Improving Trust in the Wikipedia

Thomas Rune Korsgaard

Kongens Lyngby 2007
IMM-MASTER-2007-67

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-PHD: ISSN 0909-3192

Abstract

The Wikipedia is a free online encyclopedia collaboratively edited by Internet users with a minimum of administration. Anybody can write an article for the Wikipedia and there is no verification of the author's expertise on the particular subject. This may lead to problems relating to the quality of articles, completeness and accuracy of the information in the articles, and this could result in distrust in the Wikipedia. It is our opinion that users should be able to assess the correctness, completeness and impartiality of information in the Wikipedia, and by that improve their personal trust in the Wikipedia.

In this thesis, we propose a recommendation system, which allows Wikipedia users to calculate a personalized recommendation for a specific article based on all the feedback (recommendations) provided by other Wikipedia users. Recommendations are calculated decentralized, which means that recommendations from users that one user has found useful in the past carries more weight than recommendations from unknown users or users that the user did not agree with in the past. This prevents a large population of people with similar political, social or religious norms from determining the global recommendation of all Wikipedia articles.

There are currently thousands of wiki installations through out the web, besides the Wikipedia. The introduction of a recommendation system should therefore not require any modifications to the Wikipedia engine. The proposed recommendation system is implemented in a proxy placed between the user's web-browser and the Wikipedia, for instance on the user's own machine, so there is no need to modify the Wikipedia.

A recommender system is build based on recommendations from trusted users.

The recommendation system continuously updates each trustees trust value based on the feedback given from the user.

The recommendation system has been evaluated and meets the functional requirements. The recommender system shows correct behavior. Experiments and benchmarking tests show that using the recommender system does not influence the Internet experience for its' users. In our evaluation we propose an approach to long term usability testing of the recommender system.

Resumé

Wikipedia er et frit internetbaseret encyklopædi som skrives af Internettets brugere med et minimum af adminitration. Alle kan skrive en artikel på Wikipedia og der er ingen verifikation af forfatterens ekspertise på det pågældende område. Dette kan medføre problemer med artiklernes fuldstændighed, nøjagtighed og kvalitet af artiklens information, hvilket kunne medføre mistillid til Wikipedia. Vi mener at brugere skal have mulighed for at vurdere nøjagtigheden, fuldstændigheden og objektiviteten af artiklernes information, og derigennem forbedre deres personlige tillid til Wikipedia.

Denne afhandling foreslår et anbefalingssystem, som tillader Wikipedias brugere at udregne en personlig anbefaling for en enkel artikel, baseret på anbefalinger fra andre brugere på Wikipedia. Anbefalinger er udregnet decentraliseret hvilket betyder, at anbefalinger fra brugere som brugeren tidligere har fundet brugbare, vægter mere i udregningen af anbefalingen, end anbefalinger fra brugere, som brugeren ikke kender eller ikke er enige med. Dette modvirker at store populations grupper i samfundet med samme politiske, sociale, seksuelle eller religiøse normer kan påvirke de globale anbefalinger for en artikel.

Der er tusindvis af wiki installationer, udover Wikipedia, overalt på nettet. Ved at indføre et anbefalingssystem skulle det ikke være nødvendigt at lave modifikationer til det eksisterende wiki software. Det foreslåede anbefalingssystem er udviklet som en proxy mellem brugerens browser og Wikipedia, således at det ikke er nødvendigt at foretage ændringer til softwaren der kører Wikipedia.

Vi har udviklet et anbefalingssystem baseret på anbefalinger fra betroede brugere. Anbefalingssystemet opdaterer løbende hver betroet brugers tillidsværdi baseret på de tilbagemeldinger som den betroede bruger giver.

Anbefalingssystemet er blevet evalueret og det er konstateret at det passer til de funktionelle krav der bliver stillet, og det udviser den forventede opførsel. Eksperimenter og målinger viser at brugen af anbefalingssystemet ikke har nogen nævneværdig indflydelse på Internet oplevelsen. Vi foreslår en metode til at udføre en langvarig anvendlighedstest, for at undersøge om det udviklede anbefalingssystem yder den hjælp til Wikipedia som er efterspurgt.

Preface

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 thesis deals with the aspects of creating a recommender system for the Wikipedia, that can provide its' users with recommendations based on recommendations from trusted users.

The project was completed in the period from January 1st, 2007 to July 31th, 2007 under the supervision of Associate Professor Christian Damsgaard Jensen.

An article containing the major findings from this project was submitted to the 3rd International Workshop on Security and Trust Management in conjunction with ESORICS 2007 in Dresden, with Christian Damsgaard Jensen as Co-author. Notification concerning the article was not received at the date for submission of this thesis.

Lyngby, July 2007

Thomas Rune Korsgaard
s011564

Acknowledgements

I would like to thank my superviser Christian D. Jensen for his great support throughout the entire phase of the project, and for providing ideas, solutions and general discussion on the topic.

I would also like to thank Kirstine Sandø Højland, Esben Kolind, Anders Dohn Hansen and Susanne Korsgaard for their ideas and proofreading.

Thanks also goes to the friends of IMSOR (Anders, Teis, Kristian, Jens and Aske) for providing amusement and some good table tennis matches throughout the project.

Finally a special thanks goes to Bodil for your patience throughout the project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

The Wikipedia is a free online encyclopedia where the content is written by voluntary writers from all over the world. All the articles in the Wikipedia can be edited by everybody and content can be removed or added. There are no restrictions on how to write, which content may be created and there are no requirements to the author's knowledge or writing capabilities. The Wikipedia is based on the wiki philosophy [7] that allows users to freely create and edit Web page content using a browser.

As a result of the freedom, the number of articles in the Wikipedia is growing rapidly. Alone in the English Wikipedia there are close to 1.8 million articles at the time of writing and there are around 2000 new articles emerging each day in the English Wikipedia [38]. These articles are created by voluntarily by individuals, who provide information on their own field of expertise or interest. This increases the chances of finding an article on a topic, even though there might only be a few experts in the world. Because all these articles are written by individuals, the articles can become subjective and not reflecting an objective description on the topic. This is especially the case when the article's content is political, religious, racial, sexual or otherwise dependent on taste.

The open and flexible nature of the Wikipedia has exposed weaknesses of collaborative authoring, which is that malicious or incompetent users may compromise the integrity of the documents by introducing erroneous entries or corrupting existing entries.

Jimmy Wales, the co-founder of the Wikipedia, claims to receive 10 emails every day from students who failed their courses because information cited from the Wikipedia turned out to be wrong [30].

Another example of the weaknesses in Wikipedia was in January 2006, when an IP scope from the US Congress was banned from editing in the Wikipedia because both the House and the Senate had been treating the Wikipedia as a personal battleground, fighting turf wars and repeatedly altering content about congressmen listed on the site [34].

A third example of misuse of the Wikipedia was a conflict between Adam Curry and Dave Winer, who both believed themselves to be the father of podcasting. An anonymous IP address kept making changes to the article on podcasting leaving only Curry as the Father of podcasting. The IP address was traced back to Curry [26].

A final example of misuse was an article on the Wikipedia about the assassination of president John F. Kennedy, which claiming that an innocent journalist (John Seigenthaler) had been involved in the planning and execution of the assassination. The false article was a part of the Wikipedia for four months [32].

These examples show how easy it is to be misinformed by the content of the Wikipedia. We propose a solution to filter the good content from the bad content.

1.1.1 Quality of the Content

The quality of a Wikipedia article is determined by a few simple properties, i.e., if the article complete, correct and unbiased. However, these properties are difficult to determine automatically and despite some promising work in this area [12, 39], which proposes systems that analyze an article automatically based on a predefined set of rules, we do not believe that these techniques are sufficiently mature at the moment. Instead we propose to rely on feedback from the users, i.e., to use some some recommendation system similar to the ones used by Amazon [1], IMDb [2] or the ”WOT” plugin for Firefox [4]. A recommendation system cannot prevent undesirable content from entering the

Wikipedia, but it may help readers assess the quality of Wikipedia articles and allow them to decide whether to trust the article or look for more reliable information elsewhere. Moreover, introducing a reputation system is in line with the Wiki philosophy, where we find few mechanisms to prevent malicious or accidental modification of a Wiki page; detection is left to the users and the only means of response is to restore the previous page.

In our proposal recommendations are gathered from feedback from other users that have given similar feedback, and use the feedback from these other users as recommendations for the active user. With this approach we can create a system that relies on decentralized calculations, that are carried out on the client side, instead of centralized calculations that are carried out on server side.

Relying on a central database means that large populations holding certain beliefs will dominate smaller populations with different cultural standards, e.g., feedback from the so-called Bible Belt in the Unites States (86.5 million inhabitants) will dominate feedback from a small progressive country, such as Denmark with 5.5 million inhabitants. Bearing in mind the significant cultural differences in those tow populations regarding what is considered appropriate information for young people regarding subjects like sex education and contraception we find this insufficient. We therefore believe it is essential to extend this simplistic way of approaching the issue in two ways. Firstly, for systems relying on central databases, we find it important for users to be able to choose a database reflecting values in a community that match their own culture, as far as definitions of content acceptability and unacceptability are concerned. Secondly, experience shows that combining trust based on personal experience with recommendations (direct reports of reputation) tends to give stable and reliable evaluations [22].

1.1.2 Objective

In this thesis we propose a recommender system that offers the users an evaluation of an article before they deside to read it.

The recommender system provides a central repository for feedback, which allows individual users to calculate their own subjective recommendation for a given article in the Wikipedia. The system does not calculate or distribute reputation values, but simply the recommendations (signed feedback) from other users, what brings us to refer to our system as the Wikipedia Recommender System (WRS). Through the recommendations from other people, the local component of the recommender system on the user's machine is able to calculate a recommendation, which indicates the quality of the article. The recommender system then receives feedback from the reader, which allows the WRS to determine whether the recommendations where useful and to identify the recommenders whose feedback coincided with the user. The recommender system uses this information to update the profile from the user that provided the recommendations, in order to decide how recommendations should be interpreted and provide a more precise recommendation next time.

This means that Wikipedia articles are classified on the basis of recommendations from other users, and individual users can use this classification to define their own blocking criteria. A tool that will initially be integrated with web-browser technology, but the techniques developed are generally applicable. Every user that visits an article will be offered the opportunity to classify this site using a simple classification and respond to whether the recommendation were useful or not.

The recommender system is implemented with existing Wikipedia and the idea is to give the active users a personal recommendation to an article. This recommendation is based on a set of trustees that the active user trusts. A recommendation is calculated, based on recommendations given by other users and on how much these users are trusted.

The WRS is created so that it will work with the existing Wikipedia as it is. The Wikipedia is therefore treated as a legacy system and the WRS cannot be implemented on the same server running Wikipedia. Consequently it is not possible to integrate the WRS with the existing Wikipedia software. The recommender system will have to be implemented as middleware between the Wikipedia and the browser.

The initial idea for the system is to have no configuration. Upon initialization of the system, the user should not have to enter his/hers psychological profile on how he trusts other people. The system will, through interactions with the active user, learn the users trust profile and will over time provide more and more precise recommendations to the active user.

1.1.3 Achievements

In this thesis we have implemented a decentralized recommender system, that builds a trust profile for each user in the database. The recommender system operates without configuration and is built as middleware between the user and the Wikipedia. A proxy based prototype of the WRS has been developed, which allows us to evaluate the feasibility of the proposed architecture. Experiments indicate that the computational overhead involved in verifying the recommendations and storage overhead needed by the recommendations are acceptable.

1.2 Definition of terms

In this section a set of terms are defined and used through out 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 active user. The active user is the user that uses the WRS to obtain recommendations about the articles on the Wikipedia. Also referred to as **trustor**.

The users. The term "the users" or "the other users" refers to all the other users of the Wikipedia that use the WRS, but not the active user. The active user benefits from the recommendations from the other users. Also referred to as **trustees**.

Ring of Reviewers. The Ring of Reviewers (RoR) is the set of Wikipedia users from which the active user has collected recommendations. The RoR is used to calculate the individual user's trust value.

Trust profile. Each trustee, that the trustor has in the Ring of Reviewers, 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.

Trust value. The trust value is a decimal value between -1 and 1, that describes how much the trustor user 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.

Recommendation. The recommendation is the actual mark given to an article. The mark is between 1 and 9, where 9 is the highest mark, that can be given.

Rating. The rating is the text string that is inserted in the edit page of an article.

Wikipedia. When referring to the Wikipedia this refers to the English Wikipedia, which is found on `http://en.wikipedia.org`.

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 rating from the WRS. The trustor defines through feedback if the experience is positive or negative.

1.3 Structure of this thesis

This thesis is structured as follows: This chapter (Chapter 1) contains an introduction to the Wikipedia and the concept of a recommender system. The chapter defines the objectives of the thesis.

Chapter 2 (State of the Art) contains a short description of the different technologies that are used in this project. It gives an overview of the research that has been put into trust and trust management. The chapter gives an overview of existing recommender systems.

Chapter 3 (Analysis) contains a specification of the requirements for the system, based on a scenario. The chapter gives an analysis of the existing recommender systems and points out the pros and cons of these systems. In the chapter we analyze some of the major problems that the specifications of requirements give.

Chapter 4 contains the trust model for the WRS and how it has emerged. The chapter describes why it is important to have trust model. Furthermore, it describes the necessary parts in a trust model, and how these parts are formalized so they can be implemented in software.

Chapter 5 describes the design of the WRS. The chapter describes the analysis of the Wikipedia and the result of this analysis shows how the middleware can fit into the Wikipedia. The design chapter also describes how the WRS is designed internally and which measures have to be taken in place to ensure security and privacy.

Chapter 6 describes the implementation of the WRS. The chapter goes through the different components of the WRS software, the general setup of the WRS, and the requirements in order to get the system running.

Chapter 7 describes the evaluation of the WRS. The chapter gives an overview of the white box and the black box test carried out. The chapter contains a discussion of the general need for a large scale usability test and propose an approach to perform such a large long-term test, with feedback from the users.

Chapter 8 gives an overview of which areas that are in need of further research and which interesting areas that have come to our attention, which could be a future project.

Chapter 9 describes the conclusion to this thesis and point out our major findings to this thesis.

Chapter 2

State of the Art

This chapter gives an insight to the essential technologies and research that are used in this thesis. First we review some of the basic ideas of trust and management of trust. We cover definitions of trust and the fundamentals for a trust model, discuss the components of the trust model and look at some experiments on trust.

Secondly, we list some existing recommender systems and point out the important considerations in the give recommender system in order to be able to analyze the needs for the WRS in chapter 3.

Thirdly, we look at some of the technologies and security measures that we use in this thesis and theories behind them.

Finally we look at some research within linguistics and statistics which we use in the implementation of our recommender system.

2.1 Theory and Research on Trust and Trust Management

A lot of research has carried out in the field of trust and trust management. This section identifies the most important definitions and components needed in a trust model

2.1.1 Definitions of Trust and Trust Management

Jøsang, Keser and Dimitrakos [21] describe the fundamental terms, which have to be defined when building a trust management system, and the emphasize the importance of having a proper and robust trust management system. They describe the need for trust as following:

> Lack of trust is like sand in the social machinery, and represents a real obstacle for the uptake of online services, for example for entertainment, for building personal relationships, for conducting business and for interacting with governments. Lack of trust also makes us waste time and resources on protecting ourselves against possible harm, and thereby creates significant overhead in economic transactions. [21]

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” [21].

Jøsang et al. define several useful terms and definitions, that will be used in the thesis:

A trustee is a term that is borrowed from the legal terminology. The trustee is a user that states some information that can be trusted or not.

Trustor is the active user, or some other ”thinking entity”, who has trust in a trustee. The trustor evaluates (to some degree) the information that a trustee has given, based on how much the trustor trusts the trustee.

Trust management is the term used about a system, that allows the parties to extract information about each other in order to obtain a degree of how much a trustor trust a trustee. The model that underlies the trust management system is referred to as the trust model.

2.1.2 Formalization of Trust

In his Ph.d. thesis Marsh [23] introduces a method to formalize trust as a computational concept. Marsh presents a model where trust can be represented as a decimal number between -1 and 1. In this interval -1 represents complete distrust and 1 represents blind trust. 0 represents initial trust, where it is not determined if the trustee have trust or distrust. 0 is the neutral starting point when the trustee is inserted in the trust management system.

2.1.3 Trust Model

A trust model is designed to represent the way individuals trust each other. Jonker and Treur [19] base their work in Marsh's model of formalizing trust [23], and introduce a framework for a trust model and conclude that the trust model is not static. The trust model must be able to change over time, and therefore the trust model must continuously process the inputs given to the model in order to determine the degree that a trustee is trusted. Consider figure 2.1 as a simple trust model. A user's trust can evolve over time and therefore it must be updated by verification and validation constantly over time. A plus (+) means that there has been a positive experience and a (-) means that there has been a negative experience.

The user can move between the four different states of the trust model. It is the trust characteristics that define how a user moves up and down in the model. Jonker and Treur argue that any trust model is defined by three different parts: Initial trust, trust dynamics and trust evolution. The trust dynamics define the actual development in trust, and the trust evolution function defines how this development progress.

2.1.3.1 Initial trust

Initial trust defines how trust has to be initialized. When a trustor has interactions with a trustee for the first time, the trustor relies on the default configuration in the trust management system. Jonker and Treur claim that there can be two possibilities for setting op initial trust.

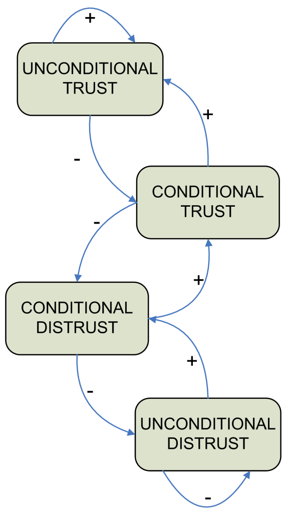


Figure 2.1: Simple trust model as presented by Jonker and Treur [19]

Initially trusting. Without any previous experience the trustee has a positive trust value. This trust value will have to be determined by configuration.

Initially distrusting. Without any previous experience the trustee is distrusted from the start. This trust value will have to be determined by configuration.

2.1.3.2 Trust dynamics

The trust dynamics determine how trust progresses over time, and how much trust is worth when it age. Jonker and Treur distinguish six types of trust dynamics:

Blindly positive defines a trust profile, where a trustor trusts a trustee blindly after a set of positive experiences. After this set of experiences the trustee is trusted blindly for all future interactions, no matter what.

Blindly negative defines the opposite of blindly positive. After a number of negative experiences the trustor will never trust the trustee again, and the trustee will have unconditional distrust, no matter what

Slow positive, fast negative dynamics, define a trustor that takes a lot of positive experiences to build trust to a trustee, but only takes a few negative experiences to spoil the build up trust.

Balanced slow defines a trustor that progresses slowly on building trust and slowly loosing trust.

Balanced fast defines a trustor the progress fast on building trust and looses it fast as well.

Fast positive, slow negative dynamics define a trustor that takes a few positive experiences to build trust to a trustee but takes a lot of negative experience to spoil it again.

2.1.3.3 Trust evolution

One of the central properties in Jonker's and Treur's definition on a trust based on experiences is the trust evolution function. The idea is that this trust evolution function is dynamic. It can change over time, if trust to a given user changes. This means that at some point in time the active user can be a trusting person, where trust progresses fast, but over time the active user can change to a more sceptical approach and therefore trust will not progress that fast. This change could be due to external factors in everyday life.

Jonker and Treur describe a formal framework that helps us define a trust evolution function. They define 16 properties that define the trust evolution function. Ten of these properties are briefly summarized below:

Future independence. A trust evolution function is future independent. This means that the output of the function only depends on the experiences in the past.

Monotonicity. The trust evolution function is monotonic.

Indistinguishable past. It is not possible to determine what actions led to the output.

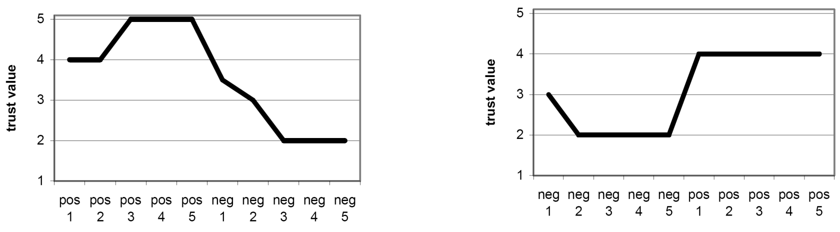
Maximal initial trust. There is a maximum on how much trust a trustee can have initially.

Minimal initial trust. There is a minimum on how little trust a trustee can have initially.

Positive trust extension. The trust can progress positively.

Negative trust extension. The trust can progress negatively.

Degree of memory based. The trust evolution function will forget about the past, leaving old experiences not as valuable as new ones.



(a) Trust dynamics, forgetabillity favors the negative experience.

(b) Trust dynamics, fogetabillity favors the positive experience

Figure 2.2: Forgetabillity in trust dynamics

Degree of trust dropping. The acceleration of distrust differs from trustee to trustee.

Degree of trust gaining. The acceleration of trust differs from trustee to trustee.

In an experiment carried out by Jonker, Treur, Shalken and Threeuwes [18] published in 2004 (five years after the first article [19]) some of the original proposed theories were empirically verified.

In the experiment the test subjects are presented a set of positive and negative interactions with an object. Throughout these interactions the subjects are asked to evaluate how much they trust that given object. The conclusion drawn from the experiment is that the final trust value differs a lot depending on the order that the interactions are presented to the user. On figure 2.2(a) and 2.2(b) two of the results are shown.

We see that the order of the interactions is not indifferent. The final trust value is different in the two cases, even though the interactions are the same. The experiment shows that the interactions, which a subject has had with an object weight more the closer they are in time. The trust dynamics need to contain some way to represent that old interactions count for less than new interactions. This property is defined as *forgetabillity*.

2.2 General Research on Recommender Systems

Reputation systems for the Internet have been proposed ever since the content of the Internet grew too large to keep track of. Several companies have introduced software that helps users assess the quality of information and services on the Internet. This section describes different solutions for recommender systems, which have been proposed. Later (in section 3.2.2) we discuss the pros and cons of these suggested solutions.

2.2.1 Content Analysis through Attributes

The European Consumer Center Denmark (ECCD) has as a part of a cross European-country project introduced an internet based shopping assistant called Howard [33]. Whenever a customer wants to make a purchase in an internet shop, Howard can provide information on when the company was started, and if the company holds some of 29 different European trust marking schemes. Howard also links to information on the company and previous versions on the site, through `archive.org`. ECCD has made the shopping assistant in order to help the consumers avoid fraudulent and frivolous web traders, get good advice on shopping online and knowing your rights when shopping online in Europe.

Dondio et al [12] propose a system minded on the Wikipedia. The system is designed to analyze a set of attributes of an article on the Wikipedia. This system looks on the history of the article, and evaluates an article on how is has evolved. There are several properties that are considered in the history. A selection of these properties is:

- Written by an expert.
- Clear leadership in the development.
- Constantly reviewed by authors.
- The article is stable.
- The article's length.
- The number of other articles that links to this article (importance).
- The article is well referenced.

All these properties are assessed in a computational approach. The proposed algorithm evaluates all the attributes or the lack of them, and creates an estimate to the trust worthiness of the article. This evaluation is made by a set of logical rules that determines the quality of the article, such as:

1. IF leadership is high AND dictatorship is high THEN warning
2. IF length is high AND importance is low THEN warning
3. IF stability is high AND (length is low OR edits is low OR importance is low) THEN warning

These rules are interpreted as following (rule no. 1): There is one author that has written the majority of an article is written by one person and the same person reverts a lot of changes that other users contributes with. This indicates that there is a chance that the article is probably not neutral, and therefore a warning flag is raised.

2.2.2 Using Trust in a Recommender System

IWTrust [39] is used to introduce trust into a question answering environment. The idea is that IWTrust tries to proof that an answer to a given question is correct. The IWTrust introduces a TrustNet, which is a network of trusted users, who can contribute to a proof. Every user that is connected to this network has a trust value - a degree of how much this user is trusted. IWtrust uses Proof Markup Language (PML), as defined by Pinheiro et al [10], to determine if the content is of high quality or not. Proofs that originate from trusted users weigh more, and there fore trusted users will influence the result more. IWtrust has a answering engine that aggregates all these proofs to a final answer to a posed question.

2.2.3 Rating Content

MyWOT [4] is a free product from Against Intuition Inc. It is an extention for the Mozilla Firefox browser. The extention uses user feedback to gather reputation information about websites. Through this extension the users can give a recommendation on how a visited site is as a business partner, how it keeps personal information, and how safe this site is for children. There are several trust and privacy issues to the myWOT approach to reputation. The reputation system relies on a single database, which collects the reputations given, and therefore the users browsing history will be stored at the myWOT database.

MovieLens is a movie recommendation Internet portal [1], maintained by Group-

[1] http://movielens.umn.edu/

Lens Research at the University of Minnesota[2]. MovieLens carries out an experiment [24] on the users of the MovieLens site, where the users are asked how high error rate is accepted before the users will be annoyed by the recommendations. The experiment has several findings. When a movie has about 80 ratings, then the error rate (the chance that the recommendation is wrong) is about 5%, and this is accepted by the users. If movies have a lower number of ratings there will be a higher rate of errors which annoy the users. However, the users accept this high error rate as long at they are informed that this recommendation is insecure, because of a low number of ratings. The experiment shows that as long as the users are informed on calculations risk, and how calculations are made, they will not get annoyed with the recommendations given. When calculating recommendations to its' users, the MovieLens calculates a recommendation based on users that rate similar to the active user. In this way the recommendations are calculated based on the decentralized database.

2.2.4 General Recommender Systems

Jøsang, Ismail and Boyd [20] have conducted a survey of a wide range of existing online reputation systems. Basically, this survey points out four basic criteria that the quality of a reputation system can be judged on. (1) Accuracy for long term performance, (2) Weighting towards current behavior, (3) Robustness against attacks, (4) Robustness against single votes. Jøsang argues that making sure that these criteria are implemented satisfactory, will give a good reputation system, that the users will have confide in.

2.3 Security

This section gives a brief overview of the security measures used in this thesis, to secure the ratings and the WRS from attacks that can compromise the system and the ratings.

2.3.1 Asymmetric Cryptography

Asymmetric cryptography is a form of cryptography in which a user has a pair of cryptographic keys - a public key and a private key. Asymmetric cryptography is also referred to as public key encryption.

[2] http://www.grouplens.org/

The private key is kept secret, whereas the public key is not secret. This key is normally kept in a certificate that the owner can publish. The public key derives from the private key, but the private key cannot be found from the public key. Asymmetric cryptography is used for two purposes:

1. Exchanging symmetric keys, which is used to symmetric key encryption, where the same key is used for decryption and encryption.
2. Creating and verifying digital signatures.

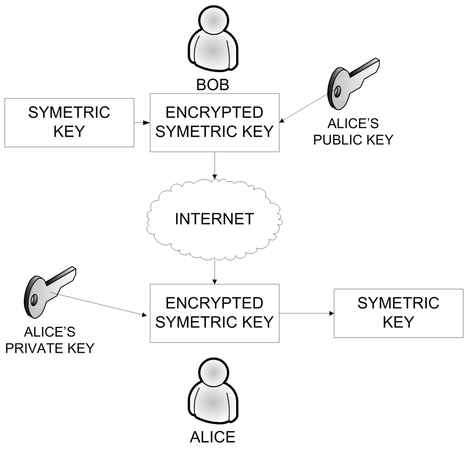


Figure 2.3: Encryption an decryption with a public and private key cryptosystem

Exchanging symmetric keys are shown on figure 2.3. Bob chooses a symmetric key of his choice and encrypts this with Alice's public key. Bob sends the encrypted message to Alice and she decrypts the key with her private key. Because Alice keeps her private key private, she alone is able to decrypt the message. Alice and Bob now have a symmetric key that they can use for transmitting data between one and other, with the symmetric key.

As shown on figure 2.4 asymmetric cryptography can also be used to as a digital signature scheme. A digital signature scheme simulates the security properties of a signature in digital form (rather than written form). Digital signature schemes consist of two different algorithms: One algorithm for signing data and one for verifying data. When Alice wants to send some information to Bob, she wants to make sure that the message, that she sends is not modified or some one is able to send information on her behalf. When sending a message to Bob,

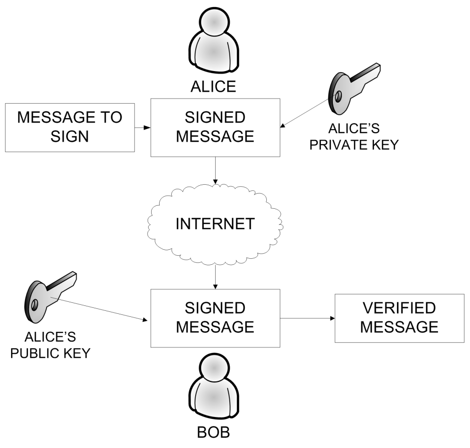


Figure 2.4: Signing and verification of a message with digital signature algorithm

Alice signs (the same procedure as encryption) this message with her private key. She now sends the encrypted message along with the clear text message to Bob. As in this case Alice and Bob do not have the need for the message to be secret, but only to be able to determine that the sender is truly Alice. Bob can now verify that Alice sends this message by decrypting the message with Alice's public key. If the decrypted message is the same as the one in the clear text, Alice must be the one that has send the message, as she is the only one that has the private key that was used to encrypting the message.

2.3.2 Key Management

Key management includes all of the following actions in a crypto system: Key generation, key exchange, key storage, key safeguarding, use, replacement of keys and infrastructure there are between the keys.

Sun Microsystems have developed the tool KeyTool [6], which can be used for key management. KeyTool offers the possibility of generating keys, storing keys, safeguarding keys and keeping an infrastructure on the keys.

In order to use a digital signature scheme as described in section 2.3.1, we need to have a public key infrastructure (PKI). The PKI arrange for entities without prior contact to be authenticated to each other, and to use the public key information in a public key certificates to encrypt messages to each other or digitally sign messages to each other. In other words, the PKI enables the users in the PKI to identify each other.

In a PKI a Certificate Authority (CA) created a certificate and signed the certificate by itself (a self signed certificate).

The CA can issue certificates to the users and these certificates are signed by the root certificate. The new certificates can now issue a new level of certificates and then build up a large hierarchy of users with a certificate that trust each other. This is due to the fact that they at some point further up in the hierarchy will have a common entity that they both believe in. A PKI hierarchy are shown in figure 2.5.

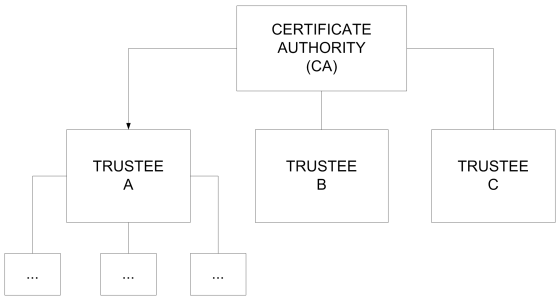


Figure 2.5: A model of Public Key Infrastructure. Trustee A trusts Trustee B because they both got their certificate from the same CA.

Web of Trust ia an alternative way of ordering the certificates. In the Web of Trust concept users build a trusted network based on derived trust. Users build up their network of trusted users by trusting other users that their trustees trust.

A third approach is not to have a PKI at all. All users have their own self-signed root certificate that they use to verify their identity.

2.3.3 Attacks

When designing a system cith some sort of communication between two entities there are three kinds of attacks that a system needs to be resistant to.

In this thesis we will only focus on the active attacks and not on the passive attacks. A passive attack (like eavesdropping) will not benefit the attacker, as there is no information that has to be kept secret.

2.3.3.1 Fabrication

Nobody should be able to send unauthorized messages. When Alice receives a message from Bob, she has to trust that Bob is the sender. It should not be possible for any other entity to send a message on behalf of Bob. See figure 2.6. We do not want anybody to be able to fabricate any false messages.

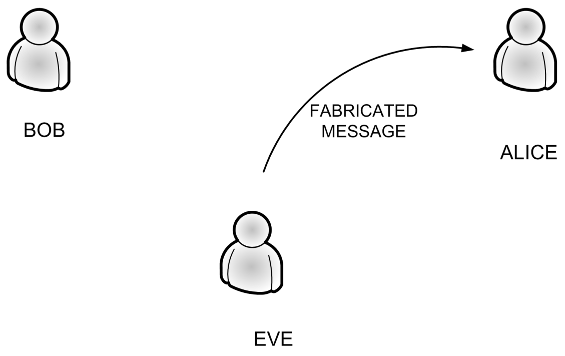


Figure 2.6: Eve fabricates a message to Alice, making it look like it originates from Bob

2.3.3.2 Modification

When Bob sends a message to Alice, she should be confident that the content of this message is from Bob. A system should be resistant to attacks where unauthorized persons can alter the content of a message, as shown on figure 2.7.

2.3.3.3 Deletion/Denial of Service

A system should be secured, in a way that malicious entities is not able to prevent authorized users to use the system. If a person is able to delete all the messages that Alice sends off to Bob and vice versa the system is useless. This is shown of figure 2.8.

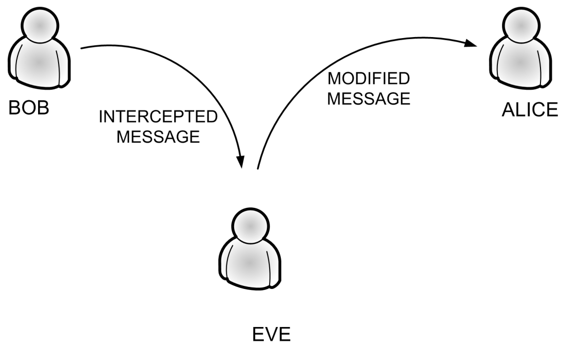


Figure 2.7: Eve intercepts a message that Bob sends off to Alice. Eve alters the content and forwards it to Alice

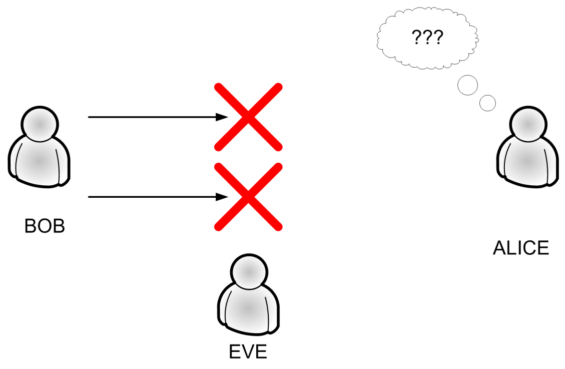


Figure 2.8: Eve intercepts all message that Bob sends off to Alice. All messages are deleted, and Alice never receives a message

2.4 Programmable Proxies

There are a few programmable proxies available as open source. The three proxies that have been analyzed in order to be determined if it could be used in this project are. PAW [28], Muffin [25] and Scone [36].

2.4.1 PAW

PAW (Pro-Active Webfilter [28]) is a Open-Source filtering HTTP proxy based on the Brazil Framework[3]. Paw is very light and it is easy to write plugins to the proxy. Paw is developed to work with Java 1.1 and supports costume made plugins. Paw only supports HTTP connections and not SSL connections.

The light programmable proxy seemed ideal for the development of the WRS. However, PAW is based on Java 1.1 and it does not compile with Java 1.5, as a lot of the operations used are deprecated. Developing with 1.1 would introduce a lot of problems with threads and network connections that have been implemented in later versions of Java. Furthermore, documentation of PAW is very sparse.

2.4.2 Muffin

Muffin [25] is a web filter written in Java 1.1 and supports both HTTP and SSL connections, and offers the possibility to create costume made plugins. Muffin is designed to enhance web surfing experience, and can be used to filter out banners, Java applets, protect from privacy threads.

Muffin is a bit heavier than PAW. It offers a range of pre-programmed filtering options, and runs with 1.5. However, the Muffin proxy is only intended as a filtering proxy and does not offer the possibility to run internal code, perform calculations and interact with a database. Furthermore, last development has been in the early 2004.

2.4.3 Scone

Scone is a programmable proxy that is written in Java and is based on the WBI technology from IMB reseach [17]. WBI is an architecture and framework for creating intermediary applications on the web. Scone is designed as a plugin to the WBI framework, with the intention to develop new web technologies that will enhance the browsing experience. On figure 2.9[4] shows the architecture of Scone.

Scone downloads pages from the web into the proxy. The proxy basically analyzes the streams which flow through the proxy. The HTML pages requested is

[3] http://www.experimentalstuff.com/Technologies/Brazil/index.html

[4] The figure is taken from http://scone.de/architecture.html

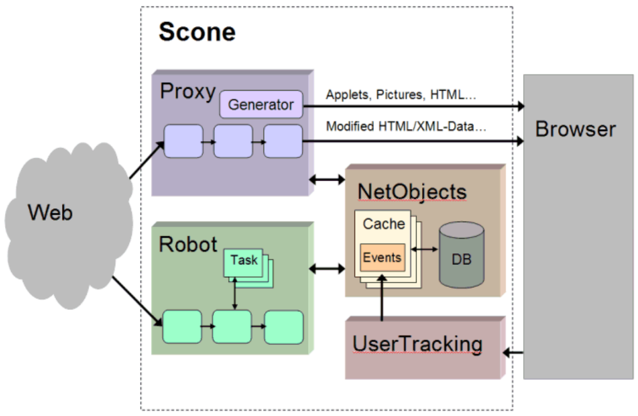


Figure 2.9: The Scone framework

downloaded in to the proxy and is tokenized into token streams and there by the requested page can be manipulated by the proxy. The proxy can operate with NetObjects that can be used to store information about the user, store elements in a database and store previous events and requests.

In addition Scone offers the possibility of having a Robot that can perform tasks and harvest information on behalf of the user. Information harvested can be stored in the database and used for later manipulation in the proxy.

These functionalities can be offered through the programmable interface that Scone provides. Plugins can be written that will enhance the users browsing experience.

Scone offered a large API. It is large proxy based on the WBI framework from IBM. Development of Scone have been continuous and it offered a quite a comprehensive documentation (majority in English and some in German). Scone offered full database access, and a supporting structure for threaded code and network development. Scone seemed the ideal choice for the WRS.

2.5 MediaWiki

A wiki is a web application designed to allow multiple authors to add, remove, and edit content. The wiki is run by a wiki engine, that renders the HTML pages to the browser. There are several wiki engines, but the most popular and the most used is MediaWiki which runs the Wikipedia. [8]

MediaWiki is written in the PHP programming language, and can use either the MySQL or PostgreSQL relational database management system. The general architecture of MediaWiki is shown in figure 2.10[5]. MediaWiki is distributed under the terms of the GNU General Public License.

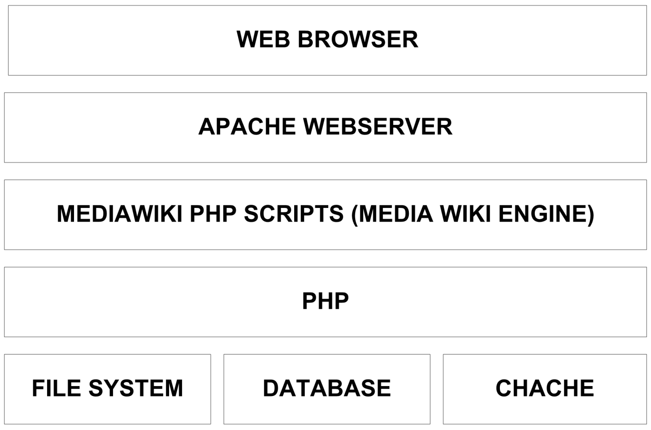


Figure 2.10: The general architecture of MediaWiki.

2.6 Resilient Aggregation

David Wagner has written an article on aggregating of feedback from sensor nodes [35]. The article deals with the fact that in a sensor node network, there may be some nodes that have gone rogue or have been compromised. In either case the node will produce a result, which is not what it was supposed to produce.

Consider a set of sensors in a building measure the temperature and send their result back to a main server in order to calculate the average temperature in the building. If someone holds a lighter close to one of the sensors, then the sensor sends back a very high temperature and the calculated average will be increased, and therefore the node has been compromised.

By using classic ideas from robust statistics, Wagner points out that calculating

[5]The figure is inspired by http://meta.wikimedia.org/wiki/MediaWiki_architecture

a simple average all the nodes are not enough, some robust aggration method is needed. Taken from the field of robust statistics, Wagner points out that a 5% trimmed average, where the upper and the lower 5% are trimmed off and the average is calculated from the rest of the data, is a robust method and will provide a reliable result.

Consider an example set of sorted integer numbers: [2, 2, 3, 3, 4, 4, 5, 5, 5, 6, 6, 7, 7, 7, 7, 8, 9, 12, 12, 12, 37] of 20 numbers. The average of this set is : 8.15 and the 5% trimmed average is: 6.9. The last number in the set, 37, has 17% influence on the untrimmed average. Using the trimmed mean provides a measure for removing outliers that would affect the average significantly. We will use this approach when aggregating the several recommendations in to one, in order to minimize possible outliers affecting the result significantly.

2.7 Semantic Similarity between Sentences

In 1973 Richard P. Honeck published his article ”Semantic Similarity Between Sentences” in Journal of Psycholinguistic Research [15]. This article presents a method to measure if two sentences are semantically similar. Consider the two sentences: *John's uncle shot the sheriff* and *The brother of John's mother shot the sheriff.* The sentences are not the same, but the meaning is the same. After Honeck published this article a lot of research was made to this topic and in 1985 psychology professor George A. Miller from Princeton University began development on WordNet [5]. WordNet is a semantic lexicon for the English language, which groups English words into sets of synonyms. The purpose of WordNet is to support automatic text analysis and artificial intelligence applications. WordNet groups words like shown of figure 2.11 [6]

Honneck's work and the research provided with WordNet, can be used to determine of two articles in the Wikipedia contain the same information, even though they are not identical. We use this to evaluate if previous given ratings are still valid for the current version of the article.

[6]The graph is inspired by http://www.codeproject.com/cs/library/semanticsimilaritywordnet.asp

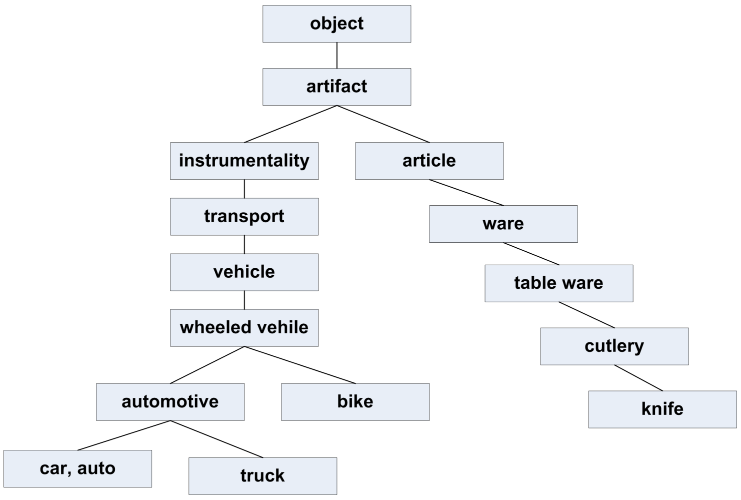


Figure 2.11: Structure of the words on WordNet

2.8 Summary

In this chapter we look at some recent research on trust and trust management, and emphasize a set of definitions and terms that are central for this thesis. Secondly, we go through resent research and proposals of recommender systems that are relevant to our project, and outline the different perspectives that are proposed in this research as being important to creating a recommender system. Thirdly, we give an overview of the security measures and techniques that are relevant for use in this project. Fourthly, we analyze a set of open source proxies that could be relevant for development of the WRS, and sum up the reasons for choosing the Scone Proxy over the other proxies proposed. Finally we look at some areas of research that are not within the area of computer science, but still are relevant to this project.

CHAPTER 3

Analysis

In this chapter we present a general scenario for the use of the WRS. We describe the different challenges that will have to be addressed in the implementation and we analyze of the Wikipedia as it is. In this chapter a specification of requirements for the WRS will be presented based on the analysis of the different aspects of the WRS.

3.1 The Scenario

As described in section 1.1.2, the general purpose of the WRS is to the user with a personalized recommendation based on trusted users.

When a user downloads an article from the Wikipedia the user is presented with a recommendation that informs the user about the quality of the article. This recommendation is created by gathering all the recommendations that are assigned to that article for analysis. First the ratings have to be verified, to prevent an attacker from inserting false ratings into the system. When the ratings have been verified the ratings provided by users, which the active user has had interacted with before, are extracted and aggregated into the combined recommendation that is presented to the user. The aggregation is based on the trust value that the other users have (the trustees). If a trustee for instance has a high trust value this trustees recommendation will influence the aggregation more than a the trustees that have a low trust value.

The active user is now asked to perform some feedback on the recommendation and on the article. This feedback is used to update the trust values of existing trustees and evaluate potential new trustees.

3.2 Specification of Requirements

This section define the functional and the non-functional requirements of the WRS. First we describe a set of basic requirements and after this other recommender systems are discussed. Finally, we describe which areas will be in focus in the development.

3.2.1 Basic Requirements to a Recommender System

Roger Dingledine [27] has proposed at set of basic criteria to asses the quality and robustness of a reputation system. In a survey, by Jøsang, Ismail and Boyd of online reputation systems [20], the four most important of these criteria are outlined as the following:

1. Accuracy for long-term performance. The system must reflect the confidence of a given score. It must also have the capability to distinguish between a new entity of unknown quality and an entity with poor long-term performance.

2. Weighting toward current behaviour. The system must recognise and reflect recent trends in entity performance. For example, an entity that has behaved well for a long time but suddenly goes downhill should be quickly recognised as untrustworthy.

3. Smoothness. Adding any single recommendation should not influence the score significantly.

4. Robustness against attacks. The system should resist attempts of entities to manipulate reputation scores.

In the requirements we will try to satisfy these four criteria in order to get a recommender system, which is robust and so the recommendations that are provided to the users will satisfy them.

3.2.2 Other Recommender Systems

Section 2.2 gives a brief overview of existing recommender systems. The pros and cons of these recommender systems are discussed in this section. This leads to the identification of the attributes which we think is valuable and will try to build into the WRS, and which attributes we should try to avoid.

When using Howard the Shopping Assistant [33] (section 2.2.1), the active user has to start a separate browser window, and enter the information on the internet shop (the URL and the CBR number[1]). A problem with this approach is that this solution is centrally controlled, and it is up to the ECCD to update the database. Furthermore, the user has to find the CBR number, which could be hard if it is a fraudulent shop. The users have to make their own decision, based on the facts provided. Centrally controlled values, like requirements to obtain a trust marking scheme, tend to favor the majority. In addition Howard's usability is not the best solution as the users have to open a separate bowser window and search for information by them selves

Dondio presents a recommender system for the Wikipedia [12], which analyzes the articles attributes. We believe that evaluation of the content based solely on the content attributes (as Dondio and Howard does it) is not enough to evaluate the content. In such an automated system there is no room for ”soft” issues. Such as the language of the article, neutrallity, containing present day information etc. These soft values can only be detected by a human reader, and therefore we believe that recommendations will be the better choice to determine trust.

IWTrust [39] (See section 2.2.2) proposed a network of trusted users. We adopt the idea of having a network of trusted users, where each member has a trust value, in the WRS. With this trust value users find other users that are similar to themselves.

Although some simple tools based on reputation, such as the WOT extension for the Mozilla Firefox browser [4], are starting to appear, they typically rely on a single database for all reputation information, and all user feedback is collated into the same database, and an overall average calculated from all

[1]Central Business Register - `http://www.cvr.dk/Site/Forms/CMS/DisplayPage.aspx?pageid=21`

the recommendations. As earlier mentioned, there are obvious problems with relying on a single centralized database of feedback for a recommender system that should provide useful information across national, political, social, religious and cultural boundaries. Storing recommendations in one single database gives the advantage that the recommendations are always available and tamper proof. But it gives the disadvantage that the result is also calculated centrally, and therefore a minority is not able to remove the influence from the majority from their recommendation. Therefore centrally calculated recommendations only favor the majority of the users, which is not preferable. Experiments with the myWOT extension show that when giving ratings to sites, which are not in the WOT database, this single rating will be displayed to subsequent visitors. For example, when a site that is not in the WOT database is rated the lowest possible mark, the site will be tagged as a malicious site and it will not be recommended to subsequent users. They will be presented to a warning that this site is unsafe and are urged not to carry on their actions.

The MovieLens project addresses the problem with centralized calculation (as outlined above), by analyzing how a user rates movies, and tries to establish a collection of raters that rates similar as the user. The rating from the users that rate alike can be used as recommendations. We want to use this approach in the WRS, by only using the recommendations from users that rate alike and by calculating ratings based on a decentralized database.

3.2.3 Functional requirements

- The WRS should have a trust model implemented that maintains trust values and perform trust update operations
- The WRS should give the active users recommendation based on other recommendations from trusted users.
- The WRS should keep information on the trustees and calculate trust values.
- The WRS should be able to continuously update trust information through the user feedback.
- The WRS should be able to secure ratings to prevent them from being falsified (masquerade attack).
- The WRS should be able to verify that ratings have not been tampered with (modification attack).
- The WRS should be able to determine if a rating is too old to contribute to the aggregation, due to the large amount of change in the article.

- The WRS should work with an out-of-the-box installation of MediaWiki.

3.2.4 Non-Functional requirements

- The WRS should work as middleware between the user and the Wikipedia.
- The WRS needs to be platform independent and browser independent.
- The user's browsing experience should not be worsened by the use of WRS.
- Ratings should be stored centrally.
- Recommendations should be calculated decentralized.

3.2.5 Area of Focus

In this thesis we have chosen to focus on the topics that are most relevant to the computer science and the trust management part of WRS. Therefore there are some areas that are still in need of research and further development. This is summarized in section 8.1.

The main focus has been on development, implementation and testing of a realistic trust management system, how a decentralized database is extracted from these trust values, how the trust management system deals with a no-configuration requirement, and on how the feedback from the user should be interpreted. Furthermore developing the WRS as middleware, implemented with a open source proxy has been one of the main areas of focus. As a result of this large focus on the development there has also been a large focus on testing and benchmarking the WRS.

Secondly there has been focus on development, implementation and testing of the security measures, infrastructure and the functionality that is needed in the WRS.

3.3 Wikipedia Architecture

It is a requirement that the WRS is implemented as middleware, because the Wikipedia is treated as a legacy system. It is a requirement that the system works with a clean MediaWiki installation, and no changes should be required in the underlying Wiki engine.

In order to access data from the Wikipedia, an HTTP connection should be used, and the facilities that comes with submitting data over the HTTP forms. This section gives an analysis on the options available from the Wikipedia.

An article on the Wikipedia is a description on a topic that is presented to the active user like a normal encyclopaedic entry. An article has several sub-pages, which are useful in the WRS. Please consider figure 3.1.

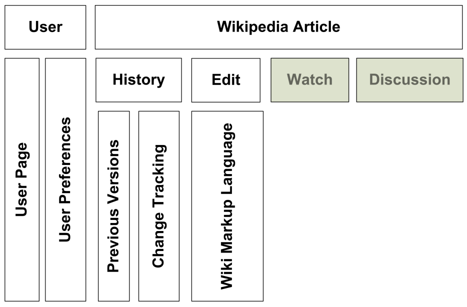


Figure 3.1: Simplified overview of the Wikipedia architecture

Each article has the main Wikipedia article presented to the viewer. Further more there are 4 pages that are related to each article. The history page, the edit page, the watch page, and the discussion page. The watch page is not visible unless the user is logged in. In the WRS, only the edit page and the history page are used.

The basic philosophy behind the a wiki is that everyone should be allowed to edit everything, but that it should be easy to restore the document to its prior state if the modifications are considered undesirable. The traditional security process is based on prevention, detection and response, where security mechanisms are introduced to prevent unauthorised access to protected resources. Auditing procedures and intrusion detection systems are introduced to detect unauthorized use of the system. A combination of automatic and manual procedures are used to stop unauthorized access and return the system to a consistent state. Applying this to the wiki philosophy, we see that there are few mechanisms to prevent malicious or accidental modification of a Wiki article. Detection is left to the users and the only means of response, is to restore the previous page.

The edit page is where the active user can alter the content of the article. The main content in the edit page is a HTML textarea, where the article can be written in plain text. It is not possible to format the article in a rich text editor. The textarea, however, supports Wiki Markup Language (WML), which is parsed by the wiki engine into normal HTML in order to make the article displayable by the browser. The WML has a wide range of tags that can be used to write Wikipedia articles.

The history page contains all the previous versions of an article. In the history page it is possible to see which users (or IP addresses, if the user is not logged in), have made modifications to the article, when these changes were made and a small summary of the changes. The history page can be used to show the difference between two prior versions of an article. Wikipedia stores all the previous versions of an article. Finally, the history page also holds the possibility to revert a page prior state if the current version has been vandalized.

The watch page is used to keep a page under surveillance. If a watched page is changed then the user, who has that page under surveillance, is informed of the change. This helps vandalism to be reverted fast. **The discussion page** is used to discuss possible changes or additions in an informal tone. The watch page and the discussion page is not used in the WRS, and are therefore not taken into further consideration.

When a user is logged in to the Wikipedia there is a personal user page available. On the active user page the user can set up preferences for the Wikipedia and keep a personal homepage, where the user can write some personal information. Due to the nature of the wiki philosophy all user pages are editable by everybody else. Wikipedia users cannot lock their user page for further editing by others. The only way to protect a user page is to keep it under watch.

3.4 Key Challenges

The specification of requirements set up some challenges that will have to be addressed. This section points out a solution to the challenges and why this solution is chosen.

3.4.1 Network Layout

Implementing on top of the legacy system, without modifying in the Wikipedia source code, leaves us with several options. One of the obvious solutions is to create a plug-in to a browser. An other option is to create a proxy, between the user and the Wikipedia, that keeps track of trustees, parses ratings, calculates trust values and calculates averages. In this scope of the project the proxy is chosen, because developing an extension will take up too much time and is not part of the objective of this thesis. Furthermore a proxy also supports multiple user, when integrated in a network environment, as network users would only have to point their browser to the network address of the proxy. The network layout is shown in figure 3.2.

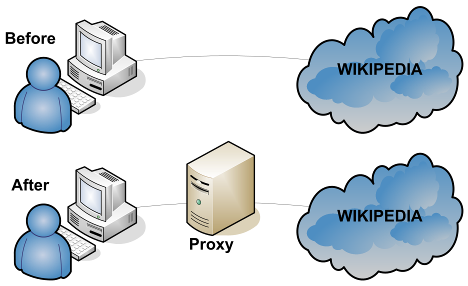


Figure 3.2: Before and after the proxy is inserted in the the network.

3.4.2 Recommendation Repository

The recommender system requires a repository that stores feedback (the recommendations) from users and makes these recommendations available to other users. This repository may either be implemented as a distributed database running on the proxies or as a centralized database running on a single separate server. Neither of these solutions are desirable because they introduce additional complexity and dependencies into the system, as some sort of P2P system between the proxies would be required. However, the Wikipedia itself provides a repository of information that can be read and updated by all its users, so it should be possible to store recommendations on the Wikipedia servers. We do not wish to modify the MediaWiki system to include recommendations, but instead propose to store recommendations as WML comments on the edit pages of the Wikipedia articles. This allows us to store the recommendations in a central location, available to everyone, without modifying the MediaWiki system or interfering with Wikipedia users who do not wish to avail of the recommender system (the WML comments will not be rendered by the user's browser). This solution also allows the WRS, running on the user's proxy, to create a subset of these recommendations based on trust in the recommender and use this subset as a decentralized database.

3.4.3 Security

Some measurements towards security have to be taken in order to prevent falsification of the recommendations, because we store the recommendations on in the edit page. As described in section 3.3, every one is able to change what is in the edit page. Therefore we need some security measures to protect the ratings and a security infrastructure to enable authentication between entities.

There are three different approaches to making a security infrastructure. The first is to have a public key infrastructure (PKI) where a Certificate Authority (CA) issues certificates to the users. The second approach is to user Web of Trust, and the third approach is not to have PKI at all. All the users have their own self-signed root certificate that they use to verify their identity. These schemes are described more closely in section 2.3.2.

The choice of security scheme falls upon having no PKI at all. There are several reasons for this. Making it up to the users to make a certificate themselves removes the administration and maintenance of a CA related to this project. If a CA would have to be operated, there would still be problems in validating the user's identity. Everybody that has an email-address can create an account on the Wikipedia, and if this is the criteria for validating the users identity there would be no point in having a CA. Therefore it is administrative much easier not to have a PKI. Finnally, one of the ideas in the WRS is that the active users have to determine which users to trust and which not to trust through the feedback mechanism provided.

As pointed out by Seigneur et al. [31] a system that has a weak recognition scheme towards identification, will also lack accountability. Consequently, a non-sophisticated identification scheme, like the one implemented the Wikipedia, where it is easy to create an identity, there is no value in the identity. However, there should not be any value in knowing the identity of the trustees that a trustor interacts with, because it is not the identity of our trustees that lead to the trust value, it is the actions that they perform that lead to a trust value, and therefore the actual identity of the trustee is indifferent. The identity (Wikipedia username) is merely used to identify the public key. Therefore the identifier could simply be the public key, but this introduces some problems with the size of the ratings and an unsolved issue when users want to revoke their certificate.

All users therefore keep and maintain their own self signed certificate that contains the public key, which is used to verifying the ratings and a keystore that contains a private key which is used to create signed ratings.

In order for other users to verify a rating, users have to make their certificate public. To publish a certificate the user will need to create a page where the certificate is stored. This is defined as a sub page named `cert` on the user page (`http://en.wikipedia.org/wiki/User:Username/cert`). The certificate will have to be BASE64 encoded, so it will become displayable by the browser. In the user page the certificate is surrounded by <nowiki> </nowiki> tags. This prevents that there would be combination in the certificate that is parseable by the WML.

3.4.3.1 Preventing Attacks

There are several kind of attacks that need to be prevented in order to satisfy the criteria set in 3.2.1 about making the recommendation system robust to attacks, and give the user trust that the recommender system cannot be compromised. The primary attacks that we want to prevent are the sybil attack [13], where an attacker tries to circumvent a reputation system, by creating a large amount of fake identities and use them to introduce a large number of recommendations that will influence the recommendation result significantly towards the attackers favour. In the WRS the sybil attack is attempted prevented by ignoring recommendation from unknown users, with whom there have been no other interactions.

As Wikipedia articles, including the WML comments, can be modified by anyone, we need to secure the ratings, to prevent attacks described in section 2.3.3. Ratings have to be secured against masquerading, modification and deletion. Masquerading and modification can be prevented by introducing cryptographic measures, where every rating is signed with a private key and in order to verify the rating other users have to download a public key from the user page on the Wikipedia (as described in section 3.4.3). The wiki philosophy states that everybody can delete everything, which makes the WRS open to DoS attacks. It is, however, always possible to revert the existing article to a previous version of the article if this happens. Normally when a page is vandalized the page restores quite fast by users of the Wikipedia [14]. As another result of the wiki philosophy the certificates that are stored on the user page can be modified by everybody. Modifying the certificate leaves it useless as the public key cannot be extracted and therefore ratings cannot be verified. In order to prevent this the WRS only downloads certificates that are modified by the user that owns this certificate. This means that the user will have to be logged in to the Wikipedia to make changes to the certificates, such as revoking it.

3.4.4 Article Versioning

Another problem is that the contents of an article change over time, so ratings that were submitted for a prior version of an article may not longer be valid, because the contents of the article have changed. It is obvious that ratings should still be valid if changes only are minor (e.g., typos, adjectives, punctuation etc), but the rating should no longer be valid if the changes are extensive or contains words that may completely change the meaning of the text (e.g., words like not, no, don't, without, does not, un-). We therefore introduce a threshold, which defines a limit to acceptable change. As default this percentage is set to 15 % difference.

It is quite a complex problem to determine the difference between two sentences. A method to determine this difference is described by Honeck [15] and is outlined in section 2.7. In this implementation we use Wikipedia's history page to obtain the difference between two versions of an article. The implementation counts the number of changed words and holds it up against how many words that are in the article. The percentage of changed words gives the percentage of the change. There are basically three kinds of changes:

Reversal. If the some of the following words are added to the paragraph, then the paragraph is consideres reversed: "not", "dont", "don't", "doesnt", "doesn't", "no", "without", "wont" or "won't". If there are added following to a word: "n't", "esn't" or "un", the paragraph is also considered reversed. If the meaning of the content is reversed, then the entire paragraph is considered changed.

Amplification and typos. If a change is a single character is it likely to be a punctuation correction, spelling correction, bracket, or other minor change. Therefore these changes does not count as a change.

Addition. Extra words in a paragraph is counted and added to he total amount of changed words.

This will provide a naive idea of how changed much a text is changed. However, more in-depth research is needed by a linguist (See section 8.1).

3.5 Summary

This chapter provides a general scenario that describes how the WRS should operate and what functionality it should provide. We take a look on some general research on recommender systems and point out the most important general requirements to a recommender system. With the general requirements in mind we make an analysis of some existing recommender systems and point out what kind of functionality the WRS should have. With basis in this analysis we create a set of functional and non-functional requirements for the WRS.

We make an analysis of the Wikipedia architecture as it is, and analyzes some of the challenges that come with implementing the WRS as middleware. This analysis covers the choice for creating the WRS as a proxy, the choice for storing the recommendations centrally on the Wikipedia pages and the security measures that have to be taken in order to make these choices feasible.

Chapter 4

Trust Model

This chapter shows how trust is managed in WRS. The chapter describes the reason and background for having a dynamic trust function and the motivation behind. It describes the three different parts of the trust model. The initial trust, trust dynamics and the trust evolution model. The chapter also describes the mathematical ideas behind the trust model and what considerations that have been taken in order to define this model.

4.1 Model Background

When working with people in everyday life we tend to trust some people more that others. Often we do not know what is the determining factor that makes us trust or distrust this person. If we have known a person for a long time it is easier to decide whether this person is a trusted person or not, and if we have just met the person it is harder tell if that person is reliable or not.

The trust model for the WRS is based on the same basic principle. When meeting a new person or entity the active user has not had any prior interactions with, it is not possible to determine whether that person is trusted or not. This has to be determined through the actions of the users. Consequently, it can be hard to tell if a user, that the active user only has one or two interactions with should be trusted or not. However, the more interactions we perform with a person the more we know whether to trust this person or not.

There are several motivating factors for making such a trust model. One of the obvious is that if trustors only take account of the trustees that they know and they trust are prevented from having their result polluted by dishonest users.

An other motivation for this approach is to build up relation management without configuration. Configuration is often used to determine what kind of a person a user is:

- Does the user want to make quick progress, is the user more careful and a cautious user? A cautious user needs a lot more positive interactions with a user, than a optimistic user needs, in order to obtain the same trust value.
- Does it take long time to develop trust in another person or does the user only need one or two positive results?
- When a trusted person suddenly acts differently that expected, will this destroy the relation ship or are the user more forgiving and needs several betrayals in order to loose trust.

With a system that requires an initial trust configuration, upon installation or initiation the user would have to decide on the questions above. This would determine what kind of a person the user is in terms of trust. But these things change. Users might change their opinion over time, and become a more optimistic or cautious person, but the user will probably not know that they changed their mind. Another point is that this configuration might not be the same for all the people in the user's ring of trusted persons.

Therefore it would be more realistic to make a trust model for each person that the user is interacting with. Each trustee that a trustor interacts with has a different trust model, that will change over time, based on the feedback that the user gives to the model. Every time a user is given a recommendation for a page the user is asked to tell whether the provided recommendation is satisfactory and what the user would have rated this article. This feedback can be used to determine the kind of relationship between the trustee and the trustor. And with this feedback the trust function is updated. See section 4.3.3.

When a user downloads an article the user gathers ratings from the different trustees that have rated the page. Some of these reviewers have a higher trust value and therefore the recommendation from that reviewer counts more.

This gives the user weighted recommendations, and therefore the result is subjective. Because a user can choose (based on which users the active user trusts) which recommendations should be incorporated in the average the result will be different from user to user.

This approach has a downside. It has a steep learning curve. In order to provide the user with proper results the user has to put some effort in teaching the system about the user's preferences, and therefore the system will not work out of the box.

4.2 General Architecture

In order to create a formal way of representing trust, a set of definitions has to be made. Stephen Paul Marsh defines trust as a variable in the interval $]-1;1[$, where -1 is complete distrust and 1 is complete trust [23]. There are several advantages to this approach:

Sensitivity. When using a small interval, a small change in the trust value can cause a large difference in the final result, This is also applicable for an everyday situation, for example reading a article in the newspaper about a politician can totally change your point of view of that politician, even though you concurred with him or her for years.

Formalizing. The interval defines how much a trust a user has. A number is easy to interpret and implement. The statement ”I trust trustee A a lot” is not easy to implement.

On the other hand Marsh mentions some disadvantages as well:

Subjectivity. It may be different from one user to another user what a drop of for instance. 0.4 in trust means.

Fudging. If the trust value is a simple number it is easier to manipulate and perform attacks on compared to statements (Such as: ”I trust user A a lot”)

4.2.1 Model Basis

In order formalize trust model more dynamic a coordinate system from -1 to 1 on both the X axis and the Y axis is introduced. (Cf. Figure 4.1) This will improve the representation of trust, making it easier to explain and understand. Furthermore, it will ease the implementation of trust.

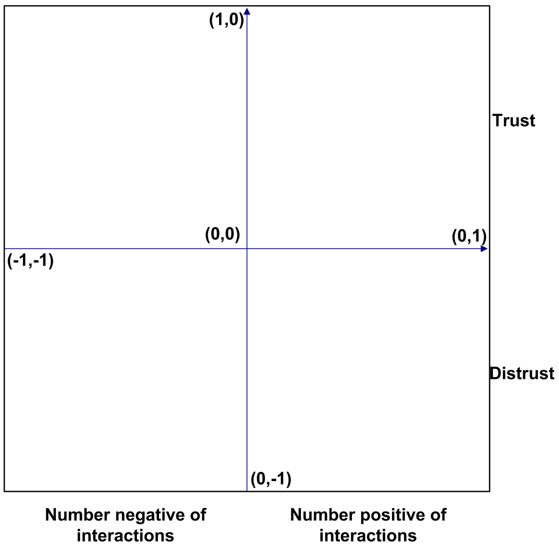

(1,0)
Trust
(0,0)
(0,1)
(-1,-1)
Distrust
(0,-1)
Number negative of interactions
Number positive of interactions

Figure 4.1: Co-ordinate system where trust is represented

Trust is represented as a function, where the trust value is represented on the Y-axis, determining if the trustee is in trust or distrust. Each trustee has his/her own coordinate system. The X-axis determines if there has been a majority of positive interactions or negative interactions. Calculating an X value from the previous interactions, gives the possibility of calculating a trust value as function of the X value. These definitions result in the trust function, which is present in the 1st quadrant and in the 3rd quadrant. For instance, it does not make sense that a user is in distrust and there is a majority of positive interactions. This would be the case if the function was placed in the 4th quadrant. Likewise a situation where the function is present in the 2nd quadrant would be a scenario where the user has trust and majority negative interactions. The construction of the actual trust evolution function is specified in section 4.4 on page 52.

4.2.2 Ring of Reviewers

Each trustee that the trustor has interactions with has a different trust profile and therefor a different curve, because of interactions and feedback differ from user to user. Each curve is regarded as an instance of the trust model that represents each trustee. In order to keep track of these curves we have a trust management system that keeps track of all the curves. We need to be able to store all the data in one place, where the active user can access previous interactions and extract a trust value. In this trust model we define a *Ring of Reviewers (RoR)*, that keeps track of all the information that is extracted from encounters with other users of the WRS. The RoR operates as a database and performs all the operations with respect to trust initialization and trust updating.

4.3 Structure of the Trust Model

The trust model is based on Jonker's and Treur's model of trust dynamics [19] and uses Marsh's formalized way of representing trust [23].

The trust model has three main parts as described in section 2.1.3.

- Initial trust. Represents how a new trustee in the Ring of Reviewers is initialized.
- Trust dynamics. Represents the speed that a trustee in the Ring of Reviewers progresses in trust.
- Trust evolution model. The model defines a function for trust evolution and how this function can evolve to describe to progresses in trust.

We adopt this model because this would give a more flexible trust model than presented in figure 2.1 on page 12. Furthermore Jøsang, Ismail and Boyd [20] points out, that if a recommender system is too simple the users will loose faith in it and the model will not fulfil its purpose.

4.3.1 Setting up Initial Trust

Reviewing articles on the Wikipedia opens the possibility that the newly encountered users can become trusted users. When a new user is introduced into the system, the user has to have some sort of initial trust. Junker and Treur have two options when defining initial trust. Initially trusting and initial distrusting, where both these approaches requires a configuration in trust.

The idea is to have a system, which do not need configuration upon initialization. Therefore the system cannot predict if a user is initially trusting or distrusting. In order to avoid configuration the system has a neutral view of new users. Therefore all new users that are given the trust value of 0. And since there is no interactions with this new user, the user starts the trust function in point (0,0).

4.3.2 Trust Dynamics

Trust Dynamics describes the development in trust. As described in section 2.1.3.2 Junker and Treur defines six ways that trust can progress: Blindly positive, Blindly negative, Fast positive - slow negative, Balanced slow, Slow positive - fast negative and Balanced fast.

These six approaches describe how the X-value on the X-axis moves, by introducing granularity on the X-axis. For example, if the user is a fast positive and slow negative, then when encountering a positive experience then the user moves a big step forward in the trust evolution function (described in section 4.3.3) and a small step back if the experience was negative. This granularity could be used to model the trust dynamics towards the trustee.

However, choosing this approach to trust dynamics would be difficult to implement, due to the requirement that the implemented system has to be able to work without configuration. It is difficult to determine through the users actions what kind of the six approaches the trust towards a trustee can progress.

Instead the trust dynamics is based on later research by Jonker, Schalken, Treeuwes and Treur [18]. With this approach is trust dynamics is based on experience that can change over time and is not defined by different granularities in progress. Instead Jonker et al. sugges that trust dynamics are influenced by forgettabillity as described in section 2.1.3. Our trust model applies this approach to trust dynamics. They perform an experiment where an object has the same positive and negative interactions, the only difference is in which order they are encountered. The experiment shows that the further back in time an event is the more insignificant this event is. There are two ways to approach this:

Amount defined implies that only the latest number of interactions have influence. For example only the last 25 interactions that have influence on the final trust value. If for instance if a trustor has had 25 interactions within a week with a trustee, they are the only ones that count, even though there might be hundreds in the past. On the other hand if the trustee and trustor only have an interaction every second months, the trust value would be based on 2 year old interactions.

Time defined. After a certain point in time (e.g. a year) the interactions do not count any more, no matter how many interactions there have been. If there is only one interaction with a trustee within a year, then that is the only one that will be used to calculate a trust value.

The model adopted in our trust model is a weighted time definition. This means that the longer an item is back in time the less it means. This gives the most realistic approach to a trust system.

4.3.3 Trust Evolution Model

The trust evolution model consists of a trust evolution function and a methods for adjusting the function, in order to have a dynamic evolution function on stead of a static function.

Junker and Treur define 16 properties for the trust evolution function. Ten of these are listed in section 2.1.3.3. These 10 properties will be the basis for developing the trust evolution function. The last 6 properties are not considered as they deal with approximation of the trust evolution function. But as we see later our trust evolution curve does not originate from a data set but from a mathematical formula, there are not need for approximation.

When designing a system that has no configuration, it is impossible to tell what kind of a person is using the system. Therefore the system takes a neutral point of view when a new user is introduced in the Ring of Reviewers. The first curve is a linear function, f(x) = x.

This approach is neutral because the first interaction will lead to the same relative growth in final trust, whether this is a negative or a positive experience.

Every time the user has read an article the user is asked to rate this article, and state if he is content with the initial result given. This feedback is used to update the trust evolution function and helps the recommender system determine what kind of trust profile the user has.

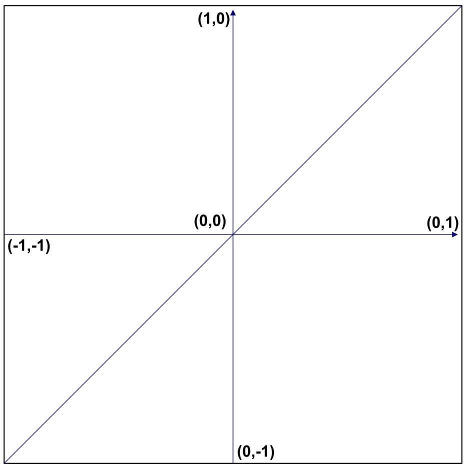


Figure 4.2: Initial linear trust evolution function.

In general people can be more optimistic or cautious when it comes to evolving trust. A user is optimistic if the user tends to trust a person based on a few set recommendations. If a person is more cautious it takes more interactions to obtain a high trust value, where optimistic persons get a high trust value fast. The optimistic and cautions curve is shown in figure 4.2.

We see that both curves ends in trust value 1, but the optimistic curve will have higher trust values while approaching 1. Basically this means that an optimist have higher trust value on a fewer recommendations. This on the other hand leads to recommendations that are based on fewer reviews and therefore is more likely to be misinforming. Fewer interactions mean that the user is not that acquainted with the user. A set of possible curves are seen on figure 4.4.

4.3.4 Adjusting the Trust Evolution Model

As described above the trust function can change over time through feedback from the user. This is done through the recommendations given by the user. The user is asked to give the page a recommendation and state if this presented recommendation is satisfying or not. This gives an idea on what fault tolerance the user has. For instance, if the WRS provide the recommendation 7 (on a scale from 1 to 9), and the user gave the recommendation 5, and stated that the user is content with feedback. This gives an indication that the user is a more forgiving person that does not mind a little slack and therefore the curve is

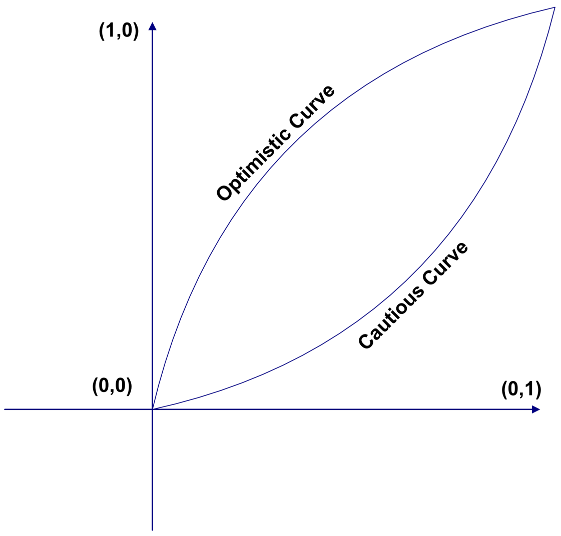


Figure 4.3: Trust evolution function, the cautious and the optimistic curve

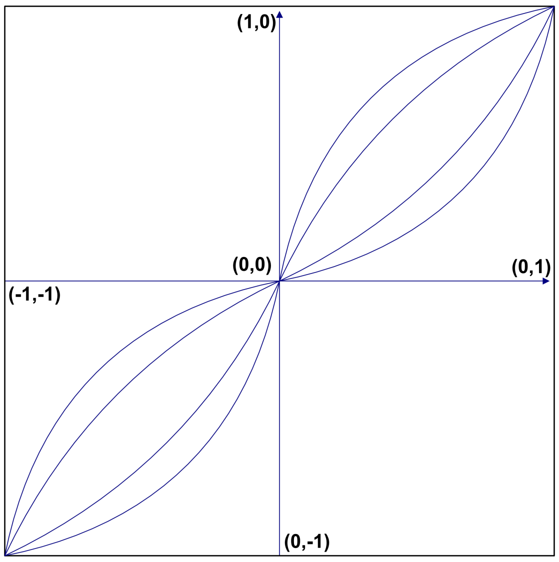


Figure 4.4: Trust evolution function, several possible curves

adjusted towards a more optimistic curve. On the other hand, if the user states that he do not like that the recommendation is 2 steps off, then this states that a user does not like to be betrayed. Hence a more cautions curve should be applied to the users in RoR.

Basically the system should deal with six different situations when getting feedback from the user, and altering the user trust values in RoR

- A user can be in the trust region (1st quadrant) and the recommendation can be a positive experience.
- A user can be in the trust region (1st quadrant) and the recommendation can be a negative experience.
- A user can be in the distrust region (3rd quadrant) and the recommendation can be a positive experience
- A user can be in the distrust region (3rd quadrant) and the recommendation can be a negative experience.
- The user can be in Origin (0, 0), and has a positive experience.
- The user can be in Origin (0, 0), and has a negative experience.

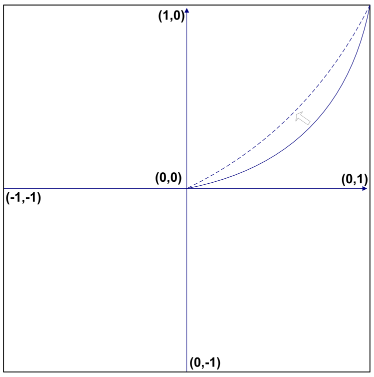
In the scenario where the user is in the trust region and has a positive experience, then this must lead to an improvement in trust for people who rated likewise the user. For the optimistic and the cautious curve the development in the evolution curve would look like figure 4.5(a) and 4.5(b).

The evolution function extends from the full-line curve to the dashed curve. We see that there is a difference in the progress of trust, depending on whether the user is an optimistic person or cautions person.

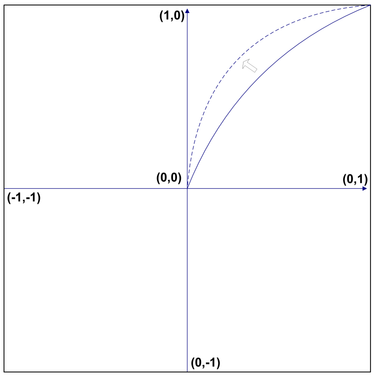
On the other hand if the experience is negative then trust value must be decreased for that users, who provided the information which is unsatisfactory. This is shown on figure 4.6(a) and 4.6(b).

The function moves from the full-line curve to the dashed, and again there is a different progress in trust depending on the user is cautious or optimistic.

When adopting this way of changing the evolution function, there are some definitions that have to be made. In this model it is only when a negative episode have occurred that it is possible to have a cautious curve, and only possible to have an optimistic curve if there has been an positive interaction. This also makes sense to the human nature.

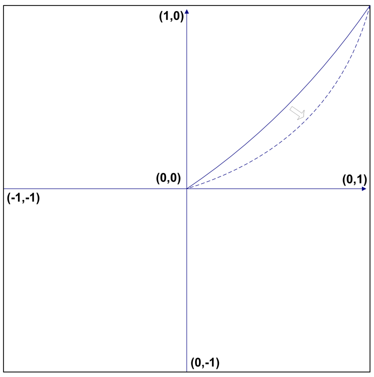


(a) A user with a cautious curve has a positive experience, and the curve is updated accordantly.

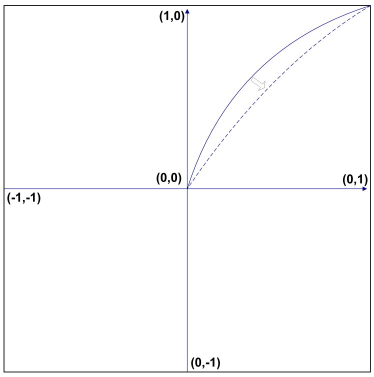


(b) A user with a optimistic curve has a positive experience, and the curve is updated accordantly.

Figure 4.5: Positive experience in trust



(a) A user with a cautious curve has a negative experience, and the curve is updated accordantly.



(b) A user with a optimistic curve has a negative experience, and the curve is updated accordantly.

Figure 4.6: Negative experience in trust

This approach for changing the trust evolution function does also apply to distrust region. If the user has a negative experience then the trust evolution curve is pushed further down towards (0, -1) and if the user encounters a positive experience, then the curve is pushed further up against (-1, -1).

4.4 Formalizing the Model

We want a mathematical interpretation of the trust model as defined by the trust evolution function and the trust dynamics.

In this trust model there are two parameters that can be adjusted in order to calculate a trust value. The first is the X value, which is based on the trust dynamics, and adjustments are based on how the interactions with the trustees are interpreted. A positive interaction is defined by the trustee and the trustor give an article a similar recommendation. The N value is based on the experience with the provided recommendation. If the trustor is content with the provided recommendation, those trustees that contributed significantly have their N value adjusted towards a more positive curve.

4.4.1 Trust Evolution Function

As described in section 4.3.3 the initial trust function is determined be the mathematical function f(x) = x. This is assumed in order to have a neutral basis upon initialization of a user in the RoR.

Using the linear formula as suggested does not bring all the possibilities as wanted.

When the user's profile is changed in to a more optimistic or cautious curve, then this is not possible to make this representation with a linear curve. Another approach would be to represent the functions as a polynomial function with different degrees power. The disadvantage of using a polynomial function is that it is not mirrored in $f(x) = -x + 1$ and $f(x) = -x - 1$, as shown in figure 4.7. By having a function expression that is not weighted equally the steps closest to 0 on the X axis is much less significant to that the steps closest to 1 and -1, which would not be fair as step towards trust should only depend on the curves parameters and not the functional expression of the curve.

A third approach (and the approach chosen) is to represent the trust function as

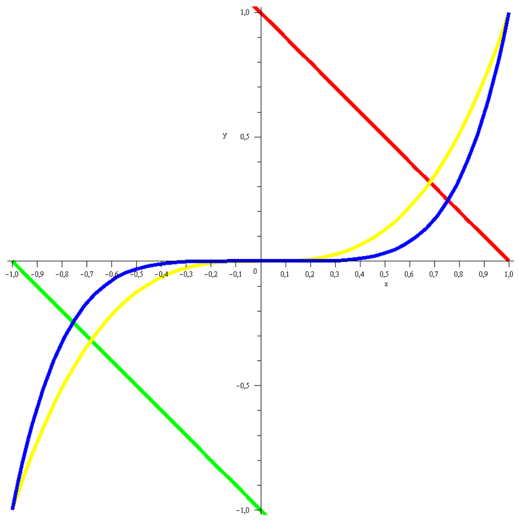


Figure 4.7: Trust Evolution Function represented with a polynomial expression

a superellipse (or Lamé curve). The superellipse is represented by the formula:

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

The superellipse can be adjusted by tweaking the parameters a,b and n. The parameter a and b represent the radius of of the superellipse, and if they are set to 1 then the radius is 1, which fits the function with in the interval defined in section 4.3. In this case it is only needed to operate in the interval -1 to 1, and therefore is is not needed to adjust the a and b parameter. Hence only the parameter n is needed to adjust the n-value in order to update the trust evolution function.

However the superellipse needs some adjustments in order to fit the desired curves as shown on figure 4.4. In the original form the super ellipse for $n = 4$ is plotted on figure 4.8 and defined by the expression:

$$|x|^4 + |y|^4 = 1$$

In order to fit the different part of the superellipse to the trust function, as it is shown on figure 4.4, it needs some adjustments. Four sets of functions are

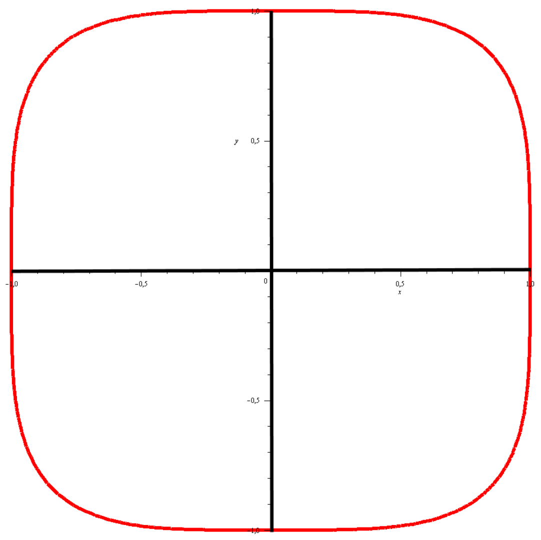


Figure 4.8: The superellipse plotted where $a = 1$, $b = 1$ and $n = 4$

defined in order to fit the different curves needed:

- Optimistic curve in trust. See Equation 4.1.
- Cautious curve in trust. See Equation 4.2.
- Optimistic curve in distrust. See Equation 4.3.
- Cautious curve in distrust. See Equation 4.4.

The four different curves are defined like this, where the parameter a and b are set to 1

$$|x-1|^n + |y|^n = 1 \tag{4.1}$$

$$|x|^n + |y-1|^n = 1 \tag{4.2}$$

$$|x|^n + |y+1|^n = 1 \tag{4.3}$$

$$|x+1|^n + |y|^n = 1 \tag{4.4}$$

Combining equation 4.4, 4.2, 4.3 and 4.1 into a piecewise function that is only are defined in 1st and 3rd quadrant, gives the trust function:

$$Trust(x) = \begin{cases} |x-1|^n + |y|^n = 1, |x|^n + |y-1|^n = 1 & \text{for } x > 0 \text{ and } y > 0 \\ |x+1|^n + |y|^n = 1, |x|^n + |y+1|^n = 1 & \text{for } x < 0 \text{ and } y < 0 \end{cases} \tag{4.5}$$

This function plotted looks like figure 4.9, where a and b are 1 and n is 2:

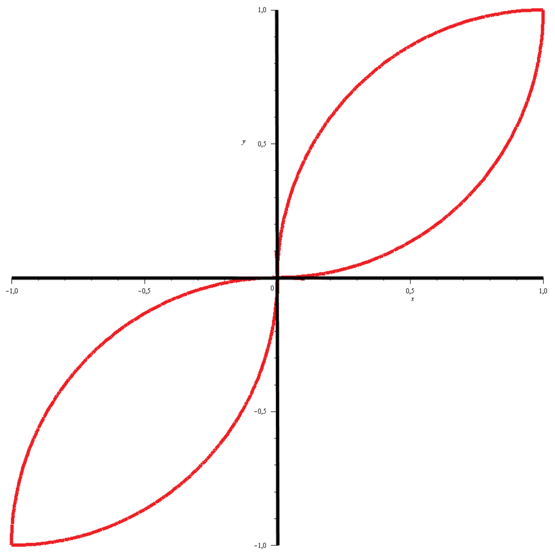


Figure 4.9: The trust function plotted where $a = 1$, $b = 1$ and $n = 2$

This implementation has the advantage that the neutral trust function, that has been assumed to be linear, can be represented for n=1.

4.4.2 Trust dynamics

As described in section 4.3.2 trust dynamics are based on forgetability over time. In our approach to trust dynamics we have made some assumptions based on the experimental research by Jonker, Treur, Theeuwes and Schalken [18]. From the research we see that after 5 successive positive interactions and no previous interactions, the trust value is almost at a maximum. Therefore we make the definition that after 10 successive positive interactions the trust value should be at a maximum.

Based on the coordinate system where the X value goes from -1 to 1, we define a positive experience to an increase to be $\frac{1}{10}$ of the possible interval. As the experiment is based on no previous actions this interval would be from 0 to 1, hence a positive experience should increase the x value with 0.1.

The same experimental research shows that negative interactions reach almost maximum distrust on 5 recommendations as well. Based on this we define a negative interaction should decrease the X value of 0.1.

We define that after 3 months an interaction is only worth 50% of its' original value and after 9 months it is only 25% worth and after a year it is not worth anything any more. For example, if a person has 2 positive experiences within the last week, a negative experience that is 4 months old, a negative experience that is 10 months old and a positive experience that is 2 years old the X value will be calculated as following

$$XValue = 0.1 + 0.1 - 0.1 \cdot 50\% - 0.1 \cdot 25\% + 0.1 \cdot 0\% = \underline{0.125}$$

There is no scientific approach of these time limits and the decrease factors. Consequently, this area needs more research. Please see section 8.1.

4.5 Conclusion on the Trust Model

These definitions on the trust evolutions functions fulfil the 10 properties which were defined as a requirements by Jonker and Treur (See section 2.1.3.3).

Future independence. This definition of the trust evolution function is only dependent on the previous interactions as defined by the trust dynamics.

Monotonicity. The trust evolution function is monotonic, because the functions make sure that a higher X value can never give a lower trust value than the actual trust value. The trust value will always progress positively when the X value progress positively.

Indistinguishable past. We cannot determine the trust evolution curve (N value) or the trust dynamics (X value) by analyzing the from the trust value

Maximal initial trust. There is a maximum trust value of 1.

Minimal initial trust. There is a minimum trust value of -1.

Positive trust extension. The trust can progress positively.

Negative trust extension. The trust can progress negatively.

Degree of memory based. The trust evolution function will forget about the past with the definition of the trust dynamics.

Degree of trust dropping It is possible to change the acceleration of trust dropping by the feedback from the user. By having a larger N value trust can drop faster.

Degree of trust gaining It is possible to change the acceleration of trust gaining by the feedback from the user. By having a larger N value trust can be gained faster.

In appendix A on page 105 an example for calculating the trust values are given.

4.6 Summary

This chapter describes the backgrounds for having a trust model that represent the trust between a trustor and a trustee. First we define a coordinate system between -1 and 1 on both axis, that helps us represent the trust as function. We define a trust model that contains three elements: *Initial trust*, that defines how trust is initialized, *trust dynamics*, which defines the actual development in trust and the *trust evolution model* which is a function expression for how the actual trust value should be calculated. The trust evolution model contains the functionality to change according to user feedback. The trust model is formalized in order to make it easier to implement into the WRS software, and follows resent research.

CHAPTER 5

Design

This chapter describes the design of the system and the problems that are outlined in the analysis (chapter 3) are solved.

5.1 Internal Architecture of the Proxy

The internal design of the WRS are split in to four central modules. The central modules in the system are sketched in figure 5.1 and described here:

- The HTTP module handles the communication between the user and the between proxy, and the communication proxy and the Wikipedia.
- The Page Module handles the HTML pages which are used to find the edit and history page, and the modification of the HTML pages.
- The Rating Module handles the rating that defines the recommendation an article has been given, how these are created, verified and parsed.
- The Trust Module handles the trustees, the trust values that are assigned to trustees, how they are calculated, updated and used to modify the ratings.

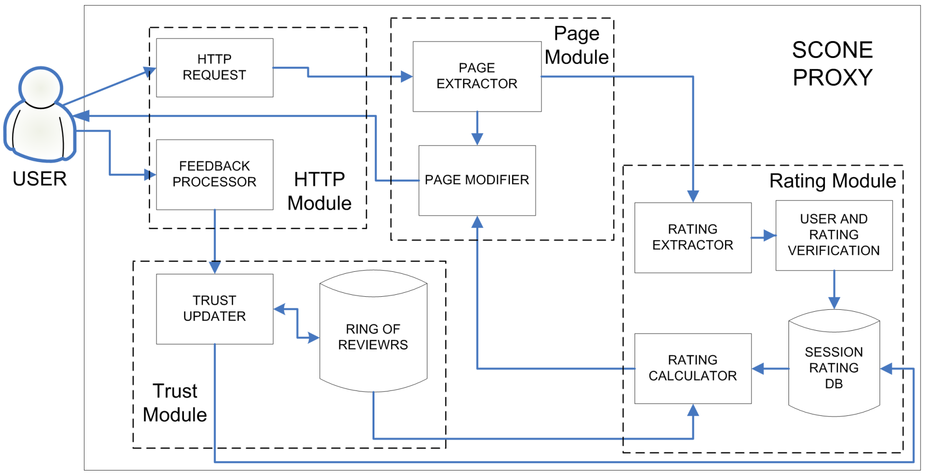


Figure 5.1: Internal architecture of the proxy

5.1.1 Flow in the Proxy

All browser based traffic goes through the proxy. The proxy filters out traffic that is not sent from or originates from the wikipedia.org domain. When a HTTP request comes in to the proxy, it is analyzed and if is not for the Wikipedia domain, it is rerouted from the proxy. If the request is to the Wikipedia domain, the proxy retrieves the HTML page and passes the HTML page along to the Page Extractor module. The Page Extractor then extracts the URL for the edit page and for the history page. The URL for the edit page is then passed on to the Rating Extractor, which extracts the ratings from the WML in the edit page, and stores them in a Session Rating Database, which operates as a temporary database that contains the ratings from an article that are relevant for this session only. The Ratings are then verified with help of user certificates, which are stored in the user page on Wikipedia. The Session Rating Database is updated and the ratings which are not valid or is below the threshold that defines them as unacceptable for this version of the page is removed. (See section 3.4.4). A sequence diagram for finding the ratings can be seen on figure 5.2.

The temporary Session Rating Database is then compared to the trustees that the user previously has had interactions with. The interactions with these trustees are stored in the Ring of Reviewers. These trustees already have a certain trust value, because of previous interactions. The Rating Calculator now calculates a personal recommendation based on the ratings in the temporary Session Rating Database and the trust values in the RoR. This recommen-

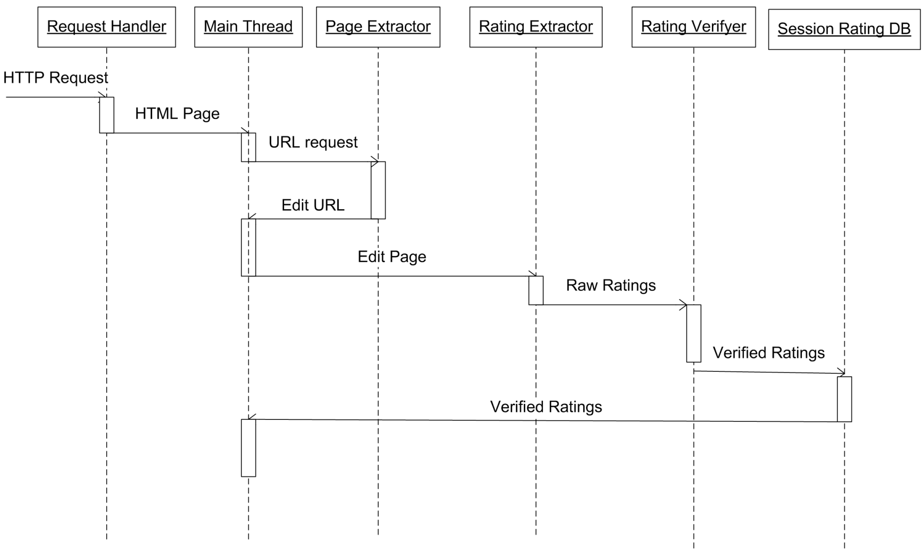


Figure 5.2: Sequence diagram: Finding the ratings.

dation is passed on to the Page Modifier. The Page Modifier then inserts the calculated recommendation into the HTML page along with the feedback option to the user. On figure 5.3 a sequence diagram shows how a recommendation is calculated and given to the user.

The user now gives feedback on the quality of the article and feedback about the satisfaction with the provided recommendation. from the WRS. The feedback is used to update the trust values in the Trust Updater, and the updated trust values and the potential new users are inserted into the RoR. The feedback is the only way to know if the used trust values are correct or not, and is thereby used to determine how trust should evolve for each user in the RoR. Figure 5.4 shows a sequence diagram of the trust update process.

5.2 The HTTP Module

The HTTP module contains the tools for general communication between the proxy and Internet and between the proxy and the user.

The Request Handler takes care on the initial HTTP request by the user. The Request Handler filters out all the requests that are not for the Wikipedia

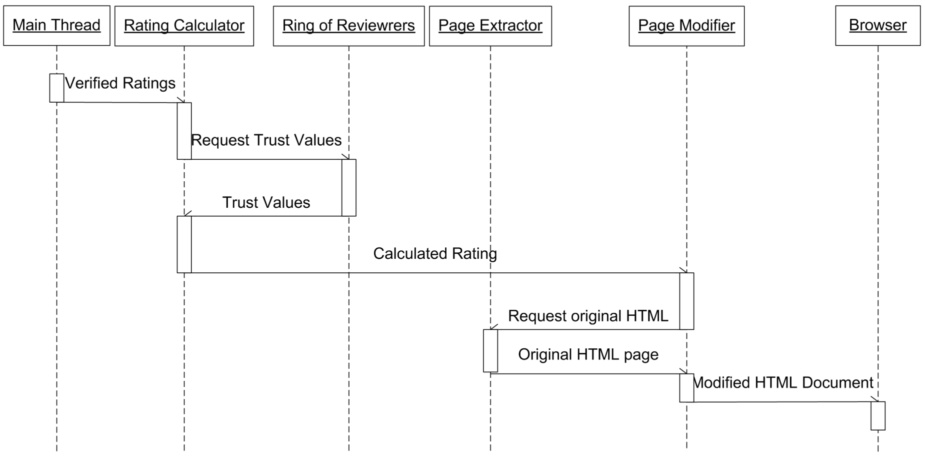


Figure 5.3: Sequence diagram: Calculating the recommendation and feeding the recomendation in a modified HTML document back to the user.

domain. These requests are passed trough the proxy.

When a HTTP request enters the proxy to the English Wikipedia, and the request is for an article, the Request Handler downloads the requested article and then passes the raw downloaded page to the Page Extractor. If the request is for the edit page, discussion page, or any other page that is not with in the scope of the recommendation software the page is simply passed along as if it is a page from out side the Wikipedia domain, because the subpages do not have any recommendations affiliated.

The HTTP Module also contains the Feedback Processor. The Feedback Processor deals with the feedback from the user. In the modified page (see section 5.3) the feedback mechanism is embedded. The user is given two feedback options.

The first feedback option is the possibility to rate a page. The user clicks a link on the Wikipedia article to give the page a recommendation. The feedback processor then creates a rating, as described in section 5.4, and puts this rating into the edit page, and submits the changes to the Wikipedia.

The second feedback option that the Feedback Processor gives the user is a satisfactory feedback. The user can, through a click, tell the system what he/she thinks of the recommendation. This reply is then passed on to the Trust Updater. See section 5.5.

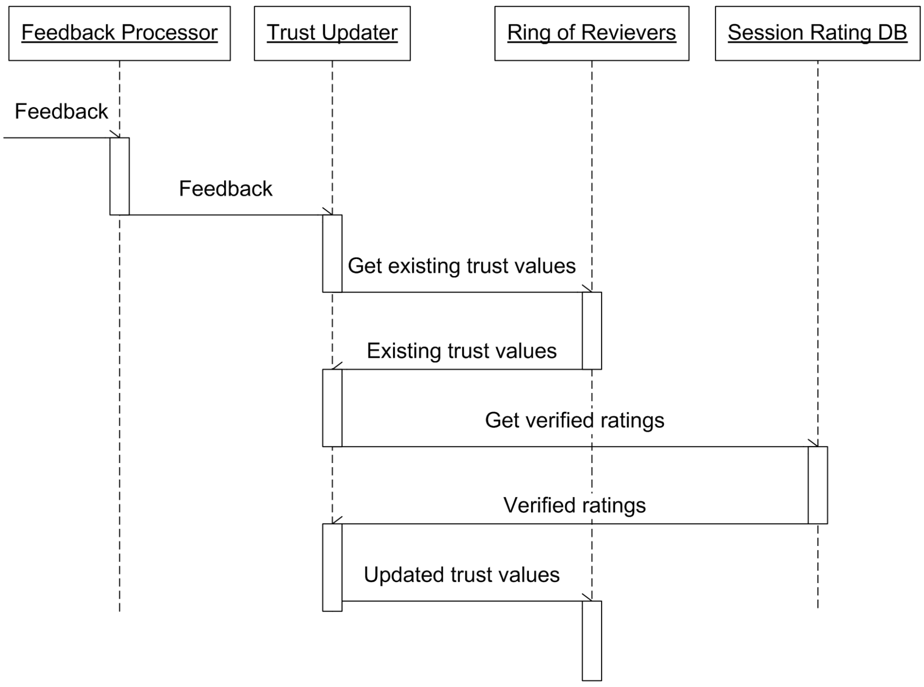


Figure 5.4: Sequence diagram: Updating the trust values based on the userfeedback.

5.3 The Page Module

The Page Module contains the components that are used to work with the HTML pages. The Page Extractor is used to obtain URLs for the edit and the history page, and scans through the HTML documents for the markers that the Wikipedia use to mark the two pages. After the scanning the original HTML document is stored, for use by the Page Modifier.

The Page Modifier is used to change the HTML page that is obtained from the Page Extractor. The Page Modifier inserts the recommendation calculated in the rating module. Furthermore, the Page Modifier inserts the HTML that contains the feedback mechanism as described in section 5.2. The HTML that is introduced into the webpage must have several properties.

The feedback mechanism must be placed somewhere on the webpage, where it does not interfere with the article. Consequently, it must be placed somewhere

where it can be seen at once, and so that the active user will not have to scroll through to the bottom of the page to see the recommendation. A Wikipedia article is a quite dense webpage, where there is a lot of information. The structure of the articles change a lot, and the users setup changes a lot as well (resolution, browser, etc). Therefore it would be preferable if the recommendation could be moved around on the webpage or minimized.

5.4 The Rating Module

The rating is what binds the users together and enables collaboration between users. The rating consists of five parts:

- The recommendation. The mark that the user has given the article. A numeric value between 1 and 9.
- The user. The name of the user who put the rating on the article. This is the registered Wikipedia user name. It is a necessity to have a registered user account, in order to keep a certificate.
- The version. This describes which version of the article, that the rating was added.
- Name of the article. Because the ratings are stored in the WML, they can be copied and inserted in to a any article. The name of the article is inserted in to the rating, in order to prevent that ratings are copied to other articles.
- A digital signature. The signature protects the recommendation from fraud. The title of the page, the mark, the username and the version are concatenated and signed with the users private key. This security measure prevents the ratings are tampered with, moved to other pages or moved to or to a later version of an article.

The Rating Extractor is given the URL of the edit page. The Rating Extractor then initiates a new URL connection to the URL of the edit page. The HTML code of the edit page is then downloaded, and the source code is now scanned for the marking that indicates a recommendation. The recommendations are now parsed in order to validate them.

The recommendations are filtered in a specific order, due to some of the filtering take more computation power that others. This order will make the filtering fastest.

Firstly, the recommendations are filtered by name and recommendation. In the rating the name of the article is inserted. The ratings are filtered by name to avoid that ratings from other articles are inserted in to the article. Hence, the rating is discarded if the recommendation is not within the limit. If an recommendation is not an integer between 1 and 9 (both included) they are removed. This is a computational easy task, as is is only a string match that is required.

Secondly the ratings are filtered on duality. This prevents a user from inserting more that one rating into an article, by removing the oldest recommendation from the page.

Thirdly, the signatures are validated with the signer's public key. The public key is stored in X.509 certificate that the signer must keep on his user page.

Finally, the ratings that are below a certain threshold are removed. The threshold are described in section 3.4.4. This filtering are done last because it requires to open an URL connection to obtain the previous versions of the acticle. Opening an URL connection takes time and therefore we would like to do this as few times as possible.

After the ratings have been filtered the ratings that are left, are inserted into the Session Rating Database. The raw ratings are parsed, and stored as an object in the database. This is a small temporary database that is only kept during the HTTP session. The Session Rating DB is stripped of the security measures, because the ratings have already been verified.

The Session Rating Database is now passed on to the Rating Calculator which calculates the final rating to the user. The Rating Calculator queries the Ring of Reviewers to obtain the trust values from the users of the ratings that are in the Session Rating Database. To obtain a final recommendation, the ratings that are in the Session Rating Database needs to be aggregated robustly. There are several ways to attack the system that could affect the result. See section 3.4.3. These attacks are prevented by filtering the ratings and the security measures in place. However there might still be some false or pollutes ratings inserted into the final Session Rating Database. Therefore the ratings need to be aggregated securely. David Wagner has in [35] proposed a way to aggregate measurements from a sensor network, based on robust statistics. We take the same approach in aggregating our recommendations. Firstly the Rating Calculator adjusts the recommendations according to trust values in the Ring of Reviewers and then aggregates these signatures a 5% trimmed average. An example of the calculated average, 5 % mean trim and ratings adjusted for trust values can be found in appendix A.2 on page 106.

5.5 The Trust Module

The trust module deals with the management of trust in the WRS. The trust module has two components: The Trust Updater and the Ring of Reviewers. The detailed description of the trust model is documented in chapter 4. This section describes how this trust model is fitted into the WRS.

As described in section 5.1, the Ring of Reviewers (RoR) is basically a database of trustees that the active user previously has had some interaction with. These trustees are defined as Reviewers. Each reviewer is basically defined by a username and a trust value. This trust value can change over time, according to the Trust Model. As the RoR evolve over time it is important that the values obtained through the feedback can be stored. In this case it would be ideal to have a database to store the different values as the RoR could become large over time. However, in order to ease the implementation the RoR will be stored as file on the disk on the proxy. Upon initialization of the proxy the RoR is read from the disk and old values are restored.

The Trust Updater takes the feedback from the active user and uses this feedback along with the existing ratings to calculate the updated trust values and to calculate if there are any new potential users that could be inserted in the RoR. If new trustees can be found a trust value is calculated, so the system is ready next time the WRS encounters a rating from that user.

5.5.1 Dealing with the Feedback

Dealing with the feedback from the user is performed in the Trust Updater as well. The user has some feedback options in the browser. This feedback can adjust the N and the X values in the trust model. This is explained in section 4.3.4.

The feedback mechanism distinguishes between interactions and experiences. Experiences adjust the N value and interactions adjust the X value.

An interaction is based on the recommendation that the active user gives the article. We define two intervals: One interval is less that ± 1 of the recommendation that the trustor has given the article and one interval is more than ± 3 of the recommendation that trustor has given. All other users that have rated similar (within the ± 1 interval) will be seen as a positive interaction and will influence the X value (trust dynamics) positively. All recommendations that are far away (more than $\pm$ 3) from the given recommendation will be seen as a negative interaction and decreases the X value. Recommendations that are not within the defined interval will not be dealt with. See an example in section 6.4.6.3.

Similar is the adjustment of the N value, that is based on the user is content about the result or not. If the active user gives positive feedback on the recommendations that have had a high influence on the mark, the N value is adjusted towards a more optimistic curve and if it is a negative experience then the N value is adjusted towards a more cautious curve.

5.6 Security design

As described in section 3.4.3 some implementation of security is required in the WRS. In order to prevent the described attacks the ratings have to be secured. We cannot rely on the Wikipedia to safeguard our ratings. Therefore users have to secure their ratings themselves.

5.6.1 Initialization of Security

In order to set up the WRS with the necessary security features, each user has to have a public and a private key. As described in section 2.3.2 the WRS does not have a central PKI. Each user creates a public and private key and creates a self signed X.509 certificate.

In order to use the WRS the generated keys will have to be distributed properly. The private key is used to sign ratings that are put into the articles. It has to be inserted into the proxy and the public key will have to be distributed to the Wikipedia, so other users can verify the rating that the active user had given

The public key will have to be available to everyone that need to verify ratings, so storing the public key on the Wikipedia seems a proper solution.

5.6.2 Domain

WRS are designed to work with one single wiki at a time. In this project our focus is on English Wikipedia, mainly because it is the largest and has the most users.

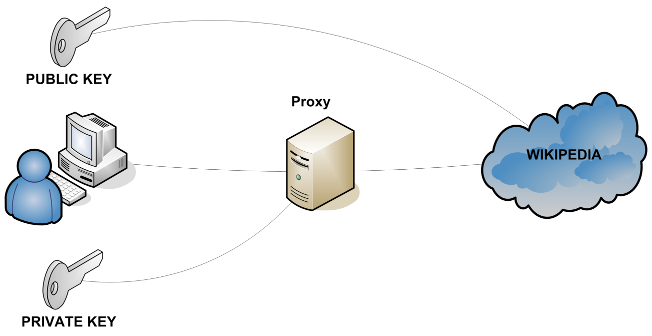


Figure 5.5: Distribution of the public and private key

There is, however, also a quite important security aspect to this design choice. When using the WRS, there has to be an associated user name. There has to be issued as an identifier to the public key. This is used to identify who put the ratings on a page, in order to get different trust value. And in order to identify persons we have to validate identity through their certificates. This certificate have to be stored somewhere where it is public available. In the WRS it is obvious to store this certificate on the user page, and then user the Wikipedia username as identifier that will be associated to the public key.

It would be preferred if we were to be able to use the same certificate and identifier on a different wiki (for instance the Danish wikipedia, or Simple Wikipedia). This would enlarge to possible number of trustees, and there by we could get more trustees in our RoR and by that, users would get better and more reliable results. But this opens up to a set of security related problems.

Identity theft. One username does not give access to all wikipedia databases. A login to the English Wikipedia cannot be used on the Danish Wikipedia and vice versa. Two people can register the same username on two different databases. Registering some one else's username, access to that database is prevented and the user who got his/hers name registered is prevented from obtaining information and submit recommendations on that wiki. This way the system would be prone to a denial of service attack.

Ratings Moving. The name of the article is included in the rating. This introduces a challenge when the same words have different meanings on different languages. For example, the English word "coin" in French means corner. Naturally the two articles would be totally different. If a users identity covered the all possible wikis then it would be possible to move the rating from one article to another, and there would be no way of knowing which one is real.

A solution to this problem could be to use the public key as identifier. This way a public key would be trusted instead of a username and each key could be used on several wikis. This however introduces another set op problems. With this approach all ratings that are put on the Wikipedia would be much larger and take up much more space. A 1024 bit key takes up 128 bytes compared to a 12-14 byte username. Furthermore a user would not be able to revoke his certificate and issue a new because this would lead to a new identity and all trust previously build in that user would be lost.

5.7 Summary

The design chapter describes the internal design of the WRS. We describe how a request from a user, is processed, and the different operations that are performed in order to provide the active user with a personalized recommendation. We describe how the feedback, which the user gives is processed and how it influences the trust management system in the WRS.

This chapter also describes general security measures that have to be taken in order to protect the WRS from being prone to attacks.

CHAPTER 6

Implementation

This chapter concern the implementation of the WRS. This chapter shows that implementation of the proposed system is feasible, within the described requirements and design. However, the implementation should not be seen as a final product that is ready to be deployed.

6.1 Technologies Used

In the development of the WRS several technologies have been used. This section lists the technologies and the reason for choosing that specific technology.

Java SE Java is chosen as development language mainly because the WRS is an internet application and Java offers a good and solid framework for development of Internet based applications. Furthermore, Java is platform independent which suits the requirements of a proxy well. The WRS will fit on any operating system, as long as the operating system can run a Java Virtual Machine. With this platformindependence the WRS will be able to fit into any network environment required. With the choice of Java, several other development tools become clear.

Junit. As a unit testing framework JUnit is a excellent choice to perform white box testing. Read more in chapter 7

Java Remote Invocation Method. For performing communications over a network Java RMI offers a simple and direct way of communication and invoking methods over a network.

WIKI bot. There are several frameworks for development of bots to the Wikipedia. One of the larges and most used is Java Wiki Bot Framework [3]. However this framework is quite large and although it can perform a large set of different tasks. However, our need is restricted to perform page edits. Therefore we user WIKIbot by MER-C [16], because it is much smaller and more light.

Scone Proxy. Scone is programmable proxy that is written in Java, which is developed to create new web technologies that will enhance the browsing experience.

ADC parser. Arthur Do from Standford University has written an library for parsing HTML documents [11]. This library is used by Scone.

WBI Framework. IMB has developed a framework for developing web applications in Java. Scone uses this framework.

6.2 Scone

As outlined in section 2.4 there are three proxies that have been evaluated in order to find a proper proxy that could be used for developing the WRS, but as outlined on section 2.4 the choice fell upon the Scone proxy.

The Scone API allows plugins to be written. Plugins ease the insertion of code and new functionalities to the proxy, and the removal of such again. The WRS are written as a plugin to Scone. Figure 6.1[1] shows a general overview of the plugin setup for Scone. Implementing as a plugin gives access to all features and components of Scone. These features are: The TokenHandler, Generator, RAS-Handler and Robot Task, where the WRS are written, with the use of TokenHandler.

Plugins can easily manipulate the HTML streams flowing through the proxy, gather and process data from the Internet and serve this information as static or dynamic documents to the browser.

[1]The figure is taken from `http://scone.de/architecture.html`

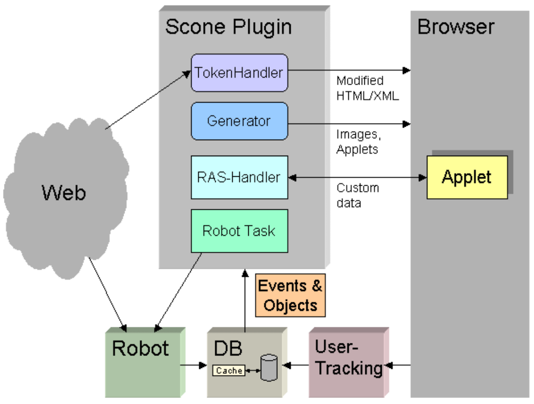


Figure 6.1: Overview of the plugin setup.

6.2.1 Scone Proxy API

The architecture of scone is described in section 2.4.3. This section gives a brief overview of the four modules in Scone: Proxy, Robot, NetObjects, and UserTracking. In the implementation of the WRS it is only the proxy module that is used. This section describes the technical details of the proxy API. An overview of the of the different components in the Scone API can be seen on figure 6.2 [2]. The Scone Proxy API consists of a set of interfaces, that can be inherited and in the development of a plugin to the Proxy. Plugins that are inserted to the proxy are controlled by the `SconePluginInterface`. Scone extends the `HttpProxy` interface that is provided by WBI, and enables all the networking operations. Consequently, Scone contains an `AccessTrackingMeg`, `GeneralResourceGenerator` and a `TokenHandlerMeg`. A Meg is a component within the WBI framework that can monitor, edit, or generate requests.

The `AccessTrackingMeg` is used to obtain the actions performed by the user in the browser. The `GeneralResourceGenerator` is used to transfer the documents from and to the user. The `TokenHandlerMeg` is used to manipulate HTML documents and other data send over a HTTP connection.

The `TokenHandlerMeg` is used to do the actual manipulation of the HTML doc-

[2]The figure have been taken from http://scone.de/doc/scone/scone/proxy/package-summary.html

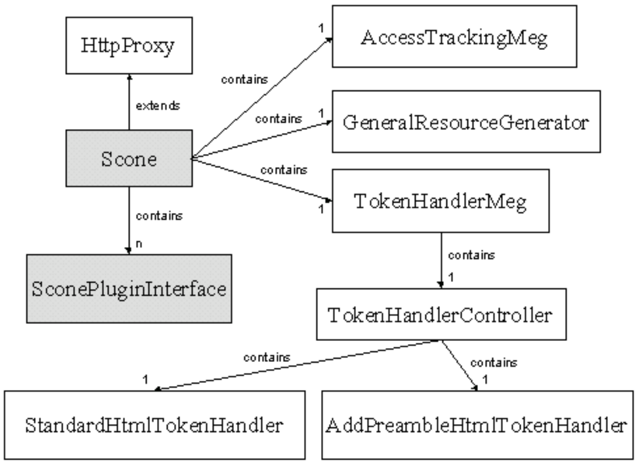


Figure 6.2: Overview of the Scone Proxy API.

uments. The manipulation is controlled by the `TokenHandlerController`, and the HTML header is manipulated by the `AddPreambleHtmlTokenHandler` and the body of a HTML document is manipulated in `StandardHtmlTokenHandler`.

6.3 Implementation of WRS

The WRS is as mentioned above implemented as a plugin that uses the Scone API to handle the HTML documents. Setting up a plugin for Scone is done like this:

```
public class WRSPlugin extends Plugin{
   public void init(){
      WRS wirtu = new WRS(this);
       wirtu.setup("WikipediaRecommenderSystem",HTDOCCONDITION,60);
       addMeg(wirtu);
  }
```

An instance of the plugin is created in the `init()` function. The `WRS` object contains all the code that the proxy needs to execute. The `WRS` object extends the `HtmlTokenEditor`:

```
public class WRS extends HtmlTokenEditor {
    WRSPlugin plugin = null;
```

The `HtmlTokenEditor` enables the manipulation of HTML documents. The method `handleRequest()` is extended from the `HtmlTokenEditor`. This metod is invoked by the `GeneralResourceGenerator`. Everytime a request comes in to the proxy the handleRequest method is invoked and executed in a thread. Two streams are initialized in order to extracting the HTML document which is downloaded to the proxy. One stream is for reading the content and one is for writing the content. The streams are obtained over a `SconePipe`:

```
public void handleRequest(SconePipe pipe) {
    TokenInputStream in = pipe.getTokenInputStream();
    TokenOutputStream out = pipe.getTokenOutputStream();
```

And HTML tokens can now be read from the in stream like this:

```
Token t = in.read();
```

And written to the browser like this:

```
out.write(token);
```

6.4 Implementation Overview

Implementing the design of the proxy reflects the design described in chapter 5. Each module described in the internal architecture is implemented as a Java package with some modification. Consider the package diagram shown on figure 6.3.

- The HTTP module is implemented by the `sconeplugin` package, which is also the main thread. The Feedback Processor from the design has been implemented in the remote package.
- The page module is implemented by the `page` package.
- The rating module is implemented by `rating` package.
- The trust module is implemented by the `trust` package.
- The package `statictools` contains a set of tools that is used throughout the implementation. Such as Security tools, network tools (which is not provided by Scone), bot tools etc.
- the adc.parser contains tools to parse a HTML page.

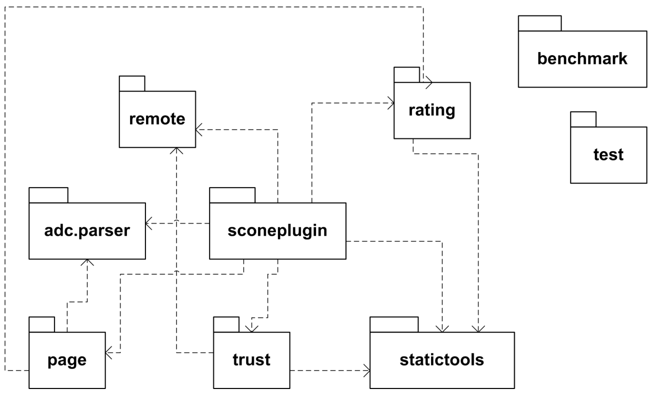


Figure 6.3: Overview of the packages that are used in the WRS.

- The benchmark package contains the benchmarking tests described in chapter 7.
- The test package contains the white box tests described in chapter 7.

The actual code is not described in detail here. The code is very well commented and should be self-explaining. See appendix C for the code. In appendix F a complete package diagram and class diagram can be seen, as well as a sequence diagram for flow in the proxy.

6.4.1 Sconeplugin package

The sconeplugin contains the main thread and from here the `handleRequest()` is invoked. The WRS initializes all the objects from the other packages that are needed to calculate the recommendation. The WRS inserts a Java applet for the feedback mechanism. A simplified class diagram is shown on figure 6.4

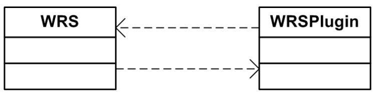


Figure 6.4: Overview of the Sconeplugin classes

6.4.2 Page package

The page package contains three Java classes, which do not have any interaction with each other but all three of them perform some operation on an HTML document.

6.4.2.1 PageExtractor Class

The PageExtractor class extracts the URLs for the edit and the history page. Furthermore, it extracts the page title and the current version number. Both values are needed to put a rating on a page.

The information is found by scanning for markers in the Wikipedia HTML. These markers are used to identify information fields.

<LI `id="ca-edit"`> marks the URL of the edit page.

<LI `id="ca-history"`> marks the URL of the history page

<LI `id="t-permalink"`> marks the URL of a permanent link, where the version number can be found. This marker can also be used to obtain the title.

An example that scans for the edit page marker is shown here:

```
//scanning through the tokens
for (int i = 0; i < HTMLdocument.size(); i++) {
Token t=(Token)HTMLdocument.elementAt(i);
//If the token is a HTML Tag and not text
    if (t instanceof HtmlTagToken) {
        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("ca-edit")) {
                    //Perform task ...
```

6.4.2.2 ExtractRatings Class

The ExtractRatings class is used to extract the ratings from the Edit URL that was obtained from the PageExtractor. Due to the fact that the user does not download the edit page, where the ratings are stored, a URL connection has to be opened and to the Edit url and the ratings have to be extracted. A rating looks like this:

```
<!-- WikiTrustComment. Read more on:
http://en.wikipedia.org/wiki/User:Korsgaard
;Korsgaard;7;94723853;Bass_Strait;MCwCFBRZ3bjvzQk5qygSaOd8k1FNnzTeAhROuRCDktVV033/4xRLA==
-->
```

ExtractRatings scans for a comment with the marker ”WikiTrustComment”, and then extracts the required information (username, recommendation, version, article name, and signature)

According to the design chapter extracting the ratings and verifying them are two different operations. In the implementation these operations have been put together, so that ratings are parsed and verified at the same time. This speeds up the process due to the fact that ratings do not have to be stored to the disk in between operation.

6.4.3 PageModifier Class

The PageModifier class is used to modify the content of the page shown to the active user. The PageModifier needs to insert two things: The recommendation that is calculated based on the trust values, and the feedback mechanism. The HTML that needs to be inserted to the modified webpage is located in two static text files that are located in the `static_textfiles/` directory in the root of the proxy. See appendix C.13. These files contain markers that are replaced at runtime, with the relevant information. These markers identify:

The recommendation. The marker `###RATING###` is replaced by the calculated recommendation.

The IP address. The marker `###IP###` is replaced by the IP address that hosts the applet that is used for obtaining feedback from the user.

The modified applet is inserted into the webpage with the use of Yahoo! UI Library: Drag & Drop Library [3]. In this way the recommendation and the feedback mechanism can be moved around on the Wikipedia article, by dragging it with the mouse, in case of the recommendation should cover some text in the article.

[3] http://developer.yahoo.com/yui/dragdrop/

6.4.4 Rating Package

The rating package administrates ratings and storing of ratings in the WRS. A simplified overview of the classes can be seen on figure 6.5.

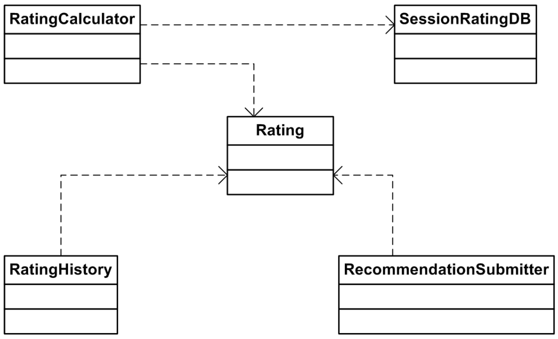


Figure 6.5: Overview of the Rating classes

6.4.4.1 Rating Class

The rating class is the central class in the Rating package. The rating is basically a data type that contains all the information relevant for a rating:

The version number of the Wikipedia article that the rating is put on.

The recommendation is the actual mark that the article is given.

The username of the Wikipedia user who put the rating on the page. This username is also the key identifier to the certificate used for validating the rating.

Article URL contains the URL from where the article is obtained.

Experience: The feedback about the experience relevant to this article. The experience indicates if the user felt the recommendation provided a relevant contribution or not. The experience value is used to alter the trust evolution function (N value).

Interaction: Interaction is used to describe if the interaction with this recommendation is positive or negative. A positive interaction occurs if the active user has given a recommendation that is similar to other recommendations that are related to the article. The interaction are used to determine the X value in the trust dynamics.

The date is the time that a rating was first discovered and inserted into the database. The date is used to calculate how old a rating is and also to calculate trust dynamics (the X value).

6.4.4.2 RatingHistory Class

The RatingHistory class is used to store the ratings in an array. Every time a rating is put into the RatingHistory an X value is calculated based on the design of the trust dynamics, which is described in section 4.4.2. This way X values can easily be extracted for use in the Trust evolution function.

6.4.4.3 SessionRatingDB class

The SessionRatingDB class is used to keep a temporary set of ratings, that is extracted from the edit page. Once they are verified they are put into the temporary database.

6.4.4.4 RatingCalculator class

The RatingCalculator class calculates a recommendation based on the description in section 5.4. The RatingCalculator uses the temporary database (SessionRatingDB) to calculate a 5% trimmed average.

In order to give recommendation the weight of their trust value, we generate an empty set of recommendations and for each reviewer that should contribute to the final recommendation we put the recommendation into the set, one hundred multiplied with trust value of recommendations. If for instance, a trustee has a trust value of 0.34 and gave the recommendation 2 to the article, then 34 2-recommendations are put into the set of recommendations. The set is now sorted and the lower and upper 5% of the ratings are trimmed off and an average is calculated from the rest.

A numerical example on how a recommendation is calculated is presented in appendix A on page 105.

6.4.4.5 RecommendationSubmitter class

This class is used to submit a recommendation to the Wikipedia. The RecommendationSubmitter generates a rating with the provided tools in the SecurityProvider, and then submits the recommendations to the Wikipedia by using a bot provided by MER-C [16].

6.4.5 Remote Package

The remote package is used to obtain the feedback from the user. The feedback mechanism is implemented with Java RMI technology. A class diagram is seen on figure 6.6.

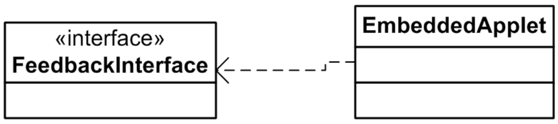


Figure 6.6: Overview of the Remote classes

6.4.5.1 FeedbackInterface class

The FeedbackInterface is the interface that the WRS class (from the sconeplugin package) implements. The interface consists of a methods that is invoked when the active user performs feedback through the browser.

6.4.5.2 EmbeddedApplet Class

The EmbeddedApplet class contains the applet that is inserted in to the proxy. The applet is build with the `java.applet` [4] package and the `java.awt` [5] package.

Whenever an action is invoked over the ActionListener, a RMI connection is set up and the code related to the feedback mechanism is invoked in the WRS object. With this architecture code execution is left within the proxy and the external applet only invokes the code.

[4] http://java.sun.com/applets/
[5] http://java.sun.com/j2se/1.5.0/docs/api/java/awt/package-summary.html

6.4.6 Trust Package

The trust package is an implementation of the Trust Model that is described in chapter 4. The trust package holds the RoR and the Trust Updater. An overview is shown in figure 6.7

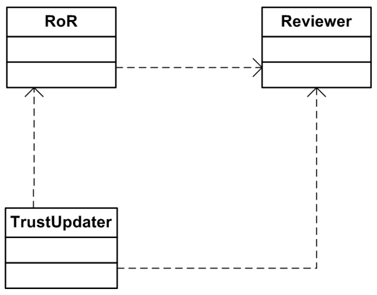


Figure 6.7: Overview of the Trust classes

6.4.6.1 Reviewer class

The reviewer class is a datatype that holds all information on each trustee in the Ring of Reviewers. Furthermore, the reviewer class holds the implementation of the trust model. Every time a new rating is inserted in to a reviewer the trust evolution function is updated, and a new trust value in calculated.

The reviewer class maintains the variables in the trust model about the trustee. It keeps and updates information about the evolution curve (cautious or optimistic), it keeps track of whether the trustee is in trust or distrust, and continuously updates the trust value based on the input.

We need to solve the four equations presented in section 4.4.1, in order to isolate the y value, which is our trust values. We use MAPLE [6] to solve the equations, and get the following equations, which are straight forward to implement with the `java.lang.Math` package. Equation 6.1 solves 4.1, equation 6.2 solves 4.2, equation 6.3 solves 4.3, and equation 6.4 solves 4.4. We get the following equations:

$$y = \left(-\left|(x-1)^n\right| + 1\right)^{\frac{1}{n}} \tag{6.1}$$

[6] http://www.maplesoft.com/

$$y = -\left(-\left|x^n\right| + 1\right)^{\frac{1}{n}} + 1 \tag{6.2}$$

$$y = \left(-\left|x^n\right| + 1\right)^{\frac{1}{n}} + 1 \tag{6.3}$$

$$y = \left(-\left|(x+1)^n\right| + 1\right)^{\frac{1}{n}} \tag{6.4}$$

The reviewer class only holds the methods for dealing with positive and negative experiences. The invocation of these functions is done from the Trust Updater class, which contains the logic for determining positive and negative experiences.

6.4.6.2 RoR Class

The RoR class contains the Ring of Revievers. The class is serializable in order to be able to store it and retrieve it from the disk. The RoR is implemented as a Java HashMap where the username is the key and the Reviewer class is the value.

The RoR caches the users X.509 certificates. The certificates are cached for performance reasons. If the WRS should download a certificate every time it needed, then it would take a long time to calculate a recommendation, because it takes a substantial amount of time to download a certificate and parse a certificate. If an article contains 100 ratings, then 100 certificates would have to be downloaded. Therefore the RoR needs to be able to cache certificates and use the cached certificates when verifying a signature. This introduces a problem, as users should be able to revoke their certificate. Therefore the RoR are programmed to update certificates every time the proxy is initialized. As a result of the wiki philosophy, then every one can alter a page. This opens the possibility that an attacker can perform a DoS attack by modifying certificates on the user pages. Therefore when the WRS downloads a certificate, it scans the history of the certificate page and obtains the certificate from the version that the owner last edited. This is possible because the history page states which users made which edits. This gives a longer start uptime, but a much better running time. See section 7.3.2 on benchmarking.

6.4.6.3 TrustUpdater Class

The TrustUpdater class contains the logic and methods for dealing with the feedback from the user. The TrustUpdater can adjust the N value in the trust model based on the experience with this recommendation, and the X value can be adjusted based on the interactions.

If a user states that he is content with the result then the users that have put ratings on the article, within a ±1 interval of the provided recommendation, we define this as a positive experience. And their N value is adjusted towards a more optimistic curve.

If a user states that he is not content with the result then user that have put ratings on the article within a ±1 interval of the provided recommendation we define this as a negative experience. Their N value is adjusted towards a more cautious curve.

Consider a page with three ratings from three different users. User A has rated the page 4, User B rated 5 and user C rated 8. The calculated recommendation to the user is 4.8 based on the trust values in the RoR. The user states that he is content with the result. Now user A and B have their N value adjusted towards a optimistic curve because their recommendation of 4 and 5 are with 3.8 and 5.8 (±1). User C is unadjusted as he has not influenced the result enough to tell if it is significant.

The same actions are taken with the interactions. Ratings that are on the article within ±1 of what the active user rates, are considered as a positive interaction. The recommendations that are ±3 or more away from what the active user have rated are considered as a negative interaction. The ratings in between are not classified.

Consider a page with three ratings from three different users. User A has rated the page 4, User B rated 5 and user C rated 8. The active user rates the article 3. User A is regarded as a positive interaction and trust dynamics (and there by X value) are increased. User B is not classified and Users C is classified as a negative interaction because it is more that 3 score points away.

6.4.7 Statictools Package

The statictools contains a set of tools that are used through out the implementation. The largest is the SecurityProvider class. A class diagram is shown on figure 6.8.

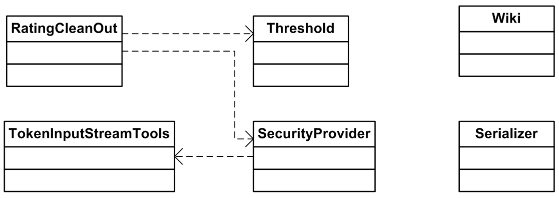


Figure 6.8: Overview of the Statictools classes

Some of these classes are small and very straight forward and will only be described shortly here:

TokenInputStreamTools is used to open up a URL connection to a webpage and to create a token stream from the HTML page. This is used when information is needed from a page that a user does not navigate through. Such as obtaining ratings, downloading certificated etc.

Wiki contains the bot that is used to submit ratings to the Wikipedia. This class is provided by MER-C [16].

Serializer is used to store and retrieve the RoR to and from the disk.

6.4.7.1 Threshold Class

The Threshold class calculates the difference between two versions of an article. The Threshold uses the function in the Wikipedia that can provide the active user with the difference between to versions of an article. Calling an article with the parameter `&diff=current&oldid=1234`, where 1234 is the previous version, returns an overview of the difference between that version and the current version. The changes are marked with red words. The threshold is based on the number of present the page has changed. The design is described in section 3.4.4.

6.4.7.2 RatingsCleanout Class

The RatingCleanOut contains three methods for cleaning out ratings that should not influence the result: RemoveRatingsTitleMismatch() removes ratings where the title does not match, RemoveRatingsBelowThreshold() removes ratings that are below a threshold (described above), and RemoveUnvalidableRatings() that removes ratings where the signature cannot be verified.

6.4.7.3 SecurityProvider Class

The SecurityProvider is the largest of the classes in statictools and covers all security related operations. The security operations is performed by using Javas build in classes and methods from the package `java.security`. The methods are straight foreward and well commented. The SecurityProvider class can be found in appendix C.6.2 on page 139.

6.5 WRS Setup

There is quite a bit of configuration needed in order to run the WRS. Please see appendix B for installation and configuration manual for Scone and registration of the WRS plugin.

6.5.1 Implementing Key Infrastructure

In order to have a key intrastructure, the users must generate their own keys and keep them properly and deploy their certificate properly

6.5.1.1 Generate Keys, keystore and certificates

Key generation is done with Suns `Keytool` [6]. Executing the following command generates a selfsigned X.509 level 1 certificate.

```
$ keytool -genkey -alias wiki_username -keystore .keys
```

Where wiki_username is the username of the wikipedia account that should put ratings on the articles. The keystore file .keys are stored in the root of the proxy. Generation of a public X.509 certificate issue the following command:

```
$ keytool -storepass my-keystore-password -alias myalias -export
          -rfc -file outfilename.cer
```

This generates a public certificate encoded with BASE64, and the certificates are deployed by copying it to the userpage as described in section 3.4.3.

6.5.1.2 Password Management

In the root of the webserver a file `password.txt` should be stored. The structure of the file should be like the example shown in appendix C.13.3. The marker `KeyStorePass` marks the password used to unlock the keystore and the marker `WikiUserPass` marks the username and password that ratings are submitted with. This is naturally not a proper password management system, and upon deployment this has to be stored in some keystore that could be unlocked with a password. However this is not the objective in this project.

6.5.2 Applet Deployment

The applet that invokes the RMI method needs to be deployed to a webserver. In this implementation we have used NanoHTTPD[7], but any webserver will do. The webserver has to be accessible by ipadress and RMI registry has to be started on port 1099. The remote package will has to be present in the root directory and readable. Read detailed instructions in appendix B.

6.6 Summary

The implementation chapter describes how the WRS has been implemented in Java SE, as a plugin to the Scone Proxy. The chapter described which technologies that makes the implementation possible, and we give a description of the Scone API, which is used for developing the WRS.

The chapter goes through how the WRS modules are developed with the Scone API and we describe how the design is implemented in packages. In each package there are a number of classes that are described. The complex classes are explained regarding how the problems were solved.

[7]http://elonen.iki.fi/code/nanohttpd/

Finally we describe how the general setup of the WRS are performed, including key generation, certificate generation, certificate distribution and password management.

Chapter 7

Evaluation

This chapter describes the evaluation of the WRS. The chapter analyzes whether the requirements (described in section 3.2.3) have been met, it describes the white and black box testing that has been performed on the software, and we describe how the proxy have been benchmarked and give an evaluation on the performance of the proxy.

The chapter discusses the general requirements to a recommender system, the lack of long term evaluation of the WRS and finally we propose a method to perform a long term usability test.

7.1 White box testing

White box testing is performed by looking internally on the source code and the system design to create test cases based on internal structure. In this project we have build a large test suite that tests each class. The tests are build upon the JUnit 4.3.1 framework[1], and a test package has been written to each package that is described in section 6.4 (`test.trust`, `test.page`, `test.sconeplugin`, `test.rating` and `test.statictools`). The `test` packages, contain a test suite

[1] http://www.junit.org/index.htm

Class	method	What is tested	Success
TestExtractRating	testExtractRatings()	Tests if ratings can be extracted from an edit page.	YES
PageExtractorTest	testPageExtractor()	Tests if the edit url and the history url can be extracted from an article.	YES
PageModifierTest	testInsertYUIandRating()	Test if text can be inserted in to a webpage.	YES

Table 7.1: White box test for the `test.page` package

for running all the 37 tests, in the above described packages, and the tests show that all the test succeed. The names of the tests and their testing area are summarized in the following tables: Table 7.1, 7.2. 7.2, and 7.4.

7.2 Black Box Testing

In order to justify that the WRS runs as expected, a set of black box tests are created, which will test the usability of the WRS. These tests have the purpose to show that the WRS performs normally and does not show any errors through ordinary use. This section can not prove that there are no errors in the software, because this would require that the program should be tested in all branches of the code.

The tests have been conducted where the browser and the proxy is running on the same machine. The following tests have been conducted.

Get a recommendation. The user can obtain a recommendation based on the trust values. In appendix D.1 a screenshot of a provided recommendation is seen and the four recommendations it is based on.

Put a recommendation. The user can submit a recommendation by clicking one of the buttons on the feedback mechanism. The new rating is created and inserted into the edit page of the article. See appendix D.2 for screenshots.

Give feedback. The user can provide feedback on the article and trust values are adjusted based on the feedback. Screenshots are provided in appendix D.2.

Moving feedback mechanism. The feedback mechanism can be dragged around in the window if the recommendation and feedback mechanism blocks important text in the article.

Class	method	What is tested	Success
RatingTest	testRating()	Tests that a simple rating can be created.	YES
	testRating2()	Tests that a full rating can be initialized.	YES
	testSetExp()	Tests if the Experience value can be set.	YES
	testSetInteraction()	Tests if the interaction value can be set.	YES
RatingCalculatorTest	testComputeAverage	Tests if a rating can be properly calculated.	YES
RatingHistoryTest	testRatingHistory()	Tests that a RatingHistory object can be initialized.	YES
	testInsertRating()	Tests that a rating can be inserted int a RatingHistory.	YES
	testGetXValue()	Tests that a X value can be obtained based upon the ratings put in to the RatingHistory.	YES
TestSessionRatingDB	testSessionRatingDB()	Tests if SessionRatingDB can be created.	YES.
	testPush()	Tests that a rating can be inserted in to the a SessionRatingDB.	YES
	testPop()	Tests that a rating can be removed from a SessionRatingDB.	YES
	testSize()	Tests if a size can be calculated from a SessionRatingDB.	YES

Table 7.2: White box test for the `test.rating` package

Class	method	What is tested	Success
ReviewerTest	testReviewer()	Tests if a Reviewer object can be initialized.	YES
	testInsertRating()	Tests if ratings can be inserted to the Reviewer and if trust values calculated is correct.	YES
	testGetTrustValue()	Tests if a trust value can be obtained from a Reviewer.	YES
RoRTest	testRoR()	Tests is a RoR can be initialized and a reviewer inserted.	YES
	testReadRoRFromDisk()	Tests that the RoR can be written to disk and then retrieved again.	YES
TrustUpdaterTest	testTrustUpdater()	Tests that trust can be updated based on the logic in the TrustUpdater	YES

Table 7.3: White box test for the `test.trust` package

Class	method	What is tested	Success
RatingCleanOutTest	testRemoveRatings-TitleMismatch()	Tests that ratings are removed based on if the title does not match.	YES
	testRemoveRatings-BelowThreshold()	Tests that ratings are removed if the version of the article have changed too much.	YES
	testRemoveUnvalidable-Ratings()	Tests that ratings are removed if their signature cannot be verified.	YES
SecurityProviderTest	testInitKeyStore()	Tests that a KeyStore can be opened from the disk.	YES
	testInitCertificate()	Tests that a certificate can be created from the KeyStore.	YES
	testInitKeyPair()	Tests that a private and public key can be generated from the KeyStore.	YES
	testCreateSignature-AndVerify()	Tests that a signature can be created and verified.	YES
	testSignatureEncoding-AndDecoding()	Tests that signatures can be encoded to BASE64 encoding and decoded back.	YES
	testGenerate-Certificate()	Tests that X.509 certificates can be generated.	YES
	testCreateRating-AndVerification()	Tests that the content of a rating can be created and verified.	YES
	testGetUser-Certificate()	Tests that a User certificate can be downloaded and parsed.	YES
	testGenerate-Ratings()	Tests that ratings can be generated.	YES
	testGetWikiUser-PassFromDisk()	Tests that Wikipedia username and Password can be retrieved.	YES
	testGetKeyStore-PassFromDisk()	Tests that keystore password can be retrieved from disk.	YES
SerializerTest	testWriteAndRead-RoRToDisk()	Tests that a RoR can be written to the disk and retrieved again.	YES
ThresholdTest	testThreshold-Calculator()	Tests that a threshold can be calculated.	YES

Table 7.4: White box test for the `test.statictools` package

Tampering with ratings. There have been several attempts to tamper with the ratings on the articles (Modification, moving and falsification). All attempts failed.

All these tests show that a general use of the proxy is possible. Sceenshots from the use of the proxy can be seen in appendix D.

7.3 Benchmarking

We have implemented a prototype of the WRS, which allows us to determine the feasibility and performance of the proposed architecture.

7.3.1 Impact on the Wikipedia

The size of the ratings depends on the length of the username and the title of the article that is rated. The average size of usernames and article titles indicates that a rating has an average size between 110 and 140 characters. Assuming a worst case scenario, we presume that a rating is 140 characters. If we assume that one third of the article on the English Wikipedia, has 80 ratings each. This gives us:

$$\frac{1}{3} \cdot 1.8 \text{ million articles} \cdot 80 \text{ ratings pr. article} \cdot 140 \text{ bytes} = 6712 \text{ million bytes}$$

An estimated 6.7 GB will be spend on storing the ratings in the Wikipedia. From the MediaWiki Foundation download page [37] a compressed database dump can be downloaded. These dump takes up 84.6 GB, and users are warned that uncompressed it will take up 20 times more space. This gives an uncompressed size of the Wikipedia of 1692 GB. This suggests that the ratings will take op 0.3% of the total size of the Wikipedia, which we regard as acceptable.

7.3.2 Benchmarking the Proxy

In the benchmark test the proxy and the browser are located on the same machine and are connected to the Internet through a 100 Mbit/s Fast Ethernet connection. The proxy is running on a AMD Athlon 2000+ machine with 512 MB ram, which represents a modest configuration for modern PCs. Load times have been measured with Ethereal [2] and the load times can be found in table 7.5. Two kind of benchmark test have been made. One for measuring the load times of a webpage and one for measuring the time it takes to initialize the proxy. The proxy takes a long time to initialize because all cached certificates are updated to ensure that no certificates have expired or been revoked. The initialization of the proxy is normally only done when the proxy is rebooted, so it is not considered a problem that these load times are quite large. Load times for initialization can be seen in table 7.6.

Test no.	Proxy	No. of Ratings	Time for a page to load
1	No	0	2.74 seconds
2	Yes	0	5.23 seconds
3	Yes	3	6.2 seconds
4	Yes	12	6.6 seconds
5	Yes	100	8.5 seconds

Table 7.5: Loadtimes for the proxy based on number of ratings related to an article

Test no.	Proxy	No. of certificates	Proxy initialization
1	No	0	2.74 seconds
2	Yes	0	5.23 seconds
3	Yes	3	9.9 seconds
4	Yes	12	21.54 seconds
5	Yes	100	96.27 seconds

Table 7.6: Initialization times on the proxy based on the number of certificates that have to be downloaded

The load times for the webpages are quite good. As described by Mc Nee et al. [24] an entity must have around 80 recommendations in order to give a confident result. We assume that a general article in the Wikipedia has between 80 and 100 recommendations, in order to give a proper result. This gives us an extra load time of 1.5 second, which is quite acceptable. In November 2006 Akamai and JupiterResearch published an extensive report about their research within acceptance of loadtimes of webpages [9]. They conclude that there is a 4 second limit, where users loose interest in a page. Each article can have up to 300 recommendations before the loadtime surpasses the allowed time limit.

[2] http://www.ethereal.com/

7.4 Requirements

Jøsang et al [20] identify four criteria that are important for the creating a recommender system (mentioned in section 2.2.4 and analyzed in section 3.2.1). The creteria are:

- Accuracy for long-term performance.
- Weighting toward current behavior.
- Smoothness.
- Robustness against attacks.

As described above the WRS has not been tested for long time performance. But initial testing shows no indications that the system would not work over long time.

The WRS is implemented with a trust model and therefore recommendations from trustees that have high trust value weigh more, than newly encountered users. Furthermore, a trustee who has only had a few interactions with others, might have the same trust value as trustees with a lot of interactions. The trust value is low for the trustee that is new due to uncertainty and the trust value is low to the old trustee because of previous disagreements. It will take longer time to build up trust for the old trustee because the negetive interaction that lies in the past weigh down. Therefore the WRS has the capability to distinguish between a new entity of unknown quality and an entity with poor longterm performance. Because of the implemented forgettabillity the old trustee can come into trust again.

The WRS builds up information about the users that it interacts with. The more information is gathered the more smooth the WRS run, and the less a single recommendation will over time not influence the trust so significantly.

Tests in section 7.2 and 7.1 show that the system is resistant to attacks and will be able to resist the most basic attacks.

In sum, three out the four criteria is fulfilled by the WRS and with further testing the fourth criteria can be fulfilled. This gives an indication of the robustness of the WRS, and the possibility of becomming a permanent recommender system for the Wikipedia.

7.5 Long Term Usability Testing

Testing the software functionality and determining if the software meets the specified requirement, are normally not a proper way to determine if the software is useful. This relies on where the requirements originate from. If the requirements originate from a thorough analysis of the demands of potential users (which in this case could be Wikipedia users that need a helping hand in evaluating the content of the Wikipedia), there is a good chance that that the developed software is useful. But if the the requirements originates from technical people who intend to create a proof-of-concept, which is the case in this project, there is a chance that the software will not fulfil its' purpose. Based on the cases described in section 1.1, there is a lot of cases where people have had a loss due to the false information on the Wikipedia, and that tells us that a system like the WRS is needed. Determing if the proposed system is exactly what is needed to improve the individual's trust in the Wikipedia is difficult. At this point in time there is no hard evidence that the system will provide the trust, the Wikipedia users are looking for.

It is clear that it cannot be determined if the WRS will satisfy its' users by doing a theoretical analysis. We need to perform a large scale test, where feedback is collected from the users of the WRS. We propose a method to perform such a large scale test, but this method is not implemented due to time constraints.

The idea is simply too keep a log of all operations that are performed by the WRS and a log of the user feedback and then gather these logs from all the users to perform a statistical analysis.

The log would simply be a text file with comma separated values (`.csv`), that would be easy to insert into a database or some statistical software. For each time the WRS calculates a recommendation to the user, the log is updated with every trust update information performed. This means there is a line in the log for each rating there is on an article. The log should contain 18 elements, which are:

1. Wikipedia username of user who put the rating on the article.
2. Article name of the evaluated article.
3. Own recommendation given.
4. WRS calculated recommendation.
5. Content feedback (did the user like the result).

6. Curve state before feedback (cautious or optimistic).
7. Curve state after feedback (cautious or optimistic).
8. Trust state before feedback (trust or distrust).
9. Trust state after feedback (trust or distrust).
10. N value value before feedback.
11. N value value after feedback.
12. X value value before feedback.
13. X value value after feedback.
14. User trust value before feedback.
15. User trust value after feedback.
16. Classification of interaction with this rating (positive or negative).
17. Classification of experience with this rating (positive or negative).
18. Timestamp.

This introduces some privacy issues, as users of the WRS properly do not want to provide information on what pages they visit and which users they agree with. A solution to this would be to use a one-way hash function to hash the usernames and the article names. A one-way hash function will always hash a string to the same value, but it is not possible to determine what string the hash originates from. Introducing this measure, ensures that all the names are are still comparable, but it will not be posible to track the users history from the logs. This way usernames and article names can be hashed before send to a central location, and it would still be posible to compare usernames and article titles without interfering with the users privacy.

A lot of the logged values are before and after values (no. 6 to 15.), which allows us to analyze if the test group can actually make progress in trust, and how trust typically will progress. The **user name** is used to keep track of how trust progress in the individual user, and the **article name** can be used to see if there is a general tendency towards some articles provide better feedback that others. The **calculated recommendation**, the **given recommendation** and the **content feedback** are used to get an general idea about the users content about the system, and to be able to analyze if the intervals that have been set for defining positive and negative values are realistic or if they need adjustment.

The **timestamp** can be used to keep track of how much and how often the WRS are used.

We believe that through analysis of this feedback we could get a general idea if the system would provide usable recommendations, or if there need to be some adjustments in the system.

7.6 Summary

This chapter shows the white and black box tests performed on the WRS, and all tests show that the WRS works as intended.

We have performed some benchmarking tests on the proxy and these show that the performance of the proxy is acceptable, and will not make users loose interest in the pages due to long load and computation times.

We discuss the general requirements to a recommender system and evaluate if these requirements have been met by the WRS. We give a general discussion on the usability of the WRS and point out that a long term usability test is needed. We propose a method for evaluating, if the WRS provides the results that is acceptable by its' users.

Chapter 8

Future Work and Research

This chapter describes the areas that are in need of further research and future work that is within the scope of this project.

8.1 Areas in Need of Research

Throughout the thesis there have been areas where the research has been scaled down, which leaves some areas somewhat incomplete. In the implementation it has been made easy to change these areas, whenever the proper research has been made. These areas include:

8.1.1 Semantic differences in sentences

The articles change in the Wikipedia and as described in the requirements (and outlined in section 3.4.3.1) the WRS should be able to determine if a rating has become invalid. The approach that is implemented is not a very sophisticated solution, as it only counts how many words have changed and looks into if there has been inserted or removed any negating words. A thorough analysis to determine the difference between two versions of the same article is needed, and would properly need the help of a linguist.

8.1.2 15% threshold accepted difference between documents

In this document we have defined that if two versions of an article differ more that 15% the rating is not accepted. This threshold is chosen with no backgrounds or research on how much change in an article is accepted by humans, and therefor this area is in need of research, possibly by an experiment.

8.1.3 Forgettabillity in trust dynamics

The in trust dynamics, forgettabillity has been implemented with a naive approach, where the interactions are worth 50% after 3 months, 25% after 9 months and worth 0% after 12 months. There is no justification for choosing these limits and research is needed to determine the correct interval for lowering the values, and research is needed to determine if the degree that the interactions have been lowered, is correct.

8.2 Future Work

In connection with the development of the WRS, it has come to our attention, that there are several areas that would be interesting to examine. We give a short description of these areas and a potential solution.

8.2.1 Discussion Page as Repository

After finishing the implementation of the WRS, the community of the Wikipedia started complaining about the ratings that polluted the edit page. Many editors on the Wikipedia believe that the edit page should not be filled up by comments. A solution to this problem is quite simple. The discussion page is similar to the edit page, and holds a forum for discussion about the article, in more informal tone, where the ratings properly would be accepted. A direct problem with moving the ratings to the discussion page is that the wiki bot, which is used to submit ratings to the edit page is not currently programmed to store text on the discussion page, and is therefore in need of reprogramming.

8.2.2 Browser Plugin

An interesting task would be build the recommender system as a extension for a browser, especially if the recommender system were to face commercially deployment. Creating an extension for a browser, such as the Mozilla Firefox is an open possibility that however requires the code to be converted into C++ or JavaScript which are currently the only supported languages for writing a Firefox Extension.

8.2.3 Confidence Displays

Introducing confidence displays in the provided recommendations, where the user is warned if the provided recommendation is calculated from a few ratings, would give the users an idea about how precise this provided recommendation is. The more ratings an article has, the more precise the provided recommendation will be. Research show that users are more likely to forgive that a rating is wrong if they are warned beforehand by the confidence display [24].

8.2.4 Cross Wiki WRS

As mentioned in the introduction there are thousands of wiki installations cross the web, and a WRS that could operate across the different wiki installations would increase the possible number of reviewers and thereby better recommendations could be provided. As mentioned in section 5.6.2 there are several security issues that will have to be addressed to provide this functionality. Furthermore the WRS would have to be language independent, as the provided implementation only supports English wiki installations.

8.2.5 Wiki Robots for Maintenance

As described earlier the Wikipedia supports the feature of creating bots, that can perform automated alterations to the wiki. Creating a bot that could perform maintenance tasks, as for instance removal of unverifiable ratings, would ease the load times for the users and thereby give a better experience using the WRS.

8.2.6 WRS outside a Wiki Domain

An improvement of the WRS could be to implement the WRS outside of a wiki domain. For instance, the popular open source forum phpBB [29], is widely used on the Internet and supports the markup language `BBCode`, which provides the functionality of inserting comments. Porting the WRS to fit other systems gives the possibility to create a recommender system that is independent of domain and there by provide more trust in the general information provided by the Internet.

Chapter 9

Conclusion

In this thesis we address the problems of incomplete, incorrect or biased articles in the Wikipedia. Simple attribute based article verification schemes are not considered adequate, so we propose a mechanism based on feedback from the users of the Wikipedia, where users of the recommender system build a network of trusted users. Users store information about the recommendations of reviewers that they have seen before and use this information to calculate an individual and subjective trust value, which influences the weight that feedback from that recommender will have in future trust calculations. The recommender system is developed as a proxy that is located between the user and the Wikipedia, and therefore no modifications to the Wikipedia are needed. The recommender system is supposed to give the user a recommendation about the if user should trust the content or whether he should seek information elsewhere.

In this thesis we present a general analysis of the challenges that accompany this approach to a recommender system. We address the problems with developing the recommender system as middleware by introducing a proxy, and choose to use the Wikipedia as a central recommendation repository. We use a set of cryptographic measures to secure these recommendations from attacks, and we create the ratings as comments in the Wikipedia, which makes them transparent to Wikipedia users that do not want to use the recommender system. We also address the problem that comes with the continuous change in the Wikipedia articles, by introducing a method for determining how much an article is changed, in order to determine if older ratings still are valid.

We present a trust model that fulfils the general requirements, that recent research show is required to build a trust model. We introduce a model that can build trust in users without configuration, but is solely based on the feedback that the users give the trust model. We present a formalized model, based on the super ellipse curve, which can be used to represent trust dynamics and trust evolution. The trust model is based on satisfactory feedback from the user and the recommendation that a user puts on an article.

We present a full design and an implementation of the recommender system based on the open source proxy Scone. A prototype of the recommendation system is build on top of the existing Wikipedia without requiring modifications to the existing Wikipedia system. The recommender system i developed in Java SE and is developed as a plugin to the Scone proxy. The implementation relies on a flat public key infrastructure, where the users is responsible for generating and maintaining their certificates and private keys themselves.

Evaluation of the proxy is done by a large scale white box testing and a smaller set of black box tests, where all the tests succeed. However, the recommender system lacks a long term usability testing to ensure that the recommender system delivers a desired solution the problems stated with the Wikipedia. We propose a method to perform such a long term test.

Experiments with the implemented prototype indicate that the overhead of the proposed architecture is negligible (both with respect to computational overhead and storage space) and the system is unlikely affect the user's experience while reading articles in the Wikipedia.

We have demonstrated the ability to integrate a recommender system with an existing wiki implementation, by storing recommendations in invisible comments on the wiki pages. This type recommendation systems allow clients to implement a subjective ranking algorithm, which prevents a (possibly artificially created) majority of users from dominating the recommendations made by the system. Moreover, the recommender system is designed such that it will not influence the content of the system, and users that do not want to use this recommender system, will not see the recommendations.

Appendix A

An Example

This appendix goes through a small simple example on how trust values are updated based on feedback from users and an example where trust values are aggregated.

A.1 Trust Updating

Consider two users, *Alice* and *Bob*, which have never had any interactions before. They both reads and gives feedback to an articles *Alpha*. Alice however knows the user *Charlie*.

A.1.1 Introduction to the RoR

First Bob reads the article Alpha, and because there is not recommendations on the page he cannot get an recommendation. However he thinks that the article is pretty average and gives it the recommendation *5*. The same day Charlie reads the article and gives the page a recommendation. Two days after Alice reads the article Alpha, but as she does not know Bob, she does not benefit from the recommendation that Bob put on the page. However, she knows Charlie and based on the recommendation that Charlie put on the page Alice i presented with the recommendation 5.0. Alice rates the article 4, and states that she is happy about the recommendation provided. The fact that Alice and Bob rates similar (Bob rated 5) makes Bob a potential reviewer and therefore Bob is now introduced into Alice's Ring of Reviewers. The actions taken are as following:

1. Bob is introduced as a neutral reviewer, with a trust value of 0, N value of 0 and an X value of 0.
2. Because the interaction is positive (they both rated similar), the X value is adjusted upwards by 0.1, from 0.0 to 0.1,
3. Because the experience is positive (Alice stated she was content) the N value is adjusted 0.1 upwards to 1.1
4. Bob's trust value is now calculated to $(-|(0.1-1)^{1.1}|+1)^{\frac{1}{1.1}} = 0.134$

A.1.2 Another Encounter

Now Alice comes across another article that Bob have reviewed, and rated 6. Alice also rated the article 6 and states that she is content about the result. N value and X value are adjusted accordantly to $N = 1.2$ and $X = 0.2$, and the trust value can be updated again to $(-|(0.2-1)^{1.2}|+1)^{\frac{1}{1.2}} = 0.299$. And so on Alice will keep on adjusting her trust value towards Bob every time she interacts with Bobs ratings.

A.2 Calculating a Racommendation

Consider four users *Alice*, *Bob*, *Charlie* and *Diana*, that have had interactions with each other and therefore they have different trust values towards each other. Diana's and Charlie's trust values towards Alice and Bob can be seen in table A.1 and A.2.

Now consider an article *Alpha* that have been rated by Alice and Bob. These article ratings can be seen on table A.3.

Calculating Dianas personal recommendation for article Alpha is done like following with use of the 5% trimmed mean as described in section 2.6.

First we have to create a set of recommendations that are used to adjusted to the trust value. This is done by multiplying the trust value of each person in the RoR by 100 and insert that amount of recommendations in to a set. In our example there would be a set of 77 recommendations of 5 and 43 recommendations of 7. This set of recommendations are sorted and then trimmed 5% in each end. This removed 6 5-recommendations and 6 7-recommendations. We calculate the average from the rest:

$$\begin{aligned} 71 \cdot 3 &= 81 \\ \underline{37 \cdot 7} &= \underline{259} \\ 108 &= 340 \end{aligned}$$

This gives a 5% trimmed average of:

$$\frac{340}{108} = \underline{3.15}$$

However if Charlie visits the page he will be presented with the trust value 5.93 based on the same calculations, but the trust values are totally different and therefor their result will be different as well.

User	Trust value
Alice	0.77
Bob	0.43

Table A.1: Diana's RoR, which showing the trust values towards the other users

User	Trust value
Alice	0.29
Bob	0.71

Table A.2: Charlies's RoR, which showing the trust values towards the other users

User	Recommendation
Alice	3
Bob	7

Table A.3: Ratings on article Alpha

Appendix B

Installation Instructions

This appendix describes how Scone and the WRS should be installed. The installation procedures for Scone are taken from the Scone website http://scone.de/download.html#install.

A complete and ready to run installation is provided on the CD enclosed. Please see appendix E.

B.1 Components needed

You need to download following:

Scone. Can be downloaded from http://scone.de/download.html

Java Platform, Standard Edition Version 1.5, which can be downloaded from http://java.sun.com/javase/downloads/index.jsp.

The WBI framework. The framework can be downloaded from http://www.alphaworks.ibm.com/tech/wbidk/. The framework is released under

IBM ALPHAWORKS LICENSE AGREEMENT [1], and is basically free for non-commercial use.

MySQL database. Version 4 or later. Can be downloaded from `http://dev.mysql.com/downloads/index.html`

B.2 Installation

If not already present on your System, install the Java Development Kit (JDK) first. More information on the JDK and its installation is available from Sun.

Install Scone by unpacking the downloaded zip-archive to a directory of your choice.

Install WBI by unpacking the WBI-archive in the `run` directory of Scone installation. The name of the target directory has to be `wbij45`. (an directory with this name, a readme file and some essential configuration files can already be found there.)

Install MySQL 4.0. MySQL is not mandatory, but several functions of Scone are not available without database, like persistent objects and advanced queries. You should set following MySQL variable:

```
max_allowed_packet=16M
```

The variable have to be set in the `[mysqld]`-section in the `my.ini` file. (See setup directory of Scone.)

Now compile Scone and WRS. They can both be build with the `ANT`[2] tool. The build file for Scone is present in the Scone installation directory and the build file for the WRS can be found on the CD-rom or in appendix C.13.4.

Now copy all the WRS directories (`adc`, `benchmark`, `rating`, `sconeplugin`, `remote`, `static_textfiles`, `statictools`, `test` and `trust`) into the `run` directory of Scone.

If not already running, start the MySQL daemon. Setup the Scone database from the definitions that can be found in `setup/sconedb.sql`. This can easily be done by calling the following commands from the command promt:

[1] `http://www.almaden.ibm.com/cs/wbi/License.html`

[2] http://ant.apache.org/

```
$ mysql scone -u root -p < sconedb.sql
```

and

```
$ mysql scone -u root -p < setUserRights.sql
```

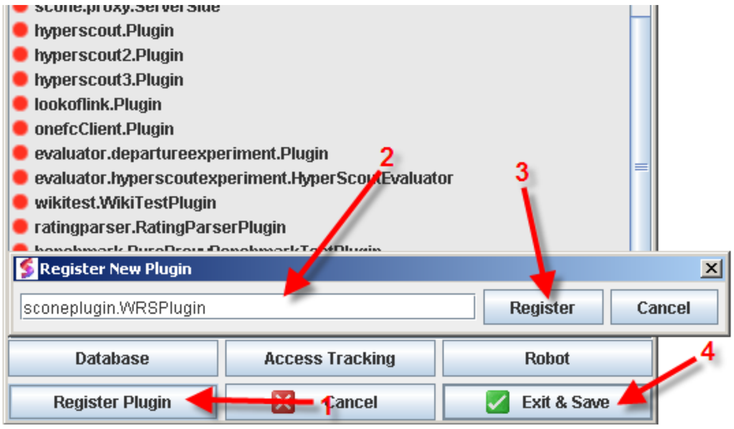
Configure your browser: You have to set the proxy of your browser to port 8088 (if Scone runs on your local machine the address is http://localhost:8088). It is also recommended to disable the hard disk cache of the browser and set it to "check page for update at every access".

B.3 Register plugins

In order to use Scone and WRS, plugins are needed to be registered:

First, register Scone-Plugins in WBI. Use "`runScone.bat -g`" in the run directory, or in the WBI-GUI use "Plugins" → "Register..." and select `scone.reg` from the setup-directory.

Then register the WRS plugin by running the `sconeConfig.bat` in the run directory. In the *"Configure Scone Properties"* window, as shown on figure B.1, do the following:


scone.proxy.ServerSide
hyperscout.Plugin
hyperscout2.Plugin
hyperscout3.Plugin
lookoflink.Plugin
onefcClient.Plugin
evaluator.departureexperiment.Plugin
evaluator.hyperscoutexperiment.HyperScoutEvaluator
wikitest.WikiTestPlugin
ratingparser.RatingParserPlugin

Register New Plugin

sconeplugin.WRSPlugin

Figure B.1: Register the WRS plugin

1. Click the ”Register Plugin” button.
2. Type ”sconeplugin.WRSPlugin” in the text box.
3. Click Register.

The plugin should now be registered as seen on figure B.2.

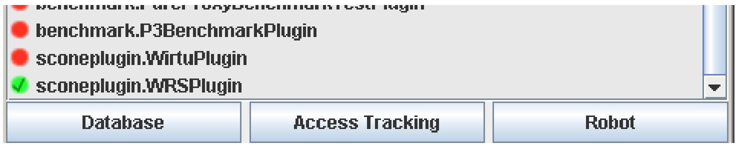
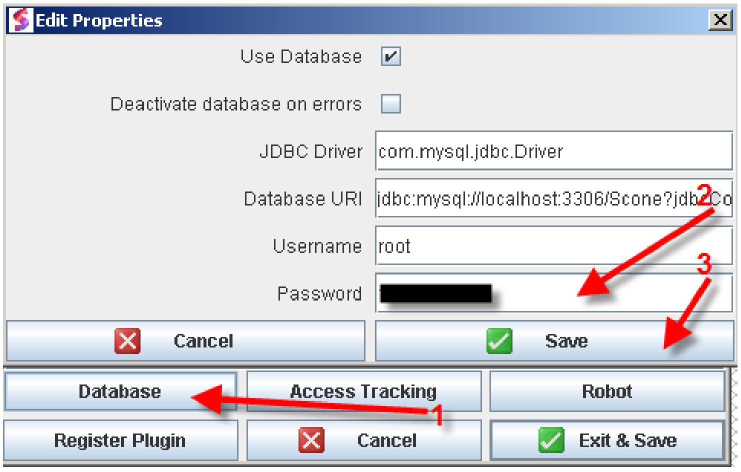

🔴 benchmark.P3BenchmarkPlugin
🔴 sconeplugin.WirtuPlugin
✓ sconeplugin.WRSPlugin

Figure B.2: WRS plugin successfully registered

Finally, make sure that the database is configured with the correct username and password as shown on figure B.3:


Edit Properties

Use Database	✓
Deactivate database on errors	☐
JDBC Driver	com.mysql.jdbc.Driver
Database URl	jdbc:mysql://localhost:3306/Scone?jdbcCo
Username	root
Password	[redacted]

Figure B.3: Setup the username and password for the database

1. Click ”Database”
2. Type in username and password for the Scone database. The username should be the same as the user that is used in the command promt (with `-u` switch) to insert the sql files to the database.

3. Click ”Save”.

Click Exit & Save.

Now compile NanoHTTPD or use an other Webserver, and copy the remote package into that directory.

```
$ javac nano/HanoHTTPD.java
```

Now run the start up scripts. `zWebserver.bat` starts the webserver, to starts the rmiregistry run z`Rmiregistry.bat` and `zScone.bat` starts Scone.

```
$zWebserver.bat
$zRmiregistry.bat
$zScone.bat
```

Point your browser to the Wikipedia and start surfing.

Appendix C

Code

The appendix contains all the code developed for the project.

C.1 Benchmark Package

C.1.1 P3Benchmark.java

```
package benchmark;

import java.io.IOException;
import java.util.Hashtable;
import java.util.Set;

import page.ExtractRatings;
import page.PageExtractor;
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.Serializer;
import statictools.TokenInputStreamTools;
import trust.RoR;

/**
 * P3Benchmark.java is used to preform benchmarking tests on a number of ratings
 * The benchmarking test only tests the parsing and verificatoin of ratings
 *
 * @author s011564
 *
 */

public class P3Benchmark extends HtmlTokenEditor implements Cloneable {
  P3BenchmarkPlugin plugin = null;

  /**
   * @param plugin
   */
  public P3Benchmark(P3BenchmarkPlugin plugin) {
    this.plugin = plugin;
  }

  /*
   * Handle request is inherieted from scone.HtmlTokenEditor and is used to
   * modify the output from the webpage wisited and pass it on th the webserver
   *
   * @see
   *     scone.proxy.HtmlTokenEditor#handleRequest(scone.util.tokenstream.SconePipe)
   */
  public void handleRequest(SconePipe pipe) {

    try {
      // Initiate a temp token
      Token t = null;
      // The two stream that tokens are read and written to
      TokenInputStream original = pipe.getTokenInputStream();
      TokenOutputStream out = pipe.getTokenOutputStream();
      // Obtaining the URL of the page through the metainformation on the
      // inputtokenstream
      Hashtable ht = original.getMetaInfo();
      Set s = ht.keySet();
      Object[] s1 = s.toArray();
      Object url = ht.get(s1[0]);
      String page_url = url.toString();
      // Extract the different URLs
      PageExtractor pex = new
          PageExtractor(TokenInputStreamTools.CreateTokenInputStreamFromURL(page_url),
           page_url);
      // Initiate a test RoR from disk
      RoR ror = Serializer.readRoRFromDisk();
      // Extract the ratings
      ExtractRatings exr = new ExtractRatings(pex.extractEditPage(), ror);
      exr.getRawRatings();
      System.out.println(pex.extractEditPage());
      // Write everything to the browser
      while ((t = original.read()) != null) {
        out.write(t);
      }

    } catch (IOException e) {
      e.printStackTrace();
    }

  }

}
```

C.1.2 P3BenchmarkPlugin.java

```
package benchmark;

import scone.Plugin;

public class P3BenchmarkPlugin extends Plugin {

  // requirements inherited from plugin
  public int getRequirements() {
    return PARSEDOCUMENT | CONSIDERLINKS;
  }

  // init() implemented from Plugin interface
  public void init() {
    P3Benchmark p3BT = new P3Benchmark(this);
    p3BT.setup("ParseThreeRatingsBenchmarkTest", HTDOCCONDITION, 60);
    addMeg(p3BT);
  }
}
```

C.1.3 PureProxyBenchmarkTest.java

```
package benchmark;
```

```
import scone.Plugin;

public class P3BenchmarkPlugin extends Plugin {

  // requirements inherited from plugin
  public int getRequirements() {
    return PARSEDOCUMENT | CONSIDERLINKS;
  }

  // init() implemented from Plugin interface
  public void init() {
    P3Benchmark p3BT = new P3Benchmark(this);
    p3BT.setup("ParseThreeRatingsBenchmarkTest", HTDOCCONDITION, 60);
    addMeg(p3BT);
  }
}
```

C.1.4 PureProxyBenchmarkTestPlugin.java

```
package benchmark;

import scone.Plugin;

public class P3BenchmarkPlugin extends Plugin {

  // requirements inherited from plugin
  public int getRequirements() {
    return PARSEDOCUMENT | CONSIDERLINKS;
  }

  // init() implemented from Plugin interface
  public void init() {
    P3Benchmark p3BT = new P3Benchmark(this);
    p3BT.setup("ParseThreeRatingsBenchmarkTest", HTDOCCONDITION, 60);
    addMeg(p3BT);
  }
}
```

C.2 Page Package

C.2.1 ExtractRatings.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 rating.Rating;
import rating.SessionRatingDB;
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 statictools.RatingCleanOut;
import trust.RoR;

// TODO: Denne fil kræver meget oprydning

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>();

// The RoR where user information is optained
RoR ror = null;

public ExtractRatings(String editUrl, RoR ror) {
  this.ror = ror;

  // cout out the title between the first = and the first &
  // Eg http://en.wikipedia.org/w/index.php?title=Bass_Strait&action=edit
  String pageTitle = (editUrl.split("=")[1]).split("&")[0];
  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 in to 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 one of the tokens is a <textarea> tag and not the end
        // tag
        if (tempHtmlTagToken.getTagType() == HtmlTagToken.T_TEXTAREA &&
            !tempHtmlTagToken.isEndTag()) {
          HtmlTextToken signature = null;
          String sig = null;
          log.fine("Found <textarea>. Looking for TrustComments");
          // read the next elemnt in the tokenstream, and keep on
          // reading as long at it is of the type HtmlTextToken
          do {
            tempToken = tokens.remove(0);
            // If it is an instance og HtmlTextToken and it
            // starts with TrustComment
            if (tempToken instanceof HtmlTextToken) {
              signature = (HtmlTextToken) tempToken;
              sig = signature.getText();
              // if the length of the HtmlTextTokens are
              // longer that 12 chars and they starts with
              // "TrustComment"
              if (sig.length() >= 16) {

                if (sig.substring(0, 16).equals("WikiTrustComment")) {
                  // verify ratings
                  // Insert the verified ratings in to the
                  // SessionRatingDB
                  String raw_rating = "";
                  // Remove the 5 words: ikiTrustComment.
                  // Read more on:
                  // http://en.wikipedia.org/wiki/User:Korsgaard
                  for (int i = 0; i < 5; i++)
                    raw_rating += ((HtmlTextToken) tokens.remove(0)).getText() +
                        " ";
                  raw_ratingsVector.add(raw_rating);
                }

              }
            }

          } // ends do-while loop associated whit the do scope
          while (tempToken instanceof HtmlTextToken);
        }
```

```
          }
        }

      } catch (MalformedURLException e) {
        // If the URL are not correct formed
        e.printStackTrace();
      } catch (IOException e) {
        // If there is a general read write error.
        e.printStackTrace();
      }

      // Cleaning out the ratings
      // Remove Ratings that dont have the correct table
      raw_ratingsVector = RatingCleanOut.RemoveRatingsTitleMismatch(pageTitle,
           raw_ratingsVector);
      // Remove ratings that are unverifiable
      raw_ratingsVector = RatingCleanOut.RemoveUnvalidableRatings(raw_ratingsVector,
           ror);
      // Remove Ratings that are above a certain threshold
      // TODO: Make this a propor threashold. Make configuration.
      raw_ratingsVector = RatingCleanOut.RemoveRatingsBelowThreshold(0.1,
           raw_ratingsVector, pageTitle);
      // remove dublicate ratings. Take away the ratings that were added by
      // the same person.
      // Leave only the newest
      // TODO: implement raw_ratingsVector(vector)
      // raw_ratingsVector =
      // RatingCleanOut.raw_ratingsVector(raw_ratingsVector);

      // Setup the sessionRatingDB
      sessionRatingDB = new SessionRatingDB();

      Rating r = null;
      String raw_ratingString[];
      // Insert the rating in to the sessionRatingDB
      while (!raw_ratingsVector.isEmpty()) {
        raw_ratingString = raw_ratingsVector.firstElement().split(";");
        r = new Rating(raw_ratingString[1], new Integer(raw_ratingString[2]),
             raw_ratingString[3],
            "http://en.wikipedia.org/wiki/" + raw_ratingString[4]);
        sessionRatingDB.push(r);
        raw_ratingsVector.remove(0);
      }
    }

    /**
     * getRawRatings() returns the raw raings in string format
     *
     * TODO: Is this method used outside the package
     *
     * @return Vector
     */
    public Vector getRawRatings() {
      return raw_ratingsVector;
    }

    /**
     * getSessionRatingDB() gives the filtered ratings in for this session
     *
     * @return
     */
    public SessionRatingDB getSessionRatingDB() {
      return sessionRatingDB;
    }
}
```

C.2.2 PageExtractor.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
 *
 * @param tokens
 */
public PageExtractor(TokenInputStream in, String URL) {
  base_page = URL;
  log.setLevel(Level.FINE);
  // Vector to store the tokens from the stream
  vec_analyze = new Vector<Token>();
  Token t = null;
  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("ca-edit")) {
            log.fine("Found LI tag with id=\"" + tag.getParam("id") + "\"
                 parameter");
            // Find the link from the next <a> tag
            HtmlTagToken temp_Token = (HtmlTagToken) vec_analyze.elementAt(i + 1);
            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("/");
  edit_page = "http://" + temp[2] + relative_url;
  edit_page = edit_page.replace("&amp;", "&");
  // using that edit an history called the sameway as 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;
  }

}
```

C.2.3 PageModifier.java

```
package page;

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 scone.util.tokenstream.HtmlTagToken;
import scone.util.tokenstream.HtmlTextToken;
import scone.util.tokenstream.Token;

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

  Vector<Token> htmlPage;

  /**
   * Default contstructor 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() {
  }

  /**
   * ReplaceTag 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 insertYUIandRating(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 temp = "";
      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;
      }

      // replacing ###RATING###
      toInsertInBody = replaceTag("###RATING###", toInsertInBody,
           String.valueOf(rating));

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

      // 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 HeadHtml 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 BodyHtml text before
          // this.
          if (tempHtmlTagToken.getTagType() == HtmlTagToken.T_BODY &&
               tempHtmlTagToken.isEndTag()) {
            HtmlTextToken body = new HtmlTextToken(toInsertInBody);
            htmlPage.insertElementAt(body, i);
            i++;
          }

        }
        i++;
      }

    } catch (FileNotFoundException e) {
      // IF the files that are to be inserted are unavailable
      e.printStackTrace();
    } catch (IOException e) {
      // If a general Rea. or write error occurs
      e.printStackTrace();
    }

  }

  /**
   * Returns the htmlPage vector when it is ready to be written to the output
   * stream
   *
   * @return
   */
  public Vector<Token> getHtmlPageVector() {
    return htmlPage;
  }

}
```

C.3 Rating Package

C.3.1 Rating.java

```
package rating;

import java.util.Date;

public class Rating {
  private Date d1;

  private int rating;

  private int experience;

  private int interaction;

  private String username;

  private String article_url;

  private String version;

  /**
   * @param hashValue
   * @param d1
   * @param rating
   * @param experience
   */

  /***************************************************************************
   * Use this constructor when initiating rations in to the SessionRatingDB. Use
   * setExp() to set experience later on
   *
   * @return
   */
  public Rating(String username, int rating, String version, String article_url) {
    this.username = username;
    this.rating = rating;
    this.version = version;
    this.article_url = article_url;
    this.experience = 2;
    this.d1 = new Date();
    this.interaction = 2;

  }

  /**
   * This constructor is only for testing purposen. Opens the possibillity to
   * insert old ratings, with costimized dates! USE WITH CARE, and block commet
   * this method in final release!!
   *
   * @param hashValue
   * @param rating
   * @param date
   * @param experience
   */
  public Rating(String username, int rating, String version, String article_url,
      Date date, int experience) {
    this.version = version;
    this.d1 = date;
    this.rating = rating;
    this.experience = experience;
    this.username = username;
    this.article_url = article_url;

  }

  /**
   * Default constructor! Do not use this constructor! Only used for test
   * purposes!
   *
   * TODO: Can this constructor be deleted??
   */
  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 expeirence 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
   * 0 marks a negative
   *
   */
  public int getInteraction() {
    return interaction;
  }

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

  // 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;
  }
}
```

C.3.2 RatingCalculator.java

```
package rating;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Vector;

import trust.RoR;

/*************************************************************************
 * The ratingCalculator puts together a rating based on the active ratings in
 * the sessionRatingDB and the trust values in RoR.
 *
 * The average is calculated on a 5% trimmed mean based on David Wagner's
 * Resilient Aggregation in Sensor Networks
 *
 *
 *
 * @author s011564
 *
 */
public class RatingCalculator {

  // The ratings extracted in this session
  SessionRatingDB sessionRatingDB;

  // Arraylist to simulate the Sensor readings
  ArrayList<Integer> ratings;

  // The asociated RoR
  RoR ror;

  // Constructor
  public RatingCalculator(SessionRatingDB sessionRatingDB, RoR ror) {
    this.sessionRatingDB = sessionRatingDB;
    ratings = new ArrayList<Integer>();
    this.ror = ror;
  }

  /*************************************************************************
   * cleanSessionRatingDBForUsersWithNegativeTrustValues() removes the
   * sessionRatingDB for any ratings that have been given from a user that have
   * a negative trust value
   *
   */
  private void cleanSessionRatingDBForUsersWithNegativeTrustValues() {
    Vector<Rating> temp = new Vector<Rating>();
    int sessionRatingDBSize = sessionRatingDB.size();
    // Check the SessionRatingDB through for users with a negative trust
    // value
    for (int i = 0; i < sessionRatingDBSize; i++) {
      Rating r = sessionRatingDB.pop();
      if (ror.getTrustValueFromUsername(r.getUserName()) > 0.0) {
        // add Ratinge given from trusted users to a temporary vector
        temp.add(r);
      }
    }
    // Insert the temp vector int othe now empty vector
    for (int i = 0; i < temp.size(); i++) {
      sessionRatingDB.push(temp.elementAt(i));
    }
  }

  /*************************************************************************
   * Computethe average
   *
   * @return
   */
  public double computeAverage() {
    // DecimalFormat used to an integer
    DecimalFormat df = new DecimalFormat("##");
    // Cleanout
    cleanSessionRatingDBForUsersWithNegativeTrustValues();
    // set up the arraylist of ratings
    int sessionRatingDBSize = sessionRatingDB.size();
    for (int i = 0; i < sessionRatingDBSize; i++) {
      Rating r = sessionRatingDB.pop();
      // Multiply acording to trust values
      double trustValue_for_user_for_Rating_r =
          ror.getTrustValueFromUsername(r.getUserName());

      // String double_numbers_to_be_copied =
      // df.format(trustValue_for_user_for_Rating_r);
      int numbers_of_copies_to_insert = (int) (double) new
          Double(df.format(trustValue_for_user_for_Rating_r * 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] = (int) 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_present_threshold = (int) (double) new Double(df.format(threshold));

    // Create an integer array that if 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_present_threshold)];

    int sum = 0;
    // Calculate the sum of the ratings
    for (int i = 0; i < rating_integer_array_trimmed.length; i++) {
      rating_integer_array_trimmed[i] = ratings_integer_array_untrimmed[i +
           five_present_threshold];
      // Updating the sum
      sum += rating_integer_array_trimmed[i];
    }

    double average = ((double) sum) / ((double)
         rating_integer_array_trimmed.length);

    // compute average
    return average;

  }
}
```

C.3.3 RatingHistory.java

```
package rating;

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

public class RatingHistory implements Serializable {

  private final long month_milis = new Long("2592000000"); // 30*24*60*60*1000

  private final long halfYear_milis = new Long("15552000000"); // 6*30*24*60*60*1000

  private final long year_milis = new Long("31104000000"); // 12*30*24*60*60*1000

  private ArrayList<Rating> history;

  private int positive_one_month, negative_one_month, positive_half_year,
       negative_half_year, positive_one_year,
      negative_one_year;

  private double xValue;

  public RatingHistory() {
    history = new ArrayList<Rating>();
    positive_half_year = 0;
    positive_one_month = 0;
    positive_one_year = 0;
    negative_half_year = 0;
    negative_one_month = 0;
    negative_one_year = 0;
    xValue = 0.0;
  }

  public void insertRating(Rating r) {
    history.add(r);
    updateTrustDynamics();
  }

  public double getXValue() {
    return xValue;
  }

  public int getHistorySize() {
    return history.size();
  }

  private void updateTrustDynamics() {
    Date now = new Date();
    long thisTime = now.getTime();
    resetCounterValues();
    // Set month, halfYear and year limits
    long month = thisTime - month_milis;
    long halfYear = thisTime - halfYear_milis;
    long year = thisTime - year_milis;
    // System.out.println(new Date(month).toString());
    // System.out.println(new Date(halfYear).toString());
    // System.out.println(new Date(year).toString());
    for (int i = 0; i < history.size(); i++) {
      long ratingAge = history.get(i).getDate();// thisTime -
      // history.get(i).getDate()
      // ;
      // If it is positive expirence
      if (history.get(i).getInteraction() == 1) {
        if (ratingAge > month)
          positive_one_month++;
        if (ratingAge < month && ratingAge > halfYear)
          positive_half_year++;
        if (ratingAge < halfYear && ratingAge > year)
          positive_one_year++;
      }
      // If it is a negative experience
      if (history.get(i).getInteraction() == 0) {
        if (ratingAge > month)
          negative_one_month++;
        if (ratingAge < month && ratingAge > halfYear)
          negative_half_year++;
        if (ratingAge < halfYear && ratingAge > year)
          negative_one_year++;

      }
    }
    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;
  }

  private void resetCounterValues() {
    positive_half_year = 0;
    positive_one_month = 0;
    positive_one_year = 0;
    negative_half_year = 0;
    negative_one_month = 0;
    negative_one_year = 0;
  }
}
```

C.3.4 RecommendationSubmitter.java

```
package rating;

import java.io.IOException;
import java.security.PrivateKey;

import statictools.SecurityProvider;
import statictools.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) {

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

    try {
      // Create an instance og the WIKIbot for the english Wikipedia as
      // default
      Wiki bot = new Wiki();
      // Login to the wiki
      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;
      // insert the new rating
      bot.editPage(name, text, "WikiTrustComment", true);
    } catch (IOException e) {
      System.out.println(e.getStackTrace());
      e.printStackTrace();
    }

  }
}
```

C.3.5 SessionRatingDB.java

```
package rating;

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 ArrayList<Rating> ratings;

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

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

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

  /***********************************************************************
   * size() returns the size of the SessionRatingDB
   *
   * @return
   */
  public int size() {
    return ratings.size();
  }
  /**
   * prettyPrintAllSignatures prettyprints all the signatures found in a
   * wikipedia edit page.
   *
   * @param signatures
   * @return a string that have to be printed to a log og terminal
   */
  public String prettyPrintAllSignatures() {
    String toReturn = "\n";
    for (Rating i : ratings) {
      toReturn += i.prettyPrint();
    }
    return toReturn;
  }

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

C.4 Remote Package

C.4.1 EmbeddedApplet.java

```
package remote;

import java.applet.Applet;
import java.awt.Component;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Font;
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 javax.swing.JButton;
import javax.swing.JLabel;

public class EmbeddedApplet extends Applet implements ActionListener {
  JButton button_YES;

  JButton button_NO;

  JLabel label_2;

  JButton button_rate9;

  JButton button_rate2;

  JButton button_rate3;

  JButton button_rate4;

  JButton button_rate5;

  JButton button_rate6;

  JButton button_rate7;

  JButton button_rate8;

  JButton button_rate1;

  JLabel label_3;

  BufferedWriter out = null;

  public void actionPerformed(ActionEvent ae) {
```

```
if (ae.getSource() == button_YES) {
  String host = null;
  try {
    Registry registry = LocateRegistry.getRegistry(host);
    FeedbackInterface stub = (FeedbackInterface)
         registry.lookup("FeedbackInterface");
    // String response = stub.sayHello();
    stub.clickYes();
    button_NO.setEnabled(false);
  } catch (Exception e) {
    e.printStackTrace();
  }
}
if (ae.getSource() == button_NO) {
  String host = null;
  try {
    Registry registry = LocateRegistry.getRegistry(host);
    FeedbackInterface stub = (FeedbackInterface)
         registry.lookup("FeedbackInterface");
    // String response = stub.sayHello();
    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();
    stub.clickRating(7);
    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();
  }
}
if (ae.getSource() == button_rate8) {
  String host = null;
  try {
    Registry registry = LocateRegistry.getRegistry(host);
    FeedbackInterface stub = (FeedbackInterface)
         registry.lookup("FeedbackInterface");
    // String response = stub.sayHello();
    stub.clickRating(8);
    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);
  } 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();
    stub.clickRating(1);
    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);
  } 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();
        stub.clickRating(9);
        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_rate7.setEnabled(false);
      } 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();
        stub.clickRating(2);
        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);
      } 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();
        stub.clickRating(3);
        button_rate1.setEnabled(false);
        button_rate2.setEnabled(false);
        button_rate7.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();
      }
    }
    if (ae.getSource() == button_rate4) {
      String host = null;
      try {
        Registry registry = LocateRegistry.getRegistry(host);
        FeedbackInterface stub = (FeedbackInterface)
             registry.lookup("FeedbackInterface");
        // String response = stub.sayHello();
        stub.clickRating(4);
        button_rate1.setEnabled(false);
        button_rate2.setEnabled(false);
        button_rate3.setEnabled(false);
        button_rate7.setEnabled(false);
        button_rate5.setEnabled(false);
        button_rate6.setEnabled(false);
        button_rate8.setEnabled(false);
        button_rate9.setEnabled(false);
      } 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();
        stub.clickRating(5);
        button_rate1.setEnabled(false);
        button_rate2.setEnabled(false);
        button_rate3.setEnabled(false);
        button_rate4.setEnabled(false);
        button_rate7.setEnabled(false);
        button_rate6.setEnabled(false);
        button_rate8.setEnabled(false);
        button_rate9.setEnabled(false);
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
    if (ae.getSource() == button_rate6) {
      String host = null;
      try {
        Registry registry = LocateRegistry.getRegistry(host);
        FeedbackInterface stub = (FeedbackInterface)
             registry.lookup("FeedbackInterface");
        // String response = stub.sayHello();
        stub.clickRating(6);
        button_rate1.setEnabled(false);
        button_rate2.setEnabled(false);
        button_rate3.setEnabled(false);
        button_rate4.setEnabled(false);
        button_rate5.setEnabled(false);
        button_rate7.setEnabled(false);
        button_rate8.setEnabled(false);
        button_rate9.setEnabled(false);
      } catch (Exception e) {
        e.printStackTrace();
      }
    }

  }

  public void init() {
    // try {
    // out = new BufferedWriter(new FileWriter("error.log"));
    // } catch (IOException e) {
    // e.printStackTrace();
    // }
    AppletLayout customLayout = new AppletLayout();

    setFont(new Font("Helvetica", Font.PLAIN, 12));
    setLayout(customLayout);

    button_YES = new JButton("Yes");
    add(button_YES);
    button_YES.addActionListener(this);

    button_NO = new JButton("No");
    add(button_NO);
    button_NO.addActionListener(this);

    label_2 = new JLabel("Was This Information Usefull to You?");
    add(label_2);

    button_rate9 = new JButton("9");
    add(button_rate9);
    button_rate9.addActionListener(this);

    button_rate2 = new JButton("2");
    add(button_rate2);
    button_rate2.addActionListener(this);

    button_rate3 = new JButton("3");
    add(button_rate3);
    button_rate3.addActionListener(this);

    button_rate4 = new JButton("4");
    add(button_rate4);
    button_rate4.addActionListener(this);

    button_rate5 = new JButton("5");
    add(button_rate5);
    button_rate5.addActionListener(this);

    button_rate6 = new JButton("6");
    add(button_rate6);
    button_rate6.addActionListener(this);

    button_rate7 = new JButton("7");
    add(button_rate7);
    button_rate7.addActionListener(this);

    button_rate8 = new JButton("8");
    add(button_rate8);
    button_rate8.addActionListener(this);

    button_rate1 = new JButton("1");
    add(button_rate1);
    button_rate1.addActionListener(this);

    label_3 = new JLabel("Please rate this article");
    add(label_3);

    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 = 182 + 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(3);
    if (c.isVisible()) {
      c.setBounds(insets.left + 200, insets.top + 112, 48, 32);
    }
    c = parent.getComponent(4);
    if (c.isVisible()) {
      c.setBounds(insets.left + 8, insets.top + 144, 48, 32);
    }
    c = parent.getComponent(5);
    if (c.isVisible()) {
      c.setBounds(insets.left + 56, insets.top + 112, 48, 32);
    }
    c = parent.getComponent(6);
    if (c.isVisible()) {
      c.setBounds(insets.left + 56, insets.top + 144, 48, 32);
    }
    c = parent.getComponent(7);
    if (c.isVisible()) {
      c.setBounds(insets.left + 104, insets.top + 112, 48, 32);
    }
    c = parent.getComponent(8);
    if (c.isVisible()) {
      c.setBounds(insets.left + 104, insets.top + 144, 48, 32);
    }
    c = parent.getComponent(9);
    if (c.isVisible()) {
      c.setBounds(insets.left + 152, insets.top + 112, 48, 32);
    }
    c = parent.getComponent(10);
    if (c.isVisible()) {
      c.setBounds(insets.left + 152, insets.top + 144, 48, 32);
    }
    c = parent.getComponent(11);
    if (c.isVisible()) {
      c.setBounds(insets.left + 8, insets.top + 112, 48, 32);
    }
    c = parent.getComponent(12);
    if (c.isVisible()) {
      c.setBounds(insets.left + 8, insets.top + 80, 240, 24);
    }
  }
}
```

C.4.2 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) throws RemoteException;

}
```

C.5 Sconeplugin Package

C.5.1 WRS.java

```
package sconeplugin;

import java.io.IOException;
import java.rmi.AccessException;
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 page.ExtractRatings;
import page.PageExtractor;
import page.PageModifier;
import rating.RatingCalculator;
import rating.RecommendationSubmitter;
import rating.SessionRatingDB;
import remote.FeedbackInterface;
import scone.proxy.HtmlTokenEditor;
```

```java
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.RoR;
import trust.TrustUpdater;

/*****************************************************************************
 * 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;

  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) {
    System.out.println("Clicked " + rating);
    if (rating == 1) {
      tu.ClickedR1();
      rs.SubmitStringToForm(1);
    }
    if (rating == 2) {
      tu.ClickedR2();
      rs.SubmitStringToForm(2);
    }
    if (rating == 3) {
      tu.ClickedR3();
      rs.SubmitStringToForm(3);
    }
    if (rating == 4) {
      tu.ClickedR4();
      rs.SubmitStringToForm(4);
    }
    if (rating == 5) {
      tu.ClickedR5();
      rs.SubmitStringToForm(5);
    }
    if (rating == 6) {
      tu.ClickedR6();
      rs.SubmitStringToForm(6);
    }
    if (rating == 7) {
      tu.ClickedR7();
      rs.SubmitStringToForm(7);
    }
    if (rating == 8) {
      tu.ClickedR8();
      rs.SubmitStringToForm(8);
    }
    if (rating == 9) {
      tu.ClickedR9();
      rs.SubmitStringToForm(9);
    }
  }

  // handleRequest() is inherited from HtmlTokenEditor
  public void handleRequest(SconePipe pipe) {

    // Set up a database to store the ratings captured
    SessionRatingDB sessionRatingDB = new SessionRatingDB();
    // Read a previously stored RoR from the disk
    RoR ror = Serializer.readRoRFromDisk();
    // The tokenstreams are initiated from the SconePipe
    in = pipe.getTokenInputStream();
    out = pipe.getTokenOutputStream();
    // Page URL are obtained from the inputstreams' meta information
    Hashtable ht = in.getMetaInfo();
    Set s = ht.keySet();
    Object[] s1 = s.toArray();
    Object url = ht.get(s1[0]);
    String page_url = url.toString();
    // The history and Edit URL are extracted from the page
    PageExtractor pex = new
         PageExtractor(TokenInputStreamTools.CreateTokenInputStreamFromURL(page_url),
          page_url);
    // Ratings are extracted from the Editpage and filtered with rhe RoR
    ExtractRatings exr = new ExtractRatings(pex.extractEditPage(), ror);
    // The extracted ratings are stored in the SessionRatingDB
    sessionRatingDB = exr.getSessionRatingDB();
    // An rating is calculated based on the trust values from the RoR
    RatingCalculator rc = new RatingCalculator(sessionRatingDB, ror);
    // TOken Vector is initiated from the TokenInputStream for modification
    Vector<Token> htmlPage = initTokenVector();
    // PageModifier is initiated
    PageModifier pm = new PageModifier(htmlPage);
    // Avareage is insertet allong with the HTML to cast own vote and
    // feedback
    double computedAverage = rc.computeAverage();
    pm.insertYUIandRating(computedAverage);
    // Stream is written
    writeOutToken(pm.getHtmlPageVector());
    // Prepare the recommendation submitter
    String wiki_username = (SecurityProvider.getWikiUserPassFromDisk())[0];
    String wiki_password = (SecurityProvider.getWikiUserPassFromDisk())[1];
    String version = pex.extractionVersion();
    String keyStorePass = SecurityProvider.getKeyStorePassFromDisk();
    KeyStore ks = SecurityProvider.InitKeyStore(".keys", keyStorePass);
    // System.out.println(wiki_username+" "+wiki_password+" "+keyStorePass);
    KeyPair keyPair = SecurityProvider.InitKeyPair(ks, wiki_username, keyStorePass);
    PrivateKey privateKey = keyPair.getPrivate();
    String article_name = pex.extractTitle();
    rs = new RecommendationSubmitter(wiki_username, wiki_password, version,
         article_name, privateKey);

    tu = new TrustUpdater(sessionRatingDB, computedAverage, ror);
    // Setting up RMI connections to wait for input from the user
    // Wirtu obj = new Wirtu();

    try {
      FeedbackInterface stub = (FeedbackInterface)
           UnicastRemoteObject.exportObject(this, 0);

      // Bind the remote object's stub in the registry
      Registry registry = LocateRegistry.getRegistry(1099);
      registry.bind("FeedbackInterface", stub);

      System.err.println("Waiting for Feedback");
    } catch (AccessException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    } catch (RemoteException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    } catch (AlreadyBoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    } catch (NullPointerException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    // Write the newny calculated trust values to the disk!
    Serializer.writeRoRToDisk(ror);

  }

  /**
   * initTokenVector is used to read the tokenstream to a Vector
   *
   * @return
   */
  private Vector<Token> initTokenVector() {
    Vector<Token> htmlPage = new Vector<Token>();
    Token t = null;
    try {
      while ((t = in.read()) != null) {
        htmlPage.add(t);
      }

    } catch (IOException e) {
      System.out.println("Initialization of Token Vector failed");
      e.getStackTrace();
    }
    return htmlPage;
  }

  /**
   * WriteOutToken() is used to write tokens to a TokenOutPutStream
   *
   * @param htmlPage
   */
  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 TokenOutputStream failed!");
      e.printStackTrace();
    }
  }
}
```

C.5.2 WRSPlugin.java

```
package sconeplugin;

import scone.Plugin;

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

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

C.6 Statictools Package

C.6.1 RatingCleanOut.java

```
package statictools;

import trust.RoR;

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 s011564
 *
 */

public class RatingCleanOut {

  /*****************************************************************************
   * RemoveTitleMismatch removes the ratings where the title in the rating dont
   * match to titel 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()) {
      // If the pagetitle maches the title in the rating
      String firstElement = ratings.firstElement();
      if ((firstElement.split(";"))[4].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 page_title
   * @return
   */
  public static Vector<String> RemoveRatingsBelowThreshold(double acceptedThreshold,
       Vector<String> ratings,
      String page_title) {
    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 = Threshold.ThresholdCalculator(page_title,
          temp_string_array[3]);
      // If the calculated threshold is lower that the excepted, then
      // insert
      // the rating into the cleaned ratings
      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, RoR
      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;
  }
}
```

C.6.2 SecurityProvider.java

```
package statictools;

import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileReader;
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.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.SignatureException;
import java.security.UnrecoverableKeyException;
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.RoR;

public class SecurityProvider {

  /**
   * Initiates a keystore from the disk. The Keystore is used to store the
   * private and public key of the user. The keystore is created with keytool
   * that is included in JDK
   *
   * InitKeyStore takes the filename of the keystore and the passeword that is
   * used to encrypt the keys
   *
   * @param filename
   * @param password
   * @return
   */
  public static KeyStore InitKeyStore(String filename, String password) {
    try {
      // The keystore is an instance og Java KeyStore
      KeyStore ks = KeyStore.getInstance("JKS");
      String fileName = filename;
      // Convert the password to Chararray
      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;
    } catch (CertificateException e) {
      e.printStackTrace();
      return null;
    } catch (FileNotFoundException e) {
      e.printStackTrace();
      return null;
    } catch (IOException e) {
      e.printStackTrace();
      return null;
    }

  }
```

```
/***************************************************************************
 * Iniates a certificate from a loaded keystore
 *
 * @param ks
 *            KeyStore
 * @param user
 *            Username in keystore
 * @return Certificate
 */
public static Certificate InitCertificate(KeyStore ks, String user) {
  try {
    Certificate cert = ks.getCertificate(user);
    return cert;
  } catch (KeyStoreException e) {
    e.printStackTrace();
    return null;
  }
}

/**
 * InitKeyPair retirns a Keypair that is loaded from a keystore. the username
 * and the passphrase is required to accecs the keys from the keystore
 *
 * The KeyPair returned consists of a private and a public key
 *
 * @param ks
 * @param user
 * @param passPhrase
 * @return
 */

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) {
    e.printStackTrace();
  } catch (UnrecoverableKeyException e) {
    e.printStackTrace();
  }
  return null;
}

/***************************************************************************
 * createSignature Creates a signature with a private key on a given string
 *
 * @param whatToHash
 * @param privateKey
 * @return
 */
public static byte[] createSignature(String whatToHash, PrivateKey privateKey) {
  try {
    // lools up the algorithm that the privatekey is grenerated from
    Signature sig = Signature.getInstance(privateKey.getAlgorithm());
    // Initiates the Signature object withe 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();
  } catch (SignatureException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }
  return null;
}

/**
 * createBase64EncodingFromSignature creates a base64 representation of the
 * signature that is prinable for the web
 *
 * @param signature
 * @return
 */
public static String createBase64EncodingFromSignature(byte[] signature) {
    BASE64Encoder b64enc = new BASE64Encoder();
    return b64enc.encode(signature);
  }

  /***************************************************************************
   * getSignatureFromBase64Representation takes a base64 encodes string and
   * decodes it to a byte array
   *
   * @param encodedSignature
   * @return
   */
  public static byte[] getSignatureFromBase64Representation(String
       encodedSignature) {
    try {
      return new BASE64Decoder().decodeBuffer(encodedSignature);
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    return null;
  }

  /***************************************************************************
   * verifySignature verifyes a signature on byte array form with a public key
   *
   * @param whatToVerify
   * @param signature
   * @param publicKey
   * @return
   */
  public static boolean verifySignature(String whatToVerify, byte[] signature,
       PublicKey publicKey) {

    try {
      // lools up the algorithm that the privatekey is grenerated from
      Signature sig = Signature.getInstance(publicKey.getAlgorithm());
      // Initiates the Signature object withe the public key
      sig.initVerify(publicKey);
      // Inserts what needs to be verified into the Signature object
      sig.update(whatToVerify.getBytes(), 0, whatToVerify.getBytes().length);
      // Returns true or false on the verification
      return sig.verify(signature);
    } catch (InvalidKeyException e) {
      e.printStackTrace();
    } catch (NoSuchAlgorithmException e) {
      e.printStackTrace();
    } catch (SignatureException e) {
      e.printStackTrace();
    }

    return false;
  }

  /***************************************************************************
   * Generate certificate is used to generate a certificate for the user to
   * insert on the wiki user page. The certificate is base64 encoded
   *
   * @param cert
   * @return
   */

  public static String generateCertificate(Certificate cert) {
    BASE64Encoder myB64 = new BASE64Encoder();
    try {
      return ("-----BEGIN CERTIFICATE-----\n" + myB64.encode(cert.getEncoded()) +
           "\n-----END CERTIFICATE-----");
    } catch (CertificateEncodingException e) {
      e.printStackTrace();
    }
    return null;

  }

  /**
   * getUserCertificate() finds the usercertificate and downloads it from the
   * Wikipedia. getUserCertificate finds the latest entry by the user to the
   * /cert page and obtains the certificate on this version of the page
   *
   * @param user
   * @return
   */

  public static Certificate getUserCertificate(String user) {
    Certificate cert;
    try {
      String certificateUrl = "";
      // Look up usercertificate history site
```

```
String historyUrl = "http://en.wikipedia.org/w/index.php?title=User:" + user +
      "/cert&action=history";
// Create an URLConnection to the history page
URLConnection historyUrlConnection = new URL(historyUrl).openConnection();
historyUrlConnection.setUseCaches(false);
InputStream his = historyUrlConnection.getInputStream();
InputStreamReader hisr = new InputStreamReader(his);
// Create a token stream for parsing
TokenInputStream htis = new TokenInputStreamTokenizerImpl(hisr);
Token temp = null;
// Start finding the newest certificate
while ((temp = htis.read()) != null) {
  if (temp instanceof HtmlTagToken) {
    HtmlTagToken htmlTag = (HtmlTagToken) temp;
    // If the token is HTML tag and of the type <A>
    if (htmlTag.getTagType() == HtmlTagToken.T_A) {
      String params = htmlTag.getParam("href");
      // Match the first partof the link
      String toMatch = "/w/index.php?title=User:" + user + "/cert&amp;oldid=";
      int matchSize = toMatch.length();
      // If the link has some parameters and the length is
      // correct
      // Matching excact length in order to avoid to 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)) {
          // Move 8 token foreward to the <A> tah the
          // holds the link to the valid certificate.
          for (int i = 0; i < 8; i++) {
            temp = htis.read();
          }
          if ((temp.toString()).equals(user)) {
            // Set the certificateUrl to the latest url
            // that the user has edited
            certificateUrl = "http://en.wikipedia.org" +
                params.replace("&amp;", "&");
            // Once the latest version of the
            // certificate i found close the stream
            while (htis.read() != null) {
            }
          }

        }
      }
    }
  }
}

// Once the URL are inplace, download the HTML page whare the
// certificate is stored and retrice the certificate.
cert = null;
// Create a certificatefactory that is 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 that 9 chars and
      // it equals "-----BEGIN"
      if (text.getText().equals("-----BEGIN")) {
        // Found certificate
        String string_cert = text.getText() + " " + ((HtmlTextToken)
            tis.read()).getText();
        // read the certificate into string
        do {
          text = (HtmlTextToken) tis.read();
          string_cert += "\n" + text.getText();

        } while (!text.getText().equals("CERTIFICATE-----"));
        // Create an inputstream for the certificateFactory
        // to use
        ByteArrayInputStream bs = new
            ByteArrayInputStream(string_cert.getBytes());
        cert = cf.generateCertificate(bs);
        return cert;
      }
```

```
          }
        }

      }

    } catch (CertificateException e) {
      e.printStackTrace();
    } catch (MalformedURLException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }

    return null;
  }

  /*************************************************************************
   * Create rating puts together a rating to be inserted in to the Wikipedia
   *
   * @param wiki_username
   * @param mark
   * @param version
   * @param page_title
   * @param privateKey
   * @return
   */
  public static String createRating(String wiki_username, int mark, String version,
       String page_title,
      PrivateKey privateKey) {
    // The string that needs to be signed
    String toSign = ";" + wiki_username + ";" + mark + ";" + version + ";" +
         page_title + ";";
    // The byte array containing the signatiure
    byte[] signature = createSignature(toSign, privateKey);
    // Encode the signature on base64 form
    String encodedSignature = createBase64EncodingFromSignature(signature);
    // Return the WikiTrustComment
    return ("<!-- WikiTrustComment. Read more on:
         http://en.wikipedia.org/wiki/User:Korsgaard \n" + toSign
        + encodedSignature + " -->");
  }

  /*************************************************************************
   * Validate rating takes a raw rating and validates
   *
   * @param rating
   * @param ror
   * @return
   */
  public static boolean validateRating(String rating, RoR 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];
    // Decodec signature
    byte[] signature = getSignatureFromBase64Representation(string_signature);
    // Retrive the certificate from the cache
    Certificate cert = getUserCertificateFromCache(userName, ror);
    // Setting together what needs to be verified
    String whatToVerify = ";" + rating_parts[1] + ";" + rating_parts[2] + ";" +
         rating_parts[3] + ";" + rating_parts[4]
        + ";";
    // 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, RoR ror) {
    return ror.getCertificateFromUsername(username);
  }

  /*************************************************************************
   *
   */
  public static String getKeyStorePassFromDisk() {
    try {
      FileReader fr = new FileReader("passwords.txt");
      BufferedReader br = new BufferedReader(fr);
      String line = null;
      String[] lineArray;
      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 "";
  }

  /***************************************************************************
   *
   */
  public static String[] getWikiUserPassFromDisk() {
    try {
      FileReader fr = new FileReader("passwords.txt");
      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;
        }
      }
    } catch (FileNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

    return null;
  }

}
```

C.6.3 Serializer.java

```
package statictools;

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.RoR;

/***************************************************************************
 * Serializer is used to read and write the RoR to and from the disk.
 *
 * @author s011564
 *
 */

public class Serializer {

  public static RoR readRoRFromDisk() {
    try {
      FileInputStream f_in = new FileInputStream("static_textfiles/myRoR.dat");
      ObjectInputStream obj_in = new ObjectInputStream(f_in);
      RoR ror = (RoR) obj_in.readObject();
      return ror;
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    } catch (ClassNotFoundException e) {
      e.printStackTrace();
    }
    return null;

  }

  public static void writeRoRToDisk(RoR r) {
    try {
      FileOutputStream f_out = new FileOutputStream("static_textfiles/myRoR.dat");
      ObjectOutputStream obj_out = new ObjectOutputStream(f_out);
      obj_out.writeObject(r);
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

C.6.4 Threshold.java

```
package statictools;

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) {
    String string_url = "http://en.wikipedia.org/w/index.php?title=" + page_title +
        "&diff=current&oldid="
        + 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<ArrayList> diff_addedline = new ArrayList<ArrayList>();
    ArrayList<String> currentRevision = new ArrayList<String>();
    int i = 0;
    int number_of_words = 0;
    double threshold = 0.0;
    try {
      while ((temp = tis.read()) != null) {
        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.isEndTag()) {
              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.isEndTag()) {
                    // System.out.println(tag.toString());
                    if (tag.hasParam("class")) {
                      if (tag.getParam("class").equals("diff-addedline")) {
                        // Found the <td
                        // class="diff-addedline"> tags
                        // now work with them until
                        // </td> tag
                        // Create an arrayList of the
                        // tokens at but that in to the
                      // diff_Addedline
                      ArrayList<Token> changed_line = new ArrayList<Token>();
                      while (true) {
                        temp = tis.read();
                        // Checking if we reached
                        // the </td> that ends the
                        // diff added line marker
                        if (temp instanceof HtmlTagToken) {
                          tag = (HtmlTagToken) temp;
                          if (tag.getTagType() == HtmlTagToken.T_TD &&
                               tag.isEndTag()) {
                            break;
                          }
                        }
                        changed_line.add(temp);
                      }
                      diff_addedline.add(changed_line);
                    }
                  }
                }
                // When the </table> is found, then break
                // the loop
                if (tag.getTagType() == HtmlTagToken.T_TABLE && tag.isEndTag()) {
                  break;
                }
              }

            }
            // }
          }
        }

      }
      // Cound 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());
      }
      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 red letter words
    int number_of_red_letter_words = 0;
    for (ArrayList line : diff_addedline) {
      number_of_red_letter_words += numberOfRedLetterWords(line);
    }
    // System.out.println("number of red words: "+
    // number_of_red_letter_words);

    threshold = (double) number_of_red_letter_words / (double)
         currentRevision.size();

  } catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }

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

  // Calculate the precentage changed:

  return threshold;
}

private static int numberOfRedLetterWords(ArrayList<Token> list) {
  String[] string_amp = { "[[", "]]", "{{", "}}", "((", "))", "[", "]", "{", "}",
       "(", ")", ".", ",", ";", ":", "s",
      "'" };
  String[] reversal_words = { "not", "n't", "dont", "don't", "esn't", "doesnt",
       "doesn't", "no", "without", "wont",
        "won't", "un" };
    Set list_of_amplications = 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) {
                num_of_total_words++;
                text = (HtmlTextToken) word;
                String s_test = text.getText();
                // Analyse the text
                // Ingnore small changes - amplications
                if (!list_of_amplications.contains(s_test)) {
                  // count single word changes - additions
                  num_of_red_words++;
                  // System.out.print(s_test+ " ");
                }
                if (list_of_reversals.contains(s_test.toLowerCase())) {
                  found_reversal = true;
                }

                // find revesal word - reversal - husk små
                // bogstaver

              }
              // 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;
  }
}
```

C.6.5 TokenInputStreamTools.java

```
package statictools;

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 {
      // Set the URL
      URL url = new URL(string_url);
      URLConnection urlconnection = url.openConnection();
      urlconnection.setUseCaches(false);
      // Open up the Stream
      InputStream is = urlconnection.getInputStream();
      InputStreamReader isr = new InputStreamReader(is);
      // Create the input stream
      TokenInputStream tis = new TokenInputStreamTokenizerImpl(isr);
      return tis;
    } catch (MalformedURLException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
    return null;
  }
}
```

C.6.6 Wiki.java

```
package statictools;

/**
 * @(#)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
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
 */

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

/**
 * Denotes talk pages for image description pages. Has the prefix "Image
 * talk:".
 *
 * @since 0.03
 */
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;

private String query;

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

// internal data storage
private Map namespaces = null;

/**
 * Creates a new connection to the English Wikipedia.
 *
 * @since 0.02
 */
public Wiki() {
  this("");
}

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

/**
 * Logs in to the wiki
 *
 * @param username
 *            a username
 * @param password
 *            a password (as a char[] due to JPasswordField)
 * @return whether the login succeeded
 * @throws IOException
 *             if something goes wrong
 */
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 + "?title=Special:Userlogin";
  URLConnection connection = new URL(URL).openConnection();
  grabCookies(connection);
```

```
    // find the target
    URL = domain + "?title=Special:Userlogin&action=submitlogin&type=login";
    connection = new URL(URL).openConnection();
    setCookies(connection);
    connection.setDoOutput(true);
    PrintWriter out = new PrintWriter(connection.getOutputStream());

    // now we send the data
    out.print("wpName=");
    out.print(username);
    out.print("&wpPassword=");
    out.print(ps);
    out.print("&wpRemember=1&wpLoginattempt=Log+in");
    out.close();

    // make it stick by grabbing the cookie
    grabCookies(connection);
    BufferedReader in = null;
    try {
      in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
    } catch (IOException e) {
      if (!(connection instanceof HttpURLConnection))
        throw e;
      InputStream 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)
      if (line.indexOf("Login successful") != -1)
        return true;
    return false;
  }

  /**
   * Logs out of the wiki.
   */
  public void logout() {
    cookies.clear();
  }

  /**
   * Gets the raw wikicode for a page. WARNING: does not support special pages.
   *
   * @param title
   *           the title of the page.
   * @throws IOException
   *            if something stuffs up the connection between here and wiki
   * @return the raw wikicode of a page
   * @throws IllegalArgumentException
   *            if you try to retrieve the text of a Special: page or a Media:
   *            page
   */
  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 + "?title=" + title + "&action=raw";
    URLConnection connection = new URL(URL).openConnection();
    connection.connect();
    BufferedReader in = new BufferedReader(new
         InputStreamReader(connection.getInputStream()));

    // get the text
    String line;
    StringBuffer text = new StringBuffer();
    while ((line = in.readLine()) != null)
      text.append(line);
    return text.toString();
  }

  /**
   * Edits a page by setting its text to the supplied value.
   *
   * @param text
   *           the text of the page
  * @param title
  *           the title of the page
  * @param summary
  *           the edit summary
  * @param minor
  *           whether the edit should be marked as minor
  * @throws IOException
  *            if something stuffs up the connection between here and wiki
  * @throws IllegalArgumentException
  *            if you try to edit a Special: page or a Media: page
  */
public void editPage(String title, String text, String summary, boolean minor)
     throws IOException {
  // pitfall check
  if (namespace(title) < 0)
    throw new IllegalArgumentException("Cannot edit Special: or Media: pages!");

  // sanitise
  title = URLEncoder.encode(title, "UTF-8");
  summary = URLEncoder.encode(summary, "UTF-8");
  text = URLEncoder.encode(text, "UTF-8");

  // what we need to do is get the edit page and fish out the wpEditToken,
  // wpAutoSummary
  // wpStartTime and wpEditTime values
  String URL = domain + "?title=" + title + "&action=edit";
  URLConnection connection = new URL(URL).openConnection();
  setCookies(connection);
  connection.connect();
  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, watchRetrieved = false;
  boolean watched = false;
  while ((line = in.readLine()) != null) {
    if (line.indexOf("name=\"wpAutoSummary\"") != -1) {
      int x = line.indexOf("value=\"") + 7;
      wpAutoSummary = line.substring(x, line.indexOf('\"', 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;
    } else if (editRetrieved && summaryRetrieved && startRetrieved &&
         timeRetrieved && watchRetrieved)
      break; // bandwidth hack
  }

  // this is what accepts the text
  URL = domain + "?title=" + title + "&action=submit";
  connection = new URL(URL).openConnection();
  setCookies(connection);
  connection.setDoOutput(true);
  PrintWriter out = new PrintWriter(connection.getOutputStream());

  // now we send the data

  out.print("wpTextbox1="); // ok
  out.print(text);
  out.print("&wpSummary="); // ok
  out.print(summary);
  if (minor)
    out.print("&wpMinoredit=1"); // ok
  if (watched)
    out.print("&wpWatchthis=1"); // ok
  out.print("&wpEdittime="); // ok
  out.print(wpEdittime);
  out.print("&wpEditToken="); // OK
  out.print(wpEditToken);
  out.print("&wpStarttime="); // ok
```

```
    out.print(wpStarttime);
    out.print("&wpAutoSummary="); // OK
    out.print(wpAutoSummary);
    // additions by thomas
    out.print("&wpSection=");
    out.print("");
    out.print("&wpScrolltop=");
    out.print("");
    // done, give the servers a rest
    out.close();
    try {
      Thread.sleep(2000);

      // it's somewhat strange that the edit only sticks when you start
      // reading the
      // response...
      in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
    } catch (IOException e) {
      if (!(connection instanceof HttpURLConnection))
        throw e;
      InputStream err = ((HttpURLConnection) connection).getInputStream();
      if (err == null)
        throw e;
      in = new BufferedReader(new InputStreamReader(err));
    } catch (InterruptedException e) {
      // nobody cares
    }
    in.readLine();

    // for debugging and/or todo purposes
    // String line;
    // while ((line = in.readLine()) != null)
    // {
    // System.out.println(line);
    // }
  }

  /**
   * Prepends something to the given page. A convenience method for adding
   * maintainance 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)
   * @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.
   *
   * @param name
   *            the name of the category (e.g. Candidates for speedy deletion, not
   *            Category:Candidates for speedy deletion)
   * @return a String[] containing page titles of members of the category
   * @throws IOException
   *             if something goes wrong
   * @since 0.02
   */
  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") + "&cpnamespace="
        + 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 MAIN_NAMESPACE;
  String namespace = title.substring(0, title.indexOf(':'));

  // 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);
    while (line.indexOf("<ns") != -1) {
      int x = line.indexOf("<ns 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('"')));
      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; // unintelligble 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("; ");
    }
    u.setRequestProperty("Cookie", cookie.toString());
  }
}
```

C.7 Trust Package

C.7.1 Reviewer.java

```
package trust;

import java.security.cert.Certificate;
import java.io.Serializable;
import rating.Rating;
import rating.RatingHistory;
import statictools.SecurityProvider;

public class Reviewer implements Serializable {

  private static final long serialVersionUID = -6356205467037013515L;

  // 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 posetime 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;

// Cachec certificate
private Certificate cachedCertificate;

/**
 * Constructor to generate an initial Reviewer with no prior history
 *
 * @param value
 * @param value2
 */
public Reviewer() {
  noOfPosInteractions = 0;
  noOfNegInteractions = 0;
  curve = Curve.OPT;
  state = State.TRUST;
  xValue = 0.0;
  nValue = 1.0;
  trustValue = 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) {
  rh.insertRating(r);
  xValue = rh.getXValue();

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

// represents the consequences of a positive experience - positive feedback
private Object positiveExp() {
  noOfPosInteractions++;
  // User in trust and an optimistic curve
  if (curve == Curve.OPT && state == State.TRUST) {
      nValue = nValue + 0.1;
      return null;
    }
    // user in trust and a cautious curve
    if (curve == Curve.CAU && state == State.TRUST) {
      nValue = nValue - 0.1;
      // If the curve changes from Cautious to neutral
      if (nValue == 1.0 && noOfPosInteractions == noOfNegInteractions) {
        curve = Curve.OPT;
      }
      return null;
    }
    // user in distrust and a optimistic curve
    if (curve == Curve.OPT && state == State.DISTRUST) {
      nValue = nValue + 0.1;
      return null;
    }
    // user in distrust and a cautious curve
    if (curve == Curve.CAU && state == State.DISTRUST) {
      nValue = nValue - 0.1;
      // If the curve changes from cautoius to neutral.
      if (nValue == 1.0 && noOfPosInteractions == noOfNegInteractions) {
        curve = Curve.OPT;
      }
      return null;
    }
    return null;
  }

  private void negativeExp() {
    noOfNegInteractions++;
    // user in trust and a cautious curve
    if (curve == Curve.CAU && state == State.TRUST) {
      nValue = nValue + 0.1;
    }
    // user in distrust and a cautious curve
    if (curve == Curve.CAU && state == State.DISTRUST) {
      nValue = nValue + 0.1;
    }
    // user in distrust and a optimistic curve
    if (curve == Curve.OPT && state == State.DISTRUST) {
      nValue = nValue - 0.1;
      if (nValue == 1.0 && noOfPosInteractions == noOfNegInteractions) {
        // nValue = nValue+0.1;
        curve = Curve.CAU;
      }

    }
    // User in trust and an optimistic curve
    if (curve == Curve.OPT && state == State.TRUST) {
      nValue = nValue - 0.1;
      if (nValue == 1.0 && noOfPosInteractions == noOfNegInteractions) {
        // nValue = nValue+0.1;
        curve = Curve.CAU;
      }
    }

  }

  // Calculates the actual trust value
  private void calcTrustValue() {
    // trust + opt. F4
    if (curve == Curve.OPT && state == State.TRUST) {
      trustValue = Math.pow((-1.0 * Math.abs(Math.pow(Math.abs(xValue - 1.0),
           nValue)) + 1.0), (1.0 / nValue));
    }
    // trust + cau. F2
    if (curve == Curve.CAU && state == State.TRUST) {
      trustValue = -1.0 * Math.pow(-1.0 * Math.abs(Math.pow(xValue, nValue)) + 1.0,
           (1.0 / nValue)) + 1.0;
    }
    // distrust + opt. F1
    if (curve == Curve.OPT && state == State.DISTRUST) {
      trustValue = Math.pow(-1.0 * Math.abs(Math.pow(Math.abs(xValue), nValue)) +
           1.0, (1.0 / nValue)) - 1.0;
    }
    // distrust + cau. F3
    if (curve == Curve.CAU && state == State.DISTRUST) {
      trustValue = -1.0 * Math.pow((-1.0 * Math.abs(Math.pow(Math.abs(xValue + 1.0),
            nValue)) + 1.0), (1.0 / nValue));
    }

  }

  // Returns the trustValue
  public double getTrustValue() {
    return trustValue;
  }

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

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

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

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

}
```

C.7.2 RoR.java

```
package trust;

import java.io.Serializable;
import java.security.cert.Certificate;
import java.util.HashMap;
import java.util.Iterator;

import rating.Rating;

public class RoR implements Serializable {
  private static final long serialVersionUID = -7149681994584389094L;

  // ArrayList<Reviewer> ror;
  HashMap<String, Reviewer> ror;

  /**
   * Constructor to initialization of Ring of Reviewers
   *
   */
  public RoR() {
    ror = new HashMap<String, Reviewer>();
  }

  // Inserts a new reviewer in to the RoR
  public void insertReviewer(Reviewer rv) {
    ror.put(rv.getUsername(), rv);
  }

  public void insertRatingToExistingUser(Rating r) {
    for (Iterator i = ror.keySet().iterator(); i.hasNext();) {
      // Finding the username in the ror that matches the rating
      if (((String) i.next()).equals(r.getUserName())) {
        // Insert the rating
        ror.get(r.getUserName()).insertRating(r);
      }
    }
  }

  // Return the trustvalue for a given user
  public double getTrustValueFromUsername(String username) {
    double trustValue = ror.get(username).getTrustValue();
    return trustValue;
  }

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

  // get a certificate from the cache
  public Certificate getCertificateFromUsername(String username) {
    return ror.get(username).getCertificate();
  }
  boolean hasUser(String username) {
    return ror.containsKey(username);
  }

}
```

C.7.3 TrustUpdater.java

```
package trust;

import java.util.ArrayList;

import rating.Rating;
import rating.SessionRatingDB;

public class TrustUpdater {

  public enum Exp {
    YES, NO
  };

  public enum Vote {
    R1, R2, R3, R4, R5, R6, R7, R8, R9
  };

  private Exp exp;

  private Vote vote;

  SessionRatingDB srDB;

  RoR ror;

  double ratingServed;

  ArrayList<Rating> ratingsToBeInserted;

  boolean clickedExp;

  boolean clickedInteraction;

  public TrustUpdater(SessionRatingDB sessionRatingDB, double ratingServed, RoR
      ror) {
    srDB = sessionRatingDB;
    this.ratingServed = ratingServed;
    this.ror = ror;
    ratingsToBeInserted = new ArrayList<Rating>();
    clickedExp = false;
    clickedInteraction = false;
  }

  public void ClickedYes() {
    /*
     * //Look through sessionRatingDB and se which ratings +1 to -1 of the
     * rating. //These should have their nValue adjusted Rating temp; for (int i
     * =0;i<srDB.size();i++){ temp = srDB.elementAt(i); int floored_rating
     * =(int) Math.floor(ratingServed); // If the given rating is with in 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 = Exp.YES;
    clickedExp = true;
    if (clickedExp && clickedInteraction) {
      updateAndInsertRatingsToROR();
    }
  }

  public void ClickedNo() {
    exp = Exp.NO;
    clickedExp = 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();
    }
  }

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

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

  private void updateAndInsertRatingsToROR() {
    Rating temp;
    int floored_rating = (int) Math.floor(ratingServed);
    for (int i = 0; i < srDB.size(); i++) {
      temp = srDB.elementAt(i);

      // Dealing with the expirence
      if (floored_rating - 1 == temp.getRating() || floored_rating ==
           temp.getRating()
          || floored_rating + 1 == temp.getRating()) {
        if (exp == Exp.YES) {
          srDB.elementAt(i).setExperience(1);
        }
        if (exp == Exp.NO) {
          srDB.elementAt(i).setExperience(0);
        }
      }

      // Dealing with the interactions R1
      if (vote == Vote.R1 && (temp.getRating() == 1 || temp.getRating() == 2)) {
```

```
    srDB.elementAt(i).setinteraction(1);
  }
  if (vote == Vote.R1 && (temp.getRating() > 2 && temp.getRating() < 5)) {
    srDB.elementAt(i).setinteraction(2);
  }
  if (vote == Vote.R1 && (temp.getRating() >= 5)) {
    srDB.elementAt(i).setinteraction(0);
  }

  // Dealing with the interactions R2
  if (vote == Vote.R2 && (temp.getRating() == 1 || temp.getRating() == 2 ||
      temp.getRating() == 3)) {
    srDB.elementAt(i).setinteraction(1);
  }
  if (vote == Vote.R2 && (temp.getRating() > 3 && temp.getRating() < 6)) {
    srDB.elementAt(i).setinteraction(2);
  }
  if (vote == Vote.R2 && (temp.getRating() >= 6)) {
    srDB.elementAt(i).setinteraction(0);
  }

  // Dealing with the interactions R3
  if (vote == Vote.R3 && (temp.getRating() == 2 || temp.getRating() == 3 ||
      temp.getRating() == 4)) {
    srDB.elementAt(i).setinteraction(1);
  }
  if (vote == Vote.R3 && (temp.getRating() == 1 || temp.getRating() == 5 ||
      temp.getRating() == 6)) {
    srDB.elementAt(i).setinteraction(2);
  }
  if (vote == Vote.R3 && (temp.getRating() >= 7)) {
    srDB.elementAt(i).setinteraction(0);
  }

  // Dealing with the interactions R4
  if (vote == Vote.R4 && (temp.getRating() == 3 || temp.getRating() == 4 ||
      temp.getRating() == 5)) {
    srDB.elementAt(i).setinteraction(1);
  }
  if (vote == Vote.R4
      && (temp.getRating() == 1 || temp.getRating() == 2 || temp.getRating() ==
          6 || temp.getRating() == 7)) {
    srDB.elementAt(i).setinteraction(2);
  }
  if (vote == Vote.R4 && (temp.getRating() >= 8)) {
    srDB.elementAt(i).setinteraction(0);
  }

  // Dealing with the interactions R5
  if (vote == Vote.R5 && (temp.getRating() == 4 || temp.getRating() == 5 ||
      temp.getRating() == 6)) {
    srDB.elementAt(i).setinteraction(1);
  }
  if (vote == Vote.R5
      && (temp.getRating() == 2 || temp.getRating() == 3 || temp.getRating() ==
          7 || temp.getRating() == 8)) {
    srDB.elementAt(i).setinteraction(2);
  }
  if (vote == Vote.R5 && (temp.getRating() == 1 || temp.getRating() == 9)) {
    srDB.elementAt(i).setinteraction(0);
  }

  // Dealing with the interactions R6
  if (vote == Vote.R6 && (temp.getRating() == 5 || temp.getRating() == 6 ||
      temp.getRating() == 7)) {
    srDB.elementAt(i).setinteraction(1);
  }
  if (vote == Vote.R6
      && (temp.getRating() == 4 || temp.getRating() == 3 || temp.getRating() ==
          9 || temp.getRating() == 8)) {
    srDB.elementAt(i).setinteraction(2);
  }
  if (vote == Vote.R6 && (temp.getRating() == 2 || temp.getRating() == 1)) {
    srDB.elementAt(i).setinteraction(0);
  }

  // Dealing with the interactions R7
  if (vote == Vote.R7 && (temp.getRating() == 6 || temp.getRating() == 7 ||
      temp.getRating() == 8)) {
    srDB.elementAt(i).setinteraction(1);
  }
  if (vote == Vote.R7 && (temp.getRating() == 4 || temp.getRating() == 5 ||
      temp.getRating() == 9)) {
    srDB.elementAt(i).setinteraction(2);
  }
  if (vote == Vote.R7 && (temp.getRating() <= 3)) {
    srDB.elementAt(i).setinteraction(0);
```

```
      }

      // Dealing with the interactions R8
      if (vote == Vote.R8 && (temp.getRating() == 7 || temp.getRating() == 8 ||
          temp.getRating() == 9)) {
        srDB.elementAt(i).setinteraction(1);
      }
      if (vote == Vote.R8 && (temp.getRating() == 5 || temp.getRating() == 6)) {
        srDB.elementAt(i).setinteraction(2);
      }
      if (vote == Vote.R8 && (temp.getRating() <= 4)) {
        srDB.elementAt(i).setinteraction(0);
      }

      // Dealing with the interactions R9
      if (vote == Vote.R9 && (temp.getRating() == 8 || temp.getRating() == 9)) {
        srDB.elementAt(i).setinteraction(1);
      }
      if (vote == Vote.R9 && (temp.getRating() == 7 || temp.getRating() == 6)) {
        srDB.elementAt(i).setinteraction(2);
      }
      if (vote == Vote.R9 && (temp.getRating() <= 5)) {
        srDB.elementAt(i).setinteraction(0);
      }
    }

    // Insert the updated ratings into the ror

    for (int j = 0; j < srDB.size(); j++) {
      // if the user already exista in the RoR
      if (ror.hasUser(srDB.elementAt(j).getUserName())) {
        ror.insertRatingToExistingUser(srDB.elementAt(j));
      }
      // If the person is a new entity to the RoR
      else {
        Reviewer newReviewer = new Reviewer();
        newReviewer.setUsername(srDB.elementAt(j).getUserName());
        ror.insertReviewer(newReviewer);
        ror.insertRatingToExistingUser(srDB.elementAt(j));
      }

    }

  }

}
```

C.8 Test Package

C.8.1 MasterAllTests.java

```
package test;

import junit.framework.Test;
import junit.framework.TestSuite;
import test.page.ExtractRatingsTest;
import test.page.PageExtractorTest;
import test.page.PageModifierTest;
import test.rating.RatingCalculatorTest;
import test.rating.RatingHistoryTest;
import test.rating.RatingTest;
import test.rating.SessionRatingDBTest;
import test.statictools.RatingCleanOutTest;
import test.statictools.SecurityProviderTest;
import test.statictools.SerializerTest;
import test.statictools.ThresholdTest;
import test.trust.ReviewerTest;
import test.trust.RoRTest;
import test.trust.TrustUpdaterTest;

public class MasterAllTests {

  public static Test suite() {
    TestSuite suite = new TestSuite("Test for test");
    suite.addTestSuite(PageModifierTest.class);
    suite.addTestSuite(PageExtractorTest.class);
    suite.addTestSuite(ExtractRatingsTest.class);
    suite.addTestSuite(RatingTest.class);
    suite.addTestSuite(RatingHistoryTest.class);
    suite.addTestSuite(SessionRatingDBTest.class);
    suite.addTestSuite(RatingCalculatorTest.class);

    suite.addTestSuite(SerializerTest.class);
    suite.addTestSuite(RatingCleanOutTest.class);
    suite.addTestSuite(SecurityProviderTest.class);
    suite.addTestSuite(ThresholdTest.class);



    suite.addTestSuite(ReviewerTest.class);
    suite.addTestSuite(RoRTest.class);
    suite.addTestSuite(TrustUpdaterTest.class);
    return suite;
  }

}
```

C.9 Test.Page package

C.9.1 ExtractRatingsTest.java

```
package test.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 junit.framework.TestCase;

import org.junit.Test;

import page.ExtractRatings;
import scone.util.tokenstream.HtmlTokenizer;
import scone.util.tokenstream.TokenInputStream;
import scone.util.tokenstream.TokenInputStreamTokenizerImpl;
import statictools.Serializer;
import trust.RoR;

public class ExtractRatingsTest extends TestCase {

  @Test
  public void testExtractRatings() {

    try {
      String editUrl =
           "http://en.wikipedia.org/w/index.php?title=Bass_Strait&action=edit";

      URL url = new
           URL("http://en.wikipedia.org/w/index.php?title=KGYM&action=edit");
      URLConnection urlconnection = url.openConnection();
      urlconnection.setUseCaches(false);
      InputStream is = urlconnection.getInputStream();
      InputStreamReader isr = new InputStreamReader(is);
      HtmlTokenizer htmlTokenizer = new HtmlTokenizer();
      htmlTokenizer.assign(isr);
      TokenInputStream tis = new TokenInputStreamTokenizerImpl(isr);

      RoR ror = Serializer.readRoRFromDisk();
      ExtractRatings exr = new ExtractRatings(editUrl, ror);
      Vector test = exr.getRawRatings();
      exr.getRawRatings();
      assertTrue(true);
      // Token test = htmlTokenizer.nextToken();
      // System.out.println();
    } catch (MalformedURLException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }
}
```

C.9.2 PageExtractorTest.java

```
package test.page;

import static org.junit.Assert.fail;

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 junit.framework.TestCase;

import org.junit.Test;

import page.PageExtractor;
import scone.util.tokenstream.TokenInputStream;
import scone.util.tokenstream.TokenInputStreamTokenizerImpl;

public class PageExtractorTest extends TestCase {

  @Test
  public void testPageExtractor() {
    try {
      URL url = new URL("http://en.wikipedia.org/wiki/Great_Ocean_Road");
      URLConnection urlconnection = url.openConnection();
      urlconnection.setUseCaches(false);
      InputStream is = urlconnection.getInputStream();
      InputStreamReader isr = new InputStreamReader(is);
      // HtmlTokenizer htmlTokenizer = new HtmlTokenizer();
      // htmlTokenizer.assign(isr);
      TokenInputStream tis = new TokenInputStreamTokenizerImpl(isr);
      PageExtractor pex = new PageExtractor(tis, url.toString());
      assertEquals(pex.extractEditPage(),
           "http://en.wikipedia.org/w/index.php?title=Great_Ocean_Road&action=edit");
      assertEquals(pex.extractHistoryPage(),
          "http://en.wikipedia.org/w/index.php?title=Great_Ocean_Road&action=history");
      // Token test = htmlTokenizer.nextToken();
      // System.out.println(test.toString());
    } catch (MalformedURLException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

  }

}
```

C.9.3 PageModifierTest.java

```
package test.page;

import java.io.IOException;
import java.util.Vector;

import junit.framework.TestCase;

import org.junit.Test;

import page.PageModifier;
import scone.util.tokenstream.Token;
import scone.util.tokenstream.TokenInputStream;
import statictools.TokenInputStreamTools;

public class PageModifierTest extends TestCase {

  @Test
  public void testInsertYUIandRating() {
    TokenInputStream tis =
         TokenInputStreamTools.CreateTokenInputStreamFromURL("http://example.com/");
    Vector<Token> htmlPage = new Vector<Token>();
    Token t = null;
    try {
      while ((t = tis.read()) != null) {
        htmlPage.add(t);
      }

    } catch (IOException e) {
      System.out.println("Initialization of Token Vector failed");
      e.getStackTrace();
    }
    PageModifier pm = new PageModifier(htmlPage);
    assertEquals(htmlPage.size(), 54);
    pm.insertYUIandRating(3.0);
    htmlPage = pm.getHtmlPageVector();
    assertEquals(htmlPage.size(), 56);

  }

}
```

C.10 Test.Rating package

C.10.1 RatingCalculatorTest.java

```
package test.rating;

import junit.framework.TestCase;

import org.junit.Test;

import rating.Rating;
import rating.RatingCalculator;
import rating.SessionRatingDB;
import trust.Reviewer;
import trust.RoR;

public class RatingCalculatorTest extends TestCase {

  @Test
  public void testComputeAverage() {
    Reviewer reviewer1 = new Reviewer(0.765338, "user1");
    Reviewer reviewer2 = new Reviewer(0.35338, "user2");
    Reviewer reviewer3 = new Reviewer(0.165338, "user3");
    Reviewer reviewer4 = new Reviewer(-0.165338, "user4");
    RoR ror = new RoR();
    ror.insertReviewer(reviewer1);
    ror.insertReviewer(reviewer2);
    ror.insertReviewer(reviewer3);
    ror.insertReviewer(reviewer4);
    SessionRatingDB sessionRatingDB = new SessionRatingDB();
    Rating r0 = new Rating("user1", 5, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait");
    Rating r1 = new Rating("user2", 3, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait");
    Rating r2 = new Rating("user3", 7, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait");
    Rating r3 = new Rating("user4", 2, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait");

    sessionRatingDB.push(r0);
    sessionRatingDB.push(r1);
    sessionRatingDB.push(r2);
    sessionRatingDB.push(r3);

    RatingCalculator rc = new RatingCalculator(sessionRatingDB, ror);
    assertEquals(rc.computeAverage(),4.6923076923076925);
  }

}
```

C.10.2 RatingHistoryTest.java

```
package test.rating;

import static org.junit.Assert.assertTrue;
import junit.framework.TestCase;

import org.junit.Test;

import rating.Rating;
import rating.RatingHistory;

//TODO: HØJ Prioritet: Denne test skal laves, så den er ordelig!

public class RatingHistoryTest extends TestCase {

  @Test
  public void testRatingHistory() {
    RatingHistory rh = new RatingHistory();
    assertTrue(rh.getHistorySize() == 0);
    assertTrue(rh.getXValue() == 0.0);
  }

  @Test
  public void testInsertRating() {
    RatingHistory rh = new RatingHistory();
    String hashValueTemp = "0123456789abcdef";
    int rating = 5;
    int experience = 0;
    Rating r3 = new Rating();
    Rating r = new Rating("Korsgaard", 5, "123123421",
         "http://en.wikipedia.org/wiki/Bass_Strait");
    Rating r2 = new Rating("Korsgaard", 5, "123123421",
         "http://en.wikipedia.org/wiki/Bass_Strait");
    rh.insertRating(r);
    assertTrue(rh.getHistorySize() == 1);
    rh.insertRating(r2);
    assertTrue(rh.getHistorySize() == 2);
  }

  @Test
  public void testGetXValue() {
    RatingHistory rh = new RatingHistory();
    String hashValueTemp = "0123456789abcdef";
    int rating = 5;
    int interaction = 0;
    Rating r = new Rating("Korsgaard", 5, "123123421",
         "http://en.wikipedia.org/wiki/Bass_Strait");
    Rating r2 = new Rating("Korsgaard", 5, "123123421",
         "http://en.wikipedia.org/wiki/Bass_Strait");
    r.setinteraction(interaction);
    r2.setinteraction(interaction);
    rh.insertRating(r);
    assertTrue(rh.getXValue() == -0.1);
    rh.insertRating(r2);
    assertTrue(rh.getXValue() == -0.2);

  }

}
```

C.10.3 RatingTest.java

```
package test.rating;

import static org.junit.Assert.assertTrue;

import java.util.Date;

import junit.framework.TestCase;

import org.junit.Test;

import rating.Rating;

public class RatingTest extends TestCase {

  @Test
  public void testRating() {
    Rating r = new Rating();
    // assert that the default experience is positive (1)
    assertTrue(r.getExperience() == 1);
    // assert date is not older than one minute
    Date temp = new Date();
    assertTrue((temp.getTime() - r.getDate()) < 1000 * 60);
    // Assert that rating is correct
    assertTrue(r.getRating() == 0);
  }

  @Test
  public void testRating2() {
    String hashValueTemp = "0123456789abcdef";
    int rating = 5;
    int experience = 0;
    Rating r = new Rating("Korsgaard", 5, "123123421",
         "http://en.wikipedia.org/wiki/Bass_Strait");
    // assert that the experience is correct
    assertTrue(r.getExperience() == 2);
    // Assert that the username i correct
    assertTrue(r.getUserName().equals("Korsgaard"));
    // Assert that teh version is correct
    assertTrue(r.getVersion().equals("123123421"));
    // assert that rating is correct
    assertTrue(r.getRating() == 5);
    // assert that the the URL are correct
    assertTrue(r.getURL().equals("http://en.wikipedia.org/wiki/Bass_Strait"));
    // assert date is not older than one minute
    Date temp = new Date();
    assertTrue((temp.getTime() - r.getDate()) < 1000 * 60);
  }

  public void testSetExp() {
    Rating r = new Rating("Korsgaard", 5, "123123421",
         "http://en.wikipedia.org/wiki/Bass_Strait");
    r.setExperience(1);
    assertTrue(r.getExperience() == 1);
  }

  public void testSetInteraction() {
    Rating r = new Rating("Korsgaard", 5, "123123421",
         "http://en.wikipedia.org/wiki/Bass_Strait");
    r.setinteraction(1);
    assertTrue(r.getInteraction() == 1);
  }
}
```

C.10.4 SessionRatingDBTest.java

```
package test.rating;

import junit.framework.TestCase;

import org.junit.Test;

import rating.Rating;
import rating.SessionRatingDB;

public class SessionRatingDBTest extends TestCase {

  @Test
  public void testSessionRatingDB() {
    try {
      SessionRatingDB sr = new SessionRatingDB();
      Rating r = sr.pop();
    } catch (RuntimeException e) {
      assertTrue(e instanceof java.lang.IndexOutOfBoundsException);
      System.out.println();
    }
  }

  @Test
  public void testPush() {
    SessionRatingDB sr = new SessionRatingDB();
    Rating r = new Rating();
    sr.push(r);
    assertEquals(sr.size(), 1);
  }

  @Test
  public void testPop() {
    SessionRatingDB sr = new SessionRatingDB();
    Rating r = new Rating();
    sr.push(r);
    Rating r2 = sr.pop();
    assertEquals(r, r2);
  }

  @Test
  public void testSize() {
    SessionRatingDB sr = new SessionRatingDB();
    Rating r = new Rating();
    sr.push(r);
    assertEquals(sr.size(), 1);
  }

}
```

C.11 Test.Statictools Package

C.11.1 RatingCleanOutTest.java

```
package test.statictools;

import java.util.Vector;

import junit.framework.TestCase;

import org.junit.Test;

import statictools.RatingCleanOut;
import statictools.Serializer;
import trust.RoR;

public class RatingCleanOutTest extends TestCase {

  @Test
  public void testRemoveRatingsTitleMismatch() {
    String r1 = "<!-- WikiTrustComment. Read more on: "+
    "http://en.wikipedia.org/wiki/User:Korsgaard\n ;Korsgaard;"+
    "7;134276108;Bass_Strait;MC0CFFY5w/j3iZveBiBJx9GXObV0Mo7RA"+
    "hUAlwlDWMFNjqrKDp4kTYwVqrilK8w= -->";
    String r4 = "<!-- WikiTrustComment. Read more on: "+
    "http://en.wikipedia.org/wiki/User:Korsgaard\n ;Korsgaard;"+
    "7;134276108;Great_Ocean_Road;MC0CFFY5w/j3iZveBiBJx9GXObV0"+
    "Mo7RAhUAlwlDWMFNjqrKDp4kTYwVqrilK8w= -->";
    String r7 = "<!-- WikiTrustComment. Read more on: "
      +"http://en.wikipedia.org/wiki/User:Korsgaard\n;Korsgaard;"+
      "7;134276108;Bass_Strait;MC0CFFY5w/j3iZvgfiBJx9GXObV0Mo7RA"+
      "hUAlwlDWMFNjqrKDp4kTYwVqrilK8w= -->";
    Vector<String> rawRatings = new Vector<String>();
    rawRatings.add(r1);
    rawRatings.add(r4);
    rawRatings.add(r7);
    rawRatings = RatingCleanOut.RemoveRatingsTitleMismatch("Bass_Strait",
         rawRatings);
    assertTrue(rawRatings.size() == 2);

  }

  @Test
  public void testRemoveRatingsBelowThreshold() {
    String r2 = "<!-- WikiTrustComment. Read more on: "+
    "http://en.wikipedia.org/wiki/User:Korsgaard\n;Korsgaard;"+
    "7;134276108;Bass_Strait;MC0CFFY5w/j3iZveBiBJx9GXObV0Mo7"+
    "RAhUAlwlDWMFNjqrKDp4kTYwVqrilK8w= -->";
    String r5 = "<!-- WikiTrustComment. Read more on: "+
    "http://en.wikipedia.org/wiki/User:Korsgaard\n;Korsgaard;"+
    "7;134276108;Fort_Knox;MC0CFFY5w/j3iZveBiBJx9GXObV0Mo7RAh"+
    "UAlwlDWMFNjqrKDp4kTYwVqrilK8w= -->";
    String r8 = "<!-- WikiTrustComment. Read more on: "+
    "http://en.wikipedia.org/wiki/User:Korsgaard\n;Korsgaard;"+
    "7;134276108;Bass_Strait;MC0CFFY5w/j3iZveBiBJx9GgfbV0Mo7RA"+
    "hUAlwlDWMFNjqrKDp4kTYwVqrilK8w= -->";
    Vector<String> rawRatings = new Vector<String>();
    rawRatings.add(r2);
    rawRatings.add(r5);
    rawRatings.add(r8);
    rawRatings = RatingCleanOut.RemoveRatingsBelowThreshold(0.1, rawRatings,
         "Bass_Strait");
    assertTrue(rawRatings.size() == 3);
  }

  @Test
  public void testRemoveUnvalidableRatings() {
    String r3 = "<!-- WikiTrustComment. Read more on: "+
    "http://en.wikipedia.org/wiki/User:Korsgaard\n;Korsgaard;"+
    "7;134276108;Bass_Strait;MCwCFGYeyYSOlqOrQBDnKA3fFo3gmfds"+
    "AhQ+MlL183hnf/LrpEaR56U7YsPUMg== -->";
    String r6 = "<!-- WikiTrustComment. Read more on: "+
    "http://en.wikipedia.org/wiki/User:Korsgaard\n;Korsgaard;"+
    "7;134276108;War;MCwCFGYeyYSOlqOrQBDnKA3fFo3gmfdsAhQ+MlL1"+
    "83hnf/LrpEaR56U7YsPUMg== -->";
    String r9 = "<!-- WikiTrustComment. Read more on: "+
    "http://en.wikipedia.org/wiki/User:Korsgaard\n;Korsgaard;"+
    "7;134276108;Bass_Strait;MCwCFGYeyYSOlqOrQBDnKA3fFo3gmfds"+
    "AhQ+MlL183hnf/LrpEaR56U7YsPUMg== -->";
    Vector<String> rawRatings = new Vector<String>();
    rawRatings.add(r3);
    rawRatings.add(r6);
    rawRatings.add(r9);
    RoR ror = Serializer.readRoRFromDisk();
    rawRatings = RatingCleanOut.RemoveUnvalidableRatings(rawRatings, ror);
    assertTrue(rawRatings.size() == 2);
  }

}
```

C.11.2 SecurityProviderTest.java

```
package test.statictools;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.security.KeyPair;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;

import junit.framework.TestCase;

import org.junit.Test;

import statictools.SecurityProvider;
import statictools.Serializer;
import sun.misc.BASE64Encoder;
import trust.RoR;

public class SecurityProviderTest extends TestCase {

  @Test
  public void testInitKeyStore() {
    try {
      KeyStore ks = SecurityProvider.InitKeyStore(".keys", "12345678");
      assertTrue(ks.containsAlias("wiki"));
      assertEquals(ks.size(), 1);
    } catch (KeyStoreException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }

  @Test
  public void testInitCertificate() {
    try {
      KeyStore ks = SecurityProvider.InitKeyStore(".keys", "12345678");
      Certificate cert = ks.getCertificate("wiki");
      FileInputStream is = new FileInputStream("THOMAS.cer");
      CertificateFactory cf = CertificateFactory.getInstance("X.509");
      Certificate certExpected = cf.generateCertificate(is);
      assertTrue(cert.toString().equals(certExpected.toString()));
    } catch (KeyStoreException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    } catch (FileNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    } catch (CertificateException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

  }

  @Test
  public void testInitKeyPair() {
    BASE64Encoder b64 = new BASE64Encoder();
    KeyStore ks = SecurityProvider.InitKeyStore(".keys", "12345678");
    KeyPair keyPair = SecurityProvider.InitKeyPair(ks, "wiki", "12345678");
    PrivateKey privatekey = keyPair.getPrivate();
    PublicKey publicKey = keyPair.getPublic();
    String encodedPublicKey = b64.encode(publicKey.getEncoded());
    String encodedPrivateKey = b64.encode(privatekey.getEncoded());
    String expectedPublicKey =
          "MIIBtzCCASwGByqGSM44BAEwggEfAoGBAP1/U4EddRIpUt9KnC7"+
      "s5Of2EbdSPO9EAMMeP4C2USZpRV1AIlH7WT2NWPq/xfW6MPbLm1Vs14E7gB00b/JmYLdrmVClpJ+"+
      "f6AR7ECLCT7up1/63xhv4O1fnxqimFQ8E+4P208UewwI1VBNaFpEy9nXzrith1yrv8iIDGZ3RSAH"+
      "HAhUAl2BQjxUjC8yykrmCouuEC/BYHPUCgYEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0"+
      "HgmdRWVeOutRZT+ZxBxCBgLRJFnEj6EwoFhO3zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRbqN/C/o"+
      "hNWLx+2J6ASQ7zKTxvqhRkImog9/hWuWfBpKLZl6Ae1UlZAFMO/7PSSoDgYQAAoGAX+qZML6+8oZ"+
      "aJ8oCUo8qyRrFapcfsByVdshI3MaGFPVR1xqhm1QSU+XSju9wkBWVpmXWH6bHBA/XCAYzWe5j+gM"+
      "GlpfKXopNZaH/c8xG4bQ8mOzTx8uUV7etyaao/tF5dgee1yzGjP4WNYO+/fePhLIYWElYQ66tA6R"+
      "x+vGCmds=";
    String expectedPricateKey =
          "MIIBSwIBADCCASwGByqGSM44BAEwggEfAoGBAP1/U4EddRIpUt"+
    "9KnC7s5Of2EbdSPO9EAMMeP4C2USZpRV1AIlH7WT2NWPq/xfW6MPbLm1Vs14E7gB00b/JmYLdrmVCl"+
    "pJ+f6AR7ECLCT7up1/63xhv4O1fnxqimFQ8E+4P208UewwI1VBNaFpEy9nXzrith1yrv8iIDGZ3RSA"+
    "HHAhUAl2BQjxUjC8yykrmCouuEC/BYHPUCgYEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0H"+
    "gmdRWVeOutRZT+ZxBxCBgLRJFnEj6EwoFhO3zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRbqN/C/ohNW"+
    "Lx+2J6ASQ7zKTxvqhRkImog9/hWuWfBpKLZl6Ae1UlZAFMO/7PSSoEFgIUM4IMLmRuc39RKoEcs9Ds"+
    "A18Uj8Y=";
    assertEquals(encodedPublicKey.substring(0, 20), expectedPublicKey.substring(0,
          20));
    assertEquals(encodedPrivateKey.substring(0, 20), encodedPrivateKey.substring(0,
          20));

  }

  @Test
  public void testCreateSignatureAndVerify() {
    KeyStore ks = SecurityProvider.InitKeyStore(".keys", "12345678");
    KeyPair keyPair = SecurityProvider.InitKeyPair(ks, "wiki", "12345678");
    String testSignatureString = "This is a test string";
    byte[] testSignature = SecurityProvider.createSignature(testSignatureString,
          keyPair.getPrivate());
    boolean testSignatureVerified =
          SecurityProvider.verifySignature(testSignatureString, testSignature,
          keyPair
        .getPublic());
    assertTrue(testSignatureVerified);
  }

  @Test
  public void testSignatureEncodingAndDecoding() {
    KeyStore ks = SecurityProvider.InitKeyStore(".keys", "12345678");
    KeyPair keyPair = SecurityProvider.InitKeyPair(ks, "wiki", "12345678");
    String testSignatureString = "This is a test string";
    byte[] testSignature = SecurityProvider.createSignature(testSignatureString,
          keyPair.getPrivate());
    String encodedSignature =
          SecurityProvider.createBase64EncodingFromSignature(testSignature);
    byte[] decodedSignature =
          SecurityProvider.getSignatureFromBase64Representation(encodedSignature);
    boolean testSignatureVerified =
          SecurityProvider.verifySignature(testSignatureString, decodedSignature,
          keyPair
        .getPublic());
    assertTrue(testSignatureVerified);
  }

  @Test
  public void testGenerateCertificate() {
    try {
      KeyStore ks = SecurityProvider.InitKeyStore(".keys", "12345678");
      Certificate cert = ks.getCertificate("wiki");
      String certB64 = SecurityProvider.generateCertificate(cert);
      File f = new File("THOMAS.cer");
      FileInputStream fis = new FileInputStream(f);
      BufferedReader in = new BufferedReader(new InputStreamReader(fis));
      String temp = "";
      String[] brokenCert = certB64.split("\n");
      int i = 1;
      in.readLine();
      while ((temp = in.readLine()) != null) {
        if (brokenCert[i].length() > 20)
          assertEquals(brokenCert[i].substring(0, 20), temp.substring(0, 20));
        i++;
      }

      /*
       * File f2 = new File("THOMAS - new certificate.cer"); FileOutputStream
       * fos = new FileOutputStream(f2); fos.write(certB64.getBytes());
       * fos.close(); assertTrue(f.equals(f2));
       */
    } catch (KeyStoreException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    } catch (FileNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

  }

  @Test
  /**
   * public void testCreateRating() { KeyStore ks =
   * SecurityProvider.InitKeyStore(".keys", "12345678"); PrivateKey privateKey =
   * SecurityProvider.InitKeyPair(ks, "wiki", "12345678").getPrivate(); String
   * createdRating = SecurityProvider.createRating("Korsgaard", 7, "94723853",
   * "Great_Ocean_Road", privateKey); String expectedRating = "<!--
   * WikiTrustComment. Read more on: http://en.wikipedia.org/wiki/User:Korsgaard
   * \n;korsgaard;7;94723853;Great_Ocean_Road;MCwCFDXX/OhLyKTlha8pGUiMHfmQd1RpAh
   * RS1MSAueywdbt0nl/Y+o7+DSEuxw==-->";
   * assertEquals(expectedRating, createdRating); }
   */
  public void testCreateRatingAndVerification() {
    RoR ror = Serializer.readRoRFromDisk();
    KeyStore ks = SecurityProvider.InitKeyStore(".keys", "12345678");
    PrivateKey privateKey = SecurityProvider.InitKeyPair(ks, "wiki",
        "12345678").getPrivate();
    String createdRating = SecurityProvider.createRating("Korsgaard", 7, "94723853",
          "Great_Ocean_Road", privateKey);
    assertTrue(SecurityProvider.validateRating(createdRating, ror));
  }

  public void testGetUserCertificate() {
    Certificate cert = SecurityProvider.getUserCertificate("Susansolovan");
    KeyStore susan = SecurityProvider.InitKeyStore(".testperson3", "12345678");
    Certificate cert_expected = SecurityProvider.InitCertificate(susan,
        "testperson3");
    assertEquals(cert, cert_expected);

  }

  public void testGenerateRatings() {
    KeyStore ks = SecurityProvider.InitKeyStore(".keys", "12345678");
    PrivateKey privateKey = SecurityProvider.InitKeyPair(ks, "wiki",
        "12345678").getPrivate();
    String createdRating = SecurityProvider.createRating("Korsgaard", 7,
        "134276108", "Bass_Strait", privateKey);
    // Uncomment this to create ratings
    // System.out.println(createdRating);
    assertTrue(true);

  }

  public void testGetWikiUserPassFromDisk() {
    String[] s1 = SecurityProvider.getWikiUserPassFromDisk();
    assertEquals(s1[0], "Susansolovan");
    assertEquals(s1[1], "12345678");
  }

  public void testGetKeyStorePassFromDisk() {
    String s1 = SecurityProvider.getKeyStorePassFromDisk();
    assertEquals(s1, "12345678");
  }
}
```

C.11.3 SerializerTest.java

```
package test.statictools;

import static org.junit.Assert.assertEquals;
import junit.framework.TestCase;

import org.junit.Test;

import statictools.Serializer;
import trust.Reviewer;
import trust.RoR;

public class SerializerTest extends TestCase {

  @Test
  public void testWriteAndReadRoRToDisk() {
    RoR ror = new RoR();
    // Insert Korsgaard
    Reviewer korsgaard = new Reviewer(0.45677, "Korsgaard");
    ror.insertReviewer(korsgaard);
    // Insert MarkusWennerberg
    Reviewer markus = new Reviewer(0.3122, "Markuswennerberg");
    ror.insertReviewer(markus);
    // Insert Mikael Martin
    Reviewer mikael = new Reviewer(0.86346, "Mikaelmartin");
    ror.insertReviewer(mikael);
    // Insert Susan Solovan
    Reviewer susan = new Reviewer(0.11223, "Susansolovan");
    ror.insertReviewer(susan);
    ror.updateCertificates();
    statictools.Serializer.writeRoRToDisk(ror);
    RoR newRoR = Serializer.readRoRFromDisk();

    assertEquals(newRoR.getTrustValueFromUsername("Korsgaard"), 0.45677);
    assertEquals(newRoR.getTrustValueFromUsername("Markuswennerberg"), 0.3122);
    assertEquals(newRoR.getTrustValueFromUsername("Mikaelmartin"), 0.86346);
    assertEquals(newRoR.getTrustValueFromUsername("Susansolovan"), 0.11223);

  }

}
```

C.11.4 ThresholdTest.java

```
package test.statictools;

import static org.junit.Assert.assertEquals;
import junit.framework.TestCase;

import org.junit.Test;

import statictools.Threshold;

public class ThresholdTest extends TestCase{

  @Test
  public void testThresholdCalculator() {
    String oldid = "10213349";
    String title = "Bass_Strait";
    double threshold = Threshold.ThresholdCalculator(title, oldid);
    //System.out.println("Threshold: " + threshold);
    assertEquals(threshold, 0.15531062124248496);
  }

}
```

C.12 Test.Trust Package

C.12.1 ReviewerTest.java

```
package test.trust;

import java.security.cert.Certificate;
import java.util.Date;
import junit.framework.TestCase;

import org.junit.Test;

import rating.Rating;
import trust.Reviewer;

public class ReviewerTest extends TestCase {

  @Test
  public void testReviewer() {
    Reviewer reviewer = new Reviewer();
    assertTrue(reviewer.getTrustValue() == 0.0);
  }

  @Test
  public void testInsertRating() {
    // Start test of optimistic curve in trust
    Reviewer reviewer = new Reviewer();
    String hashValueTemp = "0123456789abcdef";
    int rating = 5;
    int experience = 1;

    Rating r = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 1);
    Rating r2 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 1);
    r.setinteraction(1);
    r2.setinteraction(1);
    reviewer.insertRating(r);
    reviewer.insertRating(r2);
    assertEquals(reviewer.getTrustValue(), 0.2990636154095728);

    // test of cautious curve in trust
    // Create 2 ratings that are 3 months old
    long now = (new Date()).getTime();
    long three_months_old = now - new Long("7776000000");// (3*30*24*60*60*1000);
    Date date_3_months_old = new Date(three_months_old);
    System.out.println(date_3_months_old.toString());
    Rating r3 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", date_3_months_old,
        1);
    Rating r4 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", date_3_months_old,
        1);
    r3.setinteraction(1);
    r4.setinteraction(1);
    reviewer.insertRating(r3);
    reviewer.insertRating(r4);
    assertEquals(reviewer.getTrustValue(), 0.5132596571802928);
    long nine_months_old = now - new Long("23328000000");// (9*30*24*60*60*1000);
    Date date_9_months_old = new Date(nine_months_old);
    Rating r5 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", date_9_months_old,
        0);
    Rating r6 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", date_9_months_old,
        0);
    Rating r7 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", date_9_months_old,
        0);
    Rating r8 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", date_9_months_old,
        0);
    Rating r9 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", date_9_months_old,
        0);

    reviewer.insertRating(r5);
    reviewer.insertRating(r6);
    reviewer.insertRating(r7);
    reviewer.insertRating(r8);
    reviewer.insertRating(r9);
    assertEquals(reviewer.getTrustValue(), 0.13458872206570338);

    // test of cautious curve in distrust
    Rating r10 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 0);
    Rating r11 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 0);
    Rating r12 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 0);
    Rating r13 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 0);
    r10.setinteraction(0);
    r11.setinteraction(0);
```

```
r12.setinteraction(0);
r13.setinteraction(0);
reviewer.insertRating(r10);
reviewer.insertRating(r11);
reviewer.insertRating(r12);
reviewer.insertRating(r13);
assertEquals(reviewer.getTrustValue(), -0.4656334193014535);

// test of move from cautious curve

Rating r14 = new Rating("Korsgaard", rating, "1234",
      "http://en.wikipedia.org/wiki/Bass_Strait", date_9_months_old,
     1);
Rating r15 = new Rating("Korsgaard", rating, "1234",
      "http://en.wikipedia.org/wiki/Bass_Strait", date_9_months_old,
     1);
Rating r16 = new Rating("Korsgaard", rating, "1234",
      "http://en.wikipedia.org/wiki/Bass_Strait", date_9_months_old,
     1);
Rating r17 = new Rating("Korsgaard", rating, "1234",
      "http://en.wikipedia.org/wiki/Bass_Strait", date_9_months_old,
     1);
Rating r18 = new Rating("Korsgaard", rating, "1234",
      "http://en.wikipedia.org/wiki/Bass_Strait", date_9_months_old,
     1);
Rating r19 = new Rating("Korsgaard", rating, "1234",
      "http://en.wikipedia.org/wiki/Bass_Strait", date_9_months_old,
     1);

r14.setinteraction(1);
r15.setinteraction(1);
r16.setinteraction(1);
r17.setinteraction(1);
r18.setinteraction(1);
r19.setinteraction(1);
reviewer.insertRating(r14);
reviewer.insertRating(r15);
reviewer.insertRating(r16);
reviewer.insertRating(r17);
reviewer.insertRating(r18);
reviewer.insertRating(r19);
assertEquals(reviewer.getTrustValue(), -0.052764355442757704);

// test move from optimistic to cautoius curve in distrust
Rating r20 = new Rating("Korsgaard", rating, "1234",
      "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 0);
Rating r21 = new Rating("Korsgaard", rating, "1234",
      "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 0);
Rating r22 = new Rating("Korsgaard", rating, "1234",
      "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 0);

r20.setinteraction(0);
r21.setinteraction(0);
r22.setinteraction(0);
reviewer.insertRating(r20);
reviewer.insertRating(r21);
reviewer.insertRating(r22);
assertEquals(reviewer.getTrustValue(), -0.4959747813973638);
// test move from distrust to trust
Rating r23 = new Rating("Korsgaard", rating, "1234",
      "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 1);
Rating r24 = new Rating("Korsgaard", rating, "1234",
      "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 1);
Rating r25 = new Rating("Korsgaard", rating, "1234",
      "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 1);
Rating r26 = new Rating("Korsgaard", rating, "1234",
      "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 1);
Rating r27 = new Rating("Korsgaard", rating, "1234",
      "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 1);
Rating r28 = new Rating("Korsgaard", rating, "1234",
      "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 1);
Rating r29 = new Rating("Korsgaard", rating, "1234",
      "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 1);

r23.setinteraction(1);
r24.setinteraction(1);
r25.setinteraction(1);
r26.setinteraction(1);
r27.setinteraction(1);
r28.setinteraction(1);
r29.setinteraction(1);
reviewer.insertRating(r23);
reviewer.insertRating(r24);
reviewer.insertRating(r25);
reviewer.insertRating(r26);
reviewer.insertRating(r27);
reviewer.insertRating(r28);
```

```
    reviewer.insertRating(r29);
    Rating r30 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", date_9_months_old,
        0);
    Rating r31 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", date_9_months_old,
        0);
    Rating r32 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", date_9_months_old,
        0);
    Rating r33 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", date_9_months_old,
        0);
    Rating r34 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", date_9_months_old,
        0);
    Rating r35 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", date_9_months_old,
        0);

    r30.setinteraction(0);
    r31.setinteraction(0);
    r32.setinteraction(0);
    r33.setinteraction(0);
    r34.setinteraction(0);
    r35.setinteraction(0);
    reviewer.insertRating(r30);
    reviewer.insertRating(r31);
    reviewer.insertRating(r32);
    reviewer.insertRating(r33);
    reviewer.insertRating(r34);
    reviewer.insertRating(r35);
    Rating r36 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 1);
    r36.setinteraction(1);
    reviewer.insertRating(r36);

    assertEquals(reviewer.getTrustValue(), 0.2749999999999999);
    // test move from cautoius curve to optimistic curve in trust

  }

  @Test
  public void testGetTrustValue() {
    // Start test of optimistic curve in trust
    Reviewer reviewer = new Reviewer();
    String hashValueTemp = "0123456789abcdef";
    int rating = 5;
    int experience = 1;
    Rating r = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 1);
    Rating r2 = new Rating("Korsgaard", rating, "1234",
         "http://en.wikipedia.org/wiki/Bass_Strait", new Date(), 1);
    r.setinteraction(1);
    r2.setinteraction(1);
    reviewer.insertRating(r);
    reviewer.insertRating(r2);
    assertEquals(reviewer.getTrustValue(), 0.2990636154095728);
  }

}
```

C.12.2 RoRTest.java

```
package test.trust;

import java.security.cert.Certificate;

import junit.framework.TestCase;

import org.junit.Test;

import statictools.Serializer;
import trust.Reviewer;
import trust.RoR;

// TODO: Lav en ordenlig test af ROR
public class RoRTest extends TestCase {

  @Test
  public void testRoR() {
    RoR ror = new RoR();
    Reviewer rv = new Reviewer(0.5,"Susan");
    ror.insertReviewer(rv);
    double trustvalue = ror.getTrustValueFromUsername("Susan");
    assertEquals(trustvalue, 0.5);
  }

  @Test
  public void testReadRoRFromDisk() {
    RoR ror = Serializer.readRoRFromDisk();
    ror.updateCertificates();
    Certificate cert = ror.getCertificateFromUsername("Susansolovan");
    double trustvalue = ror.getTrustValueFromUsername("Susansolovan");
    assertTrue(cert != null);
    assertTrue(new Double(trustvalue) != null);
  }

}
```

C.12.3 TrustUpdaterTest.java

```
package test.trust;

import junit.framework.TestCase;

import org.junit.Test;

import rating.Rating;
import rating.SessionRatingDB;
import trust.RoR;
import trust.TrustUpdater;

public class TrustUpdaterTest extends TestCase {

  @Test
  public void testTrustUpdater() {

    Rating r1 = new Rating("Susansolovan", 1, "1234",
          "http://en.wikipedia.org/wiki/Bass_Strait");
    Rating r2 = new Rating("Markuswennerberg", 2, "1234",
          "http://en.wikipedia.org/wiki/Bass_Strait");
    Rating r3 = new Rating("MikaelMartin", 3, "1234",
          "http://en.wikipedia.org/wiki/Bass_Strait");
    Rating r9 = new Rating("Korsgaard", 9, "1234",
          "http://en.wikipedia.org/wiki/Bass_Strait");
    SessionRatingDB srDB = new SessionRatingDB();
    srDB.push(r1);
    srDB.push(r2);
    srDB.push(r3);
    srDB.push(r9);
    RoR ror = new RoR();
    double ratingServed = 1.5;
    TrustUpdater tu = new TrustUpdater(srDB, ratingServed, ror);
    tu.ClickedYes();
    tu.ClickedR1();
    assertEquals(ror.getTrustValueFromUsername("Markuswennerberg"),0.1338126772902034);
    assertEquals(ror.getTrustValueFromUsername("Korsgaard"), -0.09999999999999998);
    assertEquals(ror.getTrustValueFromUsername("Susansolovan"), 0.1338126772902034);
    assertEquals(ror.getTrustValueFromUsername("MikaelMartin"), 0.0);

  }

}
```

C.13 Static Text Files

C.13.1 BodyHtml.txt

```
<div id="dragDiv1" class="testSquare" > Drag Here To Move
<div id="textdiv1"> <center><h3> ###RATING### </h3> </center> <br>
<applet codebase="http://###IP###/"
        code="remote.EmbeddedApplet"
        width=250 height=182>
```

```
</applet>
</div> </div>
```

C.13.2 HeadHtml.txt

```
<!-- Dependencies -->
<script type="text/javascript"
     src="http://yui.yahooapis.com/2.2.2/build/yahoo-dom-event/yahoo-dom-event.js"
     ></script>

<!-- Drag and Drop source file -->
<script type="text/javascript"
     src="http://yui.yahooapis.com/2.2.2/build/dragdrop/dragdrop-min.js" ></script>

<script type="text/javascript">
YAHOO.example.DDApp = function() {
    var dd;
    return {
        init: function() {
            dd = new YAHOO.util.DD("dragDiv1");
        }
    }
}();
YAHOO.util.Event.onDOMReady(YAHOO.example.DDApp.init);
</script>

<!-- Style for the boxes-->
<style type="text/css">
#dragDiv1 {
    background: #E8E8E8 ;
    background-color:#6D739A;top:240px; left:105px;
    width:150px;
    position:absolute;
    top:50px;
    left:600px;
    z-index: 10;
}

#textdiv1{

        background-color:        #E8E8E8 ;
        height:182px;
        width:250px;
}
</style>
```

C.13.3 Passwords.txt

```
KeyStorePass 12345678
WikiUserPass Susansolovan 12345678
```

C.13.4 build.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- WARNING: Eclipse auto-generated file.
              Any modifications will be overwritten.
              To include a user specific buildfile here, simply create one in the
                   same
              directory with the processing instruction <?eclipse.ant.import?>
              as the first entry and export the buildfile again. -->
<project basedir="." default="build" name="OwnPlugin">
    <property environment="env"/>
    <property name="ECLIPSE_HOME" value="../../../Program Files/eclipse"/>
    <property name="junit.output.dir" value="junit"/>
    <property name="debuglevel" value="source,lines,vars"/>
    <property name="target" value="1.5"/>
    <property name="source" value="1.5"/>
    <path id="OwnPlugin.classpath">
        <pathelement location="."/>
        <pathelement location="scone.jar"/>
        <pathelement location="wbij45.jar"/>
        <pathelement location="junit-4.3.1.jar"/>
        <pathelement location="log4j-1.2.14.jar"/>
    </path>
```

```
<target name="init">
    <copy includeemptydirs="false" todir=".">
        <fileset dir="." excludes="**/*.launch, **/*.java"/>
    </copy>
</target>
<target name="clean">
    <delete>
        <fileset dir="." includes="**/*.class"/>
    </delete>
</target>
<target depends="clean" name="cleanall"/>
<target depends="build-subprojects,build-project" name="build"/>
<target name="build-subprojects"/>
<target depends="init" name="build-project">
    <echo message="${ant.project.name}: ${ant.file}"/>
    <javac debug="true" debuglevel="${debuglevel}" destdir="."
          source="${source}" target="${target}">
        <src path="."/>
        <exclude name="benchmarking/PureProxyBenchmarkTest.java"/>
        <classpath refid="OwnPlugin.classpath"/>
    </javac>
</target>
<target description="Build all projects which reference this project. Useful to
      propagate changes." name="build-refprojects"/>
<target description="copy Eclipse compiler jars to ant lib directory"
      name="init-eclipse-compiler">
    <copy todir="${ant.library.dir}">
        <fileset dir="${ECLIPSE_HOME}/plugins"
              includes="org.eclipse.jdt.core_*.jar"/>
    </copy>
    <unzip dest="${ant.library.dir}">
        <patternset includes="jdtCompilerAdapter.jar"/>
        <fileset dir="${ECLIPSE_HOME}/plugins"
              includes="org.eclipse.jdt.core_*.jar"/>
    </unzip>
</target>
<target description="compile project with Eclipse compiler"
      name="build-eclipse-compiler">
    <property name="build.compiler"
          value="org.eclipse.jdt.core.JDTCompilerAdapter"/>
    <antcall target="build"/>
</target>
<target name="AllTests (1)">
    <mkdir dir="${junit.output.dir}"/>
    <junit fork="yes" printsummary="withOutAndErr">
        <formatter type="xml"/>
        <test name="test.page.AllTests" todir="${junit.output.dir}"/>
        <classpath refid="OwnPlugin.classpath"/>
    </junit>
</target>
<target name="AllTests">
    <mkdir dir="${junit.output.dir}"/>
    <junit fork="yes" printsummary="withOutAndErr">
        <formatter type="xml"/>
        <test name="test.rating.AllTests" todir="${junit.output.dir}"/>
        <classpath refid="OwnPlugin.classpath"/>
    </junit>
</target>
<target name="CreateTestRoR">
    <java classname="test.sconeplugin.CreateTestRoR" failonerror="true"
          fork="yes">
        <classpath refid="OwnPlugin.classpath"/>
    </java>
</target>
<target name="ExtractRatingsTest">
    <mkdir dir="${junit.output.dir}"/>
    <junit fork="yes" printsummary="withOutAndErr">
        <formatter type="xml"/>
        <test name="test.page.ExtractRatingsTest" todir="${junit.output.dir}"/>
        <classpath refid="OwnPlugin.classpath"/>
    </junit>
</target>
<target name="GenerateCertTest">
    <mkdir dir="${junit.output.dir}"/>
    <junit fork="yes" printsummary="withOutAndErr">
        <formatter type="xml"/>
        <test name="_security.GenerateCertTest" todir="${junit.output.dir}"/>
        <classpath refid="OwnPlugin.classpath"/>
    </junit>
</target>
<target name="Mainfile">
    <java classname="sconeplugin.MainFile" failonerror="true" fork="yes">
        <classpath refid="OwnPlugin.classpath"/>
    </java>
</target>
<target name="MasterAllTests (2)">
    <mkdir dir="${junit.output.dir}"/>
```

```
      <junit fork="yes" printsummary="withOutAndErr">
         <formatter type="xml"/>
         <test name="test.MasterAllTests" todir="${junit.output.dir}"/>
         <classpath refid="OwnPlugin.classpath"/>
      </junit>
   </target>
   <target name="New_configuration (1)">
      <java classname="_wikitest.WikiTestMain" failonerror="true" fork="yes">
         <classpath refid="OwnPlugin.classpath"/>
      </java>
   </target>
   <target name="New_configuration (3)">
      <java classname="" failonerror="true" fork="yes">
         <classpath refid="OwnPlugin.classpath"/>
      </java>
   </target>
   <target name="PageExtractorTest">
      <mkdir dir="${junit.output.dir}"/>
      <junit fork="yes" printsummary="withOutAndErr">
         <formatter type="xml"/>
         <test name="test.page.PageExtractorTest" todir="${junit.output.dir}"/>
         <classpath refid="OwnPlugin.classpath"/>
      </junit>
   </target>
   <target name="PageModifierTest">
      <mkdir dir="${junit.output.dir}"/>
      <junit fork="yes" printsummary="withOutAndErr">
         <formatter type="xml"/>
         <test name="test.page.PageModifierTest" todir="${junit.output.dir}"/>
         <classpath refid="OwnPlugin.classpath"/>
      </junit>
   </target>
   <target name="RatingCalculatorTest">
      <mkdir dir="${junit.output.dir}"/>
      <junit fork="yes" printsummary="withOutAndErr">
         <formatter type="xml"/>
         <test name="test.rating.RatingCalculatorTest"
              todir="${junit.output.dir}"/>
         <classpath refid="OwnPlugin.classpath"/>
      </junit>
   </target>
   <target name="RatingCleanOutTest (1)">
      <mkdir dir="${junit.output.dir}"/>
      <junit fork="yes" printsummary="withOutAndErr">
         <formatter type="xml"/>
         <test name="test.statictools.RatingCleanOutTest"
              todir="${junit.output.dir}"/>
         <classpath refid="OwnPlugin.classpath"/>
      </junit>
   </target>
   <target name="RatingCleanOutTest">
      <mkdir dir="${junit.output.dir}"/>
      <junit fork="yes" printsummary="withOutAndErr">
         <formatter type="xml"/>
         <test name="test.statictools.RatingCleanOutTest"
              todir="${junit.output.dir}"/>
         <classpath refid="OwnPlugin.classpath"/>
      </junit>
   </target>
   <target name="RatingHistoryTest">
      <mkdir dir="${junit.output.dir}"/>
      <junit fork="yes" printsummary="withOutAndErr">
         <formatter type="xml"/>
         <test name="test.rating.RatingHistoryTest" todir="${junit.output.dir}"/>
         <classpath refid="OwnPlugin.classpath"/>
      </junit>
   </target>
   <target name="RatingTest">
      <mkdir dir="${junit.output.dir}"/>
      <junit fork="yes" printsummary="withOutAndErr">
         <formatter type="xml"/>
         <test name="test.rating.RatingTest" todir="${junit.output.dir}"/>
         <classpath refid="OwnPlugin.classpath"/>
      </junit>
   </target>
   <target name="RecommendationSubmitterTest">
      <mkdir dir="${junit.output.dir}"/>
      <junit fork="yes" printsummary="withOutAndErr">
         <formatter type="xml"/>
         <test name="rating.RecommendationSubmitterTest"
              todir="${junit.output.dir}"/>
         <classpath refid="OwnPlugin.classpath"/>
      </junit>
   </target>
   <target name="ReviewerTest">
      <mkdir dir="${junit.output.dir}"/>
      <junit fork="yes" printsummary="withOutAndErr">
```

```
            <formatter type="xml"/>
            <test name="test.trust.ReviewerTest" todir="${junit.output.dir}"/>
            <classpath refid="OwnPlugin.classpath"/>
        </junit>
    </target>
    <target name="ReviewerTest.update100Certificates">
        <mkdir dir="${junit.output.dir}"/>
        <junit fork="yes" printsummary="withOutAndErr">
            <formatter type="xml"/>
            <test name="test.trust.ReviewerTest" todir="${junit.output.dir}"/>
            <classpath refid="OwnPlugin.classpath"/>
        </junit>
    </target>
    <target name="RoRTest">
        <mkdir dir="${junit.output.dir}"/>
        <junit fork="yes" printsummary="withOutAndErr">
            <formatter type="xml"/>
            <test name="test.trust.RoRTest" todir="${junit.output.dir}"/>
            <classpath refid="OwnPlugin.classpath"/>
        </junit>
    </target>
    <target name="SecurityProviderTest">
        <mkdir dir="${junit.output.dir}"/>
        <junit fork="yes" printsummary="withOutAndErr">
            <formatter type="xml"/>
            <test name="test.statictools.SecurityProviderTest"
                 todir="${junit.output.dir}"/>
            <classpath refid="OwnPlugin.classpath"/>
        </junit>
    </target>
    <target name="SecurityProviderTest.testCreateRating">
        <mkdir dir="${junit.output.dir}"/>
        <junit fork="yes" printsummary="withOutAndErr">
            <formatter type="xml"/>
            <test name="statictools.SecurityProviderTest"
                 todir="${junit.output.dir}"/>
            <classpath refid="OwnPlugin.classpath"/>
        </junit>
    </target>
    <target name="SecurityProviderTest.testGenerateRatings">
        <mkdir dir="${junit.output.dir}"/>
        <junit fork="yes" printsummary="withOutAndErr">
            <formatter type="xml"/>
            <test name="test.statictools.SecurityProviderTest"
                 todir="${junit.output.dir}"/>
            <classpath refid="OwnPlugin.classpath"/>
        </junit>
    </target>
    <target name="SerializerTest">
        <mkdir dir="${junit.output.dir}"/>
        <junit fork="yes" printsummary="withOutAndErr">
            <formatter type="xml"/>
            <test name="test.statictools.SerializerTest"
                 todir="${junit.output.dir}"/>
            <classpath refid="OwnPlugin.classpath"/>
        </junit>
    </target>
    <target name="SessionRatingDBTest">
        <mkdir dir="${junit.output.dir}"/>
        <junit fork="yes" printsummary="withOutAndErr">
            <formatter type="xml"/>
            <test name="test.rating.SessionRatingDBTest"
                 todir="${junit.output.dir}"/>
            <classpath refid="OwnPlugin.classpath"/>
        </junit>
    </target>
    <target name="TestBuild">
        <java classname="" failonerror="true" fork="yes">
            <classpath refid="OwnPlugin.classpath"/>
        </java>
    </target>
    <target name="ThresholdTest">
        <mkdir dir="${junit.output.dir}"/>
        <junit fork="yes" printsummary="withOutAndErr">
            <formatter type="xml"/>
            <test name="test.statictools.ThresholdTest"
                 todir="${junit.output.dir}"/>
            <classpath refid="OwnPlugin.classpath"/>
        </junit>
    </target>
    <target name="TrustUpdaterTest">
        <mkdir dir="${junit.output.dir}"/>
        <junit fork="yes" printsummary="withOutAndErr">
            <formatter type="xml"/>
            <test name="test.trust.TrustUpdaterTest" todir="${junit.output.dir}"/>
            <classpath refid="OwnPlugin.classpath"/>
        </junit>
    </target>
    <target name="WikiTestMain">
        <java classname="_wikitest.WikiTestMain" failonerror="true" fork="yes">
            <classpath refid="OwnPlugin.classpath"/>
        </java>
    </target>
    <target name="junitreport">
        <junitreport todir="${junit.output.dir}">
            <fileset dir="${junit.output.dir}">
                <include name="TEST-*.xml"/>
            </fileset>
            <report format="frames" todir="${junit.output.dir}"/>
        </junitreport>
    </target>
</project>
```

Appendix D

Test Material

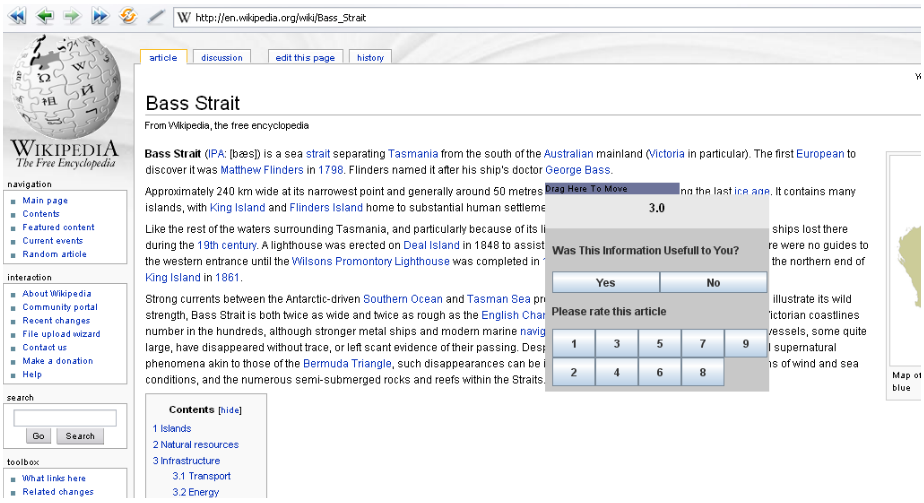
D.1 Serving a Recommendation

The recommendation provided is seen on figure D.1 and the ratings that is used for calculating is shown below:

```
<!-- WikiTrustComment. Read more on:
http://en.wikipedia.org/wiki/User:Korsgaard
;Korsgaard;7;94723853;Bass_Strait;MCwCFBRZ3bjvzQk5qygSaOd8k1FNnzTeAhROuFSrcRA1ZuRCDktVV033/4xRLA==
--> <!-- WikiTrustComment. Read more on:
http://en.wikipedia.org/wiki/User:Korsgaard
;Markuswennerberg;2;135767351;Bass_Strait;MC0CFQCAsVVaRk7rUg3JRpr5YDisWQQFmwIULnk9NYOrvsDsf1Xrnnr9KME1LVO=
--> <!-- WikiTrustComment. Read more on:
http://en.wikipedia.org/wiki/User:Korsgaard
;Mikaelmartin;3;135767351;Bass_Strait;MCwCFBnk15F+J+q3hcyEFzV+IX6+IJyQAhQLGZY7yE/OyWnQBv09tJNJxlx14A==
--> <!-- WikiTrustComment. Read more on:
http://en.wikipedia.org/wiki/User:Korsgaard
;Susansolovan;8;135767351;Bass_Strait;MCwCFFg2Xq0gC9RcgobBsotmNIbTT3IiAhQZcxGg0hAZ1wkB3+onaJOBmmOPHQ==
-->
```

D.2 Giving Feedback

A screenshot of the the WRS after feedback have been given is shown on figure D.2



Bass Strait

From Wikipedia, the free encyclopedia

Bass Strait (IPA: [bæs]) is a sea strait separating Tasmania from the south of the Australian mainland (Victoria in particular). The first European to discover it was Matthew Flinders in 1798. Flinders named it after his ship's doctor George Bass.

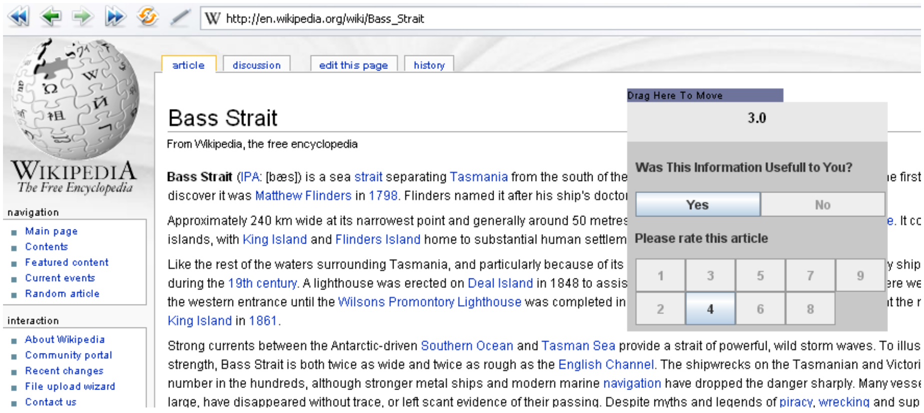
Approximately 240 km wide at its narrowest point and generally around 50 metres ng the last ice age. It contains many islands, with King Island and Flinders Island home to substantial human settleme

Like the rest of the waters surrounding Tasmania, and particularly because of its li ships lost there during the 19th century. A lighthouse was erected on Deal Island in 1848 to assist re were no guides to the western entrance until the Wilsons Promontory Lighthouse was completed in the northern end of King Island in 1861.

Strong currents between the Antarctic-driven Southern Ocean and Tasman Sea pr illustrate its wild strength, Bass Strait is both twice as wide and twice as rough as the English Char ictorian coastlines number in the hundreds, although stronger metal ships and modern marine navig vessels, some quite large, have disappeared without trace, or left scant evidence of their passing. Desp l supernatural phenomena akin to those of the Bermuda Triangle, such disappearances can be i ns of wind and sea conditions, and the numerous semi-submerged rocks and reefs within the Straits.

Contents [hide]
1 Islands
2 Natural resources
3 Infrastructure
3.1 Transport
3.2 Energy

Figure D.1: A recommendation is inserted into the browser



Bass Strait

From Wikipedia, the free encyclopedia

Bass Strait (IPA: [bæs]) is a sea strait separating Tasmania from the south of the ne first discover it was Matthew Flinders in 1798. Flinders named it after his ship's doctor

Approximately 240 km wide at its narrowest point and generally around 50 metres e. It co islands, with King Island and Flinders Island home to substantial human settlem

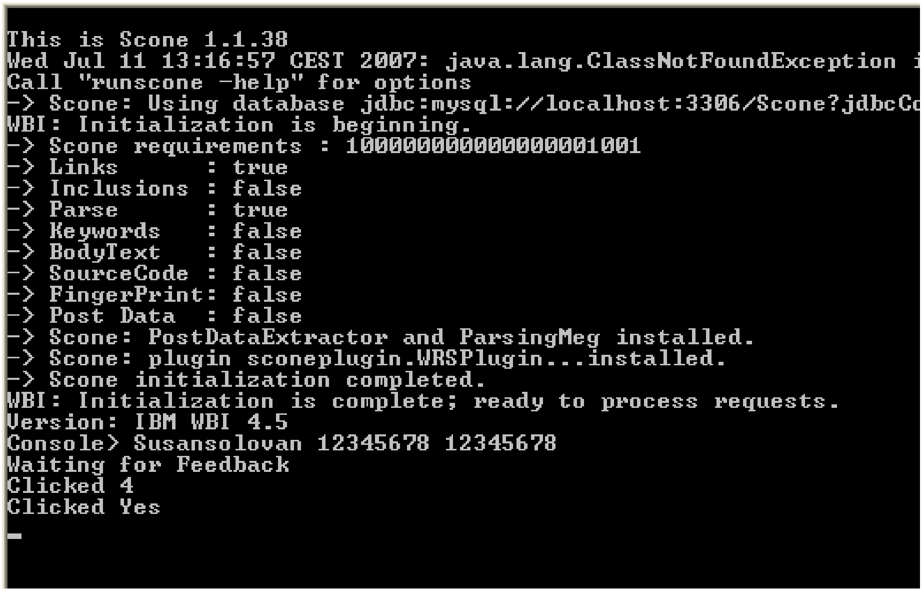
Like the rest of the waters surrounding Tasmania, and particularly because of its y ship during the 19th century. A lighthouse was erected on Deal Island in 1848 to assis ere we the western entrance until the Wilsons Promontory Lighthouse was completed in t the r King Island in 1861.

Strong currents between the Antarctic-driven Southern Ocean and Tasman Sea provide a strait of powerful, wild storm waves. To illus strength, Bass Strait is both twice as wide and twice as rough as the English Channel. The shipwrecks on the Tasmanian and Victori number in the hundreds, although stronger metal ships and modern marine navigation have dropped the danger sharply. Many vesse large, have disappeared without trace, or left scant evidence of their passing. Despite myths and legends of piracy, wrecking and sup

Figure D.2: Feedback given

D.3 Output from Scone

A screenshot from Scone after feedback have been given can be seen on figure D.3



```
This is Scone 1.1.38
Wed Jul 11 13:16:57 CEST 2007: java.lang.ClassNotFoundException i
Call "runscone -help" for options
-> Scone: Using database jdbc:mysql://localhost:3306/Scone?jdbcCo
WBI: Initialization is beginning.
-> Scone requirements : 1000000000000000001001
-> Links       : true
-> Inclusions : false
-> Parse       : true
-> Keywords    : false
-> BodyText    : false
-> SourceCode : false
-> FingerPrint: false
-> Post Data   : false
-> Scone: PostDataExtractor and ParsingMeg installed.
-> Scone: plugin sconeplugin.WRSPlugin...installed.
-> Scone initialization completed.
WBI: Initialization is complete; ready to process requests.
Version: IBM WBI 4.5
Console> Susansolovan 12345678 12345678
Waiting for Feedback
Clicked 4
Clicked Yes
```

Figure D.3: Scone output after feedback given

Appendix E

Content of the CD-ROM

The CD-ROM inclosed contains the following directories:

Code The complete source code for the WRS.

Bin Contains the binaries for the WRS, and scripts for executing the WRS. Please make sure that the system is correctly configured as explained in appendix B.

Report Contains a pdf of this report.

Appendix F

Foldout diagrams

On the next two pages two full scale ULM diagrams is found. The sequence diagram is broken up and presented in chapter 5 and the class diagram is broken up in chapter 6

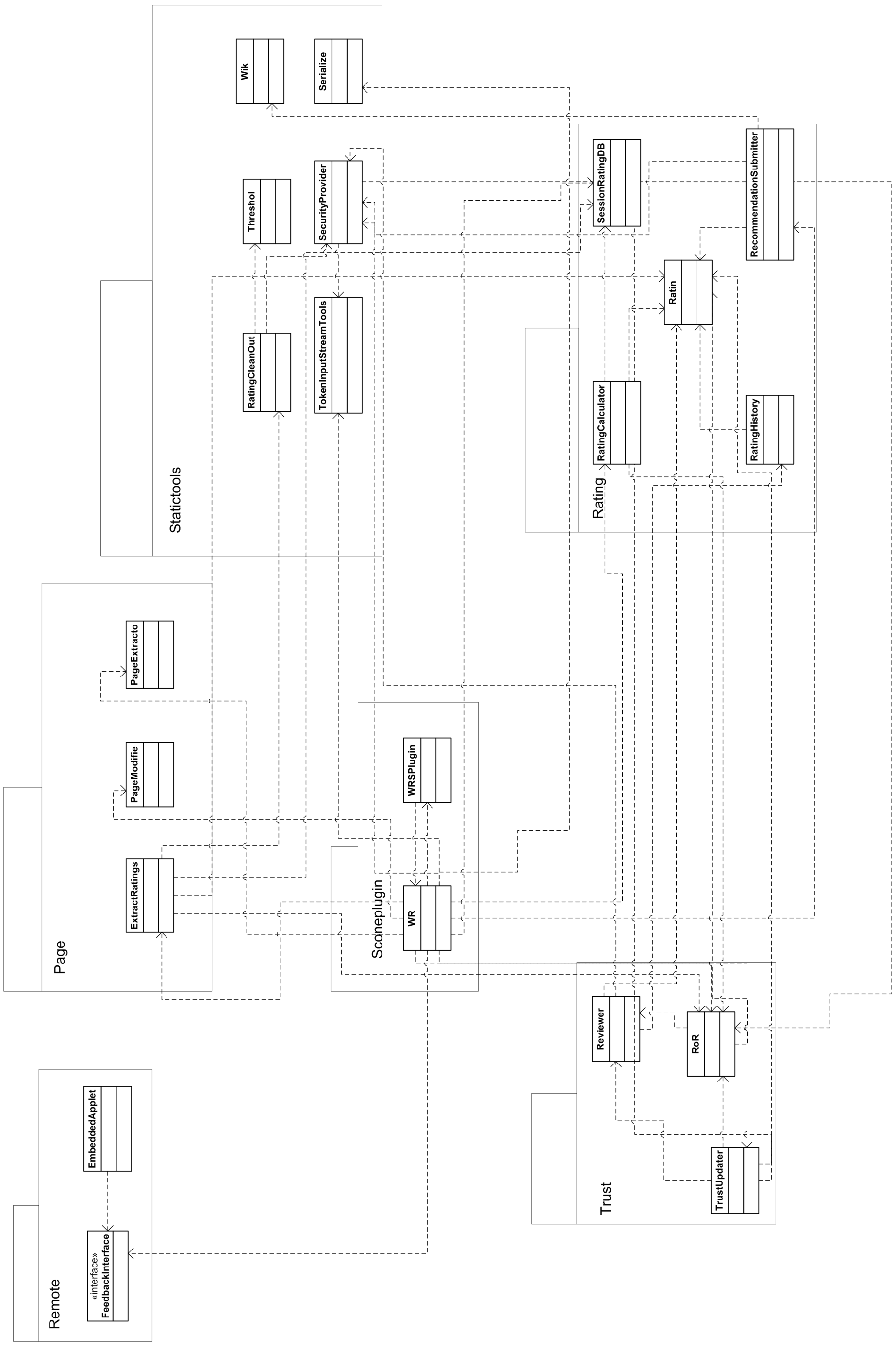


Figure F.1: Package and class diagram

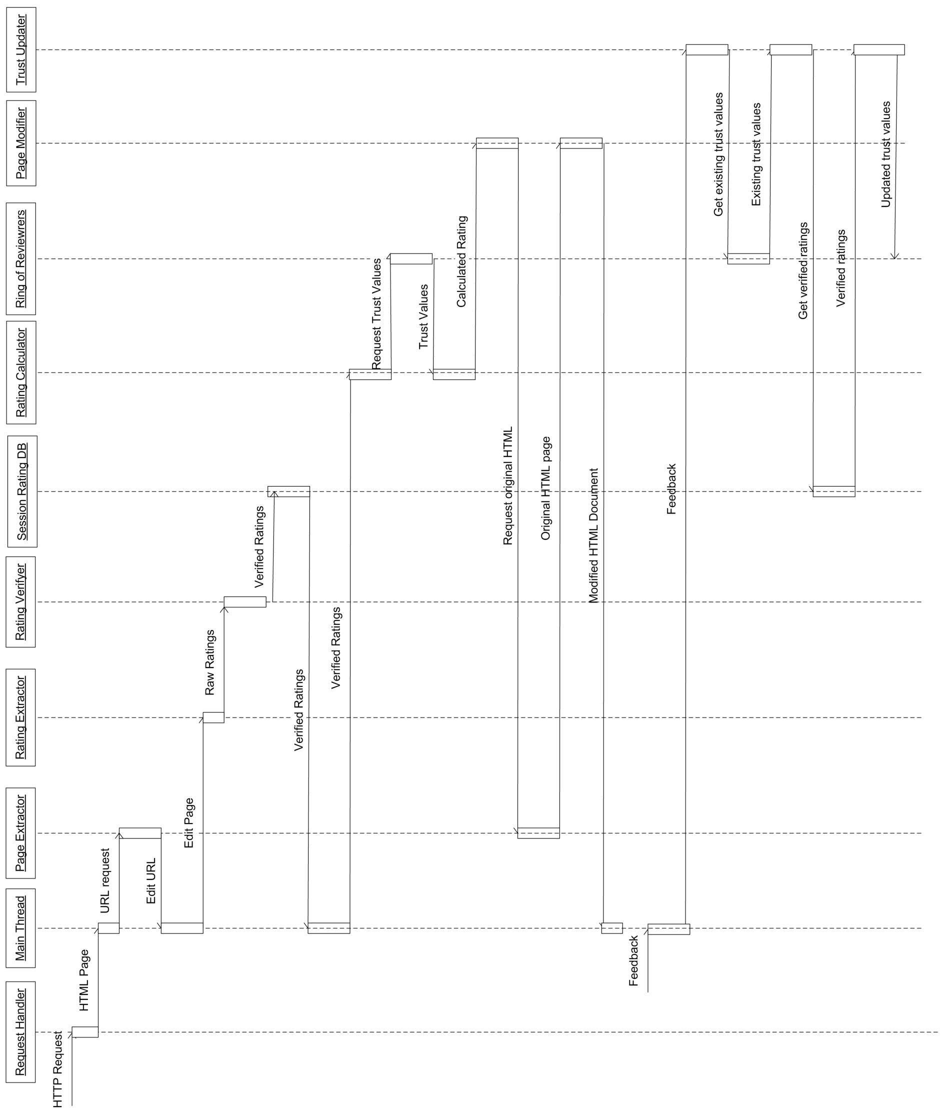


Figure F.2: A recommendation is inserted into the browser

Bibliography

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

[2] Internet movie database. http://www.imdb.com.

[3] Java wiki bot framework. http://sourceforge.net/projects/jwbf/.

[4] mywot - web of trust. http://www.mywot.com.

[5] Princeton university - wordnet. http://wordnet.princeton.edu.

[6] Sun microsystems - keytool. http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/keytool.html.

[7] Wiki philosophy. http://www.wiki.org/wiki.cgi?WhatIsWiki.

[8] Wikipedia foundation - mediawiki. http://www.mediawiki.org/wiki/MediaWiki.

[9] Akamai and JupiterResearch. Akamai and jupiterresearch identify '4 seconds' as the new threshold of acceptability for retail web page response times. http://www.akamai.com/html/about/press/releases/2006/press_110606.html, 2007 (accessed July 12, 2007).

[10] Paulo Pinheiro da Silva, Deborah L. McGuinness, and Richard Fikes. A proof markup language for semantic web services. *Inf. Syst.*, 31(4):381–395, 2006.

[11] Author Do. Adc parser. http://www-cs-students.stanford.edu/~do/.

[12] Pierpaolo Dondio, Stephen Barrett, Stefan Weber, and Jean-Marc Seigneur. Extracting trust from domain analysis: A case study on the wikipedia project. In *ATC*, pages 362–373, 2006.

[13] John Douceur. The Sybil Attack. In *Proceedings of the 1st International Peer To Peer Systems Workshop (IPTPS 2002)*, March 2002.

[14] eBlogger. Eblogger: On vandalism. `http://eblogger.blogsome.com/2005/10/24/on-vandalism/`, 2007 (accessed May 15, 2007).

[15] Richard P. Honeck. Semantic similarity between sentences. In *Journal of Psycholinguistic Research*, 1973.

[16] MER-C (http://en.wikipedia.org/wiki/User:MER C). Wikibot. `http://en.wikipedia.org/wiki/User:MER-C/Wiki.java`.

[17] IBM. Web intermediaries (wbi). `http://www.almaden.ibm.com/cs/wbi/`.

[18] Catholijn M. Jonker, Joost J. P. Schalken, Jan Theeuwes, and Jan Treur. Human experiments in trust dynamics. In *iTrust*, pages 206–220, 2004.

[19] Catholijn M. Jonker and Jan Treur. Formal analysis of models for the dynamics of trust based on experiences. In Francisco J. Garijo and Magnus Boman, editors, *Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-99)*, volume 1647, pages 221–231, Berlin, 30– 2 1999. Springer-Verlag: Heidelberg, Germany.

[20] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decis. Support Syst.*, 43(2):618–644, 2007.

[21] Audun Jøsang, Claudia Keser, and Theodosis Dimitrakos. Can we manage trust? In *iTrust*, pages 93–107, 2005.

[22] Arnd Kohrs. *Collaborative filtering on the Internet*. PhD thesis, University of Nice, Jul 2001.

[23] Paul Stephen Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, University of Stirling, 1994.

[24] Sean M. McNee, Shyong K. Lam, Catherine Guetzlaff, Joseph A. Konstan, and John Riedl. Confidence displays and training in recommender systems. Proceedings of the 9th IFIP TC13 International Conference on Human-Computer Interaction (INTERACT 2003).

[25] Muffin. World wide web filtering system. `http://muffin.doit.org/`.

[26] Podcasting News. Wikipedia caught in podfather turf war. http://www.podcastingnews.com/archives/2005/12/wikipedia_caugh_1.html, 2007 (accessed May 23, 2007).

[27] Andrew Oram. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001.

[28] PAW. pro-active webfilter. http://paw-project.sourceforge.net/.

[29] phpBB. phpbb forum. http://www.phpbb.com/, 2007 (accessed July 19, 2007).

[30] The Register. Avoid wikipedia, warns wikipedia chief: It can seriously damage your grades. http://www.theregister.co.uk/2006/06/15/wikipedia_can_damage_your_grades/, 2007 (accessed July 3, 2007).

[31] Jean-Marc Seigneur, Stephen Farrell, Christian D. Jensen, Elizabeth Gray, and Yong Chen. *End-to-End Trust Starts with Recognition*, volume 2802. January 2004.

[32] SFGate.com. The online credibility gap: Wikipedia article's false claim on jfk killing stirs debate. http://sfgate.com/cgi-bin/article.cgi?file=/c/a/2005/12/06/WIKI.TMP, 2007 (accessed June 22, 2007).

[33] Howard the Shopping Assistant. European consumer center denmark. http://www.forbrugereuropa.dk/english/facts-and-advice/ecommerce/assistant/, 2007 (accessed May 24, 2007).

[34] David A. Utter. Wikipedia bans congress. http://www.webpronews.com/topnews/2006/01/30/wikipedia-bans-congress, 2007 (accessed May 18, 2007).

[35] David Wagner. Resilient aggregation in sensor networks. In *SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, pages 78–87, New York, NY, USA, 2004. ACM Press.

[36] Harald Weinreich. Scone proxy. http://scone.de/.

[37] Wikimedia Foundation. enwiki dump progress on 20070402. http://download.wikimedia.org/enwiki/20070402/, 2007 (accessed May 29, 2007).

[38] Wikipedia. Wikipedia:size of wikipedia. http://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia, 2007 (accessed May 18, 2007).

[39] Ilya Zaihrayeu, Paulo Pinheiro da Silva, and Deborah L. McGuinness. Iwtrust: Improving user trust in answers from the web. In *iTrust*, pages 384–392, 2005.