Wikipedia, the annotation data could be considered relevant to the Wikipedia. At the time of writing\textsuperscript{2} no negative mention has been made about the presence of the annotation data. Specifically, the user Recommendations was created as an entry point into the data structure for the annotations. The annotations are stored in articles mirroring the Wikipedia in the user subpages, such that the ratings for an article such as \url{http://en.wikipedia.org/wiki/Hellerup} would be located at \url{http://en.wikipedia.org/wiki/User:Recommendations/Hellerup}. Using the user subpages allows the WRS to be entirely opt-in and never interfere with the affairs of the Wikipedians.

Annotations are estimated to be approximately 160 characters on average, this is a loose estimate based on actual annotations but is precise enough for the purpose of an example. A selection of Featured Articles have been chosen to compare against the size of annotations to give the reader an idea of the overhead required in download size, these can be seen in table 3.3. Featured Articles were chosen because they are at a level of completeness were they are not expected to grow much further and at the same time should be some of the larger articles on the Wikipedia.

It is unsure exactly what an acceptable overhead is in this case, but at a 20% overhead it will be possible to fit in 369 ratings into an Featured Article. Given the speed of current internet connections, a 20% overhead is probably a low estimate as well. The size of the English Wikipedia, only counting current article content as of May 6th 2009 is 4.8 GB in a bz2 archive. According to a test done with bz2 archives, it attains a 5:1 compression rate on pure ASCII text files, which would give a total uncompressed size of English Wikipedia articles

\begin{table}[h]
\centering
\caption{A selection of Featured Articles and their size in bytes}
\begin{tabular}{|l|l|}
\hline
7 World Trade Center & 242.210 bytes \\
Death Valley National Park & 449.365 bytes \\
Fauna of Scotland & 305.797 bytes \\
Krill & 149.169 bytes \\
Malcolm X & 329.657 bytes \\
Average & 295.239,6 bytes \\
\hline
\end{tabular}
\end{table}

\textsuperscript{2}13th of May, 2009
at 24 GB\textsuperscript{3}. Extending the 20% overhead gives the WRS room for 30 million annotations. These numbers are only guesstimates, but they serve to illustrate that the storage space required for the annotations is dwarfed by the size of the Wikipedia itself. The size guessed at above does not even consider the entire Wikipedia, such as the revision history, user pages, talk pages and so forth. The last full dump of the English Wikipedia was 1st of January 2008, which is estimated to take up 2 TB of drive space uncompressed\textsuperscript{4}.

### 3.6 Trust Propagation in Decentralized Recommender Systems

There is an important distinction to be made between the WRS and most other collaborative filtering systems. The WRS is decentralized, such that each user calculates his own recommendations whereas a centralized system contains a single, or common between users, entity responsible for making recommendations. With few exceptions recommender systems have been created around the centralized approach. In centralized approaches the annotations are stored on the centralized server, whereas in decentralized approaches the question of annotation storage becomes much more volatile. As previously mentioned, in the WRS these are stored in the Wikipedia framework. Decentralized approaches have not been as popular as the centralized approaches, however the approach is well recognized for use with sensor networks\textsuperscript{27}. However with the development and popularity of various decentralized computing methods, P2P, Grid and so forth, the decentralized approach is becoming increasingly interesting for recommender systems.

Trust propagation in decentralized networks then becomes quite a different thing, because of the nature of decentralization privacy concerns are introduced. In sensor networks this is not an issue, because sensors do not have a privacy to protect. So for trust propagation in decentralized networks to work, it necessitates information sharing between the various entities. The annotations uploaded do not contain any trust information, they need to be processed by each entity to yield trust information. Because of privacy concerns, this sharing of information is problematic because recommender systems are in general not symmetric in regards to trust. Consider a statement that $A$ trusts $B$ for 0.6, this statement implies nothing about how much $B$ trusts $A$ in an asymmetric system. Thus if $A$ trusts $B$ and wishes to propagate the trust values of $B$ to himself, $B$ needs to trust $A$ first. Because of this requirement of mutual trust,
trust propagation in this way suddenly becomes much less effective. If $A$ and $B$ trust each other enough to share trust values, it is then questionable if it as at all necessary. Remember that trust propagations primary purpose is to combat cold start problems, but if both $A$ and $B$ trust each other their trust profiles must have matured to a level where there is no longer a cold start problem. Ziegler[31], which seems to constitute the entire body of work in research on decentralized recommender systems for human subjects, circumvents the difficulties with trust propagation in decentralized recommender systems by ignoring the privacy issues. Going forward however this is obviously not something that can be ignored in future research.

In centralized systems, trust profiles can be updated continously as annotations are uploaded to the central processing entity. In decentralized systems however, each processing entity needs to check the repository for new annotations. To that end, it is important for each entity to keep a history of what the user has rated, how it is rated and any other necessary information to recalculate the contributions to the trust profile for that item. Using this history it is possible for the decentralized processing entity to check these previous items for new annotations in order to mature the trust profile. A separate task in the decentralized processing entity should keep an eye on these items.

This proposition only brings the speed at which trust profiles mature up to par with centralized approaches because it seeks to reproduce an inherent mechanism of these in decentralized approaches. However it can be considered a trust propagation scheme for decentralized recommender systems.

3.7 Summary

In this chapter we have outlined why it is important that a recommender system targeted at a large information repository such as the Wikipedia employs categories as a way to diversify users in order to determine their area of expertise. A method is proposed to determine users area of expertise by allowing users to select the article category and how this is important as a user choice. Classification schemes has been reviewed for this purpose and the Dewey Decimal Classification scheme was selected.

Lastly trust propagation in decentralized systems were discussed, it was determined that an inherent way of propagating trust in centralized recommender systems were missing in decentralized recommender systems and a method for replacing this was proposed. More importantly it was discovered that as long as privacy is a concern, traditional trust propagation schemes are impossible to
implement in decentralized recommender systems.
This chapter describes the design of the WRS, how it works as a plugin as part of the Scone proxy and the changes made.

4.1 The Scone Proxy

The Scone proxy is a framework that allows developers to quickly develop web enhancements. These enhancements are programmed as plugins to the Scone proxy. The Scone proxy has four base components, as shown in figure 4.1, that each offer a set of tools for the developer. The utility of each component is listed in figure 4.1 while a detailed look at the relationship between the components is seen in figure 4.1. The proxy is based on IBM’s Web Based Intermediary (WBI), which means that largely Scone is based upon WBI.

Many of these components are not necessary for the WRS to function properly, the only component that the WRS makes any use of is the functionality of the Scone Proxy. The WRS does not store any data about the pages visited, nor does it track user actions or employ the robot. Thus we propose that to further enable the WRS as a deployable program, to disable all other running plugins except the WRS plugin so that the user is freed of the obligation of installing
Table 4.1: The functionality of the base components of Scone

<table>
<thead>
<tr>
<th>Component</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proxy</td>
<td>Facilitates the access and manipulation of html documents.</td>
</tr>
<tr>
<td>NetObjects</td>
<td>Stores data relevant to the Scone proxy like URI’s, HTML-documents, links,</td>
</tr>
<tr>
<td>Robot</td>
<td>servers, users and their activities.</td>
</tr>
<tr>
<td>Access Tracking</td>
<td>Web-crawler populating the database with links and documents based on their</td>
</tr>
<tr>
<td></td>
<td>attributes and filtering of these.</td>
</tr>
<tr>
<td></td>
<td>Tracks user actions and generates events off of these.</td>
</tr>
</tbody>
</table>

an SQL database. The need for the SQL database by Scone makes it difficult for any Scone program to truly be deployable, thus we suggest to the authors of Scone that another solution is found such as HSQLDB. HSQLDB is able to execute within the java program itself and thus requires no installation. The author also had severe problems with making Scone able to connect to a MySQL database which further prompted this recommendation.

Essentially, Scone enables developers to intercept and manipulate HTML documents in an unlimited amount of ways with tools such as NetObjects, the robot and access tracking enabling the development of incredibly advanced information systems to enhance the browsing experience. Looking at the examples posted on the Scone website\(^1\) and the lack of Scone plugins outside of the Scone webpage it is clear that developers are not developing these plugins. Given the unique possibilities that Scone gives it is clear that the potential of Scone has not been realized.

\(^1\)http://scone.de/examples.html
4.2 The WRS Plugin

The following sections will provide a clarification on what it is that the plugin needs to do and in what order. The actions the plugin needs to perform should be categorized and based on this grouped together in modules so as to provide an easy overview of the plugin.

4.2.1 Plugin

The purpose of the main WRS module is as a main entry point of the HTML stream, where it will then invoke the necessary utility modules in proper sequence. The WRS module should first check that the HTML stream entering comes from a source of interest. That is, pages outside the Wikipedia.org domain is not interesting and in some cases pages inside the Wikipedia.org domain is of no interest either. If a page is found to be of no interest, it needs to be passed along unaltered to the browser. If the HTML stream is determined to be from a Wikipedia article, the following actions need to be taken in order. Notice that the conclusions in table 4.2 is inserted into parenthesis’ to easy correlate the conclusions to the action sequence.

1. Extract annotations for the article. (Page)
2. Process these annotations and retrieve the relevant information. (Page)
3. Determine if the user is among these annotations. (Rating)
   (a) If the user has a rating in this set, remove it. (Rating)
   (b) Update trust values according to the extracted annotations. (Trust)
4. Use the ratings to calculate a score and category for the article. (Rating)
5. Insert the score and category in the html document. (Page)
6. Insert html document into the users browser. (Plugin)

After this process has taken place, the user can read the article and use the embedded applet to give feedback. In case feedback is given, the following actions should take place.

1. Prepare the users annotation for upload and upload it. (Page)
Table 4.2: The functionality of the modules of the WRS plugin

<table>
<thead>
<tr>
<th>Module</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td>Extracts and inserts data from the Wikipedia and modifies the current HTML document.</td>
</tr>
<tr>
<td>Rating</td>
<td>Data structure and processing methods for ratings.</td>
</tr>
<tr>
<td>Plugin</td>
<td>Central coordination module.</td>
</tr>
<tr>
<td>Trust</td>
<td>Updates trust values, trust evolution functions and handles the web of trust.</td>
</tr>
</tbody>
</table>

2. Update trust values according to the extracted annotations. (Trust)

One step is shared between these two lists and can thus, in a total sequence of events, be executed twice. The first trust update implements the idea proposed in chapter 3.6 while the second trust update is the traditional trust update that happens when the user rates an item.

From the above action sequence, it is possible to derive a set of concepts that the actions relate to. From this we can conclude that the WRS plugin itself should be split into one main coordination module and three utility modules. The details of these modules can be seen in table 4.2 and the action sequence combined with the conclusions in table 4.2.

One aspect of the process in the above action sequence that was ignored is the user interaction with the WRS. Given that the Scone proxy lets us manipulate the HTML document, the most straightforward solution is to embed an applet into the webpage for the user to interact with. Inserting this necessitates that it can communicate with the WRS plugin, this can implemented through Remote Method Invocation (RMI). The applet can be injected into the web page by the same method that inserts the score and category information. The WRS module should implement methods for the embedded applet to use.

In the following sections we will describe the modules in more detail.

4.2.2 Page module

The purpose of the page module is to process HTML documents in all ways that are necessary. In the current design the following processes are determined to take place and as such tools for those purposes need to be designed.

- Annotations need to be extracted for the article.
4.2 The WRS Plugin

Figure 4.2: Flowchart for the HTML document in the WRS Plugin.

- Version number for the article needs to be extracted.
- The score, category and applet needs to be inserted into the HTML document.
- The annotation by the user needs to be prepared and inserted into the Wikipedia.

Since the analysis has shown that the proper repository for the annotations is in the userpages of the user Recommendations, a method needs to be designed that takes as input the name of the article\footnote{Since Wikipedia has a well defined naming structure, the complete URL can be constructed predictably and accurately from the article name only.} or the complete URL. The method needs to process these annotations, extract their ratings and insert them into a data structure\footnote{For example remove annotations that were added a long time ago or simply erroneous annotations.}

A method to extract necessary information from the article and saves it for later retrieval.
A method that takes the score and category information and inserts it into the HTML document is needed. This method also needs to insert the applet.

A method that takes the previously described information to form an annotation is needed. After the formation of the annotation it needs to upload this to the Wikipedia as well.

### 4.2.3 Rating module

The purpose of the rating module is to provide a data structure for ratings, both in the singular and plural; ie. a structure to hold lists of ratings. In the current version of WRS this module has expanded slightly to provide a data structure for the category as well as the Web of Trust’s interaction history.

- A data structure for ratings, needs to contain all relevant information and methods to support their use.
- A data structure facilitating the representation of the category in the WRS.
- A container to control a collection of ratings and aggregate necessary data from that collection.
- A method to aggregate score and category data.
- The data unique to the situation in which trust values are being computed need to be stored.

Ratings in the system now contain more than just a score for an article and how this is compared to the users own, it now also contains the category and also how it compared to the users own. Looking at the possible interactions as described in Chapter 3, a category interaction can have three outcomes. We define the value 0 to be a good interaction where the trustor and trustee agreed on the category, 1 to be a bad interaction in which the trustor agreed with the majority on the category and 2 is able to be either interaction. With a category interaction of 2, the user can either agree or disagree with the user on the score but they will always have disagreed on the category. Both cases trigger the same weight value, $\frac{1}{2}$. Furthermore ratings need to know for which article, what version of the the article and at what date they were given. The version is needed for determining when the rating becomes too old in an absolute sense and the date information needs to be used to calculate the effect of decay on the rating.
The category structure is simple, but should make it easy for the developer to handle. We suggest categories should be equally easy to use as either a more intuitive enumerator or a simple integer.

The container to control a collection of ratings is more specifically the need for the WRS to aggregate rating information, thus it is optimal to store this in one class. What is essential about these ratings are that they hold the interaction experience between the trustee and the trustor. So the primary function of the container, besides collecting a subset of all rating information, is to act as an aggregation tool to calculate the sum of the aforementioned experiences modified by time decay. This value is, through the trust evolution function, calculated into the trust value. The restrictions set forth in Chapter 2 on how the trust evolution function is bounded needs to be observed in the implementation.

When a page is being viewed, it needs a method to calculate an average score, determine the dominating category along with a confidence score in that category. This method needs to extract from a set of ratings, all score and category information and then average them in a reasonable way. Specifically for the score average, it has been determined by Korsgaard\(^\text{[13]}\) that a trimmed average where 5\% of the bottom and top scores gets cut away in order to make the average more resilient to outliers is used. Not commented upon is the fact that a trimmed average, inherently in eliminating outliers, also makes the average more resilient against attacks in that the general assumption is that an attack will take the form of producing an outlier in order to affect the score as much as possible.

As previously described, the WRS should be capable of updating the trustors trust profile by assimilating new ratings from articles that have previously been rated by users. To this end some information that is unique to that situation needs to be stored, so that when new ratings are being assimilated the trust updater will have all the information necessary to execute.

### 4.2.4 Trust module

The purpose of the trust module is to provide a data structure for the Web of Trust. Integral to the Web of Trust is the individuals for whom it stores trust values, a data structure needs to be implemented for them and especially the trust evolution function that updates their trust values. The trust module should also be responsible for the decision making in regards to comparing the extracted ratings to the decisions of the user, the results of these comparisons then need to be acted upon as well.
• A data structure for the Web of Trust with necessary methods, such as inserting reviewers and assign ratings to them.

• A data structure for the reviewers, most importantly the trust evolution function to update trust values.

• A class or method to implement the handling of rating comparisons and the updating of the Web of Trust.

Because we have previously defined the need for a rating history, the most logical place for this is in the reviewer data structure as the rating history is personal to the reviewer. Since the rating history class computes the x value, the obvious choice is for the reviewers data structure to become the “home” of the trust evolution function.

The Web of Trust itself then becomes a rather simple data structure, simply extending methods such as the insertion of ratings and retrieval of trust values.

The class or method to implement the handling of rating comparisons needs to implement methods so that the embedded applet can accept user information and pass this information to the class or method. With this information and the extracted ratings, comparisons and outcomes of interactions can be determined. Finally new reviewers, or ratings, can be inserted into the Web of Trust.

4.3 Summary

The design of the WRS was discussed by mapping the actions that the WRS needs to perform. This mapping was then used to devise modules to compartmentalize the tasks and these modules were assigned a set of tasks that they should be able to perform, while the order in which these are to be performed was mapped.
In this chapter the implementation details of the designs from the previous chapter will be described. The structure of this chapter will mirror that of the design chapter closely so as to make it easy to compare the implementation details to the design details.

5.1 Plugin Module

The plugin is implemented in two different classes in the plugin package, one called \texttt{WRSPlugin.java} simply takes care of setting up the plugin as per the requirements of Scone. The other, the \texttt{WRS} class, functions as the main coordination hub.

```java
public int getRequirements() {
    return PARSEDOCUMENT;
}

public void init() {
    WRS wrs = new WRS(this);
    wrs.setup("WikipediaRecommenderSystem", HTDOCCCONDITION, 60);
```
addMeg(wirtu);
}

These two methods are required by Scone. `getRequirements` tells Scone which features to have available at runtime. `PARSEDOCUMENT` tells Scone that we need the `HtmlNode` objects. The setup command gives the WRS plugin a name within the Scone environment for debugging purposes, `HTDOCCONDITION` specifies that only real HTML documents should be edited by the WRS plugin. 60 is a priority assignment in case two or more plugins run under the same condition. Originally the WRS plugin listed `CONSIDERLINKS` as a requirement, even though it was not needed.

As mentioned in the design chapter, the plugin should pass the HTML document on to the user unless we encounter a Wikipedia address.

```java
if ( !(page_url.contains("en.wikipedia.org")) ) {
    try {
        Token t;
        while ((t=in.read()) != null && !in.isClosed()) {
            out.write(t);
        }
    } catch (Exception e) {
        ...
    }
    return;
}
```

Even though it was not stated earlier, in the current version of WRS, it only processes pages belonging to the English Wikipedia. This is because there are slight differences between the various language Wikipedias, such as the German version not acknowledging a succesful login. Because of the fear for such issues arising, it was decided to focus entirely on the English Wikipedia. It should not be a major undertaking to make the WRS work for all languages, but it was considered additional work that could be spent better to stabilize the program. It is not listed in the above code snippet, but after this if statement there is a following else statement that checks what namespace in Wikipedia the url is in and rejects it, if it is not in the article namespace. This feature was added to this version, previous to this the WRS would attempt to process any page in the browser. The results of that was several illegal operations, the resulting exceptions and possibly a program crash.

```java
if (htmlPage.size() == 0) {
```
The above printout was implemented so as to notify the user if the browser fetched a cached version of the page. One weakness of the Scone proxy, at the moment, is that pages do not properly obey the *no-cache* command in HTML and continue to cache pages. Why this is so is unknown to the author, however with the above notification the user is at least made aware of the problem. This particular problem is the greatest weakness with the WRS plugin at the moment, since it requires the user to pay attention to the output from Scone and go through the tedious task of clearing the cache. Fortunately, in general usage it is not anticipated to be a major detrimental factor as it only comes into play after a user returns to a page that has been updated with ratings or to a page the user has just rated.

```java
PageExtractor pex = new PageExtractor(tokenInputStream, page_url);
String version = pex.extractionVersion();
WoT web = Serializer.readRoRFromDisk(wiki_username);

The above initiates the page extractor by giving it the input stream for the HTML tokens. The page url is used to construct various links. The result from this operation is seen above as well, the version is retrieved for later use. Something that was not mentioned in the design chapter in regards to the operation of this class is the Web of Trust, it needs to be read from drive for later use.

```java
ExtractRatings exr = new ExtractRatings(pex.extractEditPage());
sessionRatingDB = exr.getSessionRatingDB();

The rating extractor is initialized, during initialization it will extract and process the ratings. The call to `getSessionRatingDB` simply retrieves the result of the process.

```java
Rating temp = sessionRatingDB.hasOwner(wiki_username);
if (temp != null) {
    double[] interaction = new double[3];
```

\(^1\)It was attempted to insert the no-cache command into the web pages via the PageModifier.
interaction = web.getInteraction(temp.getURL());
if (interaction != null) {
    tu = new TrustUpdater(sessionRatingDB, interaction, web);
    if (interaction[3] == 1)
        tu.ClickedYes();
    else if (interaction[3] == 0){
        tu.ClickedNo();
    }
    tu.ClickedRating(temp.getRating(),
        temp.getCategory().getType().ordinal());
}

The sessionRatingDB is checked for an annotation by the user, the result of that check is returned. If the annotation is found it is removed. If the user has rated the article previously, a record of that should be found in the interaction history in the Web of Trust. If there is no trace of the interaction, that means that either the rating was faked or the user deleted his Web of Trust data file. In either case, nothing further is done. If the interaction is found, the details of it is used to simulate the interaction again to update the trust values. Old ratings are not cleared from the sessionRatingDB before insertion, the insertion method will reject it to avoid doing a recalculation of the trust value and waste CPU cycles. This is a new addition to the WRS, this is considered an essential feature to allow for the users trust profile to build faster. Without this, the WRS would behave in an unintuitive way and lose the confidence of its users quickly.

The above section of code does not fulfill all the requirements as outlined in Chapter 3.6, this would require threaded operations that the WRS does not currently support. The above should not be part of the main action sequence of the WRS, but be the job of a separate thread so that it would run concurrently with the WRS and not incur extra wait time on the user.

RatingCalculator rc = new RatingCalculator(sessionRatingDB, web);
PageModifier pm = new PageModifier(htmlPage);
// 0 is the score, 1 is category average and 2 is type
double[] computedAverage = rc.computeAverage();
pm.insertCategoryandRating(computedAverage);

The rating calculator and page modifier is initialized, the rating calculator is then asked to compute average values and the page modifier then inserts the information into the HTML document.
rs = new RecommendationSubmitter(wiki_username, wiki_password, 
    version, article_name, privateKey);

tu = new TrustUpdater(sessionRatingDB, computedAverage, web);
FeedbackInterface stub;
Registry registry;

try {
    registry = LocateRegistry.getRegistry(1099);
    stub = (FeedbackInterface)
        registry.lookup("FeedbackInterface");
} catch (NotBoundException e) {
    try {
        stub = (FeedbackInterface)
            UnicastRemoteObject.exportObject(this, 0);
        registry = LocateRegistry.getRegistry(1099);
        registry.bind("FeedbackInterface", stub);
    } catch ...
    writeOutToken(pm.getHtmlPageVector());

The recommendation submitter and the trust updater are prepared for remote 
method invocation. In the previous version of the WRS, the program would 
always try to rebind the stub to the registry resulting in a slew of exceptions 
each time a page was visited. Proper exception handling was added so that if 
the stub is not already bound, which it will not ever be on first execution, it will 
bind itself. writeOutToken is a method used to hand over the HTML document 
to the browser, it is very similar to the code described earlier on this page about 
checking for the Wikipedia domain. Finally the program will output a line that 
tells the user the WRS is ready for input.

The methods for the embedded applet was originally implemented so that there 
was one method in the WRS class for each button in the applet. This method was 
further duplicated in the TrustUpdater. The result was a sequence such that 
Applet.ClickButton7 \rightarrow WRS.ClickButton7 \rightarrow TrustUpdater.ClickButton7 
and a similar sequence was present for each interaction with the user interface. 
This was removed and replaced by a single ClickedRating function that also 
passes along the category selection, thus improving readability significantly by 
replacing several pages of code by a few lines.
5.2 Page Module

The page package consists of the four classes ExtractRatings, PageExtractor, PageModifier and RecommendationSubmitter.

5.2.1 ExtractRatings

The ExtractRatings constructor takes the URL of the edit page and constructs the necessary input stream readers. Most importantly is the token input stream reader, which it uses to load all the tokens in the HTML page into a vector. It then searches through the vector until it finds the text area token.

```java
if (sig.substring(0, 16).equals("WikiTrustComment")) {
    String raw_rating = "";
    for (String temp = ((HtmlTextToken)
        tokens.remove(0)).getText();
        !temp.contains("--&gt");
        temp = ((HtmlTextToken) tokens.remove(0)).getText()) {
        raw_rating += temp;
    }
    raw_ratingsVector.add(raw_rating);
}
```

Once the text area is found, it is searched until it finds the identifier for an annotation(WikiTrustComment). Once this is established all text is appended into a temporary string until the delimiter(–&gt) is reached. Previously this was done in a different way, such that instead of parsing until the delimiter was reached it would simply parse for five more steps. This had unintended side effects depending on annotation length and thus this more robust way of doing it was implemented. After the annotation vector has been constructed, it is processed. The current processing removes annotations that have a title different from the page title and that have been entered in a version that have since had its text edited too much for the annotation still to be relevant.

```java
while (!raw_ratingsVector.isEmpty()) {
    raw_ratingString = raw_ratingsVector.firstElement().split(";");
    r = new Rating(raw_ratingString[1],
```

\[ This character is actually \(=\rangle \), but shows up as \(=\rangle\) in html documents. \]
new Integer(raw_ratingString[2]),
new Integer(raw_ratingString[3]),
raw_ratingString[4],
sessionRatingDB.push(r);
raw_ratingsVector.remove(0);
}

After the processing the annotations are then converted into ratings usable by the system. The ratings are then entered into a sessionRatingDB, which is basically a wrapper for an ArrayList with some extra functionality. The constructor does not return anything, of course, so a getSessionRatingDB method is implemented to retrieve it.

### 5.2.2 PageExtractor

The PageExtractor class is very simple and works in much the same way as the ExtractRatings class. However it is not possible to retrieve the version of the article from the edit page, not even if the actual edit page was accessed. Instead of finding the text area this time, the method locates a link located in the Wikipedia toolbox. The toolbox is a small collection of useful links, among those one that links to the current version of this page. The information in that link is harvested to yield the current page version. The class works the same as ExtractRatings in that it finds the information, then makes it available for retrieval afterwards via a get method. Other information such as a link to the edit and history page is made available as well as the page title.

### 5.2.3 PageModifier

The PageModifier class takes the HTML token vector in its constructor, then waits for further instructions. It implements the insertCategoryandRating method which takes the calculated score, category percentage and category type and inserts it into the page along with embedded applet.

To facilitate the insertion, two files have been created that contain the necessary HTML tokens together with placeholder tags. One piece to be inserted into the head of the document and another to be inserted into the body. The head piece defines the dimensions of the body piece and bestows drag functionality
Implementation

to the body pieces. The body piece contains two placeholders. These two html
documents are then read into two strings for insertion into the main HTML
document.

DecimalFormat print = new DecimalFormat("#0.00");
if (String.valueOf(rating[0]).contains("NaN")) {
    toInsertInBody = toInsertInBody.replace("###RATING###",
            "Not enough information for a rating");
} else {
    toInsertInBody = toInsertInBody.replace("###RATING###",
            print.format(rating[0]));
}
if (rating[2] > -1.0) {
    String catType = new Category(
        Category.Type.values()[(int)rating[2]]).toString();
    toInsertInBody = toInsertInBody.replace("###CATEGORY###",
            print.format(rating[1]*100) + " % " + catType);
} else {
    toInsertInBody = toInsertInBody.replace("###CATEGORY###",
            "Not enough information for a category");
}

In the previous version, the method would always try to insert the score regard-
less if there was an actual score. If there wasn’t that would result in the rather
unhelpful message “NaN” being displayed. Proper formatting has also been
added so that numbers will not be printed with unnecessary precision. The lat-
ter part, the category, is a new addition so the information about category can
be displayed. The size of the inserted HTML box was appropriately increased
for this.

After this has been prepped for insertion as shown above, the method will find
the </HEAD> and </BODY> tags and insert the appropriate string in front of each.
The class follows the same inexplicable design pattern as elsewhere, in that it
has no return value but implements a get method to retrieve the result of the
method.

5.2.4 Recommendation Submitter

The RecommendationSubmitter class takes the relevant information to form
an annotation in its constructor except for the rating and category information.

---

3Actually three, but the third is rather unimportant for the understanding of the program.
5.3 Rating Module

The rating and category information is inserted later, if necessary, via RMI from the embedded applet. The method invoked by RMI is `SubmitStringToForm` and it relies on the two utility classes `SecurityProvider` and `Wiki`. `SecurityProvider` is used to create the annotation and `Wiki` is used to login and submit the annotation to the Wikipedia.

The `RecommenderSubmitter` makes use of the `createRating` method in the `SecurityProvider` class. This method concatenates the information required for the annotation, then signs it with a private key and appends that signature onto the annotation.

The `Wiki` class is developed by the Wikipedia user MER-C for use with Wikipedia bots. The main usage of this class in the WRS is its method to login and edit pages. However, in its original state the `editPage` method was not functioning properly. This was tracked down to the matter in which it transmitted the edit request to the Wikipedia. There are two ways to transmit forms over the internet, those are `application/xxx-form-urlencoded` and `multipart/form-data` where the `Wiki` class uses the former. Through packet sniffing it was discovered that the `wpEditToken` was being mangled and Wikipedia would not acknowledge it. To edit a Wikipedia page the `wpEditToken` needs to be recovered from the edit action and then submitted with the edit submit action before it is considered. Through packet sniffing it was also discovered that browsers were using the `multipart/form-data` transmission approach and since attempts at discovering why the `wpEditToken` was being mangled were unsuccessful it was decided to alter the `Wiki` class in such a way that it would instead transmit the edit submit request with a `multipart/form-data` request. This proved a successful approach.

5.3 Rating Module

The rating package consists mostly of data structures that are uninteresting from an implementation standpoint, so the `Rating`, `Category` and `InteractionHistory` classes will be ignored. The `SessionRatingDB` is technically a class in the rating package, however it has no basis for existing from a design perspective. In the previous version of the WRS it was an unnecessary wrapper for an `ArrayList`, however in the current version a method has been added that makes it go beyond a simple wrapper for an `ArrayList`. Still this class should be eliminated and the functionality of the method should be extracted to the single usage in the code it has.
5.3.1 RatingHistory

This class contains three methods, `insertRating`, `getXValue` and `updateTrustDynamics`. The latter function is the only interesting one, the two former ones are rather intuitive. One quick thing to notice about `insertRating` though is that it triggers `updateTrustDynamics` and as such it needs to protect itself from updating the trust dynamics if it is a duplicate rating. This has been added in the new version to prevent waste of CPU cycles. As noted by Korsgaard[13], the execution time for the WRS is not entirely insubstantial and can lead to user frustration.

```java
double categoryWeighting = 1.0;
if (rating.getInteractionCategory() == 1)
    categoryWeighting = 1.5;
if (rating.getInteractionCategory() == 2)
    categoryWeighting = 0.5;

if (rating.getInteractionRating() == 1) {
    if (ratingAge > month)
        positive_one_month += categoryWeighting;
    if (ratingAge < month && ratingAge > halfYear)
        positive_half_year += categoryWeighting;
    if (ratingAge < halfYear && ratingAge > year)
        positive_one_year += categoryWeighting;
}
```

The first part of the above section of code identifies which type of interaction there was with categories then determines the weighting as per Chapter 3. Also refer to Chapter 4 for the values of the category interaction. The base weighting is 1.0, this cover two of the five possible interactions. In the case where the category interaction has the value of 1, the category interaction can also only be 0 and thus the \( \frac{3}{2} \) weighting can never be applied in the positive. It is not shown above, but the negative follows directly after this in the code but conceptually mirrors the positive. The weighting of \( \frac{1}{2} \) can be applied in either direction as per Chapter 3.

```java
xValue = 0.1 * positive_one_month +
    0.05 * positive_half_year +
    0.025 * positive_one_year -
    0.1 * negative_one_month -
    0.05 * negative_half_year -
```
0.025 * negative_one_year;

As seen from this code snippet, each individual interaction is worth \( \frac{1}{10} \) in either direction. After six months the rating will be worth 50% and 25% after one year. After this the `trustUpdateDynamics` method would end, which is where the problems described in Chapter 2 comes in. There is no upper or lower bound on the x value. This is added with the following snippet of code.

```java
if (xValue > 1.0)
    xValue = 1.0;
else if (xValue < -1.0)
    xValue = -1.0;
```

### 5.3.2 RatingCalculator

The rating calculator class takes the `SessionRatingDB` and the Web of Trust in the constructor, then creates a local copy of the `SessionRatingDB`. There is one method in this class and that is the `computeAverage` method. It uses the values that the constructor saved locally to compute the average rating, category average and dominant category type. The rating is averaged with a 5% trim average.

```java
for (Rating r : sessionRatingDB) {
    double trustValue = ror.getTrustValueOfUser(r.getUserName(),
                                              r.getCategory().getType());
    categories[r.getCategory().getType().ordinal()] +=
        trustValue;
    ...
}
```

The above code snippet takes each rating in the `sessionRatingDB` and saves its trustvalue in an array with size equal to the amount of categories. The end result is an array that has in each slot the aggregate trust we have in each category.

```java
double cataverage = 0.0;
```

---

\(^4\)Which in this case is just an unwrapped `ArrayList<Rating>`, otherwise the for-each loop is impossible.
double catType = -1.0;
double totalTrust = 0.0;
for (int i = 0; i < categories.length && sessionRatingDB.size() > 0; i++) {
    counter += categories[i];
    if (categories[i] > cataverage) {
        cataverage = categories[i];
        catType = (double) i;
    }
}
cataverage /= totalTrust;

Before we can iterate over the length of the categories array, we need to make sure there is anything in the category array first. If the size of the sessionRatingDB is more than zero, the aggregate trust value will not be zero which will avoid a divide by zero error. The category average will sum up all the trust in totalTrust and add the specific trust in each category in the category array. catType will keep track of what the dominant category is. After the dominant category has been determined, the trust in it will be divided by the total amount of trust. This results in a trust-weighted category average. The average rating, the category average and the category type is essentially what is stored in the interaction history. This information is necessary to recreate the circumstances under which the trust values were updated. These snippets are new additions to the WRS and implemented according to the analysis in Chapter 3.3

5.4 Trust Module

The trust module has three classes, the WoT, Reviewer and TrustUpdater. As mentioned in Chapter 4 the Web of Trust class is rather uninteresting, it merely extends methods from its local objects so that they are accessible through the Web of Trust class directly.

5.4.1 Reviewer

The Reviewer class mainly focuses on handling the trust dynamics and calculating the trust values through the trust evolution function. It also implements four basic methods for retrieving and assigning the username and a method for retrieving the trust value associated with a category. The constructor for this class implements decisions from Chapter 2 such as the trust value starting at
0.0, the sum of interactions starting at 0.0 and the trustor behavior towards the trustee is defined to be trusting and optimistic.

```java
public void insertRating(Rating r) {
    if (!rh.insertRating(r)) {
        return;
    }
    int index = r.getCategory().getType().ordinal();
    xValue[index] = rh.getXValue(index);

    // update State value
    if (xValue[index] >= 0.0)
        state[index] = State.TRUST;
    if (xValue[index] < 0.0)
        state[index] = State.DISTRUST;
    // If it is a positive exp
    if (r.getExperience() == 1) {
        positiveExp(index);
    }
    // If it is a negative exp
    if (r.getExperience() == 0) {
        negativeExp(index);
    }
    // Update the trust value
    calcTrustValue(index);
}
```

The `insertRating` call at the start is the call to the `RatingHistory` equivalent, as previously described this triggers an update of the sum of interactions(x). As per chapter 2, the four different curves in the trust evolution function is distinguished between based on trust/distrust and cautious/optimistic states and one of these states is set here. Based on the rating interaction type methods are invoked to take the proper action to adjust the trust evolution function in regards to curve selection and curvature. Finally the trust value is recalculated.

```java
private void negativeExp(int index) {
    noOfNegInteractions++;
    ...
    // user in distrust and a optimistic curve
    if (curve[index] == Curve.OPT &&
        state[index] == State.DISTRUST) {
        nValue[index] = nValue[index] - 0.1;
    }
```
Implementation

```java
if (nValue[index] == 1.0 &&
    noOfPosInteractions == noOfNegInteractions) {
    curve[index] = Curve.CAU;
}
}
...
}
```

If the user is optimistic, but distrustful of the trustee the curvature of the curve is flattened, by decreasing the n value, following a negative experience. This is the reverse of what happens in figure 2.2(a) and 2.2(b). If the curve is entirely flattened, the user switches from being optimistic to cautious. The point at which the curve is flattened is at $n = 1.0$, below this point the curve will go outside the bounds previously defined and behave in unintended ways. After adjusting the curve to its proper setting, the trust value will be calculated.

There is one rather serious bug in the previous version of the WRS, which is also present in the snippets above. Remember that the initialization value of n is 1.0, when it comes time to update the curve selection in `negativeExp` the n value will be decreased to 0.9 and the check to switch curve expects a value of 1.0 to be present after the 0.1 decrement. This leads to a case where the curve will be behaving in unintended ways and produce erroneous trust values, it will persist until the n value gets incremented back to 1.0 where a curve selection will then take place. The assumption made is that the n value will approach 1.0 from a value greater than 1.0, which is not true of the initial condition.

```java
private void negativeExp(int index) {
    ...
    // user in distrust and a optimistic curve
    if (curve[index] == Curve.OPT &&
        state[index] == State.DISTRUST) {
        if (nValue[index] == 1.0) {
            curve[index] = Curve.CAU;
            nValue[index] += 0.1;
        } else {
            nValue[index] -= 0.1;
        }
    }
    ...
```

This only refers to the snippet above, the same bug is present in the positive version of this method.
The above correction will properly handle the problem described, the \( n \) value will not be decremented below the 1.0 threshold and there is no longer an assumption made by the code about the behavior of \( n \). Also the two counters `noOfPosInteractions` and `noOfNegInteractions` have been removed, their usage was self defeating. Checking these two values against each other as a second condition for the `if` statement meant that either these two were always a match when \( n = 1.0 \) or the curve would break its bounds and start producing erroneous trust values.

5.5 Security

The original WRS had a security design implemented that used asymmetric cryptography to allow for the verification of annotations. The user’s own public key was posted on a user subpage called cert. This allowed the user to sign his own annotations with the private key, upload the annotation and later anyone downloading this annotation could verify it with the public key. This approach provides excellent security for the WRS, however it did not work in the prototype provided and attempts at fixing it were unsuccessful. At first it was suspected that it was a transmission problem, that somehow the signature was garbled during up or download. This was proven wrong when the annotation was saved to disk and subsequently reloaded for verification, which failed. The BASE64 conversion algorithms were tested, both Sun’s own and an open source algorithm. For reasons unknown, saving a byte string then encoding it with BASE64 and decoding it back for comparison with the previously saved byte string failed.

Another approach to solving the security issues, possibly more elegantly and many times more efficiently is to cross-check with the revision history of the page with the annotations to compare uploader with the annotation’s associated name. It would stop the need for continually checking and updating certificates, something that Korsgaard determined to have by far the largest impact on load times.

Security is a key component of the WRS for mass deployment and it is a top priority for future work on the WRS, but due to time constraints the endeavour was put aside. However, the signing process when uploading annotations are

\[\text{\footnote{The reader is assumed to know the basics of cryptography and related issues for this section.}}\]
still operational so that when the verification process is fixed there will be no
need to cleanse the repositories and the local rating and interaction histories
unless the user wishes to do so.

5.6 Summary

In this chapter we have described key functionality by displaying the relevant
code and commenting on its function. Specifically the bounds that was deter-
mined in chapter 2.1.1.3 were implemented and the method explained. In this
chapter we have also shown the discovery of a problem with curve selection and
explained the necessary fix. A slightly simplified class diagram of the WRS is
shown in figure 5.1.

Lastly the security measures implemented in the old WRS was explained, why
it failed and a new way of handling security was proposed that could potentially
be computationally much less demanding.
Figure 5.1: Class diagram of the WRS.
In regards to the WRS there are two aspects that are considered for evaluation. One is the overhead imposed on browsing, measured by the time it takes between the time WRS receives a page and hands it over again to the browser. The other is the prediction accuracy of the WRS. Both of these will be discussed in the following sections.

### 6.1 Accuracy of the WRS

The work by Herlocker et al.\[^9\] is the seminal work of strategies for evaluating collaborative filtering systems. It reviews the key decisions in evaluating collaborative filtering systems. From this work we can get an overview of the evaluation strategies generally used, this reveals that the norm is to use a standardized data set to test algorithms. One of these popular data sets is the result of work by the GroupLens research lab at the University of Minnesota, their site also links to two other data sets. These data sets are shown in table 6.1.

These data sets are no longer compatible with the WRS, since the introduction of categories. This addition requires that new data sets are produced, the only way to come about this is to bring the WRS to such a state that it becomes deployable to a large userbase from where annotations can be harvested.
Table 6.1: The most popular data sets for collaborative filtering accuracy tests.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MovieLens</td>
<td>100,000 ratings for 1682 movies by 943 users</td>
</tr>
<tr>
<td></td>
<td>1 million ratings for 3900 movies by 6040 users</td>
</tr>
<tr>
<td></td>
<td>10 million ratings, 100,000 tags for 10681 movies by 71567 users</td>
</tr>
<tr>
<td>Book-Crossing</td>
<td>1,149,780 ratings for 271,379 books by 278,858 users</td>
</tr>
<tr>
<td>Jester Joke</td>
<td>4.1 million ratings for 100 jokes by 73,496 users</td>
</tr>
</tbody>
</table>

However, to evaluate the logic behind using categories as a choice we can construct a few scenarios to illustrate what it is that this approach contributes with.

The trustor rates the article on the Wheel, an article Alice has rated. As previously mentioned, Alice has rated it as a History article with a low score and the trustor then rates it as a Technology & Applied Science with a high score. Based on the conclusions in Chapter 3, Alice will be penalized in the History category and because the majority of trustees has rated the article as belonging to the Technology & Applied Science category she will be further penalized by half a point, to a total of $-\frac{3}{2}$ interaction. Because Alice disagreed with the majority, and the user agreed with the majority, we can now state that according to the user there is no uncertainty that Alice was wrong in categorizing the article as History. Furthermore, the low score distanced Alice from the trustor even more. The next time an article is visited where Alice has rated it as being a History article, Alice’s opinion will matter less both in determining the score of the category and the score of the article, this to a further extent than in the previous system.

Lets consider a similar scenario, but where Alice rates the Wheel article as Technology & Applied Science with a high score. The trustor this time rates it as History with a high score as well. The system will recognize the similarity in their score and determine that the overall interaction is good, both agree that the article is well written and structured. However, there is a difference of opinion in determining the category. The trustors trust profile has determined that the majority supports the Technology & Applied Science category, which means that Alice might have been correct. Because of this the system determines that Alice should have $\frac{1}{2}$ an interaction recorded for Technology & Applied Science to signify that the interaction even though not perfect was in her favor.

The most important aspect of determining whether the trust dynamics make sense, is to evaluate their intuitiveness. That is the purpose of the above two examples, to illustrate the intuitiveness of these conclusions through a scenario.
6.1 Accuracy of the WRS

Looking at trust evolution between the previous and current version, the current version will overall accrue trust value for a user at a slower rate because they are now specific to each category. If strictly contained to a single category, it depends on the difference in choice as there are now five different outcomes, $\frac{1}{2}$, $-\frac{1}{2}$, $-\frac{2}{3}$, and $+1$, $-1$.

![Figure 6.1: A sequence of actions and their effects demonstrated before and after.](image-url)

Consider encountering Alice for the first time, you rate three articles equally but Alice keeps disagreeing with the trustor on the article. In the fourth article, the trustor disagrees with Alice on both score and category. In the old system, this set of interactions would net Alice slight trust but in the new system it would net her no trust at all. Then you rate four Literature articles in a row, where you both agree on score and category. This nets you a moderate amount of trust for Alice in both systems, but because these two categories are linked in the old system Alice has 50% more trust from you than she would in the new system. And in the new system, she only has that amount of trust in Literature. Consider next browsing to an article about the Chevrolet Corvette. Alice is not very good with cars, she has determined the category in agreement with the trustor, as Technology & Applied Science. In the old system, Alice’s rating on this article would carry large weight based on the previous set of interactions but in the new system where categories are introduced the trustor will not trust
Alice in Technology & Applied Science. In the old system, this would lead to a bad experience for the trustor and he would potentially lose trust in the recommender system. The scenario described here is illustrated in figure 6.1 by displaying the sequence of interactions and their result in both the old and new systems.

Without an empirical data set, the best we can do is construct scenarios to illustrate the intuitive nature of the proposed trust model and how it differs in results from the previous model. The above examples should hopefully have demonstrated why it is important to distinguish by category and how the proposed trust model helps to do that.

### 6.2 Overhead imposed by the WRS

In the work by Korsgaard[13] the load times of pages were registered with Ethereal, however the author lacks the proficiency with Ethereal to do this task. Instead FireBug, a plugin for Firefox for web developers, was used. It allows the user to see the page loading process split up into its constituent parts, times all of them separately and also shows the total load time.

![Figure 6.2: Page loads compared, top section loading the WRS and 100 ratings and the bottom section without the WRS.](http://www.ethereal.com/)

And it is the authors opinion that it’s an overly complex tool for the task.
The WRS plugin imposes extra work on the host CPU before a page can be displayed, in cases where the page is outside the en.wikipedia.org domain the extra work is only a few milliseconds. Inside the en.wikipedia.org domain it takes around 1500-1600 milliseconds extra to process the page, load the embedded applet and extra a few ratings. Most of the overhead in extracting the ratings seem to come from the WRS establishing the connection to the repository, because even with 100 ratings extracted from the repository it adds very little load time. Figure 6.2 shows Firebug’s output for one load with WRS enabled, extracting 100 ratings, and one without WRS enabled. Note that time gets added to the “Waiting for Response” section, that’s because the browser does not see any HTML response from the server until the WRS has handed it off. Tests have been done where the WRS plugin is told to immediately hand off the HTML document to the browser, the Scone proxy then appears to be responsible for a base increase of 300-400 milliseconds. An estimate on how much time spent per entity can be seen in table 6.2.

Table 6.2: Estimated time consumption of HTML document retrieveal by entity.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Time Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRS Plugin</td>
<td>850-900 milliseconds</td>
</tr>
<tr>
<td>Scone Proxy</td>
<td>300-350 milliseconds</td>
</tr>
<tr>
<td>Internet</td>
<td>350-400 milliseconds</td>
</tr>
</tbody>
</table>

As mentioned in chapter 5.5 the security measures has been disabled, ie. the fetching of certificates and verification of signatures. The benchmarking of the original WRS concerned itself mostly with benchmarking the overhead from this activity. Unfortunately that makes comparing load times impossible. What is most unfortunate is that it is not possible to enable and disable the security measures to see the effect the process has on the page load time.

One timing that can be compared though is the load time of a webpage with the WRS enabled, without any ratings for the viewed article. The loadtime cited by Korsgaard is 5.23 seconds for an article with the WRS enabled and extracting 0 ratings, where the above results show that displaying an article and extracting 100 ratings takes 1.97 seconds. It is unclear how comparable these measurements are without a breakdown like the above, it seems unreasonable that there should be that big of a difference for the scone proxy and WRS plugin. However it seems even more unreasonable to think that the internet connection used in the test by Korsgaard was responsible for 3 seconds delay.

The goal with the WRS was never to optimize it though, so we do not put so much emphasis on the evaluation of the overhead imposed by the WRS on the browsing experience. The primary goal was to enable it to run and do so stably since it was only a prototype when picked up, optimization of the WRS
is the next step. However, it is still important to realize that it is ultimately an important measure of the WRS as research has shown that if a page takes more than 4 seconds to load a significant portion of its user base will be less likely to use a service\[23\].

6.3 Limitations of the WRS

The WRS has one severe weakness at the moment and that is in the way browsers control their cache. Because the WRS interface is injected into the webpage the way browsers control their cache is of utmost importance. Neither Internet Explorer, Firefox or Chrome works reliably in this regard, despite instructing them through their settings to never use the cache and always download a fresh page. Inserting <HTTP-EQUIV="PRAGMA" CONTENT="NO-CACHE"> into the webpage does not help and attempts at using the BrowserControl static functions of the Scone proxy to clear the browser cache between page visits has failed.

The result is that upon visiting a page that is in cache, the WRS can not show the result of its computations because the browser directly bypasses the Scone proxy when fetching the HTML document from cache. A notification has been implemented that prompts the user, in the Scone console, to delete the cache and refresh the page if anything that was expected to show up did not.

6.4 Functional Testing

A series of black box test cases have devised to test the possible actions with the WRS, these have been listed in table 6.3 along with their results. Odd exceptions do occur rarely. It is not sure whether these are tied to the WRS or to the Scone proxy as they can not be reproduced. They happen once, then do not occur again for an extended period of time.

6.5 Summary

In this chapter we have demonstrated the improved accuracy of the WRS, that it helps to discover areas of expertise of trustees. The computational overhead required by the WRS has been tested and shown to be within acceptable limits
### Table 6.3: Black box test results.

<table>
<thead>
<tr>
<th>Action</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rate an article with no previous annotations to verify upload</td>
<td>Annotation uploaded</td>
</tr>
<tr>
<td>Rate an article with previous annotations to verify own annotation being appended</td>
<td>Annotation appended</td>
</tr>
<tr>
<td>Rate an article with annotations extracted to verify trust updates</td>
<td>Trust values updated</td>
</tr>
<tr>
<td>Revisit an article that was previously rated and with new annotations</td>
<td>Annotations extracted and trust values updated</td>
</tr>
<tr>
<td>Visit an unrated article with annotations to verify computation of score and category</td>
<td>Score and category shown correctly</td>
</tr>
<tr>
<td>Visit a Wikipedia specials page and a page outside the Wikipedia domain</td>
<td>WRS disabled correctly</td>
</tr>
<tr>
<td>Rate an article, browse away and back to view category and score</td>
<td>Category and score not displayed</td>
</tr>
</tbody>
</table>

and a current limitation of the WRS is outlined. The results of black box testing have been listed as well.
The conclusions that can be drawn from the work in this thesis are related to either decentralized recommender systems or the Wikipedia Recommender System, in the following two sections the conclusions from each part will be represented.

7.1 Decentralized Recommender Systems

In this thesis we have proposed a way of applying recommender systems to a large information repository, the Wikipedia, by categorizing and using this information as a metric to compute trust values for users. The most important contribution is the novel way of extending the rating options for users by adding article category as a choice and the ways in which that information is used to compute the trust value. We have examined the way categories as a choice can be interpreted and how this can be expressed in the trust model by using weightings associated with the outcome of these interactions. It has been proposed that the choice a trustor makes in regards to a trustee in category can be relevant in how to determine the weightings and proposed weights. In chapter 6.1 it has been demonstrated how the system can succesfully assign trust in users area of expertise and how important it is in order for the recommender system to be credible to its users.
We have discovered that decentralized recommender systems have issues with regular trust propagation schemes due to privacy concerns and that decentralized approaches lack the speed of maturing trust profiles compared to centralized approaches. We have proposed a way to imitate an advantage of the centralized approach in the decentralized recommender system, by keeping a record of rating history and using this to continually check the annotation repository for the respective items rated.

The trust model underlying the WRS has been reinforced by identifying unbounded areas that unchecked would otherwise have left to unintended behavior, erroneous results and user dissatisfaction.

The most prominent classification schemes have been reviewed and the Dewey Decimal Classification scheme has been determined to be the most applicable, it is sufficiently succinct and wide in breadth to serve as a classification scheme for the Wikipedia.

We propose a way in which the recommender system can be integrated into the Wikipedia without interfering with the users who have not opted in to this system. At the same time the overall impact of the recommender system on the environment has been shown to be negligible.

### 7.2 The Wikipedia Recommender System

The work of implementing a theoretically sound recommender system for deployment is long and arduous, but with this thesis a great step has been taken in that a barely functioning prototype is now stable although with one limitation as described in chapter 6.3. The installation requirements of the WRS has been significantly cut down due to analysis of the Scone proxy and the MySQL has been identified as superfluous for the purpose of the WRS. The start up and shutdown of the WRS plugin is now single click.

At the same time the results of the research has been implemented into the system so that users can now be assessed on their expertise. The speed at which trust profiles mature is increased, so that it should match that of a centralized recommender system.
7.3 Future Research

The most important area of research for the WRS is the modeling of the trust evolution function and its components, this does not necessarily mean an empirical analysis. An empirical analysis would require a large data set, which is not available yet. It would be desirable to acquire the data set first, so as to back up any findings with empirical analysis, but it is of interest to compare the current prediction algorithm with other algorithms prevalent among recommender systems. Specifically how to incorporate the idea of the category as a choice into those algorithms.

Trust propagation is an important tool to help recommender systems deal with cold start problems, however due to the decentralized approach of the WRS there are currently no suggestions on how to extend current trust propagation schemes.

7.4 Future Work

The WRS is now in working condition, however from the developers point of view there are plenty of things to resolve. The current WRS makes use of some awkward design patterns, which means that heavy refactoring needs to be done on the WRS to eliminate these and make the code easy to read. Specifically the many classes that contain a single method with a specific purpose needs to be transformed into static utility methods, that do not require object initialization. This will also do away with the need of assigning all these classes into separate packages.

It might be possible to stop using RMI to communicate between the user interface and the WRS coordination module by using the RAS server from the Scone API. It supposedly allows passing string messages between the RAS server and client, which would mean the user interaction with the interface could be transmitted as string messages. This would mean that

The security measures of the WRS needs to be re-enabled. However it should be researched if it would not be possible to replace the rather costly asymmetric cryptography system with a cheaper system that would simply verify that an annotation has been uploaded by the correct user via the revision history. This would also serve to reduce the size of annotations by approximately 50%.

The proposed scheme to speed up trust profile maturation needs to be run in a
separate thread, so it does not interfere with the regular operation of the WRS and can run continually.
Appendix

A.1 Code

A.1.1 WRSPlugin.java

```java
package plugin;

import scone.Plugin;

public class WRSPlugin extends Plugin {
    // requirements
    public int getRequirements() {
        return PARSEDOCUMENT | CONSIDERLINKS;
    }

    public void init() {
```

Copyright (C) 2007 by Thomas Rune Korsgaard, tkorsgaard(a)gmail.com

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>.
A.1.2 WRS.java

package plugin;

import java.io.IOException;
import java.rmi.AlreadyBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.security.KeyPair;
import java.security.KeyStore;
import java.security.PrivateKey;
import java.util.Hashtable;
import java.util.Set;
import java.util.Vector;
import java.util.Date;
import page.;
import rating.RatingCalculator;
import rating.SessionRatingDB;
import rating.Rating;
import remote.FeedbackInterface;
import scone.proxy.HtmlTokenEditor;
import scone.util.tokenstream.SconePipe;
import scone.util.tokenstream.Token;
import scone.util.tokenstream.TokenInputStream;
import scone.util.tokenstream.TokenOutputStream;
import statictools.SecurityProvider;
import statictools.Serializer;
import statictools.TokenInputStreamTools;
import trust.WoT;
import trust.TrustUpdater;

public class WRS extends HtmlTokenEditor implements FeedbackInterface {

    WRSPlugin plugin = null;
    TokenInputStream in;
    TokenOutputStream out;
    TrustUpdater tu;

    WRS wirtu = new WRS(this);
    wirtu.setup("WikipediaRecommenderSystem", HTDOCCONDITION, 60);
    addMeg({wirtu});
}

WRS — Wikipedia Recommender System is a collaborative recommender system
used to rate the articles on the Wikipedia.
Copyright (C) 2007 by Thomas Rune Korsgaard, tkorsgaard[a]gmail.com
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.
You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.

/**
 * Wirtu is the Wikipedia with Ratings from Trusted Users This is the main
 * plugin for the SCONE browser
 * @author s011564
 */

public class WRS extends HtmlTokenEditor implements FeedbackInterface {

    WRSPlugin plugin = null;
    TokenInputStream in;
    TokenOutputStream out;
    TrustUpdater tu;

    WRS wirtu = new WRS(this);
    wirtu.setup("WikipediaRecommenderSystem", HTDOCCONDITION, 60);
    addMeg({wirtu});
}
RecommendationSubmitter rs;

// Default constructor
public WRS(WRSPlugin plugin) {
    this.plugin = plugin;
}

public WRS() {
}

// Implemented methods from FeedbackInterface
public String sayHello() {
    System.out.println("sayHello() invoked");
    return "Hello, world!";
}

// Implemented methods from FeedbackInterface
public void clickYes() {
    // System.out.println("Clicked Yes");
    tu.ClickedYes();
}

// Implemented methods from FeedbackInterface
public void clickNo() {
    // System.out.println("Clicked No");
    tu.ClickedNo();
}

public void clickRating(int rating, int category) {
    // System.out.println("Clicked " + rating);
    tu.ClickedRating(rating, category);
    rs.SubmitStringToForm(rating, category);
}

// handleRequest() is inherited from HtmlTokenEditor
public void handleRequest(SconePipe pipe) {
    long now = System.currentTimeMillis();
    // Set up a database to store the ratings captured
    SessionRatingDB sessionRatingDB;
    // The token streams are initiated from the SconePipe
    in = pipe.getTokenInputStream();
    out = pipe.getTokenOutputStream();
    // Page URL are obtained from the input streams’ meta information
    Hashtable ht = in.getMetaInfo();
    Set s = ht.keySet();
    Object[] sl = s.toArray();
    Object url = ht.get(sl[sl.length - 1]);
    /∗
    ∗ Do not process the page if it is not supposed to be.
    ∗
    ∗ System.out.println("Page URL = "+url.toString());
    if (!{"page_url.contains("en.wikipedia.org")}) {
        try {
            Token t;
            while ((t=in.read()) != null & !in.isClosed()) {
                out.write(t);
            }
        }
        catch (Exception e) {
            System.out.println("Page URL not relevant to WRS, processing skipped.");
            // e.printStackTrace();
        }
    }
    System.out.println("Operation took: "+(System.currentTimeMillis() - now));
    return;
    } else {
    /∗
    ∗ Listing of special cases, for some reason searches return "Special%3A" instead of Special:
    ∗
    ∗
    ∗ System.out.println("Page URL contains("Special%3A") || page_url.contains("Special") ||
    ∗
    ∗ page_url.contains("Talk") || page_url.contains("Wikipedia") ||
    ∗
    ∗ page_url.contains("Help") || page_url.contains("Media") ||
    ∗
    ∗ page_url.contains("User") || page_url.contains("Category") ||
    ∗
    ∗ page_url.contains("Portal") || page_url.contains("Main_Page") {
        try {
            Token t;
            while ((t=in.read()) != null & !in.isClosed()) {
                out.write(t);
            }
        }
        catch (Exception e) {
            System.out.println("Attempted to skip processing, if the page did not load, delete cache");
            // e.printStackTrace();
        }
    }
Appendix

82

...
A.1 Code

```java
public void writeOutToken ( Vector htmlPage ) {
    try {
        Token t = null;
        while (!htmlPage.isEmpty()) {
            t = (Token) htmlPage.firstElement();
            out.write(t);
            htmlPage.remove(0);
        }
    } catch (IOException e) {
        System.out.println("Writing to Token OutputStream failed!");
        e.printStackTrace();
    }
}
```

A.1.3 ExtractRatings.java

```java
package page;

import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.util.Vector;
import java.util.logging.Logger;

import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.util.Vector;
import java.util.logging.Logger;
```

```java
package page;

/**************************
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 **************************
```
import rating.Rating;
import rating.SessionRatingDB;
import scone.util.tokenstream.HtmlTagToken;
import scone.util.tokenstream.TokenInputStream;
import scone.util.tokenstream.TokenInputStreamTokenizerImpl;
import static tools.RatingCleanOut;

public class ExtractRatings {
    private static Logger log = Logger.getLogger(ExtractRatings.class.getName());
    // vector to store the tokens
    Vector<Token> tokens;

    // SessionRatingDB keeps a database of the active filtered ratings found on the page
    SessionRatingDB sessionRatingDB;

    // raw_ratingsVector keeps the ratings in raw string format.
    Vector<String> raw_ratingsVector = new Vector<String>();

    public ExtractRatings(String editUrl) {
        try {
            // Set up the token stream from the Edit url
            URL url = new URL(editUrl);
            URLConnection urlConnection = url.openConnection();
            urlConnection.setUseCaches(false);
            InputStream is = urlConnection.getInputStream();
            InputStreamReader isr = new InputStreamReader(is);
            TokenInputStream tis = new TokenInputStreamTokenizerImpl(isr);
            // Put the tokens from the Edit url into a Vector to process
            tokens = new Vector<Token>();
            Token t = null;
            try { while ((t = tis.read()) != null) {
                tokens.add(t);
            } catch (IOException e) {
                log.info("Initialization of Token Vector failed");
                log.info(e.getStackTrace().toString());
            }
            // initiate two temporary tokens
            Token tempToken = null;
            HtmlTagToken tempHtmlTagToken = null;
            // read all the elements from the tokenizer, and finish, whenever
            // there is no more tokens to be read
            while (!tokens.isEmpty()) {
                tempToken = tokens.remove(0);
                if (tempToken instanceof HtmlTagToken) {
                    tempHtmlTagToken = (HtmlTagToken) tempToken;
                    if (tempHtmlTagToken.getTagType() == HtmlTagToken.TEXTAREA && !tempHtmlTagToken.isEndTag()) {
                        HtmlTextToken signature = null;
                        String sig = null;
                        log.info("Found <textarea>, looking for Trust Comments");
                        // read the next element in the token stream, and keep on
                        // reading as long as it is of the type HtmlTextToken
                        do {
                            tempToken = tokens.remove(0);
                            if (tempToken instanceof HtmlTextToken) {
                                String sig = null;
                                log.info("Found a TrustComment");
                                // if it is an instance of HtmlTextToken and it
                                // starts with TrustComment
                                if (tempHtmlTagToken.getTagType() == HtmlTagToken.TEXTAREA && !tempHtmlTagToken.isEndTag()) {
                                    HtmlTextToken tempToken = (HtmlTextToken) tempToken;
                                    sig = tempToken.getText();
                                    if (sig.length() >= 16) {
                                        if (sig.substring(0, 16).equals("WikiTrustComment")) {
                                            // Insert the verified ratings in to the
                                            // SessionRatingDB
                                            String raw_rating = "*";
                                            // Remove the 5 words: wikiTrustComment.
                                            // Read more on:
                                            for (String temp = ((HtmlTextToken) tokens.remove(0)).getText();
                                                 !temp.contains("""">"""");
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
temp = ( ( HtmlTextToken ) tokens . remove ( 0 ) ) . getText ( )
{
    raw_rating += temp;
    raw_ratingsVector . add ( raw_rating );
}
}
}
}
}
// e n d s d o - w h i l e l o o p a s s o c i a t e d w i t h t h e d o s c o p e
while ( tempToken instanceof HtmlTextToken ) {
    }
}
}
}
}catch ( MalformedURLException e ) {
    // I f t h e U R L a r e n o t c o r r e c t f o r m e d
    e . printStackTrace ( ) ;
}catch ( IOException e ) {
    // I f t h e r e i s a g e n e r a l r e a d w r i t e e r r o r .
    e . printStackTrace ( ) ;
}
// c o u t o u t t h e t i t l e b e t w e e n t h e f i r s t = a n d t h e f i r s t &
// E g h t t p : / / en . w i k i p e d i a . o r g / w i k i / U s e r : R e c o m m e n d a t i o n s / H e l l e r u p & a c t i o n = e d i t
String pageTitle ;
try {
    pageTitle = ( editUrl . split (" /") [ 5 ] ) . split (" &") [ 0 ] ;
}catch ( ArrayIndexOutOfBoundsException e ) {
    // T h i s s h o u l d n ' t h a p p e n , b a d e r r o r h a n d l i n g .
    String [ ] temp = editUrl . split (" /");
    pageTitle = temp [ temp . length - 1 ] ;
}
// S y s t e m . o u t . p r i n t l n ( " R a t i n g s b e f o r e T i t l e M i s m a t c h : " + raw_ratingsVector ) ;
// C l e a n i n g o u t t h e r a t i n g s
// Remove Ratings that don't have the correct table
raw_ratingsVector = RatingCleanOut . RemoveRatingsTitleMismatch ( pageTitle , raw_ratingsVector ) ;
// System . o u t . p r i n t l n ( " R a t i n g s b e f o r e U n v a l i d a b l e R a t i n g s : " + raw_ratingsVector ) ;
// Remove Ratings that are unverifiable
// raw_ratingsVector = RatingCleanOut . RemoveUnverifiableRatings ( raw_ratingsVector , ror ) ;
// System . o u t . p r i n t l n ( " R a t i n g s b e f o r e T h r e s h o l d : " + raw_ratingsVector ) ;
// Remove Ratings that are above a certain threshold
// raw_ratingsVector = RatingCleanOut . RemoveRatingsBelowThreshold ( 0 . 1 , raw_ratingsVector , pageTitle ) ;
// remove duplicate ratings . T a k e a w a y t h e r a t i n g s t h a t w e r e a d d e d b y
// t h e s a m e p e r s o n .
// Leave only the newest
// System . o u t . p r i n t l n ( " R a t i n g s t h a t m a d e i t : " + raw_ratingsVector ) ;
// Set up the sessionRatingDB
sessionRatingDB = new SessionRatingDB ( ) ;
Rating r = null;
String raw_ratingString [ ] ;
// I n s e r t t h e r a t i n g i n t o t h e s e s s i o n R a t i n g D B
while ( ! raw_ratingsVector . isEmpty ( ) ) {
    raw_ratingString = raw_ratingsVector . f i r s t E l e m e n t ( ) . s p l i t (" ;") ;
    " http : // en . w i k i p e d i a . o r g / w i k i / " + raw_ratingString [ 5 ] ) ;
    sessionRatingDB . push ( r ) ;
    raw_ratingsVector . remove ( 0 ) ;
}
/**
 * getRawRatings () returns the raw ratings in string format
 */
public Vector getRawRatings () {
    return raw_ratingsVector ;
}
/**
 * getSessionRatingDB () gives the filtered ratings in for this session
 */
public SessionRatingDB getSessionRatingDB () {
    return sessionRatingDB ;
}
A.1.4 PageExtractor.java

```java
package page;

import java.io.IOException;
import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;
import scone.util.tokenstream.HtmlTagToken;
import scone.util.tokenstream.Token;
import scone.util.tokenstream.TokenInputStream;

public class PageExtractor {
    private static Logger log = Logger.getLogger(PageExtractor.class.getName());

    Vector<Token> vec_analyze;
    String edit_page;
    String base_page;
    String history_page;
    String version;
    String title;

    /**
     * PageExtractor looks through a HTML page and stores the edit url and the
     * history url, extracted from the source HTML.
     */
    public PageExtractor(String URL) {
        base_page = URL;
        log.setLevel(Level.FINE);
        // Vector to store the tokens from the stream
        vec_analyze = new Vector<Token>();
        try {
            while ((t = in.read()) != null) {
                vec_analyze.add(t);
            }
        } catch (IOException e) {
            log.info("Initialization of Token Vector failed");
            log.info(e.getStackTrace().toString());
        }
        HtmlTagToken tag = null;
        String relative_url = "";
        // String version = "";
        // Working through the vector of tokens looking for the edit link
        for (int i = 0; i < vec_analyze.size(); i++) {
            t = (Token) vec_analyze.elementAt(i);
            if (t instanceof HtmlTagToken) {
                tag = (HtmlTagToken) t;
                // Find the <LI id="ce-edit"> tag
                if (tag.getTagType() == HtmlTagToken.T_LI) {
                    if (tag.getParam("id") != null) {
                        if (tag.getParam("id").equals("ce-edit")) {
                            log.fine("Found LI with id=" + tag.getParam("id") + " parameter");
                            // Find the link from the next <A> tag
                            HtmlTagToken temp_token = (HtmlTagToken) vec_analyze.elementAt(i + 1);
            ```
A.1 Code

```java
relative_url = temp_token.getParam("href");
log.fine("Relative URL: " + relative_url);
}
if (((tag.getParam("id")).equals("t-permalink"))) {
HtmlTagToken temp_token = (HtmlTagToken) vec.analyze.elementAt(i + 1);
version = (temp_token.getParam("href")).split("=")[2];
// Store the title of the page
title = (temp_token.getParam("href")).split("=")[1];
title = title.split("\&")[0];
}
}
}
// Putting together the edit and the history URL
String[] temp = base_page.split("/");
// using that edit an history called the same way a PHP parameter
history_page = edit_page.replace("edit", "history");
}
// return the editURL
public String extractEditPage() {
return edit_page;
}
// Return the history URL
public String extractHistoryPage() {
return history_page;
}
public String extractionVersion() {
return version;
}
public String extractTitle() {
return title;
}
```

A.1.5 PageModifier.java

```java
package page;

/****************************************************************************************************
 * WRS − Wikipedia Recommender System is a collaborative recommender system used to rate the articles on the Wikipedia.
 * Copyright (C) 2007 by Thomas Rune Korsgaard, tkorsgaard(a)gmail.com
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.
 ****************************************************************************************************/
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.InetAddress;
import java.util.Vector;
import java.text.DecimalFormat;
```
import scone.util.tokenstream.HtmlTagToken;
import scone.util.tokenstream.HtmlTextToken;
import scone.util.tokenstream.Token;
import rating.Category;

/**
PagenModifier changes the HTML document, It inserts the tokens that
represent the ratings and the html required for the user to cast a rating
* @author s011564
*/
public class PageModifier {

    Vector<Token> htmlPage;

    /**
     * Default constructor Uses the vector htmlPage for manipulation.
     * @param htmlPage
     */
    public PageModifier(Vector<Token> htmlPage) {
        this.htmlPage = htmlPage;
    }

    /**
     * This constructor is only used for testing purposes!
     */
    public PageModifier() {
    }

    /**
     * Replace Tag is used to replace the ### tags with rating and links for
     * creating a rating
     * @param tag
     * @param line
     * @param replacement
     * @return
     */
    private static String replaceTag(String tag, String line, String replacement) {
        int b = line.indexOf(tag);
        int e = b + tag.length();
        String begin = line.substring(0, b);
        String end = line.substring(e);
        return begin + replacement + end;
    }

    /**
     * Insert the Yahoo UI Drag&Drop code and the rating
     * @param rating
     */
    public void insertCategoryandRating(double[] rating) {
        try {
            // Setting up the TOKENS to go into the HEAD of the HTML file
            File f = new File("static_textfiles/HeadHtml.txt");
            FileInputStream fis = new FileInputStream(f);
            BufferedReader in = new BufferedReader(new InputStreamReader(fis));
            String toInsertInHead = " ";
            while ((temp = in.readLine()) != null) {
                toInsertInHead += temp;
            }
            // Setting up the TOKENS to go into the Body of the HTML file
            f = new File("static_textfiles/BodyHtml.txt");
            fis = new FileInputStream(f);
            in = new BufferedReader(new InputStreamReader(fis));
            String toInsertInBody = " ";
            while ((temp = in.readLine()) != null) {
                toInsertInBody += temp;
            }

            DecimalFormat print = new DecimalFormat("#0.00");
            // replacing ###RATING###
            if (String.valueOf(rating[0]).contains("NaN")) {
                toInsertInBody = toInsertInBody.replace("###RATING###", "Not enough information for a rating");
            } else {
                toInsertInBody = toInsertInBody.replace("###RATING###", print.format(rating[0]));
            }
        }
    }
}
if (rating[2] > -1.0) {
    String catType = new Category(Category.Type.values()[int(rating[2])].toString());
    toInsertInBody = toInsertInBody.replace("###CATEGORY###", print.format(rating[1]*100) + "%" + catType);
}
else {
    toInsertInBody = toInsertInBody.replace("###CATEGORY###", "Not enough information for a category");
}
//System.out.println("Inserting:" + catPercent + ");

// Replacing ###IP###
InetAddress addr = InetAddress.getLocalHost();
String IPadress = addr.getHostAddress();
toInsertInBody = replaceTag("###IP###", toInsertInBody, String.valueOf(rating));

// Finding the Head and the Body tags, where the text from the files are inserted.
int i = 0;
while (i < htmlPage.size()) {
    Token tempToken = htmlPage.elementAt(i);
    if (tempToken instanceof HtmlTagToken) {
        HtmlTagToken tempHtmlTagToken = (HtmlTagToken) tempToken;
        // If the token is </HEAD>, then insert the Head Html text // before this tag
        if (tempHtmlTagToken.getTagType() == HtmlTagToken.T_HEAD && tempHtmlTagToken.isEndTag()) {
            HtmlTextToken head = new HtmlTextToken(toInsertInHead);
            htmlPage.insertElementAt(head, i - 1);
            i++;
        } // If the token is </BODY> insert the Body Html text before // this.
        if (tempHtmlTagToken.getTagType() == HtmlTagToken.T_BODY && tempHtmlTagToken.isEndTag()) {
            HtmlTextToken body = new HtmlTextToken(toInsertInBody);
            htmlPage.insertElementAt(body, i);
            i++;
        }
    }
    i++;
}
}

/*
* Returns the htmlPage vector when it is ready to be written to the output stream
* @return
*/
public Vector<Token> getHtmlPageVector() {
    return htmlPage;
}
import java.io.IOException;
import java.security.PrivateKey;
import static tools.SecurityProvider;
import static tools.Wiki;

public class RecommendationSubmitter {

    String username;
    String wiki_password;
    String version;
    int rating;
    String name;
    PrivateKey privateKey;

    public RecommendationSubmitter(String username, String wiki_password, String version, String name, PrivateKey privateKey) {
        this.username = username;
        this.wiki_password = wiki_password;
        this.version = version;
        this.name = name;
        this.privateKey = privateKey;
    }

    public void SubmitStringToForm(int rating, int category) {

        String rating_to_insert = SecurityProvider.createRating(username, rating, category, version, name, privateKey);

        try {
            // Create an instance of the WikiBot for the English Wikipedia as a default
            Wiki bot = new Wiki();
            // Login to the wiki
            // System.out.println("Login ");
            if (bot.login(username, wiki_password.toCharArray())) System.out.println("success!");
            // System.out.println("failed!");
            bot.login(username, wiki_password.toCharArray());
            // get the actual text of that article that have to be signed
            String text = bot.getPageText(name);
            // add the rating to that article
            text += rating_to_insert;
            // System.out.println(\"Text to insert: \" + text);
            // insert the new rating
            bot.editPage(name, text, "WikiTrustComment", true);
        } catch (IOException e) {
            System.out.println(e.getStackTrace());
            e.printStackTrace();
        }
    }
}

A.1.7 Category.java

package rating;

import java.io.Serializable;

/** * Created by IntelliJ IDEA. 
 * User: a011531 
 * Date: 03-06-2009 
 * Time: 15:41:58 
 * To change this template use File | Settings | File Templates. */
A.1 Code

```java
public class Category implements Serializable {
    private static final long serialVersionUID = 1L;
    private Type type;

    public enum Type {
        COMPUTER, PHILOSOPHYPSYCH, RELIGION, SOCIALSCIENCSE,
        LANGUAGES, SCIENCEANDMATH, TECHANDSCIENCE,
        ARTSANDRECREATION, LITERATURE, HISTORY
    }

    public Category(Type type) {
        this.type = type;
    }

    public Category(int category) {
        this.type = Type.valueOf(category);
    }

    public Type getType() {
        return type;
    }

    public String toString() {
        switch(type) {
            case COMPUTER:
                return "Computer, Information & General Works";
            case PHILOSOPHYPSYCH:
                return "Philosophy & Psychology";
            case RELIGION:
                return "Religion";
            case SOCIALSCIENCSE:
                return "Social Sciences";
            case LANGUAGES:
                return "Languages";
            case SCIENCEANDMATH:
                return "Science & Mathematics";
            case TECHANDSCIENCE:
                return "Technology & Applied Science";
            case ARTSANDRECREATION:
                return "Arts & Recreation";
            case LITERATURE:
                return "Literature";
            case HISTORY:
                return "History, Geography & Biography";
            default:
                return "Unknown category";
        }
    }
}
```

A.1.8 InteractionHistory.java

```java
package rating;

import java.util.HashMap;
import java.io.Serializable;

/**
 * Created by IntelliJ IDEA.
 * User: ThomasLefebvre
 * Date: 10-06-2009
 * Time: 19:38:37
 * To change this template use File | Settings | File Templates.
 */
public class InteractionHistory implements Serializable {
    private static final long serialVersionUID = 1L;
    private HashMap<String, double[]> history;

    public InteractionHistory() {
        history = new HashMap<String, double[]>();
    }

    public void insertInteraction(String url, double[] interaction) {
        if (!history.containsKey(url)) {
            history.put(url, interaction);
            //System.out.println("New interaction, saving in history.");
        } else {
            //System.out.println("Insertion aborted to preserve old interaction data");
        }
    }

    public double[] getInteraction(String url) {
        return history.get(url);
    }
}
```
public double getAverageRating(String url) {
    return history.get(url)[0];
}

public double getCategoryPercentage(String url) {
    return history.get(url)[1];
}

public Category.Type getCategoryType(String url) {
    return Category.Type.values()[((int) history.get(url)[2])];
}

public int getExperience(String url) {
    return (int) history.get(url)[3];
}

A.1.9 Rating.java

package rating;

import java.util.Date;
import java.io.Serializable;

public class Rating implements Serializable {
    private static final long serialVersionUID = 14L;
    private Date d1;
    private int rating;
    private int experience;
    private int interactionRating;
    
    // 0 if there's a good interaction, 1 and 2 if it's a bad interaction.
    // 1 signifies trustor chose same as majority and 2 signifies trustor
    // chose a minority and trustee either chose majority or minority.
    
    private int interactionCategory;
    private Category category;
    private String username;
    private String article_url;
    private String version;
A.1 Code

```java
/**
 * @param hashValue
 * @param dl
 * @param rating
 * @param experience
 */

/*********************
 * Use this constructor when initializing ratings into the SessionRatingDB. Use
 * setExp() to set experience later on
 * */

public Rating(String username, int rating, int category, String version, String article_url) {
    this.username = username;
    this.rating = rating;
    this.category = new Category(Category.Type.values()[category]);
    this.version = version;
    this.article_url = article_url;
    this.experience = 2;
    this.d1 = new Date();
    interactionRating = 2;
    interactionCategory = 2;
}

/**
 * This constructor is only for testing purposes. Opens the possibility to
 * insert old ratings, with customized dates! USE WITH CARE, and block comment
 * this method in final release!!
 * @param rating
 * @param date
 * @param experience
 */

public Rating(String username, int rating, String version, String article_url, Date date, int experience, Category category) {
    this.version = version;
    this.d1 = date;
    this.rating = rating;
    this.experience = experience;
    this.username = username;
    this.article_url = article_url;
    this.category = category;
}

/**
 * Default constructor! Do not use this constructor! Only used for test
 * purposes!
 * */

public Rating() {
    article_url = "";
    version = "";
    d1 = new Date();
    rating = 0;
    experience = 1;
    username = "testUser";
}

// Returns the date where the rating is given
public long getDate() {
    return d1.getTime();
}

// Returns the username for the user that created this rating
public String getUserName() {
    return username;
}

/*******************************
 * getExp returns the experience that was had when this rating was collected 2
 * marks that the experience is not yet given 1 marks a positive experience 0
 * marks a negative
 */

public int getExperience() {
    return experience;
}

// Change the experience with this rating
public void setExperience(int experience) {
    this.experience = experience;
}

******************
 * getInteraction returns the if it was a positive or negative interaction had
 * 2 marks that the experience is not yet given 1 marks a positive experience
 */
```
public int getInteractionRating() {
    return interactionRating;
}

public int getInteractionCategory() {
    return interactionCategory;
}

// Change the interactionRating with this rating
public void setRatingInteraction(int interaction) {
    interactionRating = interaction;
}

public void setInteractionCategory(int interaction) {
    interactionCategory = interaction;
}

// Returns the URL of the article
public String getURL() {
    return article_url;
}

// Returns the rating that was given in this rating
public int getRating() {
    return rating;
}

public Category getCategory() {
    return category;
}

// Returns version of the article where the rating was given
public String getVersion() {
    return version;
}

// Used to prettyPrint the rating
// Mainly used for debugging
public String prettyPrint() {
    String toReturn = null;
    toReturn = "Rating: \t" + getRating() + "\tUser: \t" + username + "\t\tExp: \t" + experience + "\tDate: \t" + \
    + d1.toString() + "\tVersion: \t" + getVersion() + "\tURL: \t" + getURL() + "\n";
    return toReturn;
}
The rating Calculator puts together a rating based on the active ratings in the session Rating DB and the trust values in RoT.

The average is calculated on a 5% trimmed mean based on David Wagner’s Resilient Aggregation in Sensor Networks.

public class RatingCalculator {
    // The ratings extracted in this session
    ArrayList<Rating> sessionRatingDB;

    // Array list to simulate the sensor readings
    ArrayList<Integer> ratings;

    // The associated WoT
    WoT wot;

    public RatingCalculator (SessionRatingDB sessionRatingDB, WoT wot) {
        this.sessionRatingDB = new ArrayList<Rating>();
        for (int i = sessionRatingDB.size() - 1; i >= 0; i--){
            this.sessionRatingDB.add(sessionRatingDB.elementAt(i));
        }
        ratings = new ArrayList<Integer>();
        this.wot = wot;
    }

    public double[] computeAverage() {
        double[] categories = new double[10];
        // DecimalFormat used to an integer
        DecimalFormat df = new DecimalFormat("##");
        for (Rating r : sessionRatingDB) {
            double trustValue = wot.getTrustValueOfUser(r.getUserName(), r.getCategory().getType());
            if (trustValue > 0.0) {
                temp.add(r);
            }
        }
        sessionRatingDB = temp;
        for (Rating r : sessionRatingDB) {
            double trustValue = wot.getTrustValueOfUser(r.getUserName(), r.getCategory().getType());
            categories[r.getCategory().getType().ordinal()] += trustValue;
        }
    }
}
int numbers_of_copies_to_insert = (int) (double) new Double(df.format(trustValue + 100));
int rating_to_be_inserted = r.getRating();
// insert the sensors
for (int j = 0; j < numbers_of_copies_to_insert; j++) {
    ratings.add(rating_to_be_inserted);
}

// Convert the Rating array to an int array
int[] ratings_integer_array_untrimmed = new int[ratings.size()];
for (int i = 0; i < ratings_integer_array_untrimmed.length; i++) {
    ratings_integer_array_untrimmed[i] = ratings.get(i);
}

// sort the ArrayList
Arrays.sort(ratings_integer_array_untrimmed);
// trim off 5%
int size_of_arraylist = ratings.size();
double threshold = (double) size_of_arraylist * 0.05;
int five_percent_threshold = (int) (double) new Double(df.format(threshold));

// Create an integer array that is 10% shorter of the original.
// to fit an array that is trimmed 5% at each end.
Integer[] rating_integer_array_trimmed = new Integer[ratings_integer_array_untrimmed.length
- 2 * five_percent_threshold];
int sum = 0;
for (int i = 0; i < rating_integer_array_trimmed.length; i++) {
    rating_integer_array_trimmed[i] = ratings_integer_array_untrimmed[i + five_percent_threshold];
    // Updating the sum
    sum += rating_integer_array_trimmed[i];
}

double average = ((double) sum) / ((double) rating_integer_array_trimmed.length);
double catAverage = 0.0;
double catType = -1.0;
double totalTrust = 0.0;
for (int i = 0; i < categories.length && sessionRatingDB.size() > 0; i++) {
    totalTrust += categories[i];
    if (categories[i] > catAverage) {
        catAverage = categories[i];
        catType = (double) i;
    }
}
catAverage /= totalTrust;
// catAverage = catAverage \% (int) catAverage;

double[] composite = new double[3];
composite[0] = average;
composite[1] = catAverage;
composite[2] = catType;
//System.out.println("Composite0 " + composite[0] + ", composite1 " + composite[1] + ", composite2 " + composite[2]);

// compute average
return composite;

A.1.11 RatingHistory.java

package rating;

świadcza i jest częścią systemu Rekomendatora Wikipedia, który jest używany do zautomatyzowania recenzji w Wikipedii.
Copyright (C) 2007 by Thomas Rune Korsgaard, tkorsgaard@gmail.com

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
import java.io.Serializable;
import java.util.*;

public class RatingHistory implements Serializable {
    private final long month_millis = new Long("25920000000"); // 30*24*60*60*1000
    private final long halfYear_millis = new Long("15552000000"); // 6*30*24*60*60*1000
    private final long year_millis = new Long("31104000000"); // 12*30*24*60*60*1000
    private HashMap<String, Rating> history;
    private double xValue;

    public RatingHistory() {
        history = new HashMap<String, Rating>();
        xValue = 0.0;
    }

    // Returns false if this is a duplicate rating, ie has been entered before.
    // Returns true if this is a unique rating.
    public boolean insertRating(Rating r) {
        if (history.containsKey(r.getURL())) {
            System.out.println("Duplicate rating by " + r.getUserName() + " found and rejected.");
            return false;
        }
        history.put(r.getURL(), r);
        System.out.println("New rating by " + r.getUserName() + " found and inserted.");
        updateTrustDynamics();
        return true;
    }

    public double getXValue() {
        return xValue;
    }

    private void updateTrustDynamics() {
        Date now = new Date();
        long thisTime = now.getTime();
        double positive_half_year = 0, positive_one_month = 0,
                negative_one_month = 0, negative_half_year = 0;
        // Set month, halfYear and year limits
        long month = thisTime - month_millis;
        long halfYear = thisTime - halfYear_millis;
        long year = thisTime - year_millis;

        Set entries = history.entrySet();
        Iterator iter = entries.iterator();
        while (iter.hasNext()) {
            Rating rating = (Rating)((Map.Entry)iter.next()).getValue();
            long ratingAge = rating.getDate();
            // thisTime -
            double categoryWeighting = 1.0;
            if (rating.getInteractionCategory() == 1) categoryWeighting = 1.5;
            if (rating.getInteractionCategory() == 2) categoryWeighting = 0.5;

            if (rating.getInteractionRating() == 1) {
                if (ratingAge > month) positive_one_month += categoryWeighting;
                if (ratingAge < month && ratingAge > halfYear) positive_half_year += categoryWeighting;
                if (ratingAge < halfYear && ratingAge > year) positive_one_year += categoryWeighting;
            } else if (rating.getInteractionRating() == 0) {
                if (ratingAge > month) negative_one_month += categoryWeighting;
            }
        }
    }
}

A.1 Code


```java
if (ratingAge < month && ratingAge > halfYear)
    negative_half_year += categoryWeighting;
if (ratingAge < halfYear && ratingAge > year)
    negative_one_year += categoryWeighting;

xValue = 0.1 * positive_one_month + 0.05 * positive_half_year + 0.025 * positive_one_year - 0.1
    * negative_one_month - 0.05 * negative_half_year - 0.025 * negative_one_year;

/∗
∗ Make sure −1.0 ≤ x ≤ 1.0 or y will go from 0 → 1 → 0 as f(2.0) = f(-2.0) = 0
∗ if (xValue > 1.0)
   xValue = 1.0;
else if (xValue < -1.0)
   xValue = -1.0;
*/
```

A.1.12 SessionRatingDB.java

```java
package rating;

/**
 * WRS – Wikipedia Recommender System is a collaborative recommender system
 * used to rate the articles on the Wikipedia.
 * Copyright (C) 2007 by Thomas Rune Korsgaard, tkorsgaard@gmail.com
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

import java.util.ArrayList;
import java.util.logging.Logger;

/**
 * SessionRatingDB is a class for storing the signatures parsed from the
 * wikipedia. It works like a FIFO stack, with push() and pop methods.
 * @author Thomas Rune Korsgaard.
 */
public class SessionRatingDB {
    private static Logger log = Logger.getLogger(SessionRatingDB.class.getName());
    private boolean hasOwner = false;
    private ArrayList<Rating> ratings;

    public SessionRatingDB() {
        ratings = new ArrayList<Rating>();
    }

    /**
     * push() inserts a rating into the arraylist
     * @param r
     */
    public synchronized void push(Rating r) {
        boolean success = false;
        success = ratings.add(r);
        if (!success) {
            log.severe("Insertion of Rating Failed, average may not be correct!");
        }
    }

    /**
     * pop() removes a rating from the ArrayList
     */
```
public synchronized Rating pop() {
    Rating r = ratings.remove(0);
    return r;
}

public Rating hasOwner(String owner) {
    for (int i = 0; i < ratings.size(); i++) {
        if (owner.equals(ratings.get(i).getUserName())) {
            hasOwner = true;
            return ratings.remove(i);
        }
    }
    return null;
}

public boolean hasOwner() {
    return hasOwner;
}

public int size() {
    return ratings.size();
}

public void prettyPrintAllSignatures() {
    String toReturn = System.out.println("The following information was extracted from the page:");
    for (Rating i : ratings) {
        System.out.println("User: " + i.getUserName() + " with a rating of " + i.getRating() + " in " + i.getCategory().toString());
        toReturn += i.prettyPrint();
    }
    return toReturn;
}

public Rating elementAt(int g) {
    return ratings.get(g);
}

package remote;

A.1.13 EmbeddedApplet.java

package remote;

ırken edit page.
Appendix

```java
import java.applet.Applet;
import java.awt.Component;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Frame;
import java.awt Insets;
import java.awt.LayoutManager;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOException;
import java.io.IOSS
FeedbackInterface stub = (FeedbackInterface) registry.lookup("FeedbackInterface"); // String response = stub.sayHello();
System.out.println("No button dimmed!");
stub.clickNo();
button.YES.setEnabled(false);
}
catch (Exception e) {
e.printStackTrace();
}
if (ae.getSource() == button_rate7) {
  String host = null;
  try {
    Registry registry = LocateRegistry.getRegistry(host);
    FeedbackInterface stub = (FeedbackInterface) registry.lookup("FeedbackInterface"); // String response = stub.sayHello();
    button_rate1.setEnabled(false);
    button_rate2.setEnabled(false);
    button_rate3.setEnabled(false);
    button_rate4.setEnabled(false);
    button_rate5.setEnabled(false);
    button_rate6.setEnabled(false);
    button_rate8.setEnabled(false);
    button_rate9.setEnabled(false);
    System.out.println("No button dimmed!");
    stub.clickRating(7, categoryList.getSelectedIndex() - 1);
  } catch (Exception e) {
e.printStackTrace();
  }
}
if (ae.getSource() == button_rate8) {
  String host = null;
  try {
    Registry registry = Locate Registry . get Registry ( host );
    FeedbackInterface stub = ( FeedbackInterface ) registry . lookup ( " FeedbackInterface " ); // String response = stub . sayHello () ;
    button_rate1.setEnabled(false);
    button_rate2.setEnabled(false);
    button_rate3.setEnabled(false);
    button_rate4.setEnabled(false);
    button_rate5.setEnabled(false);
    button_rate6.setEnabled(false);
    button_rate7.setEnabled(false);
    button_rate9.setEnabled(false);
    System.out.println("No button dimmed!");
    stub.clickRating(8, categoryList.getSelectedIndex() - 1);
  } catch (Exception e) {
e.printStackTrace();
  }
}
if (ae.getSource() == button_rate1) {
  String host = null;
  try {
    Registry registry = LocateRegistry.getRegistry(host);
    FeedbackInterface stub = (FeedbackInterface) registry.lookup("FeedbackInterface"); // String response = stub.sayHello();
    button_rate7.setEnabled(false);
    button_rate2.setEnabled(false);
    button_rate3.setEnabled(false);
    button_rate4.setEnabled(false);
    button_rate5.setEnabled(false);
    button_rate6.setEnabled(false);
    button_rate8.setEnabled(false);
    button_rate9.setEnabled(false);
    System.out.println("No button dimmed!");
    stub.clickRating(1, categoryList.getSelectedIndex() - 1);
  } catch (Exception e) {
e.printStackTrace();
  }
}
if (ae.getSource() == button_rate9) {
  String host = null;
  try {
    Registry registry = LocateRegistry.getRegistry(host);
    FeedbackInterface stub = (FeedbackInterface) registry.lookup("FeedbackInterface"); // String response = stub.sayHello();
    button_rate1.setEnabled(false);
    button_rate2.setEnabled(false);
    button_rate3.setEnabled(false);
    button_rate4.setEnabled(false);
    button_rate5.setEnabled(false);
    button_rate6.setEnabled(false);
    button_rate8.setEnabled(false);
    button_rate9.setEnabled(false);
  } catch (Exception e) {
e.printStackTrace();
  }
button_rate7.setEnabled(false);
stub.clickRating(9, categoryList.getSelectedIndex() - 1);
} catch (Exception e) {
e.printStackTrace();
}
}
if (ae.getSource() == button_rate2) {
String host = null;
try {
Registry registry = LocateRegistry.getRegistry(host);
FeedbackInterface stub = (FeedbackInterface) registry.lookup("FeedbackInterface");
// String response = stub.sayHello();
button_rate1.setEnabled(false);
button_rate7.setEnabled(false);
button_rate3.setEnabled(false);
button_rate4.setEnabled(false);
button_rate5.setEnabled(false);
button_rate6.setEnabled(false);
button_rate8.setEnabled(false);
button_rate9.setEnabled(false);
stub.clickRating(2, categoryList.getSelectedIndex() - 1);
} catch (Exception e) {
e.printStackTrace();
}
}
if (ae.getSource() == button_rate3) {
String host = null;
try {
Registry registry = LocateRegistry.getRegistry(host);
FeedbackInterface stub = (FeedbackInterface) registry.lookup("FeedbackInterface");
// String response = stub.sayHello();
button_rate1.setEnabled(false);
button_rate2.setEnabled(false);
button_rate3.setEnabled(false);
button_rate4.setEnabled(false);
button_rate5.setEnabled(false);
button_rate6.setEnabled(false);
button_rate7.setEnabled(false);
button_rate8.setEnabled(false);
button_rate9.setEnabled(false);
stub.clickRating(3, categoryList.getSelectedIndex() - 1);
} catch (Exception e) {
e.printStackTrace();
}
}
if (ae.getSource() == button_rate4) {
String host = null;
try {
Registry registry = LocateRegistry.getRegistry(host);
FeedbackInterface stub = (FeedbackInterface) registry.lookup("FeedbackInterface");
// String response = stub.sayHello();
button_rate1.setEnabled(false);
button_rate2.setEnabled(false);
button_rate3.setEnabled(false);
button_rate4.setEnabled(false);
button_rate5.setEnabled(false);
button_rate6.setEnabled(false);
button_rate7.setEnabled(false);
button_rate8.setEnabled(false);
button_rate9.setEnabled(false);
stub.clickRating(4, categoryList.getSelectedIndex() - 1);
} catch (Exception e) {
e.printStackTrace();
}
}
if (ae.getSource() == button_rate5) {
String host = null;
try {
Registry registry = LocateRegistry.getRegistry(host);
FeedbackInterface stub = (FeedbackInterface) registry.lookup("FeedbackInterface");
// String response = stub.sayHello();
button_rate1.setEnabled(false);
button_rate2.setEnabled(false);
button_rate3.setEnabled(false);
button_rate4.setEnabled(false);
button_rate5.setEnabled(false);
button_rate6.setEnabled(false);
button_rate7.setEnabled(false);
button_rate8.setEnabled(false);
button_rate9.setEnabled(false);
stub.clickRating(5, categoryList.getSelectedIndex() - 1);
} catch (Exception e) {
e.printStackTrace();
}
A.1 Code

```java
A.1 Code

```
add(label_3);
categoryList = new JComboBox(categoryStrings);
categoryList.setSelectedIndex(0);
add(categoryList);
categoryList.addActionListener(this);
setSize(getPreferredSize());
}

public static void main(String args[]) {
  EmbeddedApplet applet = new EmbeddedApplet();
  Frame window = new Frame("test");
  window.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
      System.exit(0);
    }
  });
  applet.init();
  window.add("Center", applet);
  window.pack();
  window.setVisible(true);
}

class AppletLayout implements LayoutManager {
  public AppletLayout() {
  }
  public void addLayoutComponent(String name, Component comp) {
  }
  public void removeLayoutComponent(Component comp) {
  }
  public Dimension preferredLayoutSize(Container parent) {
    Dimension dim = new Dimension(0, 0);
    Insets insets = parent.getInsets();
    dim.width = 250 + insets.left + insets.right;
    dim.height = 214 + insets.top + insets.bottom;
    return dim;
  }
  public Dimension minimumLayoutSize(Container parent) {
    Dimension dim = new Dimension(0, 0);
    return dim;
  }
  public void layoutContainer(Container parent) {
    Insets insets = parent.getInsets();
    Component c;
    c = parent.getComponent(0);
    if (c.isVisible()) {
      c.setBounds(insets.left + 8, insets.top + 48, 120, 24);
    }
    c = parent.getComponent(1);
    if (c.isVisible()) {
      c.setBounds(insets.left + 128, insets.top + 48, 120, 24);
    }
    c = parent.getComponent(2);
    if (c.isVisible()) {
      c.setBounds(insets.left + 8, insets.top + 8, 240, 32);
    }
    c = parent.getComponent(13);
    if (c.isVisible()) {
      c.setBounds(insets.left + 8, insets.top + 80, 240, 32);
    }
    c = parent.getComponent(3);
    if (c.isVisible()) {
      c.setBounds(insets.left + 200, insets.top + 144, 48, 32);
    }
    c = parent.getComponent(4);
    if (c.isVisible()) {
      c.setBounds(insets.left + 8, insets.top + 176, 48, 32);
    }
    c = parent.getComponent(5);
    if (c.isVisible()) {
      c.setBounds(insets.left + 56, insets.top + 144, 48, 32);
    }
    c = parent.getComponent(6);
  }
A.1 Code

if (c.isVisible()) {
    c.setBounds(insets.left + 56, insets.top + 176, 48, 32);
}  
c = parent.getComponent(7);  
if (c.isVisible()) {
    c.setBounds(insets.left + 104, insets.top + 144, 48, 32);
}  
c = parent.getComponent(8);  
if (c.isVisible()) {
    c.setBounds(insets.left + 104, insets.top + 176, 48, 32);
}  
c = parent.getComponent(9);  
if (c.isVisible()) {
    c.setBounds(insets.left + 152, insets.top + 144, 48, 32);
}  
c = parent.getComponent(10);  
if (c.isVisible()) {
    c.setBounds(insets.left + 152, insets.top + 176, 48, 32);
}  
c = parent.getComponent(11);  
if (c.isVisible()) {
    c.setBounds(insets.left + 8, insets.top + 144, 48, 32);
}  
c = parent.getComponent(12);  
if (c.isVisible()) {
    c.setBounds(insets.left + 8, insets.top + 112, 240, 24);
}

A.1.14 FeedbackInterface.java

package remote;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface FeedbackInterface extends Remote {
    // String sayHello() throws RemoteException;
    void clickYes() throws RemoteException;
    void clickNo() throws RemoteException;
    void clickRating(int rating, int category) throws RemoteException;
}

A.1.15 Reviewer.java

package trust;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Reviewer extends Remote {
    // String sayHello() throws RemoteException;
    void countClicks() throws RemoteException;
    void countErrors() throws RemoteException;
}
import java.security.cert.Certificate;
import java.io.Serializable;
import java.util.Vector;
import rating.Rating;
import rating.RatingHistory;
import rating.Category;
import static tools.SecurityProvider;
public class Reviewer implements Serializable {
    private static final long serialVersionUID = -6356205467037013516L;
    // The two states that the Curve can be in: Optimistic or Cautious
    public enum Curve {
        OPT, CAU
    }
    // The two states a reviewer can be in: Trust and Distrust
    public enum State {
        TRUST, DISTRUST
    }
    // Measuring the number of positive and negative interactions
    private int[] noOfPosInteractions;
    private int[] noOfNegInteractions;
    // Measuring the number of positive and negative experiences
    Variables
    private String username;
    private double[] xValue;
    private double[] nValue;
    private double[] trustValue;
    private Curve[] curve;
    private State[] state;
    // Previous ratings
    private RatingHistory rh;
    // Cache certificate
    private Certificate cachedCertificate;
    /**
     * Constructor to generate an initial Reviewer with no prior history
     *
     * public Reviewer() {
     * // Initialize arrays
     * noOfPosInteractions = new int[10];
     * noOfNegInteractions = new int[10];
     * curve = new Curve[10];
     * state = new State[10];
     * xValue = new double[10];
     * nValue = new double[10];
     * trustValue = new double[10];
     * // Initialize values in arrays
     * for (int n = 0; n < 10; n++) {
     * noOfPosInteractions[n] = 0;
     * noOfNegInteractions[n] = 0;
     * curve[n] = Curve.OPT;
     * state[n] = State.TRUST;
     * xValue[n] = 0.0;
     * }
nValue[n] = 1.0;
trustValue[n] = 0.0;
}

rh = new RatingHistory();
username = ""
/

Reviewer constructor ONLY for testing purposes!
*/
*
/**
public Reviewer(double trustValue, String username) {
    this.trustValue = trustValue;
    this.username = username;
    rh = new RatingHistory();
    noOfPosInteractions = 2;
    noOfNegInteractions = 1;
    curve = Curve.OPT;
    state = State.TRUST;
    xValue = 0.0;
    nValue = 0.0;
}
/**
** Inserts a rating into the history
*
@param r

public void insertRating(Rating r) {
    */
    * Make sure to stop the trust model from being updated in case of an old rating.
    */
    if (!rh.insertRating(r)) {
        return;
    }
    int index = r.getCategory().getType().ordinal();
    xValue[index] = rh.getXValue();
    // update State value
    if (xValue[index] >= 0.0)
        state[index] = State.TRUST;
    if (xValue[index] < 0.0)
        state[index] = State.DISTRUST;
    // If it is a positive exp
    if (r.getExperience() == 1) {
        positiveExp(index);
    }
    // If it is a negative exp
    if (r.getExperience() == 0) {
        negativeExp(index);
    }
    // Update the trust value
    calcTrustValue(index);
}

// represents the consequences of a positive experience - positive feedback
private void positiveExp(int index) {
    */
    * User in trust and an optimistic curve
    */
    if (curve[index] == Curve.OPT && state[index] == State.TRUST) {
        nValue[index] += 0.1;
        return;
    }
    */
    * User in distrust and an optimistic curve
    */
    if (curve[index] == Curve.OPT && state[index] == State.DISTRUST) {
        nValue[index] += 0.1;
        return;
    }
    */
    * User in trust and a cautious curve
    */
    if (curve[index] == Curve.CAU && state[index] == State.TRUST) {
        // If the curve changes from Cautious to neutral
        if (nValue[index] == 1.0) {
            curve[index] = Curve.OPT;
            nValue[index] -= 0.1;
        } else {
            nValue[index] -= 0.1;
        }
        return;
    }
    */
    * User in distrust and a cautious curve
    */
    if (curve[index] == Curve.CAU && state[index] == State.DISTRUST) {
        // If the curve changes from cautious to neutral
        if (nValue[index] == 1.0) {
            curve[index] = Curve.OPT;
            nValue[index] += 0.1;
        } else {
            nValue[index] -= 0.1;
        }
    }
private void negativeExp(int index) {
    // user in trust and a cautious curve
    if (curve[index] == Curve.CAU & state[index] == State.TRUST) {
        nValue[index] += 0.1;
        return;
    }
    // user in distrust and a cautious curve
    if (curve[index] == Curve.CAU & state[index] == State.DISTRUST) {
        nValue[index] += 0.1;
        return;
    }
    // user in distrust and an optimistic curve
    if (curve[index] == Curve.OPT & state[index] == State.DISTRUST) {
        if (nValue[index] == 1.0) {
            curve[index] = Curve.CAU;
            nValue[index] += 0.1;
        } else {
            nValue[index] -= 0.1;
        }
        return;
    }
    // User in trust and an optimistic curve
    if (curve[index] == Curve.OPT & state[index] == State.TRUST) {
        if (nValue[index] == 1.0) {
            curve[index] = Curve.CAU;
            nValue[index] += 0.1;
        } else {
            nValue[index] -= 0.1;
        }
    }
}

// Calculates the actual trust value
private void calcTrustValue(int index) {
    System.out.println(" Trust value for " + username + " in classification " + new Category(index) + " updated from " + trustValue[index] + " to ");
    // trust + opt. F4
    if (curve[index] == Curve.OPT & state[index] == State.TRUST) {
        trustValue[index] = Math.pow((-1.0 * Math.abs(Math.pow(Math.abs(xValue[index] - 1.0), nValue[index])) + 1.0), (1.0 / nValue[index])) + 1.0;
        System.out.println(trustValue[index]);
    }
    // trust + cau. F2
    if (curve[index] == Curve.CAU & state[index] == State.TRUST) {
        trustValue[index] = Math.pow((-1.0 * Math.abs(Math.pow(Math.abs(xValue[index] - 1.0), nValue[index])) + 1.0), (1.0 / nValue[index])) + 1.0;
        System.out.println(trustValue[index]);
    }
    // distrust + opt. F1
    if (curve[index] == Curve.OPT & state[index] == State.DISTRUST) {
        trustValue[index] = Math.pow((-1.0 * Math.abs(Math.pow(Math.abs(xValue[index] + 1.0), nValue[index])) + 1.0), (1.0 / nValue[index]));
    }
    // distrust + cau. F3
    if (curve[index] == Curve.CAU & state[index] == State.DISTRUST) {
        trustValue[index] = Math.pow((-1.0 * Math.abs(Math.pow(Math.abs(xValue[index] + 1.0), nValue[index])) + 1.0), (1.0 / nValue[index]));
        System.out.println(trustValue[index]);
    }
}

// Returns the trustValue
public double getTrustValue(Category.Type type) {
    return trustValue[type.ordinal()];
}

public double getTrustValue() {
    double average = 0.0;
    int count = 0;
    while (count < trustValue.length) {
        average += trustValue[count];
        count++;
    }
    average /= count;
    return average;
}

// Returns the username
public String getUsername() {
    return username;
}

// Sets the username
public void setUsername(String username) {
    this.username = username;
}
A.1 Code

// Downloads the certificate from the web
public void getUpdatedCertificate() {
    this.cachedCertificate = SecurityProvider.getUserCertificate(username);
}

// Returns the cached certificate
public Certificate getCertificate() {
    return cachedCertificate;
}

A.1.16 TrustUpdater.java

package trust;

import java.util.ArrayList;
import rating.Rating;
import rating.SessionRatingDB;
import rating.Category;
import statictools.Serializer;

public class TrustUpdater {
    public enum Exp {
        YES, NO
    };
    public enum Vote {
        R1, R2, R3, R4, R5, R6, R7, R8, R9
    };
    private int exp;
    private Vote vote;
    private int rating, category;
    private SessionRatingDB srDB;
    private WoT ror;
    private double[] ratingServed;
    private ArrayList<Rating> ratingsToBeInserted;
    boolean clickedExp;
    boolean clickedInteraction;

    public TrustUpdater(SessionRatingDB sessionRatingDB, double ratingServed[], WoT ror) {
        srDB = sessionRatingDB;
        this.ratingServed = ratingServed;
        this.ror = ror;
        clickedExp = false;
        clickedInteraction = false;
    }
}

WRS - Wikipedia Recommender System is a collaborative recommender system used to rate the articles on the Wikipedia.
Copyright (C) 2007 by Thomas Rune Korsgaard, thorsgaard(a)gmail.com
This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

import java.util.ArrayList;
import rating.Rating;
import rating.SessionRatingDB;
import rating.Category;
import statictools.Serializer;

public class TrustUpdater {
public void ClickedYes() {
    /*
     * Look through sessionRatingDB and see which ratings ±1 to -1 of the
     * rating. //These should have their nValue adjusted Rating temp for (int i
     * =0; i<sessionDB.size(); i++) temp = sessionDB.elementAt(i); int floored_rating
     * = (int) Math.floor(ratingServed); // If the given rating is within the
     * ±1 threshold if (floored_rating−1 == temp.getRating() || floored_rating ==
     * temp.getRating()) || floored_rating+1 == temp.getRating()) {
     * temp.setExperience(1); // Insert the rating into the temp rating Array for
     * further refinement ratingsToBeInserted.add(temp); }
     */
    exp = 1;
    clickedExp = true;
    if (clickedExp && clickedInteraction) 
        updateAndInsertRatingsToROR();
}

public void ClickedNo() {
    exp = 0;
    clickedExp = true;
    if (clickedExp && clickedInteraction) 
        updateAndInsertRatingsToROR();
}

public void ClickedRating(int rating, int category) {
    this.rating = rating;
    this.category = category;
    clickedInteraction = true;
    if (clickedExp && clickedInteraction) 
        updateAndInsertRatingsToROR();
}

public void ClickedR1() {
    vote = Vote.R1;
    clickedInteraction = true;
    if (clickedExp && clickedInteraction) 
        updateAndInsertRatingsToROR();
}

public void ClickedR2() {
    vote = Vote.R2;
    clickedInteraction = true;
    if (clickedExp && clickedInteraction) 
        updateAndInsertRatingsToROR();
}

public void ClickedR3() {
    vote = Vote.R3;
    clickedInteraction = true;
    if (clickedExp && clickedInteraction) 
        updateAndInsertRatingsToROR();
}

public void ClickedR4() {
    vote = Vote.R4;
    clickedInteraction = true;
    if (clickedExp && clickedInteraction) 
        updateAndInsertRatingsToROR();
}

public void ClickedR5() {
    vote = Vote.R5;
    clickedInteraction = true;
    if (clickedExp && clickedInteraction) 
        updateAndInsertRatingsToROR();
}

public void ClickedR6() {
    vote = Vote.R6;
    clickedInteraction = true;
    if (clickedExp && clickedInteraction) 
        updateAndInsertRatingsToROR();
}

public void ClickedR7() {
    vote = Vote.R7;
    clickedInteraction = true;
    if (clickedExp && clickedInteraction) 
        updateAndInsertRatingsToROR();
}
updateAndInsertRatingsToROR();
}

public void ClickedR8()
{
  vote = Vote.R8;
  clickedInteraction = true;
  if (clickedExp & clickedInteraction)
  updateAndInsertRatingsToROR();
}

public void ClickedR9()
{
  vote = Vote.R9;
  clickedInteraction = true;
  if (clickedExp & clickedInteraction)
  updateAndInsertRatingsToROR();
}

public void updateAndInsertRatingsToROR()
{
  /**
   * 0 = Rating average, 1 = Category Percentage, 2 = Category Type, 3 = Experience (1 good, 0 bad)
   */
  double[] interaction = new double[4];
  interaction[3] = (double)exp;
  System.arraycopy(ratingServed, 0, interaction, 0, ratingServed.length);
  
  Rating temp;
  int rounded_rating = (int)Math.round(ratingServed[0]);
  for(int i = 0; i < srDB.size(); i++)
  temp = srDB.elementAt(i);
  // Dealing with the experience.
  if ((rounded_rating - 1 == temp.getRating()) || rounded_rating == temp.getRating() || rounded_rating + 1 == temp.getRating())
  srDB.elementAt(i).setExperience((int)interaction[3]);

  if (rating_difference == 1) { srDB.elementAt(i).setRatingInteraction(1); }
  if (rating_difference > 1 && rating_difference < 3) { srDB.elementAt(i).setRatingInteraction(2); }
  if (rating_difference >= 3) { srDB.elementAt(i).setRatingInteraction(0); }
  if (ratingServed[2] >= 0) {
    givenCategory = Category.Type.values()[((int)ratingServed[2])];
    Category.Type selectedCategory = Category.Type.values()[category];
    if (selectedCategory == temp.getCategory().getType()) {
      srDB.elementAt(i).setInteractionCategory(0);
    } else {
      if (selectedCategory == givenCategory) {
        srDB.elementAt(i).setInteractionCategory(1);
      } else {
        srDB.elementAt(i).setInteractionCategory(2);
      }
    }
  }

  // Dealing with the interactions R1
  if (vote == Vote.R1 && (temp.getRating() == 1 || temp.getRating() == 2)) {
    srDB.elementAt(i).setRatingInteraction(1);
  }
  if (vote == Vote.R1 && (temp.getRating() >= 2 && temp.getRating() < 5)) {
    srDB.elementAt(i).setRatingInteraction(2);
  }
  if (vote == Vote.R1 && (temp.getRating() >= 5)) {
    srDB.elementAt(i).setRatingInteraction(0);
  }

  // Dealing with the interactions R2
  if (vote == Vote.R2 && (temp.getRating() == 1 || temp.getRating() == 2 || temp.getRating() == 3)) {
    srDB.elementAt(i).setRatingInteraction(1);
  }
if (vote == Vote.R3 && (temp.getRating() >= 3 && temp.getRating() < 6)) {
    srDB.elementAt(i).setRatingInteraction(2);
}
if (vote == Vote.R2 && (temp.getRating() >= 6)) {
    srDB.elementAt(i).setRatingInteraction(0);
}
// Dealing with the interactions R3
if (vote == Vote.R3 && (temp.getRating() == 2 || temp.getRating() == 3 || temp.getRating() == 4)) {
    srDB.elementAt(i).setRatingInteraction(1);
}
if (vote == Vote.R3 && (temp.getRating() == 7 || temp.getRating() == 8)) {
    srDB.elementAt(i).setRatingInteraction(2);
}
if (vote == Vote.R3 && (temp.getRating() >= 7)) {
    srDB.elementAt(i).setRatingInteraction(0);
}
// Dealing with the interactions R4
if (vote == Vote.R4 && (temp.getRating() == 3 || temp.getRating() == 4 || temp.getRating() == 5)) {
    srDB.elementAt(i).setRatingInteraction(1);
}
if (vote == Vote.R4 && (temp.getRating() == 3 || temp.getRating() == 6)) {
    srDB.elementAt(i).setRatingInteraction(2);
}
if (vote == Vote.R4 && (temp.getRating() == 8)) {
    srDB.elementAt(i).setRatingInteraction(0);
}
// Dealing with the interactions R5
if (vote == Vote.R5 && (temp.getRating() == 4 || temp.getRating() == 5 || temp.getRating() == 6)) {
    srDB.elementAt(i).setRatingInteraction(1);
}
if (vote == Vote.R5 && (temp.getRating() == 4 || temp.getRating() == 7)) {
    srDB.elementAt(i).setRatingInteraction(2);
}
if (vote == Vote.R5 && (temp.getRating() == 9)) {
    srDB.elementAt(i).setRatingInteraction(0);
}
// Dealing with the interactions R6
if (vote == Vote.R6 && (temp.getRating() == 5 || temp.getRating() == 6 || temp.getRating() == 7)) {
    srDB.elementAt(i).setRatingInteraction(1);
}
if (vote == Vote.R6 && (temp.getRating() == 5 || temp.getRating() == 9)) {
    srDB.elementAt(i).setRatingInteraction(2);
}
if (vote == Vote.R6 && (temp.getRating() == 7)) {
    srDB.elementAt(i).setRatingInteraction(0);
}
// Dealing with the interactions R7
if (vote == Vote.R7 && (temp.getRating() == 6 || temp.getRating() == 7 || temp.getRating() == 8)) {
    srDB.elementAt(i).setRatingInteraction(1);
}
if (vote == Vote.R7 && (temp.getRating() == 6 || temp.getRating() == 9)) {
    srDB.elementAt(i).setRatingInteraction(2);
}
if (vote == Vote.R7 && (temp.getRating() >= 3)) {
    srDB.elementAt(i).setRatingInteraction(0);
}
// Dealing with the interactions R8
if (vote == Vote.R8 && (temp.getRating() == 7 || temp.getRating() == 8 || temp.getRating() == 9)) {
    srDB.elementAt(i).setRatingInteraction(1);
}
if (vote == Vote.R8 && (temp.getRating() == 5 || temp.getRating() == 6)) {
    srDB.elementAt(i).setRatingInteraction(2);
}
if (vote == Vote.R8 && (temp.getRating() <= 4)) {
    srDB.elementAt(i).setRatingInteraction(0);
}
// Dealing with the interactions R9
if (vote == Vote.R9 && (temp.getRating() == 8 || temp.getRating() == 9)) {
    srDB.elementAt(i).setRatingInteraction(1);
}
if (vote == Vote.R9 && (temp.getRating() == 7 || temp.getRating() == 6)) {
    srDB.elementAt(i).setRatingInteraction(2);
}
if (vote == Vote.R9 && (temp.getRating() <= 5)) {
    srDB.elementAt(i).setRatingInteraction(0);
}
for (int j = 0; j < srDB.size(); j++) {
    // If the user already exists in the WoT
    if (ror.hasUser(srDB.elementAt(j).getUserName())) {
        // Insert new rating for " + srDB.elementAt(j).getUserName() + "
        ror.insertRatingToExistingUser(srDB.elementAt(j));
    } else if (!ror.isOwner(srDB.elementAt(j).getUserName())) {
        Reviewer newReviewer = new Reviewer();
        newReviewer.setUsername(srDB.elementAt(j).getUserName());
        ror.insertReviewer(newReviewer);
        ror.insertRatingToExistingUser(srDB.elementAt(j));
    }
}

// Write the new calculated trust values to the disk!
Serializer.writeRoRToDisk(ror);
}
public void insertInteraction(String url, double[] interaction) {
    interactionHistory.insertInteraction(url, interaction);
}

// Inserts a new reviewer into the WoT
public void insertReviewer(Reviewer rv) {
    System.out.println("Inserting reviewer: " + rv.getUsername());
    wot.put(rv.getUsername(), rv);
}

public void insertRatingToExistingUser(Rating r) {
    if (hasUser(r.getUsername())) {
        // Insert the rating
        System.out.println("Found user, inserting rating.");
        wot.get(r.getUsername()).insertRating(r);
    }
}

// Return the trust value for a given user
public double getTrustValueOfUser(String username, Category.Type type) {
    double trustValue = 0.0;
    if (wot.containsKey(username))
        trustValue = wot.get(username).getTrustValue(type);
    System.out.println("We trust " + username + " for " + trustValue);
    return trustValue;
}

// Update certificates from the web.
public void updateCertificates() {
    for (Iterator i = wot.values().iterator(); i.hasNext();)
        ((Reviewer) i.next()).getUpdatedCertificate();
}

public void updateCertificate(String user) {
    wot.get(user).getUpdatedCertificate();
}

// Get a certificate from the cache
public Certificate getCertificateFromUsername(String username) {
    System.out.println("Encountered " + username);
    try {
        return wot.get(username).getCertificate();
    } catch (NullPointerException npe) {
        System.out.println("First time encountering " + username + " , adding to WoT.");
        Reviewer rv = new Reviewer();
        rv.setUsername(username);
        insertReviewer(rv);
        wot.get(username).getUpdatedCertificate();
        return wot.get(username).getCertificate();
    }
}

public boolean hasUser(String username) {
    return wot.containsKey(username);
}

public boolean isOwner(String username) {
    return owner.equals(username);
}

public String toString(String username) {
    String output = "User " + i + " is " + rev.getUsername() + " with an average trust value of " + r + ".
    return output;
}

A.1.18 MultiPartformOutputStream.java
/**
 * Created by IntelliJ IDEA.
 * User: s011531
 * Date: 04-03-2009
 * Time: 11:34:35
 * To change this template use File | Settings | File Templates.
 */
import java.io.*;
import java.net.*;
/*
<code>MultipartFormDataOutputstream</code> is used to write
"multipart/form-data" to a <code>java.net.URLConnection</code> for
POSTing. This is primarily for file uploading to HTTP servers.
* @since JDK1.3
*/
public class MultiPartFormOutputStream {
    /**
     * The line end characters.
     * /
    private static final String NEWLINE = "\r\n";
    /**
     * The boundary prefix.
     * /
    private static final String PREFIX = "--";
    /**
     * The output stream to write to.
     * /
    private DataOutputStream out = null;
    /**
     * The multipart boundary string.
     * /
    private String boundary = null;
    /**
     * Creates a new <code>MultipartFormDataOutputstream</code> object using
the specified output stream and boundary. The boundary is required
 to be created before using this method, as described in the
 description for the <code>getContentType(String)</code> method.
 The boundary is only checked for <code>null</code> or empty string,
 but it is recommended to be at least 6 characters. (Or use the
 static createBoundary() method to create one.)
 * @param os the output stream
 * @param boundary the boundary
 * @see #createBoundary()
 * @see #getContentType(String)
 */
    public MultiPartFormOutputStream(DataOutputStream os, String boundary) {
        if (os == null) {
            throw new IllegalArgumentException("Output stream is required. ");
        }
        if (boundary == null || boundary.length() == 0) {
            throw new IllegalArgumentException("Boundary string is required.");
        }
        this.out = new DataOutputStream(os);
        this.boundary = boundary;
    }
    /**
     * Writes a boolean field value.
     * @param name the field name (required)
     * @param value the field value
     * @throws java.io.IOException on input/output errors
     */
    public void writeField(String name, boolean value) throws java.io.IOException {
        writeField(name, Boolean.toString(value));
    }
    /**
     * Writes a double field value.
     * @param name the field name (required)
     * @param value the field value
     * @throws java.io.IOException on input/output errors
     */
    public void writeField(String name, double value) throws java.io.IOException {
        writeField(name, Double.toString(value));
    }
}
/**
 * Writes a float field value.
 * @param name the field name (required)
 * @param value the field value
 * @throws java.io.IOException on input/output errors
 */
public void writeField(String name, float value)
  throws java.io.IOException {
  writeField(name, Float.toString(value));
}

/**
 * Writes a long field value.
 * @param name the field name (required)
 * @param value the field value
 * @throws java.io.IOException on input/output errors
 */
public void writeField(String name, long value)
  throws java.io.IOException {
  writeField(name, Long.toString(value));
}

/**
 * Writes an int field value.
 * @param name the field name (required)
 * @param value the field value
 * @throws java.io.IOException on input/output errors
 */
public void writeField(String name, int value)
  throws java.io.IOException {
  writeField(name, Integer.toString(value));
}

/**
 * Writes a short field value.
 * @param name the field name (required)
 * @param value the field value
 * @throws java.io.IOException on input/output errors
 */
public void writeField(String name, short value)
  throws java.io.IOException {
  writeField(name, Short.toString(value));
}

/**
 * Writes a char field value.
 * @param name the field name (required)
 * @param value the field value
 * @throws java.io.IOException on input/output errors
 */
public void writeField(String name, char value)
  throws java.io.IOException {
  writeField(name, Character.toString(value));
}

/**
 * Writes a string field value. If the value is null, an empty string
 * is sent (""").
 * @param name the field name (required)
 * @param value the field value
 * @throws java.io.IOException on input/output errors
 */
public void writeField(String name, String value)
  throws java.io.IOException {
  if (name == null) { // cannot be null
    throw new IllegalArgumentException("Name cannot be null or empty.");
  }
  if (value == null) {
    value = "";
  }
  // boundary
  Content-Disposition: form-data; name="fieldName">
  \r\n  <value>
  \r\n  /* write boundary
  out.writeBytes(PREFIX);
  out.writeBytes(boundary);
  out.writeBytes(NEWLINE);
public void writeFile(String name, String mimeType, File file) throws java.io.IOException {
    if (file == null) {
        throw new IllegalArgumentException("File cannot be null.");
    } if (!file.exists()) {
        throw new IllegalArgumentException("File does not exist.");
    } if (file.isDirectory()) {
        throw new IllegalArgumentException("File cannot be a directory.");
    }
    writeFile(name, mimeType, file.getCanonicalPath(), new FileInputStream(file));
}

public void writeFile(String name, String mimeType, String fileName, InputStream is) throws java.io.IOException {
    if (is == null) {
        throw new IllegalArgumentException("Input stream cannot be null.");
    } if (fileName == null || fileName.length() == 0) {
        throw new IllegalArgumentException("File name cannot be null or empty.");
    } /*
    - boundary
    Content-Disposition: form-data; name="<field name>"; filename="<filename>"
    Content-Type: <mime-type>*/
    // write boundary
    out.writeBytes(PREFIX);
    out.writeBytes(boundary);
    out.writeBytes(NEWLINE);
    // write content header
    out.writeBytes("Content-Disposition: form-data; name=" + name + "\n" + fileName + "\n" + mimeType + ";");
    out.writeBytes(NEWLINE);
    if (mimeType != null) {
        out.writeBytes("Content-Type:");
        out.writeBytes(mimeType);
    }
    byte[] data = new byte[1024];
    int r;
    while ((r = is.read(data, 0, data.length)) != -1) {
        out.write(data, 0, r);
    }
    // close input stream, but ignore any possible exception for it
    try {
        is.close();
    } catch (Exception e) {
        out.writeBytes(NEWLINE);
        out.flush();
    }
}
public void writeFile(String name, String mimeType, String fileName, byte[] data) throws java.io.IOException {
    if (data == null) {
        throw new IllegalArgumentException("Data cannot be null.");
    }
    if (fileName == null || fileName.length() == 0) {
        throw new IllegalArgumentException("File name cannot be null or empty.");
    }
    //−−boundary
    out.writeBytes(PREFIX);
    out.writeBytes(boundary);
    out.writeBytes(NEWLINE);
    //write content header
    out.writeBytes("Content-Disposition: form-data; name="+name+"; filename="+fileName+");
    out.writeBytes(NEWLINE);
    if (mimeType != null) {
        out.writeBytes("Content-Type: "+mimeType);
        out.writeBytes(NEWLINE);
    }
    out.writeBytes(NEWLINE);
    //write content
    out.write(data, 0, data.length);
    out.writeBytes(NEWLINE);
    out.flush();
}

public void flush() throws java.io.IOException {
    //out.flush();
}

public void close() throws java.io.IOException {
    //write final boundary
    out.writeBytes(PREFIX);
    out.writeBytes(boundary);
    out.writeBytes(NEWLINE);
    out.flush();
    out.close();
}

public String getBoundary() {
    return this.boundary;
}

/**
 * Creates a new <code>java.net.URLConnection</code> object from the specified <code>java.net.URL</code>. This is a convenience method which will set the <code>doInput</code>, <code>doOutput</code>, <code>&lt;</code>code&gt;java.lang.IllegalArgumentException&lt;/code&gt; will be thrown.
 * @param name the field name
 * @param mimeType the file content type (optional, recommended)
 * @param fileName the file name (required)
 * @param data the file data
 * @throws java.io.IOException on input/output errors
 */
A.1 Code

```java
* <code>useCaches</code> and <code>defaultUseCaches</code> fields to
  * the appropriate settings in the correct order.
  *
  * @return a <code>java.net.URLConnection</code> object for the URL
  * @throws java.io.IOException on input/output errors
*/
public static URLConnection createConnection(URL url)
    throws java.io.IOException {
    URLConnection urlConn = url.openConnection();
    if (urlConn instanceof HttpURLConnection) {
        HttpURLConnection httpConn = (HttpURLConnection) urlConn;
        httpConn.setRequestMethod("POST");
    }
    urlConn.setDoInput(true);
    urlConn.setDoOutput(true);
    urlConn.setUseCaches(false);
    urlConn.setDefaultUseCaches(false);
    return urlConn;
}

/**
 * Creates a multipart boundary string by concatenating 20 hyphens (−) and
 * the hexadecimal (base−16) representation of the current time in
 * milliseconds.
 *
 * @return a multipart boundary string
 * @see #createBoundary()
 */
public static String createBoundary() {
    return "- - - - - - - - - - - - - - - - - - - - " +
        Long.toString(System.currentTimeMillis(), 16);
}

/**
 * Gets the content type string suitable for the
 * <code>java.net.URLConnection</code> which includes the multipart
 * boundary string. <br />*
 * <br />
 * This method is static because, due to the nature of the
 * <code>java.net.URLConnection</code> class, once the output stream
 * for the connection is acquired, it's too late to set the content
 * type (or any other request parameter). So one has to create a
 * multipart boundary string first before using this class, such as
 * with the <code>createBoundary</code>() method.
 *
 * @param boundary the boundary string
 * @return the content type string
 * @see #createBoundary()
 */
public static String getContentType(String boundary) {
    return "multipart/form-data; boundary=" + boundary;
}
```

A.1.19 RatingCleanOut.java

```java
package static tools;

import trust.Wot;
```

---

**WRS — Wikipedia Recommender System is a collaborative recommender system used to rate the articles on the Wikipedia.**

Copyright (C) 2007 by Thomas Rune Korsgaard, tkorsgaard(a)gmail.com

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

---
import java.util.Vector;

/**
 * RatingsCleanOut is a set of static tools that is used to clean out the
 * ratings that are on a page, and does not belong there for some reason.
 * @author 501564
 */

public class RatingCleanOut {

    /**
     * RemoveTitleMismatch removes the ratings where the title in the rating
don't match to title of the page that the rating is inserted into.
     * @return
     */
    public static Vector<String> RemoveRatingsTitleMismatch(String pageTitle, Vector<String> ratings) {
        Vector<String> cleanedRatings = new Vector<String>();
        while (!ratings.isEmpty()) {
            String firstElement = ratings.firstElement();
            // If the pageTitle matches the title in the rating
            if ((firstElement.split(";"))[5].equals(pageTitle)) {
                cleanedRatings.add(firstElement);
            }
            ratings.remove(0);
        }
        return cleanedRatings;
    }

    /**
     * RemoveRatingsBelowThreshold() removes the ratings where the Threshold is
     * below an accepted limit
     * @param acceptedThreshold
     * @param ratings
     * @param pageTitle
     * @return
     */
    public static Vector<String> RemoveRatingsBelowThreshold(double acceptedThreshold, Vector<String> ratings, String pageTitle) {
        Vector<String> cleanedRatings = new Vector<String>();
        String[] temp_string_array = null;
        // Work through the raw ratings
        while (!ratings.isEmpty()) {
            temp_string_array = ratings.firstElement().split(";");
            double calculatedThreshold = ThresholdCalculator(pageTitle, temp_string_array[3]);
            // If the calculated threshold is lower than the accepted, then
            // insert
            System.out.println("Calc: " + calculatedThreshold + " accepted: " + acceptedThreshold);
            if (calculatedThreshold < acceptedThreshold) {
                cleanedRatings.add(ratings.firstElement());
            }
            ratings.remove(0);
        }
        return cleanedRatings;
    }

    /**
     * RemoveUnvalidableRatings() removes the ratings where the signature cannot
     * be verified
     * @param ratings
     * @param ror
     * @return
     */
    public static Vector<String> RemoveUnvalidableRatings(Vector<String> ratings, WoT ror) {
        Vector<String> cleanedRatings = new Vector<String>();
        while (!ratings.isEmpty()) {
            if (SecurityProvider.validateRating(ratings.firstElement(), ror)) {
                cleanedRatings.add(ratings.firstElement());
            }
            ratings.remove(0);
        }
        return cleanedRatings;
    }
}
package statictools;

/**************************************************************************
 * WRS – Wikipedia Recommender System is a collaborative recommender system
 * used to rate the articles on the Wikipedia.
 * Copyright (C) 2007 by Thomas Rune Korsgaard, thorsgaard(s)gmail.com
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 **************************************************************************/

import java.io.*;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.security.InvalidKeyException;
import java.security.Key;
import java.security.KeyPair;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.security.cert.Certificate;
import java.security.cert.CertificateEncodingException;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import scone.util.tokenstream.HtmlTagToken;
import scone.util.tokenstream.HtmlTextToken;
import scone.util.tokenstream.Token;
import scone.util.tokenstream.TokenInputStream;
import scone.util.tokenstream.TokenInputStreamTokenizerImpl;
import sun.misc.BASE64Decoder;
import sun.misc.BASE64Encoder;
import trust.WoT;

public class SecurityProvider {

    public static KeyStore InitKeyStore(String filename, String password) {
        try {
            // The keyStore is an instance of Java KeyStore
            KeyStore ks = KeyStore.getInstance("JKS");
            String fileName = filename;
            // Convert the password to char array
            char[] passPhrase = password.toCharArray();
            // Load the file of the keyStore
            File keyStoreFile = new File(fileName);
            // Load from the file to the KeyStore instance
            ks.load(new FileInputStream(keyStoreFile), passPhrase);
            return ks;
        } catch (KeyStoreException e) {
            e.printStackTrace();
            return null;
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return null;
        }
        return null;
    }
}

A.1.20 SecurityProvider.java
// Initialize a certificate from a loaded keystore

public static Certificate InitCertificate(KeyStore ks, String user) {
    try {
        Certificate cert = ks.getCertificate(user);
        return cert;
    } catch (CertificateException e) {
        e.printStackTrace();
        return null;
    }
}

// Initialize a KeyPair that is loaded from a keystore. The username and the passphrase is required to access the keys from the keystore

public static KeyPair InitKeyPair(KeyStore ks, String user, String passPhrase) {
    try {
        // Get the private key directly from the keystore
        Key key = ks.getKey(user, passPhrase.toCharArray());
        PrivateKey privateKey = (PrivateKey) key;
        // Get the public key from a certificate generated from the keystore
        PublicKey publicKey = ks.getCertificate(user).getPublicKey();
        return new KeyPair(publicKey, privateKey);
    } catch (KeyStoreException e) {
        e.printStackTrace();
    } catch (NoSuchAlgorithmException e) {
        try {
            Signature sig = Signature.getInstance("SHA1 with RSA");
            sig.initSign(privateKey);
        } catch (InvalidKeyException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (NoSuchAlgorithmException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return null;
    }
}

// Create a signature with a private key on a given string

public static byte[] createSignature(String whatToHash, PrivateKey privateKey) {
    try {
        // Looks up the algorithm that the private key is generated from
        Signature sig = Signature.getInstance(privateKey.getAlgorithm());
        // Initiates the Signature object with the private key
        sig.initSign(privateKey);
        // Inserts what needs to be signed into the Signature object
        sig.update(whatToHash.getBytes(), 0, whatToHash.getBytes().length);
        // Returns the signed bytes
        return sig.sign();
    } catch (InvalidKeyException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (NoSuchAlgorithmException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return null;
}
public static boolean verifySignature(String whatToVerify, byte[] signature, PublicKey publicKey) {
    try {
        // Looks up the algorithm that the private key is generated from
        Signature sig = Signature.getInstance(publicKey.getAlgorithm());
        // Initiates the Signature object with the public key
        sig.initVerify(publicKey);
        // Inserts what needs to be verified into the Signature object
        sig.update(whatToVerify.getBytes(), 0, whatToVerify.getBytes().length);
        return sig.verify(signature);
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    } catch (InvalidKeyException ike) {
        ike.printStackTrace();
    }
    return null;
}
private static String generateCertificate(Certificate cert) {
    BASE64Encoder myB64 = new BASE64Encoder();
    try {
        return "- - - - - BEGIN CERTIFICATE - - - - -
        " + myB64.encode(cert.getEncoded()) + " - - - - - END CERTIFICATE - - - - -";
    } catch (CertificateEncodingException e) {
        e.printStackTrace();
    }
    return null;
}

public static Certificate getUserCertificate(String user) {
    Certificate cert;
    try {
        URLConnection historyURLConnection = new URL(historyUrl).openConnection();
        historyURLConnection.setUseCaches(false);
        InputStream history = historyURLConnection.getInputStream();
        InputStreamReader historyReader = new InputStreamReader(history);
        // Create a token stream for parsing
        TokenInputStream htis = new TokenInputStreamTokenizerImpl(historyReader);
        Token temp = null;
        // Start finding the newest certificate
        while (temp != null && temp.getString().indexOf("<A") < 0) {
            String params = temp.getString().substring(0, matchSize);
            // If the link matches the <A> tag in the HTML document
            if (params != null && params.length() >= matchSize) {
                String tempstring = params.substring(0, matchSize);
                // If the link has some parameters and the length is correct
                if (tempstring.equals(toMatch)) {
                    // Move 8 tokens forward to the <A> tag to get the URL
                    int matchSize = toMatch.length();
                    // If the link has some parameters and the length is correct
                    // Matching exact length in order to avoid too much computation
                    if (params != null && params.length() >= matchSize) {
                        String tempstring = params.substring(0, matchSize);
                        // If the link matches a link to the certificate
                        if (tempstring.equals(toMatch)) {
                            // Get the certificate URL to the latest URL
                            // that the user has edited
                            certificateUrl = "http://en.wikipedia.org" + params.replace("&amp;", ";");
                            System.out.println("URL:");
                            // Once the latest version of the certificate is found close the stream
                            while (htis.read() != null) {
                                htis.close();
                            }
                        }
                        temp = htis.read();
                    }
                }
            }
        }
    }
}

*/

getCertificate is used to generate a certificate for the user to insert on the wiki user page. The certificate is base 64 encoded.

@param cert
@return
A.1 Code

// Once the URL are in place, download the HTML page where the certificate is stored and retrieve the certificate.
cert = null;
// Create a certificate factory that generates x.509 certificates
CertificateFactory cf = CertificateFactory.getInstance("X.509");
// Open an URL connection to the certificate
URLConnection urlconnection = new URL(certificateUrl).openConnection();
urlconnection.setUseCaches(false);
InputStream is = urlconnection.getInputStream();
InputStreamReader isr = new InputStreamReader(is);
// Create a Tokenstream for the URL
TokenInputStream tis = new TokenInputStreamTokenizerImpl(isr);
temp = null;
// Start looking for "−−−−−BEGIN" which initiates the certificate
while ((temp = tis.read()) != null) {
  if (temp instanceof HtmlTextToken) {
    HtmlTextToken text = (HtmlTextToken) temp;
    if (text.getText().length() > 9) {
      // If the Token is a tag, it is longer than 9 chars and it equals "−−−−−BEGIN"
      if (text.getText().equals("-−−−−BEGIN")) {
        // Found certificate
        String string_cert = text.getText() + "\n" + ((HtmlTextToken) tis.read()).getText();
        // read the certificate into string
        do {
          text = (HtmlTextToken) tis.read();
          string_cert += "\n" + text.getText();
          System.out.println("Text is: "+ text.getText());
        } while (!text.getText().equals("CERTIFICATE -−−−−"));
        // Create an input stream for the CertificateFactory
      }
      // to use
      System.out.println("My certificate is "+ string_cert);
      ByteArrayInputStream bs = new ByteArrayInputStream(string_cert.getBytes());
      cert = cf.generateCertificate(bs);
    }
  }
} catch (CertificateException e) {
  e.printStackTrace();
} catch (MalformedURLException e) {
  e.printStackTrace();
} catch (IOException e) {
  e.printStackTrace();
} return null;
/
*************************************************************************/
public static String createRating(String wiki_username, int rating, int category, String version, String pageTitle, PrivateKey privateKey) {
  // The string that needs to be signed
  String toSign = "\n" + wiki_username + ";" + rating + ";" + category + ";" + version + ";" + pageTitle;
  // The byte array containing the signature
  byte[] signature = createSignature(toSign, privateKey);
  // Encode the signature on base64 form
  String encodedSignature = createBase64EncodingFromSignature(signature);
  // System.out.println("Submitting: "+ toSign + encodedSignature);
  try {
    try {
      FileWriter fw = new FileWriter("signature.txt");
      fw.write(toSign + ";" + signature + ";" + encodedSignature);
      fw.close();
    }
  }
  catch (Exception e) {
    e.printStackTrace();
  }
  /*
  byte[] signature2 = getSignatureFromBase64Representation(encodedSignature);
  boolean validated = verifySignature(toSign, signature2, getUserCertificate("Fluxdk").getPublicKey());
  */
System.out.println("Public Cert: " + getUserCertificate("Fluxdk").getPublicKey());
System.out.println("Signature survives encode & decode: " + signature.equals(signature2));
*/
return ("<!- -→WikiTrustComment.→n" + toSign + encodedSignature + "→n→n");
*/
/**
 * Validator takes a raw rating and validates
 * @param rating
 * @param ror
 */
public static boolean validateRating(String rating, WoT ror) {
  // Split up the rating
  String[] rating_parts = rating.split(";");
  // Set the username
  String username = rating_parts[1];
  // The base64 signature
  String string_signature = rating_parts[5].trim();
  // Decode signature
  byte[] signature = getSignatureFromBase64Representation(string_signature);
  // Retrieve the certificate from the cache
  Certificate cert = getUserCertificateFromCache(username, ror);
  // Setting together what needs to be verified
  try {
    FileWriter fw = new FileWriter(new File("signature.txt"));
    fw.write(whatToVerify + "\n" + string_signature + "\n" + signature + "\n" + cert.getPublicKey());
    fw.close();
  }
  catch (Exception e) {
    e.printStackTrace();
  }
  System.out.println("What to verify:
  System.out.println("Signature:
  // Check if the signature is validated
  boolean validated = verifySignature(whatToVerify, signature, cert.getPublicKey());
  return validated;
}*/
/**
 * Obtains the user certificate from the RoR
 * @param username
 * @param ror
 * @return
 */
public static Certificate getUserCertificateFromCache(String username, WoT ror) {
  return ror.getCertificateFromUsername(username);
}*/
/**
 * Get KeyStore pass from disk
 */
public static String[] getWikiUserPassFromDisk() {
  try {
    FileReader fr = new FileReader("passwords.txt");
    BufferedReader br = new BufferedReader(fr);
    String line = null;
    String[] lineArray = null;
    while ((line = br.readLine()) != null) {
      lineArray = line.split(";
      if (lineArray[0].equals("KeyStorePass")) {
        return lineArray[1];
      }
    }
  }
  catch (FileNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }
  catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }
  return ";
}*/
/**
 * GetWikiUserPassFromDisk
 */
public static String[] getWikiUserPassFromDisk() {
  try {
    FileReader fr = new FileReader("passwords.txt");
    BufferedReader br = new BufferedReader(fr);
    String line = null;
    String[] lineArray = null;
    while ((line = br.readLine()) != null) {
      lineArray = line.split(";
      if (lineArray[0].equals("WikiUserPass")) {
        return lineArray[1];
      }
    }
  }
  catch (FileNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }
  catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }
  return ";
}*/
A.1 Code

```java
BufferedReader br = new BufferedReader(fr);
String line = null;
String[] lineArray;
while ((line = br.readLine()) != null) {
    lineArray = line.split(" ");
    if (lineArray[0].equals("WikiUserPass")) {
        String[] loginInfo = {lineArray[1], lineArray[2]};
        return loginInfo;
    }
}
}
```

A.1.21 Serializer.java

```java
package static tools;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import trust.WoT;

public class Serializer {
    private static boolean runOnce = true;

    public static WoT readRoRFromDisk(String username) {
        try {
            FileInputStream fin = new FileInputStream("static_textfiles/myWoT.dat");
            ObjectInputStream objIn = new ObjectInputStream(fin);
            WoT ror = (WoT) objIn.readObject();
            if (!ror.isOwner(username)) {
                System.out.println("Wrong owner, creating new WoT!");
                ror = new WoT(username);
                return ror;
            }
            if (runOnce) {
                System.out.println("WoT loaded successfully!");
                runOnce = false;
            }
        }
    }
}
```
return null;
}

public static void writeRoRToDisk(WoT wot) {
try {
FileOutputStream fout = new FileOutputStream("static_textfiles/myWoT.dat");
ObjectOutputStream objout = new ObjectOutputStream(fout);
//System.out.println("WoT saved: " + wot.toJSONString());
objout.writeObject(wot);
} catch (FileNotFoundException e) {
System.out.println("No WoT found, creating new!");
return new WoT(username);
} catch (IOException e) {
}
	catch (ClassNotFoundException e) {
}
}

A.1.22 Threshold.java

package statictools;
/****************************
* WRs – Wikipedia Recommender System is a collaborative recommender system    *
* used to rate the articles on the Wikipedia.                               *
* Copyright (C) 2007 by Thomas Rune Korsgaard, tkorsgaard(at)gmail.com        *
* This program is free software; you can redistribute it and/or modify       *
* it under the terms of the GNU General Public License as published by      *
* the Free Software Foundation, either version 3 of the License, or         *
* (at your option) any later version.                                       *
* This program is distributed in the hope that it will be useful,           *
* but WITHOUT ANY WARRANTY; without even the implied warranty of            *
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
* You should have received a copy of the GNU General Public License        *
* along with this program. If not, see <http://www.gnu.org/licenses/>.
* ****************************/

import java.awt.List;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;
import scone.util.tokenstream.HtmlCommentToken;
import scone.util.tokenstream.HtmlTagToken;
import scone.util.tokenstream.HtmlTextToken;
import scone.util.tokenstream.Token;
import scone.util.tokenstream.TokenInputStream;

public class Threshold {

public static double ThresholdCalculator(String page_title, String old_version_number) {
    TokenInputStream tis = TokenInputStreamTools.CreateTokenInputStreamFromURL(string_url);
    int total_words_in_textarea = 0;
    int total_words_changed = 0;
    // Start scanning for changes
    Token temp = null;
    HtmlTagToken tag = null;
    HtmlTextToken text = null;
    HtmlCommentToken comment = null;
    boolean current_marker = false;
}
ArrayList<String> currentRevision = new ArrayList<String>();
int i = 0;
int numberOfWords = 0;
double threshold = 0.0;
try {
    while (((temp = tis.read()) != null)) {
        i++;
        if (temp instanceof HtmlTagToken) {
            tag = (HtmlTagToken) temp;
            // Find the table where the changes are
            if (tag.hasParam("class")) {
                if (tag.getTagType() == HtmlTagToken.T_TABLE &&
                    tag.getParam("class").equals("diff") && tag.isEndElement()) {
                    while ((temp = tis.read()) != null) {
                        if (temp instanceof HtmlTagToken) {
                            tag = (HtmlTagToken) temp;
                            // If the tag is a TD with
                            // class="diff-addedline"
                            if (tag.getTagType() == HtmlTagToken.T_TD && tag.isEndElement()) {
                                break;
                            }
                            changedLine.add(temp);
                            diffAddedLine.add(changedLine);
                        }
                    }
                }
            }
        }
        if (temp instanceof HtmlTextToken) {
            currentRevision.add(((HtmlTextToken) temp).getText());
        } else if (temp instanceof HtmlCommentToken) {
            comment = (HtmlCommentToken) temp;
            String test = comment.getComment();
            // If <!-- end content --> is found, then break it all up.
            // We’re done
            if (comment.getComment().trim().equals("end content")) {
                break;
            }
        }
    }
    // Count the words in the rest of the article
    // Look for HtmlTextTokens from "Current revision" to <!-- end
    // content --> marker
    if (temp instanceof HtmlTextToken) {
        currentRevision.add(((HtmlTextToken) temp).getText());
    }
    // Count red letter words
    int numberOfRedLetterWords = 0;
    for (ArrayList line : diffAddedLine) {
        numberOfRedLetterWords += numberOfRedLetterWords(line);
    }
    System.out.println("number of red words: ");
    // threshold = (double) numberOfRedLetterWords / (double) currentRevision.size();
catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

// When we meet a diff marker + then enable scanning, as it marks changes
// to current version (green block)
// when we meet a diff marker - then disable scanning, as we are scanning
// the old version (yellow block)
// when finding a diff change determine if it is amplification, reversal
// or addition
// Add words to total change
// Calculate the percentage changed:

return threshold;

private static int numberOfRedLetterWords(ArrayList<Token> list) {
    String[] string_amp = {
    }
    String[] reversal_words = {
        "not ", "n't ", "dont ", "don't ", "esn't ", "doesn't ", "doesn't ", "no ", "without ", "wont ", "won't ", "un 
    }
    Set list_of_amplifications = new HashSet(Arrays.asList(string_amp));
    Set list_of_reversals = new HashSet(Arrays.asList(reversal_words));
    boolean found_reversal = false;
    // look for red words:
    int num_of_red_words = 0;
    int num_of_total_words = 0;
    Token word = null;
    HtmlTagToken tag = null;
    HtmlTextToken text = null;
    while (!list.isEmpty()) {
        word = list.remove(0);
        if (word instanceof HtmlTagToken) {
            tag = (HtmlTagToken) word;
            if (tag.hasParam("class")) {
                if (tag.getParam("class").equals("diffchange")) {
                    while (true) {
                        word = list.remove(0);
                        if (word instanceof HtmlTextToken) {
                            text = (HtmlTextToken) word;
                            String s_test = text.getText();
                            // Analyse the text
                            // Ignore small changes - amplifications
                            if (!list_of_amplifications.contains(s_test)) {
                                // count single word changes - additions
                                num_of_red_words++;
                                // System.out.println(s_test + "]");
                            }
                            if (list_of_reversals.contains(s_test.toLowerCase())) {
                                found_reversal = true;
                            }
                            // find reversal word - reversal - husk små
                            // kogslaver
                        } else {
                            // If we find a tag, then it must be an ending tag
                            if (word instanceof HtmlTagToken) {
                                if (((HtmlTagToken) word).isEndTag())
                                    break;
                            }
                        }
                    }
                    if (found_reversal) {
                        num_of_red_words = num_of_total_words;
                    }
                    return num_of_red_words;
                }
            }
        }
    }
    return num_of_red_words;

A.1.23 TokenInputStreamTools.java

package static tools;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import scone.util.tokenstream.TokenInputStream;
import scone.util.tokenstream.TokenInputStreamTokenizerImpl;

public class TokenInputStreamTools {
    // CreateTokenInputStreamFromURL creates a TokenInputStream from an URL.
    @param string url
    @return
    public static TokenInputStream CreateTokenInputStreamFromURL(String string_url) {
        try {
            URL url = new URL(string_url);
            URLConnection urlconnection = url.openConnection();
            urlconnection.setUseCaches(false);
            // Open up the Stream
            InputStreamReader isr = new InputStreamReader(urlconnection.getInputStream());
            // Create the input stream
            TokenInputStream tis = new TokenInputStreamTokenizerImpl(isr);
            return tis;
        }
        catch (MalformedURLException e) {
            e.printStackTrace();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
}

A.1.24 Wiki.java

package static tools;
//
// @(#)Wiki.java 0.03 10/06/2007
// Copyright (C) 2007 MER-C
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
// You should have received a copy of the GNU General Public License
// along with this program; if not, write to the Free Software
import java.io.*;
import java.util.*;
import java.net.*;

/*
This is somewhat of a sketchy bot framework for editing MediaWiki wikis.

@Author MER-C
@Version 0.03
*/

public class Wiki {

/*
Denotes the namespace of images and media, such that there is no
description page. Uses the "Media:" prefix.

@since 0.03
*/
public static final int MEDIA_NAMESPACE = -2;

/*
Denotes the namespace of pages with the "Special:" prefix. Note that many
methods dealing with special pages may spew due to raw content not being
available.

@since 0.03
*/
public static final int SPECIAL_NAMESPACE = -1;

/*
Denotes the main namespace, with no prefix.

@since 0.03
*/
public static final int MAIN_NAMESPACE = 0;

/*
Denotes the namespace for talk pages relating to the main namespace,
denoted by the prefix "Talk:".

@since 0.03
*/
public static final int TALK_NAMESPACE = 1;

/*
Denotes the namespace for user pages, given the prefix "User:".

@since 0.03
*/
public static final int USER_NAMESPACE = 2;

/*
Denotes the namespace for user talk pages, given the prefix "User talk:".

@since 0.03
*/
public static final int USER_TALK_NAMESPACE = 3;

/*
Denotes the namespace for pages relating to the project, with prefix
"Project:". It also goes by the name of whatever the project name was.

@since 0.03
*/
public static final int PROJECT_NAMESPACE = 4;

/*
Denotes the namespace for talk pages relating to project pages, with prefix
"Project talk:". It also goes by the name of whatever the project name was, +
"talk:".

@since 0.03
*/
public static final int PROJECT_TALK_NAMESPACE = 5;

/*
Denotes the namespace for image description pages. Has the prefix "Image:".
Do not create these directly, use upload() instead.

@since 0.03
*/
public static final int IMAGE_NAMESPACE = 6;

/*
Denotes talk pages for image description pages. Has the prefix "Image
public static final int IMAGE_TALK_NAMESPACE = 7;
/**
 * Denotes the namespace for (wiki) system messages, given the prefix
 * "MediaWiki:"
 * @since 0.03
 */
public static final int MEDIAWIKI_NAMESPACE = 8;
/**
 * Denotes the namespace for talk pages relating to system messages, given the
 * prefix "MediaWiki talk:"
 * @since 0.03
 */
public static final int MEDIAWIKI_TALK_NAMESPACE = 9;
/**
 * Denotes the namespace for templates, given the prefix "Template:"
 * @since 0.03
 */
public static final int TEMPLATE_NAMESPACE = 10;
/**
 * Denotes the namespace for talk pages regarding templates, given the prefix
 * "Template talk:"
 * @since 0.03
 */
public static final int TEMPLATE_TALK_NAMESPACE = 11;
/**
 * Denotes the namespace for help pages, given the prefix "Help:"
 * @since 0.03
 */
public static final int HELP_NAMESPACE = 12;
/**
 * Denotes the namespace for talk pages regarding help pages, given the prefix
 * "Help talk:"
 * @since 0.03
 */
public static final int HELP_TALK_NAMESPACE = 13;
/**
 * Denotes the namespace for category description pages. Has the prefix
 * "Category:"
 * @since 0.03
 */
public static final int CATEGORY_NAMESPACE = 14;
/**
 * Denotes the namespace for talk pages regarding categories. Has the prefix
 * "Category talk:"
 * @since 0.03
 */
public static final int CATEGORY_TALK_NAMESPACE = 15;
/**
 * Denotes all namespaces.
 * @since 0.03
 */
public static final int ALL_NAMESPACES = 0x09f91102;

// the domain of the wiki
private String domain;

// something to handle cookies
private Map cookies = new HashMap(10);

// internal data storage
private Map namespaces = null;
public Wiki() {
    this("*");
}

/**
 * Creates a new connection to a wiki.
 */
public Wiki(String domain) {
    if (domain == null || domain == "")
        domain = "en.wikipedia.org";
    this.domain = "http://" + domain;
    this.script = "http://" + domain + "/wiki/index.php";
    query = "http://" + domain + "/w/query.php";
}

/**
 * Logs into the wiki.
 */
public boolean login(String username, char[] password) throws IOException {
    // sanitize
    String ps = new String(password);
    username = URLEncoder.encode(username, "UTF-8");
    ps = URLEncoder.encode(ps, "UTF-8");
    // "enable" cookies
    String URL = domain + "/w/index.php?title=Special:UserLogin";
    URLConnection connection = new URL(URL).openConnection();
    grabCookies(connection);
    // find the target
    URL = domain + "/w/index.php?title=Special:UserLogin&action=submitLogin&title=Login";
    connection = new URL(URL).openConnection();
    setCookies(connection);
    connection.setDoOutput(true);
    PrintWriter out = new PrintWriter(connection.getOutputStream(true));
    // now we send the data
    out.print("wpName=");
    out.print(username);
    out.print("&wpPassword=");
    out.print(ps);
    out.print("&wpPassword=");
    out.print("&wpLoginPlugin=1&wpLoginAttempt=Log+in");
    out.close();
    // make it stick by grabbing the cookie
    grabCookies(connection);
    BufferedReader in = null;
    try {
        io = new BufferedReader(new InputStreamReader(connection.getInputStream()));
    }
    catch (IOException e) {
        if (!((connection instanceof HttpURLConnection))
            throw e;
        InputStreamReader err = ((HttpURLConnection) connection).getInputStream();
        if (err == null)
            throw e;
        in = new BufferedReader(new InputStreamReader(err));
    }
    in.readLine();
    // test for success
    String line;
    while ((line = in.readLine()) != null) {
        // status.write(line);
        if (line.indexOf("login\_successful\") != -1) {
            return true;
        }
    }
    return false;
}
public void logout() {
    cookies.clear();
}

public String getPageText(String title) throws IOException {
    // pitfall check
    if (namespace(title) < 0)
        throw new IllegalArgumentException("Cannot retrieve Special: or Media: pages!");
    // sanitise the title
    title = URLEncoder.encode(title, "UTF-8");
    // go for it
    String URL = domain + "/w/index.php?title=Special:Recommendations/" + title + "&action=raw";
    URLConnection connection = new URL(URL).openConnection();
    connection.connect();
    BufferedReader in;
    try {
        in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
    } catch (FileNotFoundException fnfe) {
        return "";
    }
    // get the text
    StringBuffer text = new StringBuffer();
    while ((line = in.readLine()) != null)
        text.append(line + "\n");
    return text.toString();
}

public void editPage(String title, String text, String summary, boolean minor) throws IOException {
    // pitfall check
    if (namespace(title) < 0)
        throw new IllegalArgumentExpection("Cannot edit Special: or Media: pages!");
    // what we need to do is get the edit page and fish out the wpEditToken,
    // wpAutoSummary, wpStartTime and wpEditTime values
    URL URL = new URL(domain + "/w/index.php?title=Special:Recommendations/" + title + "&action=edit";
    URLConnection connection = URL.openConnection();
    setCookies(connection);
    grabCookies(connection);
    BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
    // more specifically, we're looking for "name="wpEditToken"
    // "name="wpAutoSummary"
    String line, wpEditToken = "", wpAutoSummary = "", wpStartTime = "", wpEditTime = "";
    boolean editRetrieved = false, summaryRetrieved = false, startRetrieved = false, timeRetrieved = false;
    boolean watched = false;
    while ((line = in.readLine()) != null) {
        int x = line.indexOf("name="wpAutoSummary") != -1 { 
            int y = line.indexOf("value=") + 7;
            int z = line.indexOf("\n"), x);
            int x = line.indexOf("area="wpAutoSummary") != -1 {
                int y = line.indexOf("value=") + 7;
                int z = line.indexOf("\n", x);
            }
summaryRetrieved = true;
}  
else if (line.indexOf("name=" + "wpEditToken" + "\"\") != -1) {
  int x = line.indexOf("value=") + 7;
  wpEditToken = line.substring(x, line.indexOf(\"\", x));
  editRetrieved = true;
}  
else if (line.indexOf("name=" + "wpEdittime" + "\"\") != -1) {
  int x = line.indexOf("value=") + 7;
  wpEdittime = line.substring(x, line.indexOf(\"\", x));
  timeRetrieved = true;
}  
else if (line.indexOf("name=" + "wpStarttime" + "\"\") != -1) {
  int x = line.indexOf("value=") + 7;
  wpStarttime = line.substring(x, line.indexOf(\"\", x));
  startRetrieved = true;
}  
else if (line.indexOf("name=" + "wpWatchthis" + "\"\") != -1) {
  watched = (line.indexOf("checked=") != -1);
  watchRetrieved = true;
}  
  break;  // bandwidth hack
}
// this is what accepts the text
URL = new URL(domain + "/w/index.php?title=User:Recommendations/" + title + "+\action=submit");
// Create the boundary for the MultiPartForm transmission
String boundary = MultiPartFormOutputStream.createBoundary();
connection = MultiPartFormOutputStream.createConnection(URL);
connection.setRequestProperty("Accept", "+\*/\*");
connection.setRequestProperty("Content-Type", MultiPartFormOutputStream.getContentType(boundary));
collection.setRequestProperty("Connection", "+\keep\+\alive");
collection.setRequestProperty("Cookie", collection);
MultiPartFormOutputStream out =
  new MultiPartFormOutputStream(connection.getOutputStream(), boundary);
out.writeField("wpTextbox1", text);
out.writeField("wpSummary", summary);
if (minor)
  out.writeField("wpMinoredit", "+");
if (watched)
  out.writeField("wpWatchthis", "+");
out.writeField("wpEdittoken", wpEditToken);
out.writeField("wpEdittime", wpEdittime);
out.writeField("wpStarttime", wpStarttime);
out.writeField("wpAutoSummary", wpAutoSummary);
out.writeField("wpSection", "+");
out.writeField("wpScrollstep", "+");
out.close();
try {
  // FileWriter status = new FileWriter("reply.html");
  in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
  while((line = in.readLine()) != null) {
    // status.write(line);
  }
  //status.close();
  in.close();
} catch (Exception e) {
  e.printStackTrace();
}
/**
 * Prepends something to the given page. A convenience method for adding
 * maintenance templates, rather than getting and setting the page yourself.
 * Edit summary is automatic, being "+whatever".
 * @param title
 *   the title of the page
 * @param stuff
 *   what to prepend to the page
 * @param minor
 *   whether the edit is minor (a prod compared to a simple tag)
 * @param throws IOException
 *   if something goes wrong
 */
public void prepend(String title, String stuff, boolean minor) throws IOException {
  StringBuffer text = new StringBuffer();
  text.append(stuff);
  text.append(getPageText(title));
  editPage(title, text.toString() + "\*\* + stuff, minor");
}
/**
 * Gets the members of a category.
public String[] getCategoryMembers(String name) throws IOException {
    return getCategoryMembers(name, ALL NAMESPACES);
}

/**
 * Gets the members of a category.
 * @param name the name of the category (e.g. Candidates for speedy deletion, not Category: Candidates for speedy deletion)
 * @param namespace filters by namespace, returns empty if namespace does not exist
 * @return a String[] containing page titles of members of the category
 * @throws IOException if something goes wrong
 * @since 0.03
 */
public String[] getCategoryMembers(String name, int namespace) throws IOException {
    String url;
    if (namespace == ALL NAMESPACES)
        url = query + " what = category & format = xml & cptitle = " + URLEncoder.encode(name, " UTF-8 ");
    else
        url = query + " what = category & format = xml & cptitle = " + URLEncoder.encode(name, " UTF-8 ") + " & cpname=" + namespace;
    URLConnection connection = new URL(url).openConnection();
    connection.connect();
    // read the first line, as it is the only thing worth paying attention to
    BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
    String line = in.readLine();
    // parse
    ArrayList<String> members = new ArrayList<String>(10000); // enough for most cats
    while (line.indexOf(" < title > ") != -1) {
        int x = line.indexOf(" < title > ");
        int y = line.indexOf(" </ title ");
        members.add(line.substring(x + 7, y));
        line = line.substring(y + 8, line.length());
    }
    return members.toArray(new String[0]);
}

/**
 * Returns the namespace the page is in. Uses /w/query.php?what=namespaces to fetch list of namespaces.
 * @since 0.03
 * @return one of namespace types above, or a number for custom namespaces or ALL NAMESPACES if we can't make sense of it
 * @throws IOException if something goes wrong
 */
public int namespace(String title) throws IOException {
    // sanitise
    title = title.replace(" ", ",");
    if (title.indexOf(\')') == -1)
        return MAINNAMESPACE;
    String namespace = title.substring(0, title.indexOf('\') + 1);
    // all wiki namespace test
    if (namespace.equals("Project\_talk"))
        return PROJECT TALK NAMESPACE;
    if (namespace.equals("Project"))
        return PROJECT NAMESPACE;
    if (namespaces == null) {
        URLConnection connection = new URL(query + " what = namespaces & format = xml ").openConnection();
        connection.connect();
        // read the first line, as it is the only thing worth paying attention to
        BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
        String line = in.readLine();
        namespaces = new HashMap(20);
    }
    return namespaces.get("Project\_talk");
}
while (line.indexOf("<ns") != -1) {
    int x = line.indexOf("<ns\s+id="");
    if (line.charAt(x + 8) == '0') {
        line = line.substring(13, line.length());
        continue;
    }
    int y = line.indexOf("</ns>");
    String working = line.substring(x + 8, y);
    int ns = Integer.parseInt(working.substring(0, working.indexOf("""\s+""")));
    String name = working.substring(working.indexOf(""">""" + 1, working.length()));
    namespaces.put(name, new Integer(ns));
    line = line.substring(y + 5, line.length());
}

if (!namespaces.containsKey(namespace))
    return MAIN_NAMESPACE; // For titles like UN:NRV
Iterator i = namespaces.entrySet().iterator();
while (i.hasNext()) {
    Map.Entry entry = (Map.Entry) i.next();
    if (entry.getKey().equals(namespace))
        return (Integer) entry.getValue().intValue();
}
return ALL_NAMESPACES; // unintelligible title
}

/**
 * Grabs cookies from the URL connection provided.
 * @param u an unconnected URLConnection
 */
private void grabCookies(URLConnection u) {
    // reset the cookie store
    // cookies.clear();
    String headerName = null;
    for (int i = 1; (headerName = u.getHeaderFieldKey(i)) != null; i++) {
        if (headerName.equals("Set-Cookie")) {
            String cookie = u.getHeaderField(i);
            cookie = cookie.substring(0, cookie.indexOf(";"));
            String name = cookie.substring(0, cookie.indexOf("="));
            String value = cookie.substring(cookie.indexOf("=") + 1, cookie.length());
            cookies.put(name, value);
        }
    }
}

/**
 * Sets cookies to an unconnected URLConnection.
 * @param u an unconnected URLConnection
 */
private void setCookies(URLConnection u) {
    Iterator i = cookies.entrySet().iterator();
    StringBuffer cookie = new StringBuffer();
    while (i.hasNext()) {
        Map.Entry entry = (Map.Entry) i.next();
        cookie.append(entry.getKey());
        cookie.append("=");
        cookie.append(entry.getValue());
        cookie.append(";\s+");
    }
    u.setRequestProperty("Cookie", cookie.toString());
}


