# Reducing Cold Start problem in the Wikipedia Recommender System

Natasa Popovic Andersen

Kongens Lyngby 2011
IMM-MSc-2011

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

# Abstract

Wikipedia is a free online encyclopedia based on the openly editable model where Internet users all over the world, who wishes to share their knowledge about a subject, can edit the article on that subject. The open and flexible nature of Wikipedia exposes a lot of problems concerning the quality and correctness of the articles. The Wikipedia Recommender System (Korsgaard & Jensen 2008) is developed in order to help users assess the quality of articles and to provide incentive for authors to improve the quality of their articles. It is built on top of the existing Wikipedia software and it is a distributed, trust based recommender system.

The Wikipedia Recommender System suffers from a general problem in Recommendation systems, known as the "Cold Start" problem. If a new user in the system who has only interacted with a few other users wants to recommend an article, the system cannot calculate trust in the recommendation. Until a certain level of annotations has been made, ratings will be imprecise. One way of addressing this problem is to include a second level of recommendations in the system, where users can recommend other users as trustworthy recommenders. Trustworthy recommenders will propagate some of their trust values on other users, back to the truster to expand the trust profile of the truster and build a trust network.

This project aims to examine how concepts from the semantic web can be used to reduce the cold start problem, build the trust network to propagate trust and use the trust metric to calculate the trust in a previously unknown user. The Semantic Web can be used to provide the common framework that allows data to be shared across community boundaries. The semantic web of trust requires that users describe their beliefs about others. Once a person has a file that lists who they know and how much they trust them, social information can be easily compiled and uploaded to the user's own page in the Wikipedia.

The solution should extend the existing Wikipedia Recommender System software and follow the premises on which WRS is built on: it is a distributed system and it is built on the top of the existing Wikipedia software.

# Preface

This thesis was prepared at the Department of Informatics and Mathematical Modeling of Technical University of Denmark as part of the requirements for acquiring the MSc degree in Computer Science and Engineering. It has been carried out in the period from March to September 2011. The thesis has been written under the supervision of Associated Professor Christian Damsgaard Jensen.

I would like to thank my supervisor for the support and guidance during this project. Special thanks goes to Povilas Pilkauskas for introducing me to the Wikipedia Recommender System, help with setting it up and pointing out the main features of it. Thanks to the participants of the First IFIP WG 11.11 International Conference and Summer School on Trust Management for their valuable inputs and comments.

Finally, I would like to thank my family and friends for their support and encouragement during the past months and most of all, for believing in me to carry out master studies in the fourth decade of my life!

Lyngby, August 2011

_____

Natasa Popovic Andersen
s091413

# Content

# Chapter 1: Introduction

## 1.1 Introduction

"Wikipedia is a multilingual, web based, free content encyclopedia project based on the openly editable model" [12]. In other words, Wikipedia is a free online encyclopedia collaboratively edited by Internet users around the world. The Wikipedia philosophy is that anyone who wishes to share their knowledge about a subject can edit the article on that subject. No prior qualifications are needed to write an article, so the Wikipedia is able to harness the collective intelligence of all Internet users, by allowing anyone who feels knowledgeable in an area to create or correct articles about subjects in that area.

Opinions about Wikipedia are mixed. Wikipedia claims to be the most popular reference site on the Internet, a model of democratized information, 'wisdom of crowds' and something that may cause an end of the 'stodgy old commercial encyclopedia'. Others are concerned about the reliability of an uncontrolled reference work that may include any number of purposeful or accidental inaccuracies. The open and flexible nature of the Wikipedia has exposed one of the weaknesses of collaborative authoring, which is that malicious or incompetent users may compromise the integrity of the document by introducing erroneous entries and or corrupting existing entries. Jimmy Wales, the co-founder of the Wikipedia, claims to receive 10 emails every day from students who failed their course because the information cited from Wikipedia turned out to be wrong [4]. Another example of the erroneous entry and slow responsiveness of editing community is a false biography of John Seigenthaler. For four months Wikipedia depicted him as a suspected assassin of Kennedy brothers. During that time, one volunteer edited his biography by fixing the spelling mistake. The same false biography appeared at Reference.com and Answers.com. Four months later, at his request, his biography was removed from the three websites. But they don't know, and cannot find out, who wrote the toxic sentence [3].

Robert McHenry, former editor of Encyclopedia Britannica raised a question about accuracy and quality of the articles provided by Wikipedia. He thinks that community-accretion process of Wikipedia is fundamentally incapable of rising to a high standard of excellence ("The faith based encyclopedia") [1].

Regardless of which side one may take, relying on Wikipedia presents numerous risks [2]:

**Accuracy**, you cannot be sure which information is accurate and which is not;

**Motives**, you cannot know if motives of the contributors to an article are altruistic, political, or commercial or if contributors are just a practical jokers or even vandals;

**Uncertain expertise**, it is difficult to determine how qualified an article's contributors are as revision histories often identify them by pseudonyms, making it hard to check credentials and sources;

**Volatility**, contributors and corrections may be negated by future contributors;

**Coverage**, voluntary contributions largely represent the interests and knowledge of a self-selected set of contributors causing less interesting or historical topics covered inadequately or inaccurately;

**Sources**, many articles do not cite independent sources.

It is, however, apparent that the Wikipedia has gained the trust of the Internet population, despite the fact that there is nothing inherently trustworthy about the Wikipedia. It is therefore important to provide Wikipedia users with a simple and intuitive way to access the trustworthiness of the content they are reading.

Reputation systems, where users provide feedback on the quality of the articles they read, provide a powerful incentive to improve the quality of the articles and help users avoid articles that are of low quality. In order to preserve both the large investment that authors have made in terms of time and effort and the familiarity of the user interface for occasional users, the collaborative authoring system must be considered a legacy system that cannot be modified and the reputation system should only be offered to users who opt in. This means that the reputation system should be offered as an external service, which users can include as an intermediary between their browser and the collaborative authoring system, in this case the Wikipedia [6].

Wikipedia Recommender System (WRS) (Korsgaard and Jensen, 2008) is a reputation system for the Wikipedia, which provides a central repository for feedback that allows individual users to calculate their own subjective reputation value 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. The local component of the recommender system residing on the user's own machine is able to calculate a rating, using the recommendations from other people, which indicate the quality of the article. The recommender system then receives feedback from the reader, which allows the WRS to determine whether the recommendations were useful and to identify the recommenders whose feedback was similar to the feedback provided by the user. The recommender system uses this information to update active user's profile, in order to decide how recommendations should be interpreted and provide a more precise recommendation next time.

## 1.2 Wikipedia Recommender System

The Wikipedia Recommender System (WRS) allows users of Wikipedia to record and share their opinions on the quality of articles that they read in the Wikipedia. The WRS treats Wikipedia as a legacy system, which means that it has been built up on the top of the current Wikipedia system, without modifying the infrastructure or the underlying wiki engine. A brief overview of the system is presented in the following, but more detailed description and evaluation of the system is given in [5], [6] and [7].

The Wikipedia Recommender System is implemented as a proxy between the Wikipedia site and the user's browser. It is based on an extendable web proxy technology called Scone [8], which makes it possible to intercept Wikipedia requests and collect recommendations from other users about the article and present

an aggregated value that represents these recommendations to the user through the WRS user interface that is injected into the browser. The user can then use this interface to read estimated ratings as well as rate articles himself.

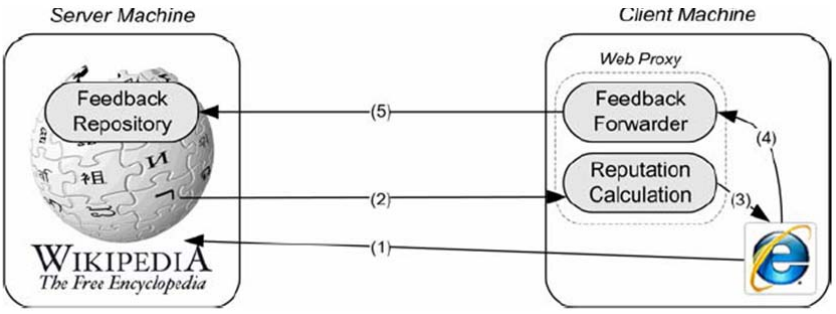



**Figure 1.2.1. Overview of the WRS [29]**

When a user requests a Wikipedia article (1), the Reputation Calculation module extracts the rating data from Wikipedia (2). It then uses this data to calculate aggregated rating of the article which is passed to the user interface (4), the user can rate the article and the Feedback Forwarder will compile the necessary data, and upload it to the appropriate location on Wikipedia.

Treating the Wikipedia as a legacy system means that there is no access to the Wikipedia's underlying SQL database, so recommendations had to be stored in a separate decentralized database or as HTML comments in the pages of the Wikipedia.

Storing recommendations directly in Wikipedia pages benefits from the fundamental Wiki philosophy of providing a central repository of information that all users can modify easily. Storing recommendations in HTML comments means that they will not be shown by existing web browsers, so the WRS is invisible to the users who do not participate in the system. This also means that comments are always online and recommendations are always stored in one place. The downside to this approach is that recommendations have to be protected against fabrication and modification attacks, but this may be done by standard cryptographic techniques for authenticity and authentication [7].

Wikipedia gives registered users their own user-page, which is part of Wikipedia and can be accessed by everybody. Therefore, recommendations are created and stored in sub-pages of the user's user-page, which mirror the rated article from the main Wikipedia. The first part of the recommendation contains the rating of the article which is readable by humans, but the second part is a BASE64 encoded digital signature by the user who submitted the rating.

As an example of the naming convention, the following article can be considered:

http://en.wikipedia.org/wiki/Albert_Einstein , which becomes

http://en.wikipedia.org/wiki/User:Recommendations/Albert_Einstein, page which contains ratings for the article.

When the user requests a Wikipedia article, The Reputation Calculator accesses the Feedback Repository and extracts the recommendations. It then verifies the recommendations, making sure that they are valid and properly signed. The Reputation Calculator then aggregates the recommendations for the article, based on the ratings and category in the recommendations and the requesting user's trust in the recommenders. The aggregated ratings are then presented to the user through the WRS user interface shown in Figure 1.2.2. The WRS user interface shows the aggregated quality indication at the top (the value "7.0") and the majority category that was assigned by users who viewed and rated the article before ("100% Science"). Below this line, the user is asked whether he agrees with this value ("Was this information useful to you?") and the user is asked to provide his own rating at the bottom of the window. The usefulness indicator and the user's own rating are used to calculate the user's trust in recommenders.

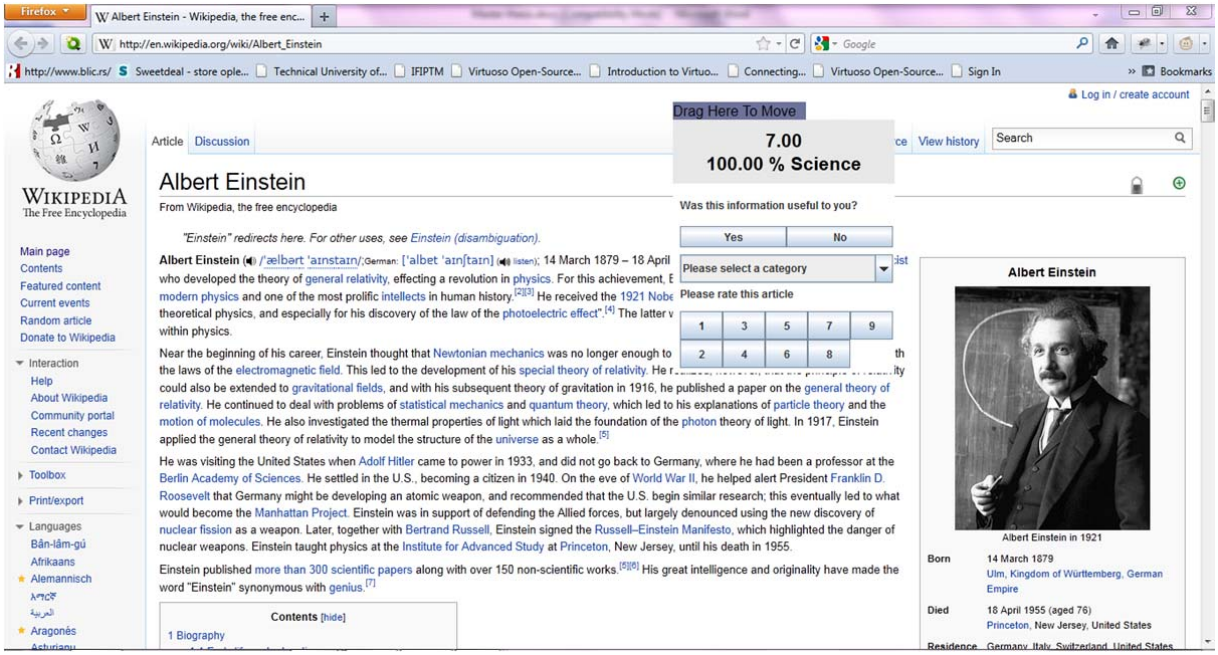



**Figure 1.2.2. The WRS user interface**

The WRS Client Proxy maintains a database of all users who have rated articles accessed by the user. When the user then rates an article on the Wikipedia, the Reputation Calculator compares this rating with the recommendations received from other users. Recommendations that contain ratings close to the user's own ratings are considered positive and result in the higher trust in these recommenders. Recommendations with the ratings that are very different from the user's own ratings will be considered negative and the user's trust in those recommenders will decrease. The user's trust in the recommender is used together with the rating in the recommendation to calculate the aggregated article rating. If the user's trust in a recommender is high, the rating will contribute more to the aggregated rating. This means that the value of the WRS to an individual user depends on the user's rating of articles, which build trust in the other users as recommenders, but at the same time contribute recommendations to the WRS community.

## 1.3 Improvements of the Wikipedia Recommender System

WRS suffers from the known problem in Recommender systems called the "Cold Start" problem. As the system uses the localized web of trust, trust can be determined only in users with whom previous interactions have been made. When a new article or user is introduced to the system, the system cannot calculate the trust value in the rating due to the fact that the article has not been rated yet, or the users trust profile did not mature enough -meaning that the user did not have prior interaction with any of the users who reviewed the article. Introduction of categories has reduced the cold start problem to the certain degree, compared to the first version of WRS which demanded only direct interactions for the selected article. Categorization gave the opportunity to establish the trust relationship based on the expertise of the reviewers in the predefined 15 categories. However, the cold start problem still exists.


Figure 1.2.3. WRS user interface with the cold start problem

This thesis proposes another way of reducing the cold start problem by introducing recommendations for other WRS users. Users can recommend other users, with whom they had positive interactions, as trustworthy reviewers and, in that way, propagate their trust in reviewer to other WRS users. The whole concept is based on the user's willingness to share part of his/her trust profile publicly with the rest of the community. As the system is designed now, trust profiles are private data stored on the user's own machine and managed by the users themselves. The solution includes the introduction of indirect or

inferred trust (trust in unknown user, recommended by somebody whom I directly trust) in addition to direct trust (trust in user based on similarity of our interactions) and metric used to calculate this trust value.

Reduction of the cold start problem is the main improvement to the system. However, during the testing of the current version of WRS several other problems occurred:

1 WRS uses a lightweight Java Wiki bot framework [24] to communicate with Wikipedia. Robots or bots are automatic processes which interact with Wikipedia as though they were human editors. A bot logs in to its own account, the same as user would do, and requests pages from Wikipedia in much the same way as a browser does. The page is not displayed on screen, but processed in memory; page data can be read, edited and submitted back to the Wikipedia.
2 WRS uses the bot to login the user, read article ratings and to submit new ratings back to the Wikipedia. The Wikimedia framework is often changed, to the point that the previous version cannot be used any more, which I faced trying to use the system for the first time. This makes system maintainability hard as every time the bot framework is changed, the WRS system has to be changed, recompiled and redistributed to the users.
3 Another problem is that Wikipedia filters became more sophisticated and more sensitive to page editing, which I also experienced during the course of development of recommendation part and performing heavy edits while testing, which resulted in Wikipedia blocking my account and IP address for editing.

It became clear that the WRS system had to be changed. The data stored in the Wikipedia had to be moved to another place. The idea of having a separate server used for storing the user data, article ratings and users recommendation was incorporated. Main concepts on which the WRS is built are preserved. All recommendations are stored to the central repository of information that all users can modify easily. Previous "User:Recommendations" pages which stored ratings in Wiki Comments are replaced with RDF files containing the data. In this way, data is centrally stored, exposed and always online. WRS web service gives the interface to the data. Data is protected of fabrication as communication with the central server is encrypted and only registered users have access to it. Wikipedia user's pages are replaced with Open Data Spaces (ODS) user's pages.

Further, WRS users don't have to be Wikipedia users; it is only expected to be registered as WRS user.

## 1.4 Structure of thesis

Chapter 1 (Introduction) contains an introduction to the credibility issues facing the Wikipedia, the concept behind the WRS, the short introduction to the problem that is the main part of the thesis and additional problems that occurred during the development of the project.

Chapter 2 (State of the Art) gives an overview of the current WRS system and theoretical and practical achievements in recommender systems concerning the solutions to fight the cold start problem.

Chapter 3 (Analysis) contains a thorough analysis of the current WRS system, reflects to the current accomplishments in cold start reduction and discusses how they can be incorporated in the WRS system. The result of the Analysis phase is a set of requirements that new WRS system should fulfill.

Chapter 4 (Introduction to Linked Data) gives an introduction bases of the semantic web. The purpose of the chapter is give a brief overview of the concepts and technologies used in the Design and Implementation chapters.

Chapter 5 (Design) describes the design of the WRS and the changes made to accommodate the proposed solution to issues discovered in Analysis and the cold start problem.

Chapter 6 (Implementation) deals with the specifics on how the additions are implemented into the existing code and also the code is made to conform to the specifications outlined in previous chapters.

Chapter 7 (Evaluation) evaluates the proposed solution both from the cold start point of view and systems functionality point of view.

Chapter 8 (Contribution) discusses the contributions made in this thesis and outlines the areas that are in need of more research as well as work on the WTS.

## 1.5 Terminology

The definition of terms used in this thesis.

**Wikipedia.** Wikipedia is an online encyclopedia and the term in the rest of thesis is referring to the English Wikipedia found on http://en.wikipedia.org.

**WRS.** Wikipedia Recommender System is a rating-based system that provides the recommendations to the users.

**Article.** This term refers to the article of Wikipedia. The article describes some information in various means of media in purpose to inform, discuss current news and general interests.

**The truster.** The truster is a user, who uses the WRS in order to obtain the recommendations about the articles in Wikipedia. He/she gets the benefit from the recommendations, which are assigned by other users.

**The trustee.** The trustee refers to all the other WRS users, but not the truster.

**Trust profile.** In the WRS each user has a trust profile. It stores the information about how many interactions user had and the experience of them. Moreover, it is saved whether the user is optimistic or cautious. All the information, which is stored on trust profile, is calculated into the trust value.

**Trust value.** The trust value is value between -1 to +1 in purpose of expressing how user(s) trusts another user(s).

**Rating.** The rating is a figure from 1 to 9 that could be assigned to a Wikipedia article by truster or trustee using WRS.

**Annotation.** The annotation consists of the article version and name assigned to the Wikipedia article. The annotations are extracted from the Wikipedia pages.

**Category.** The category refers to one of the top-level classes in classification scheme.

# Chapter 2: State of the Art

In this chapter the latest achievements in Wikipedia Recommender System are presented together with explanation of the cold start problem and ways how to fight it. Letting users to recommend other users as trustworthy reviewers and, in that way, propagate their trust in the user, is one of the ways. Explanation of the trust propagation is given together with the currently most used algorithms for calculating propagated trust in unknown user. Trust propagation works as long as a user has at least one connection that can propagate trust. Solutions how to help cold start users to make initial connections are given in the end.

## 2.1 Wikipedia Recommender System (WRS)

WRS is a rating-based collaborative filtering system developed to establish more trust in Wikipedia articles by helping users to determine credibility of an article. When the article is being viewed by the user, the system searches for users that have evaluated the article, calculates the average of ratings and the category assigned to the article by the majority of readers. After the user rates the article and classifies in belonging category, the WRS stores the given data on the central repository, which is available for everyone. This central repository is Wikipedia itself, or more precise subpages of user pages of user "Recommendations" which are mirroring the original article.
For example, recommendations for the article http://en.wikipedia.org/wiki/Albert_Einstein are stored on page http://en.wikipedia.org/wiki/User:Recommendations/Albert_Einstein.
Furthermore, the trust profile of the user is also updated depending on the ratings assigned by the user and the ratings which were assigned previously. As the system is decentralized, trust profiles are kept on the users own machine.

### 2.1.1 WRS as part of Scone

Scone is a Java Framework which allows developers to create web enhancements which help to improve navigation and orientation on the web. These enhancements are programmed as plugins to Scone proxy. Scone has four base and several support components. Base components are:

**Programmable proxy:** Interfaces for the access and manipulation of documents. The proxy is based on IBM's Web Based Intermediary (WBI).
**Net Objects:** Represent and store Web based data like URIs, HTML-documents, links, servers, users and their activities.
**Robot:** It can be used to whole web sites, but it was design to be used as user agent to gather specific information on the on client side, using a classifier-filter-concept, which allows the classification of links of documents depending on their attributes and the filtering of these objects according to these classifications
**AccessTracking**: It is used to trace the actions of a user when using a Web browser. Different kinds of access events can be generated, depending on whether he clicked a link, filled a form or pushed the back-button.

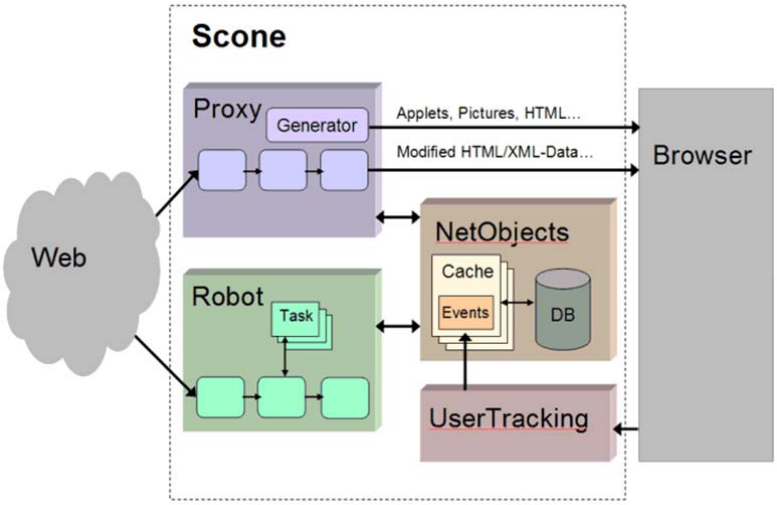



Figure 2.1.1. The Scone framework, showing the four base components

Many of these components are not necessary for the WRS to function properly. The WRS does not store any data about the pages visited (net objects, access tracking), nor does it track user actions or employ the robot. Installation of the database is not necessary as storage is not needed and connection to the database disabled.

The only component that the WRS is using the functionally of, is the Scone proxy. The proxy is used to intercept the calls to Wikipedia pages and extract the information about the visited page.

### *2.1.2 WRS Plugin*

The WRS plugin is developed as plugin for the Scone Proxy component, and presents the central governing part of the system. It consists of one main module WRS module and three utility modules. The flow of a HTML document in the WRS plugin is shown on Figure 2.1.2.1.

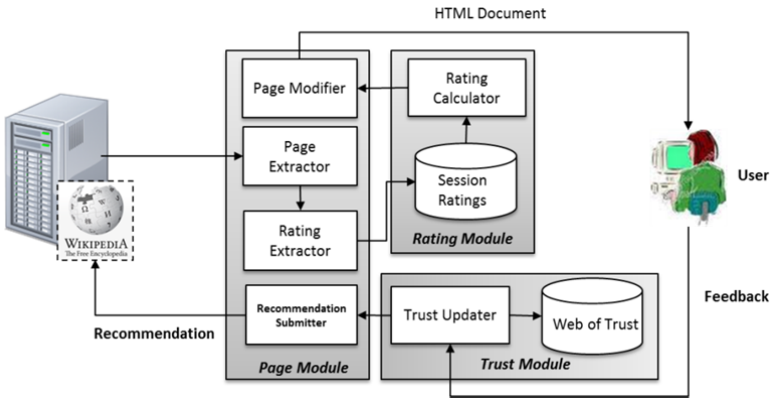



Figure 2.1.2.1 The flow of HTML document in WRS plugin

**WRS Module**. This module filtrates for incoming browser calls and manipulates only ones coming from *.wikipedia.org domain. If the page is from the wanted domain, the WRS module will coordinate page manipulation by calling other modules in appropriate sequence. Pages not coming from Wikipedia domain are ignored.

**Page Module.** The page module is responsible for all processing of HTML documents. It reads a Wikipedia article in form of HTML document extracts annotations from the mirroring recommendation page and inserts the rating, category and WRS GUI back to the HTML document containing the article [29]. All this is then forwarded to the user's browser, where the WRS GUI applet is opened in the separate window displaying article score and waiting for user feedback. Finally, when the user gives the feedback on the article, the annotations are prepared and inserted back to the Wikipedia.

**Rating Module.** When the Rating Module is invoked by Page Module, it first checks whether Wikipedia article identified by the page title and version number has already been rated by user [29]. If the user has already rated the article, the value is returned to the Page Module, otherwise, the Rating Calculator calculates a rating based on annotation retuned from Page Module. If annotations are coming from reviewers who are new to the user, ratings are stored in the Session Ratings, so that the trust value for them can be calculated when the user has provided the feedback. Rating Module facilitates the representation of the category in WRS. It provides a method (container) to aggregate rating and category data [28].

**Trust Module.** The Trust Module maintains the trust information about users with whom the WRS user had previous interactions. These interactions are positive or negative experiences the users have had, based on similarity of their ratings. Ratings closed to the user's own rating (within users rating ± 1) count as positive and the trust value in those users will increase and those that are far from user's own rating (within users rating ± 3) count as a negative experience the trust value for those users will decrease. Ratings that are between these two intervals are considered inconclusive and the trust values for those recommenders will not be updated. Rating Module includes both the local repository of trust values (users own local Web of Trust) and the functions that implement the WRS trust dynamics.

### *2.1.3 Trust metric embedded in WRS*

Using ratings and category as the choice for user credibility evaluation in WRS, two metrics are introduced. The rating metric defines the quality of writing and the structure of an article, whereas the category metric determines a user understanding of an article. The rating metric is considered to be the primary metric, because the most important is that an article is well written and structured, while the category metric is the secondary metric which is dependent on individual user`s abilities, for instance, perception of the information [28].

Introducing two metrics - rating metric as the primary and category metric as the secondary - the combination of metrics becomes complex. Five different possibilities / situations showed in Table 2.1.3.1 could have great influence on the user trust value variation. It is worth noticing that majority means the category which is supported by people the most, while minority - only few of them have assigned the category for a particular article [28].

| Weight | Rating | Category |
|---|---|---|
| 0.5 | Agree | Disagree |
| 1 | Agree | Agree |

| | | |
|---|---|---|
| -1 | Disagree | Agree |
| -1.5 | Disagree | Disagree with majority |
| -0.5 | Disagree | Disagree with minority |

**Table 2.1.3.1. Interaction with weight value**

The situation when the user agrees on rating, but disagree on category could be explained by the fact that the user has misunderstood the article. However, as the primary metric matches with the majority rating value, consequently entire interaction is positive with value 0.5. The disagreement of rating and majority of category reveals that the user misunderstood the area of expertise; therefore, the user is penalized with -1.5 on trust value [28].

The WRS containing embedded trust - metric enables us to determine the trust value for each category. This fact implies that the trust value has an impact on the weighting of the ratings. The more trusted user is within a particular category, the greater impact should the user have on the rating of articles which belong to that category. The weighting formula is [28]:

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

, where $t_{i,i}$ denotes the trust value of user *i* for category *j* and $c_{i,j}$ denotes the choice of category *j* by user *i*.

## 2.2 Trust propagation and Cold Start problem

### 2.2.1 The Cold Start problem

Cold Start is a potential problem in information systems which include automated data modeling. It occurs when the system cannot draw any inferences for the user or items about which it has not yet gathered sufficient information. Cold Start problem is widely spread in recommender systems. Typically, recommender system compares profiles of users to some reference characteristics. This characteristic can be certain characteristics of an item which are matched against features in the user profile, such as taste and preferences (content-based information filtering), or profiles of different users who share same preferences can be matched in order to propose items which like-minded users favor (collaborative filtering approach).

As WRS is a collaborative filtering type of recommender system, therefore it suffers from the cold start issue itself. Imagine the situation in which a new article is written in the Wikipedia and it has not been rated yet. A user browse the article using the WRS to find out the credibility of the article, and category in which the article is classified, but the system cannot calculate the values as ratings have not been submitted yet. After the user has rated the article, the cold start issue is reduced, as all users who have been given the

similar rating with the article reviewer, will have this rating as the starting point in determining the credibility of the article.

A similar situation occurs when a new user of the WRS browses the article. He will face the cold start problem no matter if it is a newly introduced article or the one that contains dozens of ratings. As WRS uses localized web of trust, until the new user rates the article and, in that way establishes the trust connection with like-minded users, the cold start problem will take its place. The same applies for all users with little interaction.

To summarize, cold start problem occurs when:

1. New article is introduced to the Wikipedia and has not been evaluated yet
2. New user is introduced to the Wikipedia Recommender System
3. Users try to evaluate articles with low number of reviews
4. Users with one or very few interactions with other users try to evaluate the article

The first two problems can be solved by introducing a neutral entity that the whole community would trust and let it give the first rating to the article. In that way both the new comers and mature users will have the first, neutral, but trusted reference to the reliability of the article. Another approach to this is to let users select their initial connections without having previous interactions with them. This requires data mining of the network in order to find out which recommenders would be suitable for initial connection. Such an approach is characteristic for centralized recommender system.

The second two can be solved by introducing recommendations and let users share part of their trust network by recommending users with whom they had positive interaction as trustworthy reviewers and, in that way, help their connections to faster build their network and reach for the reliable ratings. This solution can solved the problem number two too, as the top reviewers for category can be determined (ones who got the best recommendations) and their review can be used as the start for the new user.

The hybrid approach would give the best results as it solves the cold start issue entirely while the partial solutions lack objectivity or does not solve the problem with initial connections.

According to [23], Wikipedia is experiencing a slow-down in the number of new pages added each day and at the same time, number of people in developing countries who have access to the internet increases, we can expect that the demand for reliable articles will increase and therefor the number of WRS users. With the increased number of WRS users, comes the cold start problem and the challenge of connecting newcomers and building their trust network fast. Here the contribution of already established users and propagation of their trust via recommendations surface as the valuable source of solving the cold start issue.

### 2.2.2 Trust propagation

How many times have you heard an expression “It’s a small world”? As being almost a trilingual person, I can tell you that the same expression exists in English, Danish (“Det er en lille verden”) and Serbian (“Svet

je mali"), and probably in many other languages. Stanley Milgram conducted a set of experiments known as the "Small world experiment" where he examined the average path length for social network of people in United States which suggested that human society is a small world type network characterized by short path length. This experiment is often connected with the "Six degrees of separation" idea by Frigyes Karinthy everyone is on average approximately six steps away from any other person on Earth, so that a chain of, "a friend of a friend" statements can be made, on average, to connect any two people in six steps or fewer. Research conducted on Facebook, Linkedin and Twitter confirmed this theory as average degree of separation between people varied from 4 to 6.

The same idea of small world can be applied to trust networks, as trust networks are a special subset of social networks. We can assume that participants can be connected to each other in only a few steps. The fact that they can be connected does not immediately mean that they trust each other. There should be a way to determine to which degree participants in the trust network that can be connected trust each other. This is where trust metrics take place as their task is to compute a trust estimate based on the existing trust relations between linked users.

Trust metrics are often based on the assumption that trust is transitive. The question here is: "Is trust transitive"? In real life, I trust my brother Marko to know a lot about cars and he trusts Peter to be a good car mechanic. This can imply that I trust Peter to be a good mechanic and let him fix my car. But the same thing does not imply if my brother knows someone who can give a good haircut: as I trust him about cars, which does not imply that I trust him to recommend a good hairdresser. So, the trust is transitive but in the same context and under certain conditions. If I do not know anybody who can fix my car, I would probably rely on my brother's choice.

In the previous example, a clear distinction can be seen between functional and referral trust. I trust Marko's competence to access the trustworthiness of Peter as good car-mechanic (functional trust, Marko trusts Peter) and my trust in Marko to recommend a good recommender (referral trust, I trust Peter). The second distinction is made between direct and indirect trust. Marko trusts Peter as he had previous experience with him and he proved to be a good mechanic (direct trust) and I trust Peter based on Marko's recommendation (indirect trust).

It is the referral trust that allows trust to be transitive. Trust propagation is transitive chain of referral trusts paths, which ends with one functional scope.

If we go back to the small world and the idea that all participants can be reached in only a few steps, we can conclude that propagation of trust will not go to the eternity. However not all propagation paths will have the same length. Some users can be reached in the first step while other can be reached in twelfth. Intuition tells us that trust to recommender found in twelfth step will be less than trust in one found in first step. The same applies for the second, third etc. Therefore, it can be concluded that trust decreases along the chain and the highest trust significance will have the closest participants.

It has been shown in literature that issuing few trust statement, compare to the similar amount of rating information, the system can generate more and more accurate recommendations. Research has also shown that people tend to rely upon recommendations from people they trust, more than upon online recommendation systems which generate recommendations based on anonymous people similar to them.

The reason why trust-based recommendations work can partially be explained by the effect of social influence and selection in social networks. The theory of social influence states that users will behave more alike to users they interact with (i.e., become more similar), while selection (or homophile) denotes the

process of forming relationships to people who are already similar. In other words, there is an interaction between similarity and social (hence also trust) relations.

One group of trust-based recommendation approaches uses information coming from the trust network that is generated by the direct input of the users, by explicitly issuing trust statements. Such a strategy allows using trust propagation and aggregation in the network to infer the final trust values that are needed in the recommender algorithm. The other group does not use user's estimations; instead trust values are computed automatically. If no explicit information is available, one must rely on other information that is available, most often the ratings.

### 2.2.3 Trust propagation algorithms

The most common trust-based recommender systems such as Moleskiing [31] (a mounteering skiing community, Epinions [32] and FilmTrust [30] (a website that combines social networks with movie ratings.) ask their users to issue trust statement about other users and assign them certain trust value. The scale for expressing trust in another user is generally from 1 to 10 (FilmTrust) or 1-9 (Moleskiing). All systems use relations from the already established trust network to determine which opinions or ratings weight more or less in the recommendation process. These weights are estimates obtained by trust propagation or aggregation. There are several algorithms in use these days, but the most common ones and the once that have been implemented and tested in practice are TidalTrust developed by Golbeck et al. and applied in FilmTrust recommender system and MoleTrust, developed by Avesani and Massa for Moleskiing . These algorithms are selected not only because they are well-known and proved their value in practice, but also and because both are personalized and use local trust metrics the same as WRS.

**TidalTrust metric** is key mechanism implemented in FilmTrust recommender system. The authors conducted several experiments that lead to development of this algorithm [27]. In these experiments they ignored an existing trust relation from user *a* to user *b* and focused on all parts that connect users *a* and *b*. They've noticed that shorter paths yield more accurate trust estimations and, paths containing higher trust values yield better results.

If we take only first conclusion into consideration, shortest paths should give the best results. However, if limit is set on the path length, sometimes only few users can be reached. This trade-off is incorporated through a variable path length limit: the shortest path length that is needed to connect the target user *a* with user *b* that has rated the target item *i*, becomes the path depth of the algorithm. This can be achieved by performing a breadth-first search from *a* until the level is reached on which at least one user can be found who has rated i; the algorithm then knows all the shortest paths from *a* to users on that level who have rated *i*. Note that this process requires a central authority and that the depth of the breadth-first search varies from one computation to another.

One way of addressing the second observation (higher trust values on the path yield better trust estimates) is to limit the information such that only the most trusted users are taken into account. However, every user has his own behavior for issuing trust values (one user may give the maximum value quite often while another one never does), and in addition, it will often be the case that only a few paths contain the same high trust value. This is why it is difficult in practice to set a static threshold, so instead, thresholds are chosen dynamically: Golbeck et al. suggested to incorporate a value that represents the path strength (i.e., the minimum trust rating on a path leading to the user who is connected with *b*), and to compute the maximum path strength over all paths leading to the raters. This maximum (max) is then chosen as the minimum trust threshold for participation in the process. In other words, TidalTrust only takes into account

the shortest, strongest paths. The algorithm name (TidalTrust) was chosen because calculations sweep forward from *a* to *b* in the trust network, and then pull back from *b* to return the final trust value to *a*. Golbeck et al. have shown that using trust-based weighted mean in combination with TidalTrust does not necessarily offer a general benefit over computing the average or applying collaborative filtering, but that it does yield significantly more accurate recommendations for users who disagree with the average rating for a specific item.

TidalTrust algorithm calculates the trust in unknown user in the following way: The trust value from target user Alice (*a*) in unknown user (*u*) is estimated recursively as with WOT(*a*) the set of users in WOT(*a*) for whom *a*'s trust statement is greater than or equal to the given threshold max. If WOT (*a*) is empty, then trust $t_{a,u}$= 0.

$$t_{a,u} \frac{\sum_{v \in WOT(a)} t_{a,v} \cdot t_{a,u}}{\sum_{v \in WOT(a)} t_{a,v}}$$

, where WOT(a) is Alice's Web of Trust and $t_{a,v}$ is trust in user *v*.

I have selected not to present the pseudo code of the algorithm but to show the calculation on the concrete example.

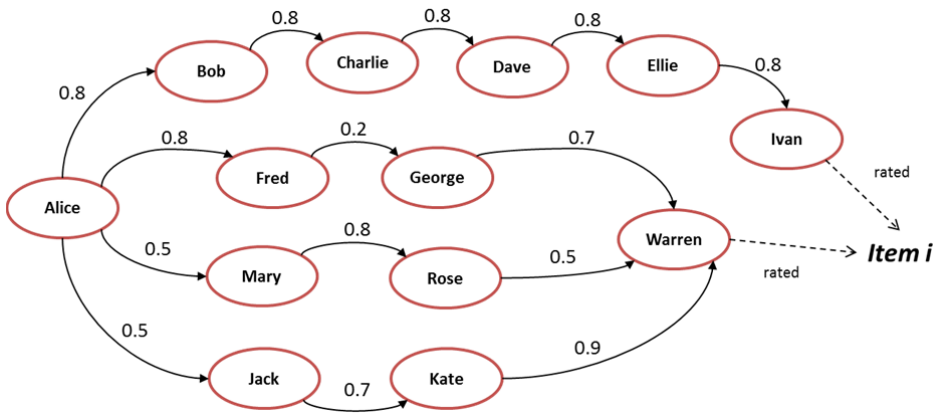



**Figure 2.2.3.1 - Example of a trust network for generating a recommendation for user Alice for item i**

In the trust network shown in the Figure 2.2.2.1, there are 4 propagation paths that lead from *Alice* to users *Ivan* and *Warren*, who have rated the item *i*. The algorithm consists of two phases; in the first phase, it will find the shortest path to the user who has rated the item. The upper path will be excluded from calculations as it is longer than shortest path to the item (length 3). Among the other 3 path only two lowest will be taken into account and contribute to final recommendation as *max = max(min(0.8,0.2); min(0.5,0.8); min(0.5,0.7)) = 0.5* and the path strength of the second path is merely 0.2. In the second phase, actual trust estimate is computed recursively as a weighted mean of trust values for all users that are a first link on the shortest path from the source to the sink.

$$trust\ (Alice, Warren) = \frac{0.5 \cdot \left(\frac{0.8 \cdot 0.5}{0.8}\right) + 0.5 \cdot \left(\frac{0.7 \cdot 0.9}{0.7}\right)}{0.5 + 0.5} = \frac{0.5 \cdot 0.5 + 0.5 \cdot 0.9}{1.0} = 0.7$$

Alice's trust in Warrens rating of the item *i* calculated using TidalTrust matric is 0.7.

**MoleTrust metric** is recommendation algorithm developed by Massa et al. and incorporated into the Moleskiing application. It consists of two phases: in the first phase all cycles are removed from the network and all users are stored in the array according to their distance from user *a*. In the second step trust computation (trust propagation) is performed on the cycle free network using the trust propagation. Massa et al. acknowledge that removing cycles from the network does remove trust statements that can be informative but claim that technique is acceptable because of time-efficiency. The main reason for removing cycles is that they would be visited over and over again until the propagation horizon is reached. This is another characteristics of MoleTrust algorithm, trust is not computed for eternity so before the computation begins, propagation horizon value must be set. Only users who can be reached within horizon are taken into account. Another parameter that needs to be set is trust threshold. On the scale (0 to 1, where 0 is complete distrust and 1 is complete trust) this value is set to 0.6. This means that only paths with the trust value equal or greater than trust threshold are taken into account. This value is static unlike the previous TidalTrust algorithm where the max value is determined dynamically. While the first phase consists of the search and storage of the users, the second phase deals with the computation of the trust estimates in users on distances m > 1. The trust in the users at distance 2 or more is calculated in a way similar to TidalTrust, in the sense that it is also a weighted mean-based algorithm. However, the details of the implementation differ significantly. In TidalTrust, a user *v* is added to WOT(*a*) only if he is on a shortest path from target user *a* to target item *i*. On the other hand, in MoleTrust, WOT(*a*) includes all users who have rated the target item and who can be reached through a direct or propagated trust relation. MoleTrust ensures that only the shortest paths from *a* to another user are taken into account, so if there is a direct trust between users, propagation will not take place. If the match is found in the second step, it will not go to third etc. The same weighted-mean formula is used for calculation:

$$t_{a,u} \frac{\sum_{v \in WOT(a)} t_{a,v} \cdot t_{a,u}}{\sum_{v \in WOT(a)} t_{a,v}}$$

MoleTrust works with a central authority and starts with a breadth-first search phase to find all shortest paths for every user u within the trust horizon d who has rated i. In the second phase, the trust values $t_{a,u}$ are computed as the weighted mean of trust values $t_{v,u}$ for all users v for whom the trust value $t_{a,v}$ is directly available, or, if not, the trust estimate $t_{a,v}$has already been computed; only trust information that is at least as high as trust threshold taken into account. In other words, the second phase in MoleTrust is implemented as a forward movement (as opposed to the backward, recursive, wave in TidalTrust.

Massa and Avesani have illustrated that MoleTrust provides better trust estimates than global trust metrics such as eBay's, especially when it comes down to estimating the trust in controversial users (who are trusted by one group and distrusted by another) They also showed that MoleTrust yields more accurate predictions for cold start users, compared to a classical collaborative filtering system.

Using the previous example of trust network, Alice's trust in Warrens rating of item *i* calculated using the MoleTrust metric (trust horizon = 0.5) is:

$$trust\ (Alice, Warren) = \frac{0.8\ \cdot 0.5 + 0.7 \cdot 0.9}{0.8 + 0.7} = 0.69$$

### 2.2.4 Guiding Cold Start users through initial connections

The easiest way of reducing the cold start problem is to let new users to rate several articles in different categories until their trust profile is boosted enough so they have enough direct connection for system to generate rating prediction without problems. However, this is a tiring and time consuming process and not many users would accept to do such a thing.

Most approaches designed to reduce the cold start problem combine rating data with content data trying to build up a hybrid system where aspects of content-based filtering is incorporated in recommendation process. But there are also other approaches that think that the trust network plays and important role in rating prediction generation for new users and that they should have be guided through connection. Here are two approaches which suggest who cold start users should aim to connect.

#### *2.2.4.1 Trust-aware recommendation system*

Massa and Avesani in [26] gave a known and extensive study about trust-based collaborative filtering conducted on the real world, large dataset (Epinions-products dataset).

The first result which they have pointed out is that considering only ratings of direct trusted users gives the smallest error with an acceptable coverage. With regard to cold start users, they showed that standard Collaborative filtering techniques fail or are not able to generate any recommendations and that considering ratings of trusted users with a very small error system is was able to produce recommendation for almost 17% of users. Massa and Avesani state that it is better to make a few connections in the trust network instead of rating a similar amount of items to bootstrap the system for cold start users. For users that have rated three (four) items, the coverage that collaborative filtering can achieve is 4% (8%), while the trust-based variant reaches 53% (59%) of all items. This is a significant indicator how the trust statement and trust propagation play an important role in generating predictions for the cold start users.

To summarize, authors proposed trust propagation to fight the cold start problem and to let users themselves to choose whom they should initially connect. This implies that the expressed trust recommendations should be available publically, so that the user can select which users he might trust and add them to his WOT as initial connections. That is where the name “trust-aware” comes from as similarity is not used in determining which users should be connected only the directly expressed trust statements and propagated trust.

#### *2.2.4.2 Key Figures in the Trust-based Recommender’s Network*

Patricia Victor in [25] suggests that cold start users should be guided through connections by connecting to trust network’s key figures. She divides key figures into three classes: mavens, frequent raters and connectors. Mavens are people who write a lot of reviews. They are knowledgeable people who want to share their wisdom to others. Out of the three classes, mavens are the most visible to new users and easiest to form opinion about. She states that: “the more reviews someone writes, the better a new user can form an opinion about him and decide to put him in his personal web of trust or not”.

The second class of key figures consists of frequent raters. They don’t necessarily write a lot of reviews but they evaluate a lot of them and therefore provide a lot of ratings which are the key element of recommender system as you can’t have predictions without ratings. Frequent raters are not always so

visible as mavens. Including a frequent rater in a trust network, more items can be reached (in WRS system more articles can be reached) and thus better system coverage.

The third class of key figures is formed by connectors. Connectors are people who connect a lot of other users and occupy a central position in the trust network. They issue a lot of trust statements (user recommendations) and are often at the receiving end as well (trustworthy recommenders). Unlike the mavens and frequent raters, their strength is not in visibility or rating capacity, but in their ability to reach a large group of users through trust propagation. When a trust propagation algorithm has to find a path from one user to another, connectors are often part of the propagation chain and propagation path will on average be shorter than other paths (mainly compare to the random user). Shorter chains, as discussed before, have better influence on accuracy of trust estimations and recommendations.

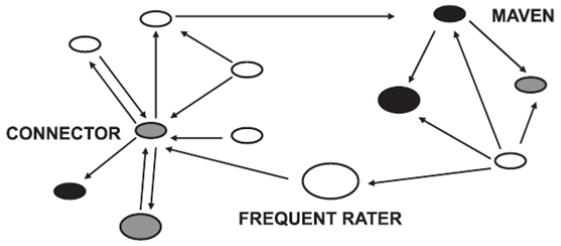



Figure 2.2.4.2.1 - Trust based recommender's system network with example of key figures [2]

## 2.3 Summary

In this chapter current achievement in WRS is described. WRS functions as part of Scone proxy, which enables WRS to intercept calls to Wikipedia pages and coordinate page manipulation by calling other modules of the system. These modules are Page Module, Rating Module and Trust Module. Page Module is responsible for processing of HTML documents, extraction of article recommendations, insertion of the GUI applet and upload of the annotations containing user's feedback back to the Wikipedia. Rating Module calculates the rating of the article, based on annotations returned from Page Module. Trust Module maintains the trust information about users with whom the WRS user had previous interactions. These interactions are positive or negative experiences the users have had, based on similarity of their ratings.

WRS uses two metrics to determine user's credibility: rating metric and category metric. Rating metric is considered primary metric and defines the quality of writing and the structure of the article, while category metric is secondary and determines a user understanding of the article. Users classify the article in one of the 15 categories (Arts, Business, Computers, Games, Health, Home, Kidsteens, News, Recreation, Reference, Regional, Science, Shopping, Society, Sport). Two metrics, combined together give an overall rating of the article percentage of belong to the category in which majority of users has assigned to it. They also enable storing two trust values per user, one general trust in user and trust for each category.

Trust propagation is introduced as way to reduce the cold start problem. Trust propagation is transitive chain of referral trusts paths, which ends with one direct trust connection. Referral trust is trust based on recommendations. Research showed that people tend to rely upon recommendations from people they trust, more than upon online recommendation system which generates recommendations based on

anonymous people similar to them. Recommending other users as trustworthy reviewers and, in that way, propagating your trust, can help cold start users reach more trusted reviews.

In order to calculate propagated trust in unknown user, appropriate algorithm is needed. Two currently most cited are presented: TidalTrust (implemented in FilmTrust recommender system) and MoleTrust (implemented in Moleskiing application). The reason for selecting to present these two algorithms is that, they are well-known in literature, they proved their value in practice and they are both personalized and use the local trust metric (the same as WRS). Trust in unknown user is calculated as weighted mean of all trust paths leading from target user to unknown user. Difference is that, TidalTrust determines minimal trust threshold on the path and path length dynamically, while MoleTrust demands these two parameters to be predefined. Therefore, MoleTrust is less exhaustive but not less precise.

For trust propagation to work, a user needs to have at least one connection. In literature proposals for how to make initial trust connections can be found. Authors of MoleTrust, Avesani and Massa, suggest that it is better to make few trust connections instead of rating the similar amount of items to bootstrap the trust network. They leave choice to user to select whom to connect initially. P. Victor suggests that cold start users should be guided through connections by connecting to trust network's key figures. She divides key figures into mavens (people who write a lot of reviews), frequent raters (people who provide a lot of evaluations) and connectors (people who issue a lot of recommendations). The prerequisite for such a solution to work is that, trust network (or part of it) is centralized and can be data-mined.

# Chapter 3: Analysis

In this chapter work presented in the State of the Art chapter is analyzed from the perspective of the WRS system itself and the main problem defined in this thesis, the cold start problem. The first part is analysis of the current version of the WRS system, its functionality, usage of the Wikipedia as a repository for storing recommendations, security issues and suggestions for necessary improvement need to be implemented to accommodate the cold start problem solution. The second part is analysis of the current solutions in theory and practice, used to fight the cold start problem, and how they can be applied in the WRS. The result is a specification containing the requirements for the improved version of WRS, which has been later elaborated in the following Design and Implementation chapters.

## 3.1 Wikipedia as a Repository

In the previous chapter we have seen that WRS uses Wikipedia as storage for storing ratings. Ratings are stored in subpages matching the article name of special user called "Recommendations". In order to do so, WRS uses a Java based API which enables the connection to Wikipedia, creating and updating of the articles and other content on the Wikipedia. As being an open source, the API is often changed by contributors to the point that the previous versions are completely unusable to the system (Last change was introduced on 26. Of March 2011, just before the work on this project has started). WRS incorporates Wiki API into the solution, so each time the API has been changed, the solution need to be recompiled and re-distributed to the users. As an example, the last version of software created by Povilas Pilauskas cannot be used as the *edit* function in the API is changed and ratings cannot be uploaded to the Wikipedia.

Another, even more serious problem occurred when trying to prepare the ground for uploading users' recommendations used to solve the cold start problem. The initial idea was to store recommendations of the particular user, submitted by other users, in the subpage "Recommendations" of the user's Wikipedia user page (ex. http://en.wikipedia.org/wiki/User:npandersen/Recommendations). During the testing of the solution, while creating multiple accounts from the same machine and several constant edits in the short time period, one of the accounts was blocked. It was spotted by the Wikipedia abuse filters and blocked. No matter if this was due to the fact that heavy edits during testing caused the problem, or Wikipedia's abuse filters got more sophisticated, the risk of having the same problem with the final solution was too high. Therefore, the decision to abandon Wikipedia as storage has been made and idea to have a separate server where ratings and recommendations can be stored was introduced.

> *R1. The WRS system should provide storage for article ratings and user recommendations independent from the Wikipedia*

Having a separate storage for ratings and recommendations removes the need to have Wiki API as part of the solution, as Wikipedia pages will not be edited any more, but leaves the problem of user accounts as

the API was used to authenticate Wikipedia users. To be a WRS user one needed to be a registered Wikipedia user.

At this point, the second decision has been made - to have user accounts stored on the separate server together with ratings and recommendations, so that WRS users do not necessarily have to be registered Wikipedia users any more.

> *R2. WRS user does not necessarily have to be a registered Wikipedia user. The WRS enables storage and management of users' profiles.*

## 3.2 Trust Propagation in decentralized system

The trust propagation in a decentralized system needs the information to be distributed amongst the users. According to the information privacy policies, trust information can be shared publicly only if user accepts to expose his trust to the other users. The current WRS does not violate this rule as annotations that are uploaded to Wikipedia do not contain any trust information. They need to be proceeded by each entity individually in order to obtain trust information. Thomas Lefevre discusses in [12] that, due to privacy policies, sharing of information in distributed recommender systems becomes problematic because the systems are not symmetric in terms of trust. Consider a statement where *Alice* trusts *Bob* for *0.7*, this statement implies nothing about how much *Bob* trusts *Alice* in an asymmetric system. Thus if *Alice* trusts *Bob* and wishes to propagate the trust values of *Bob* to herself, *Bob* needs to trust *Alice* first. Because of this requirement of mutual trust, trust propagation in this way suddenly becomes much less effective. If *Alice* and *Bob* trust each other enough to share trust values, it is then questionable if it as at all necessary. As trust propagations primary purpose is to combat cold start problems, but if both *Alice* and *Bob* trust each other their trust profiles must have matured to a level where there is no longer a cold start problem. Some authors faced with the difficulties with trust propagation in decentralized recommender systems try to bypass it by ignoring the privacy issues. This seems as a not very good idea. Privacy advocates are becoming more and more sound in their endeavor to defend people's privacy, and a good example of it is the privacy control improvement which Facebook introduced few days ago, where you cannot publish a picture of someone if you get his approval. This is something that should be taken into consideration when finding the best solution to cold start problem in WRS.

That cold start problem is a big issue in recommendations systems shows the study of Massa and Avesani [26]. In the research that they have conducted on Epinions dataset, the cold start users make more than 50% of total users. If that is the case with WRS, which can be assumed, the system would fail to provide recommendation for 50% of the users. Therefore, trust propagation is a needed but with consideration of the privacy.

> *R3. WRS system enables trust propagation by letting users to issue trust statement about other users with whom they had positive previous interactions.*

Most recommendation systems propagate trust by letting users to form a list of their trusted users and sharing it with the rest of community. The problem with this idea is that "trusted users" cannot delete themselves from such list if they do not trust the recommender or if they do not want, for some reason, to be part of the propagation chain. The idea for trust propagation in WRS is a bit different. The suggestion is that each person has its own list of recommendations formed from the recommendations that other users

submitted. In this way, a user can control its own recommendations and decide to delete or disapprove suspicions recommendations and in that way choose not to be part of the propagation chain.

> *R4. Each user has its own recommendation list formed from the recommendations submitted by the other users.*

> *R5. User can manage its own recommendation list by choosing to delete recommendations which he/she finds suspicious or unacceptable.*

Due to the privacy policy the WRS should not submit user's recommendations automatically (although this would be an easier way to generate recommendations quickly) but rather let user decide and confirm which people should be recommended. In that way, only publicly exposed trust data is trust values in users' recommendations. These values are published only if user chooses to publish them, and WRS Server should not store or publish any trust data without the user doing it self.

> *R6. WRS system does not expose user's trust values publically without user's confirmation.*

From the system performance point of view, decentralized systems decrease a great deal of load from the central server. As all calculations are done on the client, the central server is used only as storage with services which provide querying and publishing of the stored data.

> *R7. WRS is a decentralized system.*

## 3.3 Local trust metric

WRS uses local trust metric to calculate trust in recommended rating for the article. This means that the derived rating is based on the personalized calculation based on the interaction a user previously had with users who have rated the article. Only ratings of the trusted users are taken into account and the recommended rating is calculated based on their rating and the user's trust values in trusted users as good reviewers. Global trust metrics take into account all ratings of the considered article. There is no reason to change this approach and use the global trust propagation metric. Both trust metrics presented in the previous chapter are localized.

> *R8. WRS uses localized trust profiles. Web of trust is created and managed by each user locally on the user's own machine.*

TidalTrust and MoleTrust use the same weighted average algorithm to calculate the trust in unknown user. However, there are slight differences: TidalTrust runs both forward and backward as it first searches for the minimal paths and afterward calculates the values, while MoleTrust demands trust propagation horizon and minimal trust value in the path to be set in advance. Thus, TidalTrust demands more complicated calculation and longer execution time; it takes into account all propagation paths. MoleTrust runs only in one direction so the calculation is less expensive, but by limiting the length of the propagation path and trust on edges it lowers the number of paths taken into the consideration. The algorithm can be tuned by setting the minimal trust value to 0.1 (on scale [-1 , 1] where -1 is a complete distrust and 1 is a complete

trust) as this value includes all positive interactions and setting the trust horizon to 6 (following the small world experiments that every person can be reached in 6 steps). Experiments show that propagation horizon of 3 steps is good enough. According to Massa and Avensani, trust propagation horizon is a trade-off between accuracy and coverage i.e. by increasing the distance to which trust is propagated by the local trust metric the prediction coverage increases but the error increases as well.

The proposition for WRS is to use MoleTrust algorithm but to enable usage of other algorithms in the future, if they prove to be more accurate. The MoleTrust algorithm is selected for its simplicity and ability to adjust above mentioned parameters. The WRS should enable users to set these parameters easily, so that while their trust profile increases, the horizon can be lowered and minimal trust value increased in order to get more accurate predictions. The second proposal is to make WRS an open-source system and enable researchers to use the publish WRS data in testing their algorithms or maybe creating a new ones. The decentralized character of the WRS enables users to build their own client applications. Instead of limiting the system of having one or two functions for calculating the trust values, the open-source nature of WRS challenges the researchers to provide more accurate calculations and contribute to more reliable article ratings.

> *R9. WRS has an open-source nature. The solution enables researchers and other contributors to build their own client applications by exposing stored data publically and easy to process by other applications.*

The challenge in proposed solution is which trust values should be exposed for trust propagation. There are 2 trust values that current version of WRS is storing: general trust in user and trust on category. As trust in category reveal more information than general trust, users should propagate their trust in other users in certain category. These propagated trust values will be used in calculating the trust in certain category in unknown user.

> *R10. WRS calculates trust in unknown user using the weighted mean of all trust paths leading from target user to unknown user. Maximum path length and minimum trust value on the path are predefined and can be easily adjusted.*

The algorithm for calculating the trust in a user's rating in WRS can be explained by following scenario: Assume that WRS user Alice (a) wants to know the credibility of a Wikipedia article *i.* She starts the WRS system and the systems starts to search for the trusted ratings. Let's assume that user *u* has rated the article *i* and assigned it category *c*. If *a* has previous interaction with *u* in category *c*, *a*'s web of trust (*WOT(a)*) contains *u,* and trust in *u*'s ratings can be calculated. In the opposite case, *a* is facing cold start problem. The system is then extracting all users from *a*'s WOT to find those with whom *a* had previous positive interactions in category *c.* Let's assume that *v* is one of them. The two variables used in the algorithm are *trust_horizon*, used to determine propagation path length and *threshold* used to determine minimum trust value on the path. Let's assume that *trust_horizon=3 and threshold=0.1*. The system will, then, search for recommendation given by *v* to user *u* on category *c*. If such recommendation exists, trust in *u* can be calculated using the weighted mean algorithm formula.

$$t_{a,u,c} \frac{\sum_{v \in WOT(a)} t_{a,v,c} \cdot t_{v,u,c}}{\sum_{v \in WOT(a)} t_{a,v,c}},$$

where $t_{a,u,c}$ is trust of user *a* in user *u* on category *c,* $t_{v,u,c}$ is trust of user *v* in user *u* on category *c and WOT(a)* is web of trust of user *a* extended with trusted reviewers recommended by users contained in *WOT(a).*

The algorithm will stop here, if the match has been found already in the first step. Else, the calculation continues as the new WOT will be formed by all users who have recommended *v* on category *c*. For each of them, the system will search for their recommendation given to the article reviewer *u,* etc.

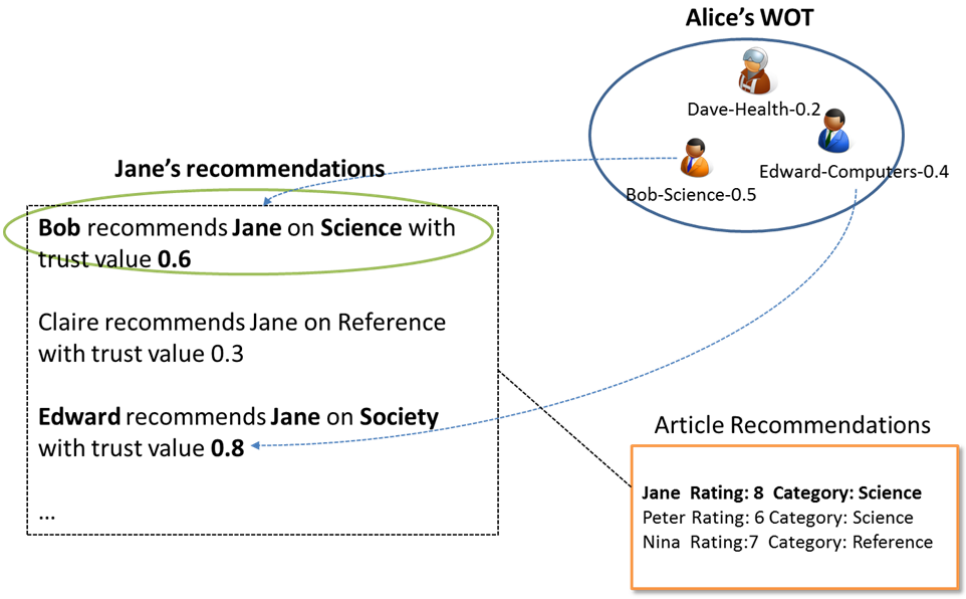



Figure 4.3.1 – Process of finding trusted recommendations

## 3.4 WRS security

The current WRS system has one pitfall that has been reported in the previous work by Povilas Pilauskas. User credential, username and password, used to login to the Wikipedia are sent unencrypted over the network. This is an important security issue which should be removed in the new version of the system. Further, in the solution which includes trust propagation and submitting of the recommendation which contain trust information, such sensitive information should be protected. This lead to the conclusion that the whole communication between client and server side of the system should be done over the secure connection.

> *R11. Communication between the client and the server side of the WRS system is encrypted and done over the secure connection.*

The valuable contribution to the project would be to perform a throughout analysis of the possible attacks to the system. One of the common attacks is in recommender systems is Sybil attack[1], where attacker creates multiple identities in order to disproportionally influence the trust network. Security analysis help determine the trustworthiness of the recommender system.

[1] http://en.wikipedia.org/wiki/Sybil_attack

## 3.5 Initial connections

The proposed solutions by Avesani and Victor for making initial connections are possible to incorporate with the current WRS system, but they do not sound as a good solution. WRS is not a complete trust-aware system as the trust in WRS user is not established just by simple trust statement, but by trust based on the similarity of the ratings given to the article. Victor's solution on the other hand would demand a data mining of the recommendations network on the WRS server in order to find, who are the most trusted reviewers of the article, or who are the people who made most recommendations etc. In the end, it leads to the same principal as Avesani suggested.

As WRS is a decentralized system which uses WRS server only as a storage, the previously mentioned solutions would demand central computational unit that would data-mine the recommendation network or the user's own effort to browse and search published ratings and recommendations. Trust propagation seems as a good enough solution to reach sparseness and the user will still face the cold start problem but in a reduced degree. Proposals for making initial connections can be discussed deeper in future work on the project.

## 3.6 Summary

In this chapter analysis of the current WRS system is conducted together with the solution for the cold start problem. Analysis is performed with the aim to discover the problems in the current implementation of the system and to define the necessary changes need to be done to include the colds start problem solution. Appropriate trust propagation strategy is determined based on the current achievement in the area presented in the State of the Art chapter. During the analysis requirements to the system are noted and complete list can be summarized as system specification in the following.

| |
|---|
| *R1. The WRS system should provide storage for article ratings and user recommendations independent from the Wikipedia.* |
| *R2. WRS user does not necessarily have to be a registered Wikipedia user. The WRS enables storage and management of users' profiles.* |
| *R3. WRS system enables trust propagation by letting users to issue trust statement about other users with whom they had positive previous interactions.* |
| *R4. Each user has its own recommendation list formed from the recommendations submitted by the other users.* |
| *R5. User can manage its own recommendation list by choosing to delete recommendations which he/she finds suspicious or unacceptable.* |

| |
|---|
| *R6. WRS system does not expose user's trust values publically without user's confirmation.* |
| *R7. WRS is a decentralized system.* |
| *R8. WRS uses localized trust profiles. Web of trust is created and managed by each user locally on the user's own machine.* |
| *R9. WRS has an open-source nature. The solution enables researchers and other contributors to build their own client applications by exposing stored data publically and easy to process by other applications.* |
| *R10. WRS calculates trust in unknown user using the weighted mean of all trust paths leading from target user to unknown user. Maximum path length and minimum trust value on the path are predefined and can be easily adjusted.* |
| *R11. Communication between the client and the server side of the WRS system is encryted and done over the secure connection.* |

**Figure 4.6.1 - WRS system requirements specification**

# Chapter 4: Introduction to Linked Data

In this chapter an introduction to the sematic web is given together with the description of the core technologies which form the semantic web. According to the W3C[2], Semantic Web is a web of data. The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. When thinking how to meet the requirement of open-source nature of the WRS, semantic web sounded as a good solution. Exposing WRS data as linked data in both human and machine readable format seemed as a reasonable solution which would in the end allow different application to easily read and process WRS data.

The decision to include this chapter in the project is made for the easier understanding of the terminology and solutions given in the Design and Implementation chapters, which are based on the technology explained here. The assumption is that the reader is already familiar with the presented concepts as they are already widely spread in the recommender systems, and the goal is just to give a brief overview.

## 4.1 Linked Data

The current World Wide Web is a mesh of “Linked Documents” connected by hypertext links. The purpose of each link is to connect one document to another. This widely used interaction pattern is facilitated by a dual resource identification/location facility called a Uniform Resource Locator (URL) and a messaging protocol known as the Hyper Text Transfer Protocol (HTTP). However, Tim Berners-Lee vision of the Web was broader than the current “Web of Documents” and implies to Web as an information space connecting disparate data spaces as part of a granular "Web of Linked Data Items". In such a form of the Web, each item of data (a datum) is uniquely identified by a resolvable identifier called an HTTP URI and applicability extends to real-world objects; basically a scheme similar to Object or Entity Identity in the distributed database management technology realm. According to Berners-Lee [36]:

*“Like the web of hypertext, the web of data is constructed with documents on the web. However, unlike the web of hypertext, where links are relationships anchors in hypertext documents written in HTML, for data they links between arbitrary things described by RDF. The URIs identify any kind of object or concept. But for HTML or RDF, the same expectations apply to make the web grow:*

1. *Use URIs as names for things*
2. *Use HTTP URIs so that people can look up those names.*
3. *When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)*
4. *Include links to other URIs. so that they can discover more things. “*

[2] http://www.w3.org/2001/sw/

Resource Description Framework (RDF) provides the foundation for Linked Data. RDF is a framework for describing Web addressable resources through associated metadata. RDF (and supporting layers like RDFS and OWL) support the Entity-Attribute-Value (aka Subject-Predicate-Object) + Classes & Relationships metadata model (EAV/CR), a model that's been used since the inception of modern computing, long before the Web.

## 4.2 Ontology

### 4.1.1 Definition

Ontology in computer science is defined as formal representation of knowledge, a set of concepts within a domain, and the relationships between those concepts. It can be used to reason about the entities within that domain, and may be used to describe the domain.

Ontology consists of four concepts: *Classes*, *Relationships* between classes, *Property* of classes and *Constrains* on relationships between the classes and properties of the classes. *Individuals* present instances of the classes. By considering the ontology as a graph, concepts are assigned to vertices and directed edges refer to relationships between concepts. Ontologies play a significant role in Semantic web technologies.

### 4.1.2 Ontology Description Languages

**RDF** (Resource Description Framework) is a general-purpose language for representing information in the Web. RDF provides mechanism for recording statement about resources so that machines can easily interpret the statements. A basic grammar rule is that a complete sentence (or statement) contains a *subject*, a *predicate* and an *object*. The subject is the who or what of the sentence, the predicate provides information about the subject and the object presents the matching value of that information. In RDF such sentences translates to RDF *triple*. In RDF, the *subject* is the thing being described; a *resource* identified by URI, the *predicate* is the property type of the resource, such as an attribute, a relationship, or a characteristic. The *object* is the value of the resource property type for the specific subject. Such a triple an RDF directed graph, forming a pattern of node-arc-node.

{subject, predicate, object}

Each RDF triple is a complete and unique fact about the resource.

*"Alice likes swimming"*

The previous sentence is an example of RDF triple where *Alice* is a subject, *likes* is a predicate and *swimming* is an object.

**RDFS** (RDF Schema) provides resources with classes and establishes connections between relations and classes in order to build class hierarchies.

**OWL** (Web Ontology Language) is a language for making ontological statements, developed as a follow-on from RDF and RDFS, as well as earlier ontology language projects including OIL, DAML and DAML+OIL. OWL is intended to be used over the World Wide Web, and all its elements (classes, properties and individuals) are defined as RDF resources, and identified by URIs. OWL not only provides the capabilities of RDF(S) but adds additional constrains that increase the accuracy of implementations of a given vocabulary. RDFS imposes fairly loose constrains on vocabularies. There is nothing in the schema that restricts the cardinality of a specific property or constrains property values. Additional information about two properties can be provided, such as that they are disjoined or transitive. Further, OWL allows additional information to be inferred about the data, though it may not be specifically recorded.

## 4.3 FOAF

The FOAF (Friend-Of-A-Friend) project is based around the use of *machine readable* Web homepages for people, groups, companies and other kinds of thing. To achieve this , "FOAF vocabulary" is used to provide a collection of basic terms that can be used in these Web pages. At the heart of the FOAF project is a set of definitions designed to serve as a dictionary of terms that can be used to express claims about the world. The initial focus of FOAF has been on the description of people, since people are the things that link together most of the other kinds of things we describe in the Web: they make documents, attend meetings, are depicted in photos, and so on.

FOAF files are simple text documents based on the conventions of the Resource Description Framework (RDF), Web Ontology Language (OWL) and may be written in XML syntax or any other of the syntaxes of RDF such as RDFa or N3.

FOAF defines categories ('classes') such as `foaf:Person`, `foaf:Document`, `foaf:Image`, alongside with their properties, such as `foaf:name`, `foaf:mbox` (ie. an internet mailbox), `foaf:homepage` etc., as well as their relationships `foaf:knows` and references to other document (`rdfs:seeAlso`), creating a linked data system.

**Table 4.3.1 - FOAF Vocabulary**

| FOAF Basics | Personal Info | Online Accounts / IM | Projects and Groups | Documents and Images |
|---|---|---|---|---|
| • Agent<br>• Person<br>• name<br>• nick<br>• title<br>• homepage<br>• mbox<br>• mbox sha1sum<br>• img<br>• depiction (depicts)<br>• surname<br>• family_name<br>• givenname<br>• firstName | • weblog<br>• knows<br>• interest<br>• currentProject<br>• pastProject<br>• plain<br>• based_near<br>• workplaceHomepage<br>• workinfoHomepage<br>• schoolHomepage<br>• topic_interest<br>• publications<br>• geekcode<br>• myersBriggs<br>• dnaChecksum | • OnlineAccount<br>• OnlineChatAccount<br>• OnlineEcommerceAccount<br>• OnlineGamingAccount<br>• holdsAccount<br>• accountServiceHomepage<br>• accountName<br>• icqChatID<br>• msnChatID<br>• aimChatID<br>• jabberID<br>• yahooChatID | • Project<br>• Organization<br>• Group<br>• member<br>• membershipClass<br>• fundedBy<br>• theme | • Document<br>• Image<br>• PersonalProfileDocument<br>• topic(page)<br>• primaryTopic<br>• tipjar<br>• sha1<br>• made(maker)<br>• thumbnail<br>• logo |

An example of the FOAF file of the person Bob Brown in RDF/XML format is given below:

```
<rdf:RDF>
<rdf:Description rdf:about="http://wrs.imm.dtu.dk/dataspace/person/bob#this">
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
</rdf:Description>
<rdf:Description rdf:about="http://localhost:8890/dataspace/person/bob#this">
    <foaf:name>Bob</foaf:name>
</rdf:Description>
<rdf:Descriptionrdf:about="http://wrs.imm.dtu.dk/dataspace/person/bob#this">
    <foaf:family_name>Brown</foaf:family_name>
</rdf:Description>
<rdf:Description rdf:about="http://wrs.imm.dtu.dk/dataspace/person/bob#this">
    <foaf:gender>male</foaf:gender>
</rdf:Description>
<rdf:Description rdf:about="http://wrs.imm.dtu.dk/dataspace/person/bob#this">
    <foaf:nick>bob</foaf:nick>
</rdf:Description>
<rdf:Description rdf:about="http://wrs.imm.dtu.dk/dataspace/person/bob#this">
    <foaf:mbox rdf:resource="mailto:bob@bob.com"/>
</rdf:Description>
<rdf:Description rdf:about="http://wrs.imm.dtu.dk/dataspace/person/bob#this">
    <foaf:mbox_sha1sum>cayrPRHGAUZYO+YSl4yBGSC7m3k=</foaf:mbox_sha1sum>
</rdf:Description>
<rdf:Description rdf:about="http://wrs.imm.dtu.dk/dataspace/person/bob#this">
    <rdfs:seeAlso rdf:resource="http://wrs.imm.dtu.dk/WRS/recommendations/bob#this"/>
</rdf:Description>
<rdf:Description rdf:about="http://wrs.imm.dtu.dk/dataspace/person/bob">
    <foaf:maker rdf:resource="http://wrs.imm.dtu.dk/dataspace/person/bob#this"/>
</rdf:Description>
<rdf:Description rdf:about="http://wrs.imm.dtu.dk/dataspace/person/bob">
    <foaf:primaryTopic rdf:resource="http://wrs.imm.dtu.dk/dataspace/person/bob#this"/>
</rdf:Description>
<rdf:Description rdf:about="http://wrs.imm.dtu.dk/dataspace/person/bob#this">
    <foaf:holdsAccount rdf:resource="http://wrs.imm.dtu.dk/dataspace/bob#this"/>
</rdf:Description>
<rdf:Description rdf:about="http://wrs.imm.dtu.dk/dataspace/person/bob#this">
    <foaf:made rdf:resource="http://wrs.imm.dtu.dk/ods/describe?uri=mailto%3Abob@bob.com"/>
</rdf:Description>
</rdf:RDF>
```

Figure 4.3.2 - Example of the FOAF file in RDF/XML format

## 4.4 RDF Query Languages

### 4.4.1 SPARQL

SPARQL is a query language for RDF. Its name is a recursive acronym that stands for Protocol and RDF Query Language [20]. It is considered a key semantic web technology and it became an official W3C Recommendation. SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. The results of SPARQL queries can be results sets or RDF graphs. It specifies four different query variations for different purposes:

*SELECT query*
: Used to extract raw values from a SPARQL endpoint, the results are returned in a table format.

*CONSTRUCT query*
: Used to extract information from the SPARQL endpoint and transform the results into valid RDF.

*ASK query*
: Used to provide a simple True/False result for a query on a SPARQL endpoint.

*DESCRIBE query*
: Used to extract an RDF graph from the SPARQL endpoint, the contents of which is left to the endpoint to decide based on what the maintainer deems as useful information.

Each of these query forms takes a WHERE block to restrict the query although in the case of the DESCRIBE query the WHERE is optional.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?email

WHERE {

  ?person a foaf:Person.

  ?person foaf:name ‘Bob’.

  ?person foaf:mbox ?email.

}
```

The previous query searches the FOAF file to find triples where subject is a `foaf:Person` that has name ‘Bob’ and returns it’s email address. Variables in SPARQL are indicated by a ‘?’ prefix.

### 4.4.2 SPARUL

SPARUL (SPARQL/UPDATE) is an update language for RDF. It uses a syntax derived from SPARQL. Update operations are performed on a collection of graphs in a Graph Store. Operations are provided to change existing RDF graphs as well as create and remove graphs with the Graph Store. It is intended to be a standard mechanism by which updates to a remote RDF Store can be described, communicated and stored.

*INSERT*
: Used to insert new triples to an RDF graph.

*DELETE, CLEAR*
Used to delete triples from an RDF graph
*MODIFY*
Used to perform a group of update operations as a single action.
*CREATE*
Used to create a new RDF Graph to a Graph Store.
*DROP*
Used to delete an RDF graph from a Graph Store.

An example of SPARUL query where an insert statement is used to add the statement that a resource `foaf:Person` with the name 'Bob' knows 'Alice' is given below:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

INSERT INTO <http://xmlns.com/foaf/0.1/>

{

  ?person foaf:knows 'Alice'

}

WHERE {

  ?person a foaf:Person.

  ?person foaf:name 'Bob'.

}
```

## 4.4 Triple Stores

"Triple Store" is a common name given to a database management system for RDF data. These systems provide data management and data access via APIs and query languages (such as SPARQL) to RDF data. Recently there has been a major development initiative in query processing, access protocols and triple-store technologies. The number of triple stores being actively developed has increased in the last 10 years, starting from Jena and Sesame in early 2000s to Jena TDB, Jena SDB, BigOWLIM, Virtuoso, Allegro Graph, Kowari, 3store, BigData etc. Some are only commercially available such as BigOWLIM and Allegro Graph, Virtuoso has both commercial and open source version, while the rest are mainly free and open source.

Triple stores differ by the way they store data and by amount of data they can store. The amount of data varies from 10-20 million triples to 1 trillion triples (Allegro Graph)[3]. Triples are stored in storage which is either a "native" triple store or a relational database with a triple store overlay. Native stores provide persistent storage with their own implementation of the databases (Virtuoso, Allegro Graph). Native stores

[3] http://www.w3.org/wiki/LargeTripleStores

have gained a lot of popularity as they give better performance when the size of the dataset is large and because they can offer in-store reasoning/inference.

## 4.5 Jena

Jena [21] is an open source JAVA framework for building the Semantic Web applications. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine. Jena can be easily integrated in any kind of Java applications.

# Chapter 5: Design

This chapter explains the design of WRS, how the client side works as a plugin and part of Scone proxy and how it interacts with WRS Server, used to store ratings and recommendations data. WRS Server is used instead of Wikipedia as storage for information about users, ratings and recommendations.

Moving away from Wikipedia and storing data on the separate server is one of the three changes introduced in this thesis. The second is introduction of recommendations as part of user trust network which he/she is willing to share with rest of the community. The rest are RDF views of data and user pages made to replace Wikipedia user pages, used by previous versions of WRS.

## 5.1 The overview of the system

The overview of the system can be best explained following the flow of the HTML Document from when it is extracted from the Wikipedia to when it is inserted to the user's browser. The flow of the HTML document in the WRS system is shown on the Figure 5.1.1.

The main part of the client side of WRS is the WRS plugin developed as a plugin for a programmable proxy called Scone, which allows manipulation of the HTML documents. The proxy intercepts the browser calls to Wikipedia pages. The WRS plugin is running on the user's machine and incorporates the whole business logic of the system and is used in communication with Wikipedia – to extract the annotations about the requested page, and with WRS Server (the central storage) – to insert and retrieve the information about ratings and recommendations.

The main governing part of the plugin is the ***WRS module***. The WRS module checks that the incoming HTML stream comes from the Wikipedia.org domain and further, from the interest point within the domain which are article pages (en.wikipedia.org/wiki/[article_name]). If the page is found of no interest, it is passed unchanged to the browser. Article pages are processed by ***Page module***. Annotations about the page such as name and version are extracted from the page and send to the ***Rating module***. This model calls the ***Service module*** which coordinates the service calls to ***WRS Service*** residing on the ***WRS Server***. Ratings of reviewed articles are stored in ***Virtuoso*** relational database running on the WRS Sever. WRS Service is an interface for retrieving and storing information about ratings and recommendations. WRS Service also provides the registration of the new user, login and logout functionality. After the ratings have been retrieved, Rating module will update the trust values according to the given data and calculate a score and category for the article. At this point if the retrieved ratings are not given by reviewers with previous interaction with the user, the Rating module will call the WRS Service, again, to search, if any of the reviewers that had interaction with the user, have recommended article reviewers as trustworthy ones and therefore noted them for inferred trust. After the calculations are done, the Page Module will insert the score and category in HTML document and Plugin will insert the document to the users browser.

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

The **Trust module** updates the trust values according to the user's feedback, updates the **Web of Trust** and calls the Service module to submit the given rating for the article. The user can choose to recommend the users with whom he had positive interaction and which case the service will be called again in order to submit the recommendations to the WRS Server.

WRS Server hosts several parts used to store, query, publish and view the information about users, their rating of the Wikipedia articles and their recommendations of other users. ***Open Link Virtuoso Server*** is used as database server to store the data, as application server to publish ***RDF views*** of data (so it can be available and easily processed by other applications and machines) and to host user pages via the **Open Data Spaces** (ODS) application.

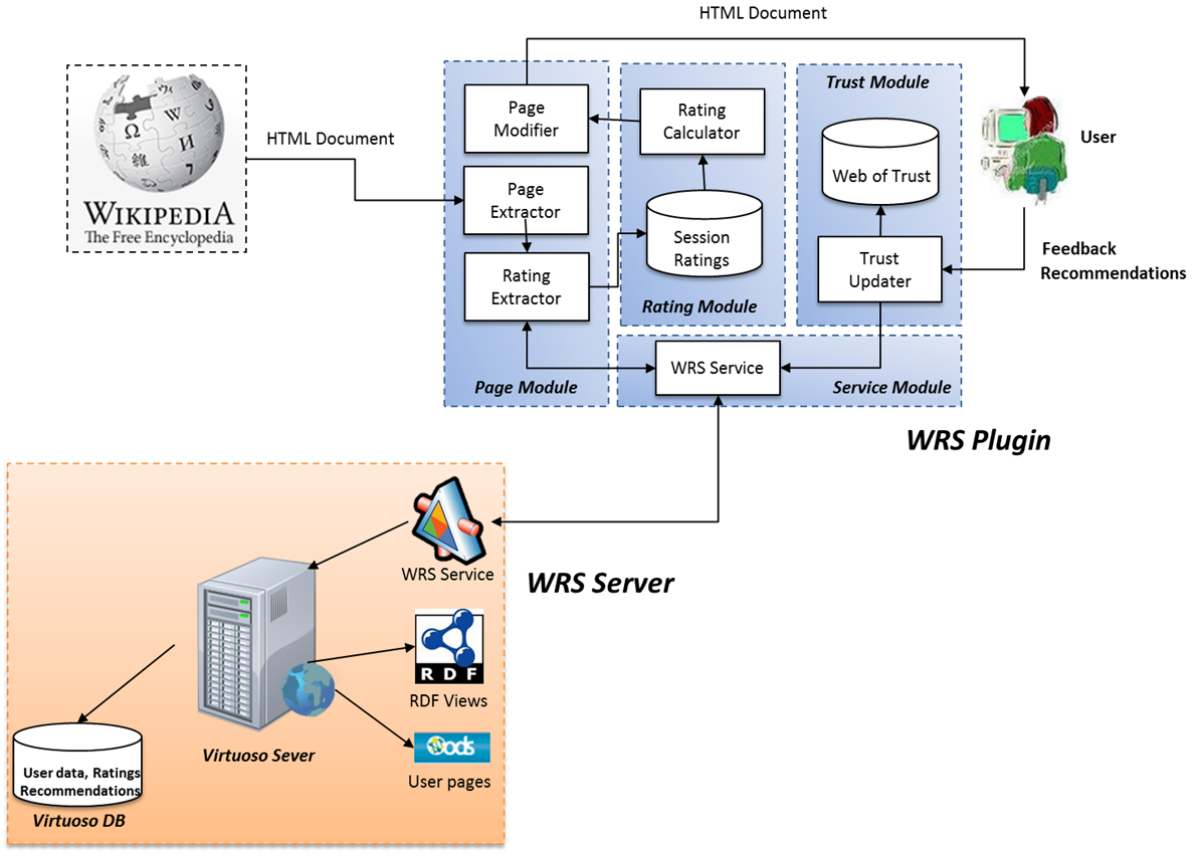



Figure 5.1.1. Flowchart for the HTML document in the WRS System.

## 5.2 The WRS Service

The WRS Service is a SOAP web service used as an interface for uploading and retrieving data between the WRS Plugin and WRS Server. It is implemented as Apache Axis2 web service and its description can be found on the following address: https://wrs.imm.dtu.dk:8181/wrs/services/WrsService?wsdl

It contains the following methods:

**register** – used to register new users to the system. The method returns the session id used to identify the user in the following calls.

**login** – used to login the user. The method returns the session id used to identify the user in the following calls.

**logout** – used to logout the user from the system and finishes the session.

**setRating** – used to store the rating for the article given by the user.

**getRatings** – used to retrieve the ratings for the given article.

**recommend** – used to upload recommendations for the selected users.

**getRecommendations** – used to retrieve all recommendation of the given reviewer.

## 5.3 Virtuoso Server

Virtuoso is a middleware and database engine hybrid developed by OpenLink Software[4]. It combines the functionality of traditional relational database, object-relational database, virtual database, RDF, XML, web application server, content management system and SOA platform functionality in the single system. There are two editions of the software, Virtuoso Universal Server which is a licensed version and Virtuoso open-source edition, known as OpenLink Virtuoso. The major difference is that open-source version does not support clustering and interoperability with other RDBMS.

In this project, Virtuoso open-source edition is used and for several purposes. Relational database containing data about articles, ratings, categories, WRS users and users' recommendation is stored using the in-built Virtuoso's RDBMS. Using the virtual database with the RDF support, RDF Views of relational data are created. Their main purpose is to expose WRS data as Linked Data and publish it publically, using the in-built web application server and RDF web service functionality. Such a data can be easily read and processed by other machines. The third usage of Virtuoso is for creating and maintaining WRS users' profiles. Virtuoso enables additional applications to be installed to the server as packages. These packages are generally tools for building Semantic Web compliant applications. One of them is OpenLink Dataspace (ODS) which enable creation and management of user profiles based on the FOAF file.

[4] http://www.openlinksw.com/

Virtuoso has been chosen, beside the above mentioned qualities, because it provides a native RDF storage that can store over 15 billion triples. It's been already used in many projects such as *data.gov* and *dbpedia.org* and showed impressive performance results in conducted Benchmarks[5]. Though its conductor package, Virtuoso provides a nice graphical interface for administration of the various part of the server. The interface is shown in the Figure 5.3.1.

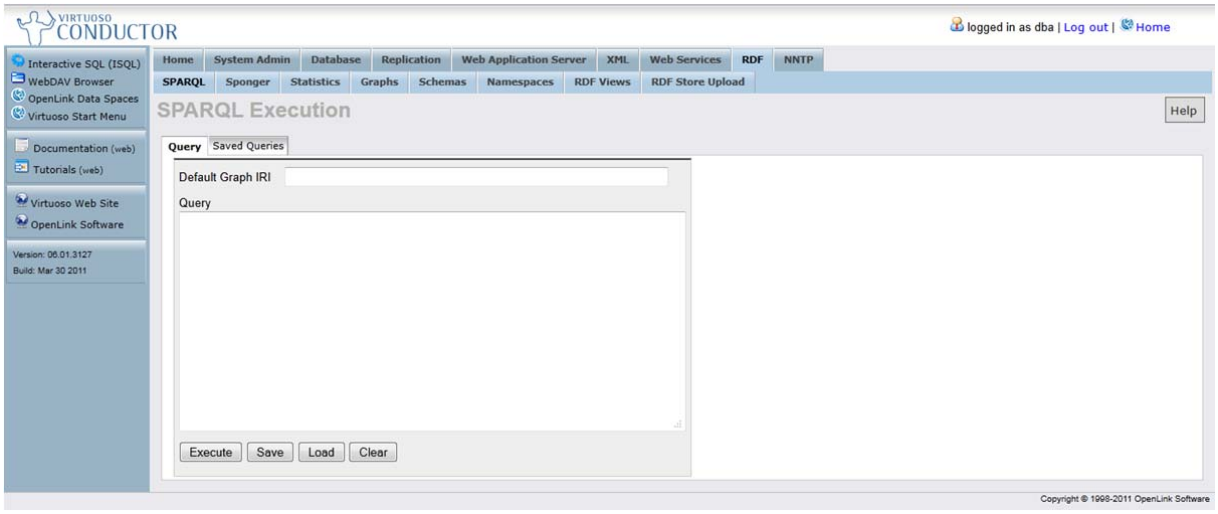



**Figure 3.3.1. Virtuoso's administration interface**

[5] http://www.w3.org/wiki/RdfStoreBenchmarking

## 5.4 WRS data as Linked Data via RDF Views

Virtuoso includes built-in RDF middleware for transforming non-RDF data into RDF "on the fly" called Sponger. Its goal is to use non-RDF Web data sources as input and create RDF as output. The purpose of this functionality is to use non-RDF data sources as Linked Data Web data sources.

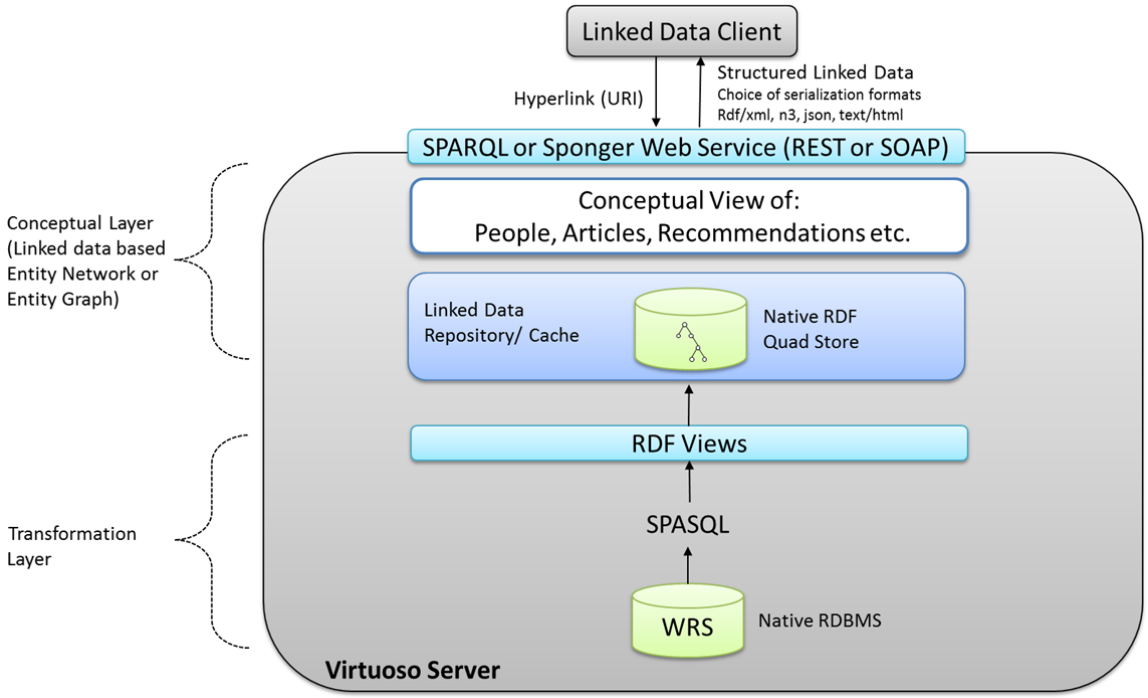



Figure 4.4.1. Linked Data generation[6]

WRS data (users, articles, ratings and recommendations) are stored in Virtuoso's native relational database. Using the SPASQL (SPARQL embedded in SQL), RDF Meta Schema and Virtuoso's Meta Schema Language - a domain specific, declarative language for mapping a logical SQL data model to a conceptual RDF data model, relational data is transformed to RDF as views. Mapping patterns used to transform SQL to RDF and graph definitions are stored in Quad storage and from there, they can be queried via SPARQL Service or publish as a Linked Data using the Proxy service.

SPARQL Service is in-built REST based web service used for SPARQL query processing. It supports both GET and POST requests. The client chooses between GET and POST automatically, using the length of query text as the criterion. If the SPARQL endpoint is accessed without any URL and requisite SPARQL protocol parameters, an interactive HTML page for capturing SPARQL input will be presented.

The service endpoint is: https://wrs.imm.dtu.dk/sparql

This service can be used from Jena, to perform SPARQL queries.

[6] Overview of the full Linked Data generation options is given at http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VirtSpongerCartridgeProgrammersGuide

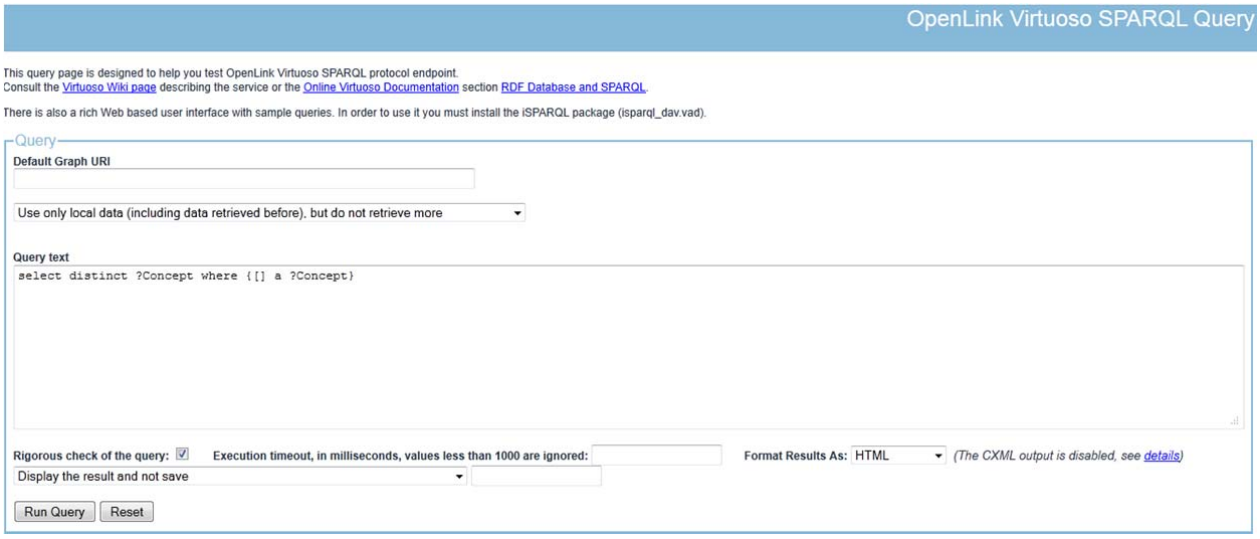



Figure 5.4.2. An interactive HTML page for capturing SPARQL queries

PROXY Service is in-built REST style web service and part of Sponger functionality. This web service takes a target URL and either returns the content "as is" or tries to transform (by sponging) to RDF. Thus, the proxy service can be used as a 'pipe' for RDF browsers to browse non-RDF sources. Path to the service is: https://wrs.imm.dtu.dk/about/[rdf|html]. The proxy service is used to expose WRS data in HTML or RDF.

- HTML description - https://wrs.imm.dtu.dk/about/html/http/localhost:8890/WRS/articles/en/Albert_Einstein exposes the ratings of Wikipedia article about Albert Einstein in HTML as LinkedData

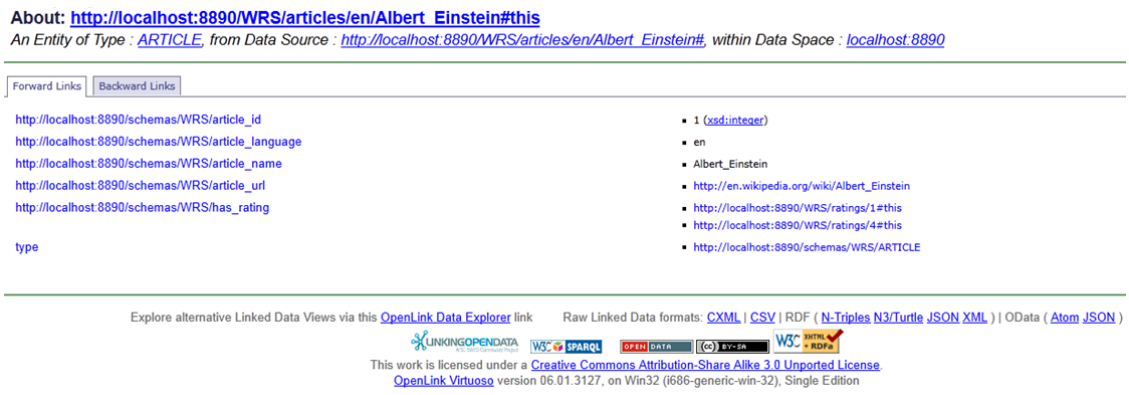



Figure 5.4.3 - HTML presentation of Linked Data exposing ratings of the article Albert_Einstein

- RDF description https://wrs.imm.dtu.dk/about/html/http/localhost:8890/WRS/articles/en/Albert_Einstein exposes the ratings of Wikipedia article about Albert Einstein in RDF, where format is one of xml, n3, nt, ttl, or json.


Figure 5.4.4 - RDF description of the ratings of the article Albert_Einstein

## 5.5 OpenLink Data Spaces (ODS)

ODS is a new-generation Distributed Collaborative Application platform for creating presence in the semantic web via Data Spaces derived from Weblogs, Wikis, Feed Aggregators, Photo Galleries, Shared Bookmarks, Discussion Forums and more [33]. They are important as they provide a cost-effective route for generating a Semantic Web Presence from Web 2.0 and traditional Web data-sources, by delivering an atomic data container for RDF Instance Data derived from data hosted in Blogs, Wikis, Shared Bookmark Services, Discussion Forums, Web File Servers, Photo Galleries, etc. ODS consist of several application components which can be installed as part of Virtuoso server. These components are:

**ODS-Framework –** An OpenID and WebID compliant framework for building Distributed Collaborative Applications that are equipped with Single-Sign-On (SSO) functionality and auto-generated RDF Data Spaces.

**ODS-Weblog –** A comprehensive blogging platform supporting all the major publishing protocols (Atom, Moveable Type, Meta Weblog, and Blogger) which includes automatic generation of content in several different formats.

**ODS-Briefcase –** A Unified Storage Solution that incorporated automated extraction and management of metadata.

**ODS-Feed-Manager** - An RSS 1.0, RSS 2.0, Atom, OPML, and OCS Feed Aggregator

**ODS-Wiki** - A Wiki Platform supporting the Atom Publishing Protocol, Twiki, MediaWiki (Wikimedia), etc.

**ODS-Mail** - A Web-based Email Client

**ODS-Calendar** - A Calendar manager

**ODS-Bookmark-Manager** - A Shared Bookmark Manager

**ODS-Gallery** - Photo & General Image Sharing

**ODS-Polls** - Polls Manager

**ODS-Addressbook** - An Address Book Manager

**ODS-Community** - Group/Community Mode service for all of the ODS-* applications

**ODS-Discussions** - Web-based Newsgroups Aggregator and Reader.

Each of these components is available as an installation package for the Virtuoso server. Together they give a powerful tool for building Semantic Web compliant applications. However, this exceeds the borders of the system that it is presented in this thesis and only ODS-Framework is used as tool for creating and managing WRS users' profiles.

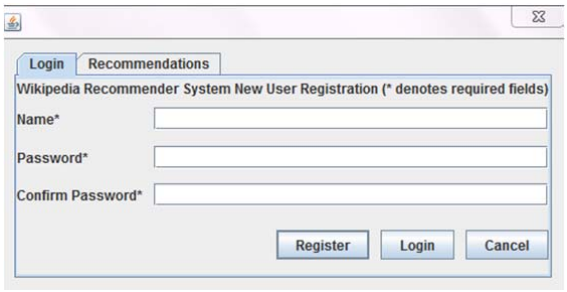
In the Semantic Web applications, each user is represented by its File-Of-A-Friend file. FOAF file contains all information about the user such as name, nickname and address, his/her social contacts throw defining

people whom he/she knows etc. One of the things that FOAF includes is security information where a user can store his/her certificate used to identify or identity token, used to identify the user instead of classical user name and password. Each user can have many FOAF files but each FOAF has a unique URL identifier.

In the previous version of WRS, Wikipedia User Pages were used to create user accounts and upload an X509. certificate. In this version of WRS, ODS-Framework is used for creating and managing of user profiles. In this way, WRS users don't need to be Wikipedia users anymore as the system requires them to be only registered WRS users.

Users can register in two ways:

1. Using the WRS client application on their own machine




2. Using the ODS application on the WRS server https://wrs.imm.dtu.dk/ods

When the user registers with WRS, his/her data space is created with unique Personal URI. This URI is of the form:

https://wrs.imm.dtu.dk/dataspace/person/<username>#this

and can be used as a link to the user in other semantic web applications or to link other resources in WRS to the user, such as recommendations. The same URI is user's WebID and can be used together with the X509certificate in WebID authentication protocol.

Provided username is actually a nickname (`foaf:nick`) in FOAF terminology, which means that the user's FOAF file is automatically generated. WRS does not require user's real name, just a unique username, and it is left to the user to fill his profile information to the degree he/she finds appropriable. The same applies for the privacy level of exposed data.

User can edit profile information by clicking on the "Profile Edit" button in the top left corner of ODS application. Visualized FOAF file will be shown, letting user to edit his profile information.

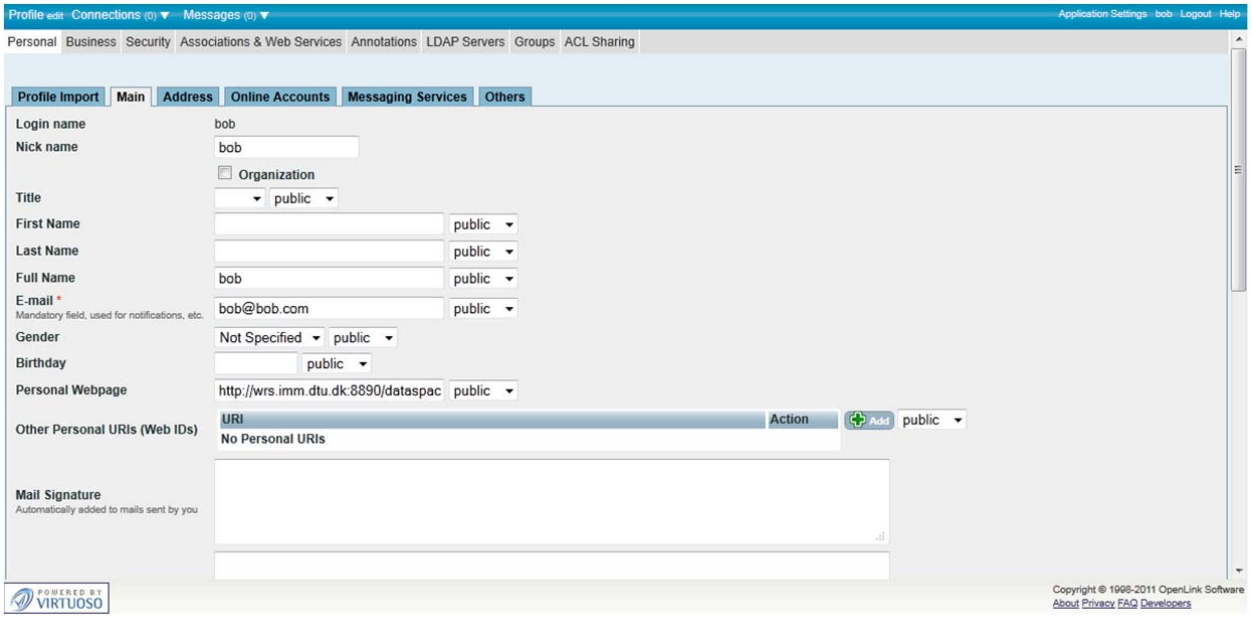



Figure 5.5.1 - Visualized FOAF file of WRS user Bob in ODS

The interesting part is a **Security** tab, where user can change his Log-In information or use a **Certificate Generator** to generate an X509.Certificate.

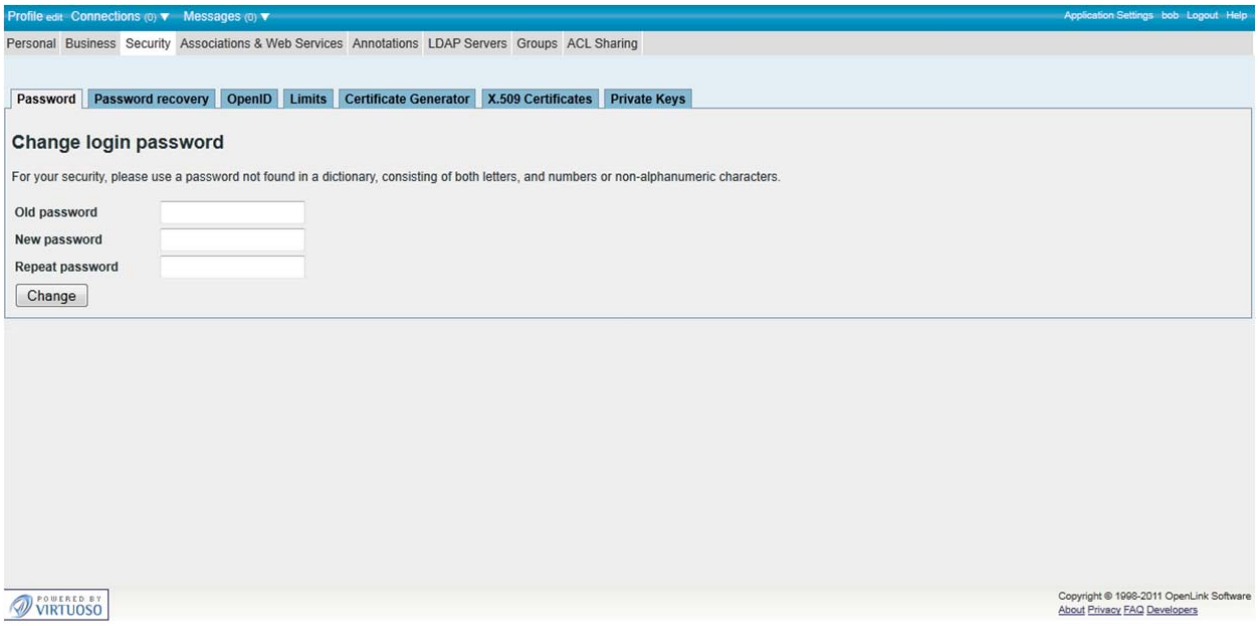



Figure 6.5.2 - Security settings in ODS

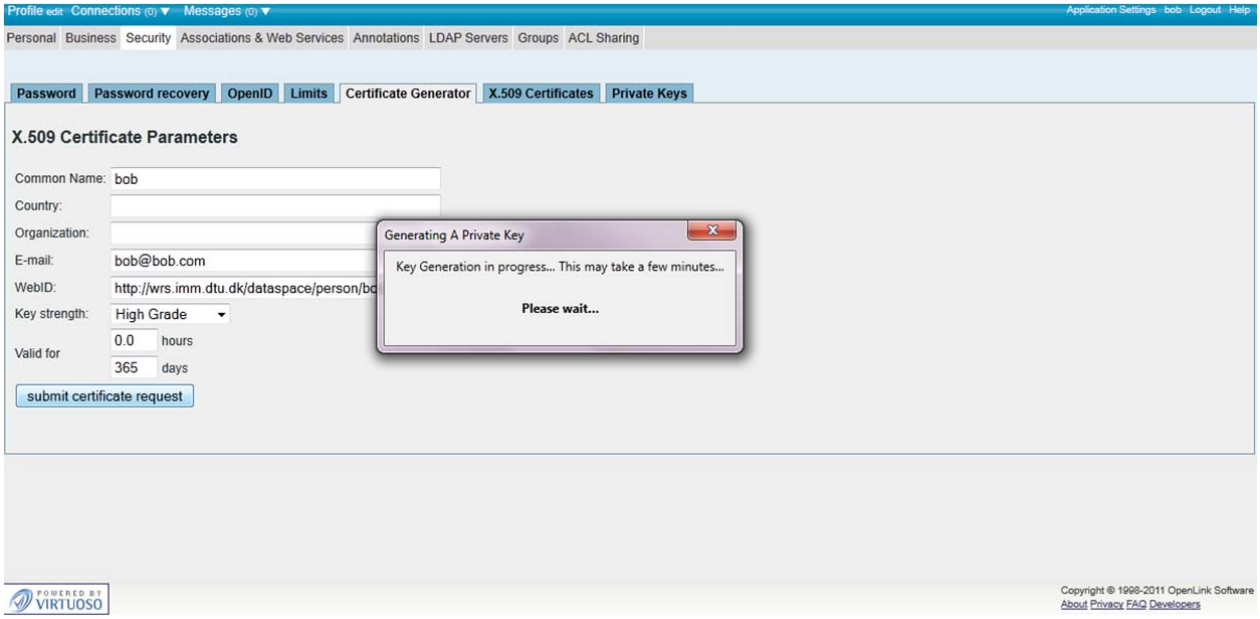



Figure 5.5.3 - X509.Certificate Generator in ODS

## 5.6 Summary

In this chapter design of the WRS system is presented. The main parts of the system are:

- WRS client application implemented as part of the Scone proxy, which is running on the user's local machine. The application contains several modules used to process Wikipedia pages, calculate ratings, store and manage trust values and communicate with WRS Service.
- WRS Service is an interface between WRS Client and WRS Server. WRS Service is used to upload and retrieve data stored on the WRS Server.
- WRS Server is occupying wrs.imm.dtu.dk domain and host Virtuoso server used to store and publish WRS data about users, articles, ratings and recommendations. Virtuoso incorporates several functionalities such as relational database server, virtual RDF storage, application server and content management system. It is used to store WRS data, publish WRS data as RDF and host ODS application.
- RDF Views are part of the Virtuoso server which enables data stored in relational database to be published as RDF Linked Data. Such data can be after easily read and processed by other applications. View can be queried using the SPARQL service and viewed using the Proxy service.
- ODS is a framework based on the FOAF concept which enables maintenance of the users' profiles and Single-Sign-On functionality of the system using the username-password, OpenID or WebID credentials.

# Chapter 6: Implementation

In this chapter the most interesting parts of the implementation are presented. As the proposed WRS solution is built on top of the existing WRS system, there is no need to go in detail with the implementation. Detail description of WRS modules and the source code can be found in work of previous authors: Povilas Pilauskas [28], Thomas Lefevre [12] and Thomas Korsgaard [5].

## 6.1 WRS Database

WRS database is a relational database stored on the WRS Server and hosted by Virtuoso. The database contains data about rated articles and user recommendations. As Virtuoso is a native storage it contains a virtual database stored in memory. This means that the database is a file virtuoso.db. Virtuoso Universal Server can connect a traditional RDBMS such as MySql but as Open Source version of the server is used which does not include this functionality, relational data is stored in memory.

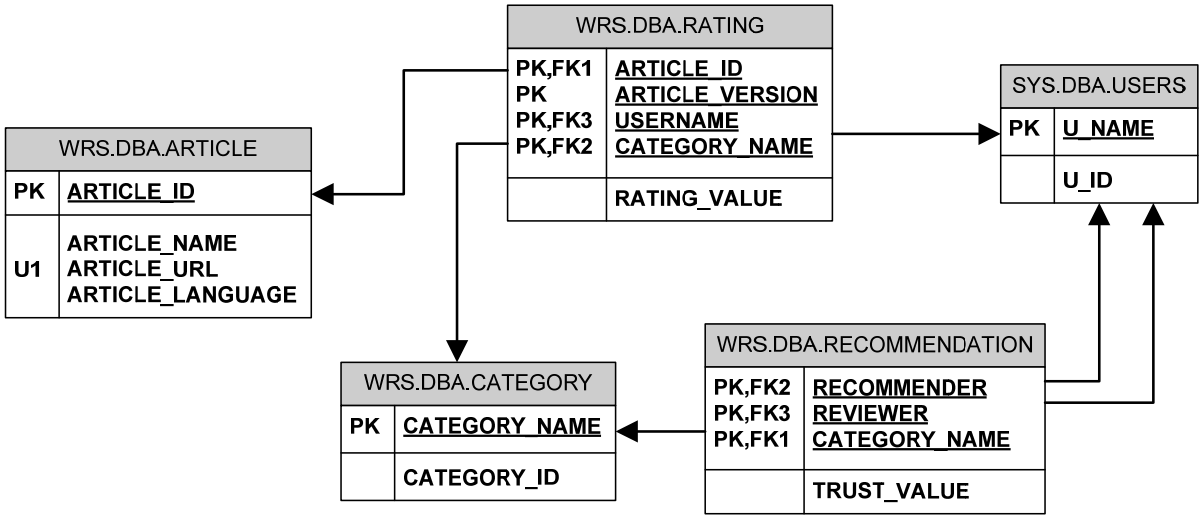



Figure 6.1.1 - WRS database schema

## 6.2 Mapping Relational Data to RDF with Virtuoso's RDF Views

RDF Views map relational data into RDF and allow RDF representation of the relational data to be customized. Virtuoso includes a declarative Meta Schema Language for defining the mapping of SQL data to RDF ontologies. The mapping is dynamic; consequently changes to the underlying data are reflected immediately in the RDF representation. This means that RDBRDF triggers enable that every update of the

database table triggers the update of corresponding RDF data and as RDF Views are created dynamically, "on-the-fly", users are always provided with the latest data.

RDF Views enable WRS article's ratings and user recommendations to be exposed publicly on the web as RDF, machine readable format that can be shared across applications and enables researches to build their own client applications using the WRS data.

Mapping operation does not require changes to the underlying relational database schema.

In the Figure 6.2.1, WRS sample data is shown. *ARTCILE* table together with the *RATING* contains data about evaluated Wikipedia articles and their ratings. *CATEGORY* contains data about classification categories and *RECOMMENDATION* table contains data about user recommendations.

**Sample Data - Table: ARTICLE**

| ARTICLE_ID | ARTICLE_NAME | ARTICLE_URL | ARTICLE_LANGUAGE |
|---|---|---|---|
| 1 | Albert_Einstein | http://en.wikipedia.org/wiki/Albert_Einstein | en |
| 2 | Rotavirus | http://en.wikipedia.org/wiki/Rotavirus | en |
| 3 | Hellerup | http://en.wikipedia.org/wiki/Hellerup | en |

**Sample Data - Table: RATING**

| ID | CATEGORY_NAME | USERNAME | ARTICLE_ID | ARTICLE_VERSION | RATING_VALUE |
|---|---|---|---|---|---|
| 1 | Science | alice | 1 | 442504748 | 7 |
| 2 | Science | bob | 2 | 452504741 | 9 |
| 3 | Reference | alice | 3 | 432504742 | 6 |
| 4 | Science | bob | 1 | 442504748 | 8 |

**Sample Date - Table: CATEGORY**

| CATEGORY_ID | CATEGORY_NAME |
|---|---|
| 1 | Arts |
| 2 | Business |
| 3 | Computers |
| 4 | Health |
| ... | ... |
| 12 | Science |
| 13 | Shopping |
| 14 | Society |
| 15 | Sport |

**Sample Data - Table: RECOMMENDATION**

| RECOMMENDER | REVIEWER | CATEGORY_NAME | TRUST_NAME |
|---|---|---|---|
| alice | bob | Science | 0.3 |
| charlie | bob | Science | 0.8 |
| | | | |

**Sample Data - Table: SYS_USERS**

| U_ID | U_NAME |
|---|---|
| 1 | alice |
| 2 | bob |

Figure 6.2.1 - WRS sample data

Let's see how the highlighted data can be presented in RDF. Entity-Relationship is a basis for designing database schemas. Typically, each entity is represented as a table, each attribute of the entity becomes a column in that table, and relationships between entities are indicated by foreign keys. Each table typically defines a particular class of entity, each column one of its attributes. Each row in the table describes an entity instance, uniquely identified by a primary key.

In RDF the constituent parts of a triple could be described as: subject, subject attribute and attribute value, rather than the usual subject, predicate, object. A thorough description of an entity in RDF would consist of a collection of triples each relating to the same subject, i.e. entity, where each predicate relates to a different entity attributes. The entity's class would be described by a further RDF statement containing the rdf:type (or a) predicate. To uniquely identify each distinct subject and predicate, the subject (ignoring blank nodes for now) and predicate must be IRI's, the object can be an IRI or a literal.

Therefore, any relational schema can be rendered into RDF by converting all primary keys and foreign keys into IRI's, assigning a predicate IRI to each column, and an rdf:type predicate for each row linking it to a RDF class IRI corresponding to the table. We can then, for each column that is neither part of a primary or foreign key, construct a triple containing the primary key IRI as the subject, the column IRI as the predicate and the column's value as the object.

In WRS database there are 5 entities ARTICLE, CATEGORY, RATING, RECOMMENDATION and USER. Users are already presented in RDF by FOAF concept and each user is presented by its FOAF file. These files are exposed publicly on the web and can be accessed by typing:

https://wrs.imm.dtu.dk/dataspace/person/bob,

in the browser. More about data spaces and user representation will be discussed in the next chapter.

The remaining 4 entities (tables) have to be described in RDF using the RDF Schema. The RDFS class for the table RATING is given below:

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix aowl: <http://bblfish.net/work/atom-owl/2006-06-06/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix virtrdf: <http://www.openlinksw.com/schemas/virtrdf#> .
@prefix WRS: <http://localhost:8890/schemas/WRS/> .

WRS: a owl:Ontology .

# WRS.DBA.RATING
WRS:RATING a rdfs:Class .
WRS:RATING rdfs:isDefinedBy WRS: .
WRS:RATING rdfs:label "WRS.DBA.RATING" .

WRS:id a owl:DatatypeProperty .
WRS:id rdfs:range xsd:int .
WRS:id rdfs:domain WRS:RATING .
WRS:id rdfs:isDefinedBy WRS: .
WRS:id rdfs:label "ID" .

WRS:is_rating_of_article a owl:ObjectProperty .
WRS:is_rating_of_article rdfs:range WRS:ARTICLE .
WRS:is_rating_of_article rdfs:domain WRS:RATING .
WRS:is_rating_of_article rdfs:isDefinedBy WRS: .
WRS:is_rating_of_article rdfs:label "ARTICLE" .

WRS:article_version a owl:DatatypeProperty .
WRS:article_version rdfs:range xsd:string .
WRS:article_version rdfs:domain WRS:RATING .
WRS:article_version rdfs:isDefinedBy WRS: .
WRS:article_version rdfs:label "ARTICLE_VERSION" .
```

```
WRS:has_reviewer a owl:ObjectProperty .
WRS:has_reviewer rdfs:range foaf:Person .
WRS:has_reviewer rdfs:domain WRS:RATING .
WRS:has_reviewer rdfs:isDefinedBy WRS: .
WRS:has_reviewer rdfs:label "USERNAME" .

WRS:has_category a owl:ObjectProperty .
WRS:has_category rdfs:range WRS:CATEGORY .
WRS:has_category rdfs:domain WRS:RATING .
WRS:has_category rdfs:isDefinedBy WRS: .
WRS:has_category rdfs:label "CATEGORY" .

WRS:rating_value a owl:DatatypeProperty .
WRS:rating_value rdfs:range xsd:int .
WRS:rating_value rdfs:domain WRS:RATING .
WRS:rating_value rdfs:isDefinedBy WRS: .
WRS:rating_value rdfs:label "RATING_VALUE" .
```

Notice that foreign keys in this table were ARTICLE_ID, USERNAME and CATEGORY_NAME are not presented in the class as datatype properties (such as ARTICLE_VERSION and TRUST_VALUE), instead relationship between linked tables is made more explicit by referencing relevant classes WRS:ARTICLE, foaf:Person and WRS:CATEGORY, where WRS is a prefix for WRS schema (http://wrs.imm.dtu.dk/schemas/WRS) containing the mentioned classes and foaf is a prefix for FOAF schema (http://xmlns.com/foaf/0.1/) which contains definition of Person class.

Virtuoso manages the conversion of column values to IRIs using IRI classes, a feature of Virtuoso's Meta Schema Language which itself is implemented as extensions to SPARQL. An IRI class defines how a column or set of columns gets converted into an IRI.

```
SPARQL
prefix WRS: <http://localhost:8890/schemas/WRS/>

create iri class WRS:article "http://wrs.imm.dtu.dk/WRS/articles/%s/%s#this" (in
_ARTICLE_LANGUAGE varchar not null,in _ARTICLE_NAME varchar not null) .

create iri class WRS:category "http://wrs.imm.dtu.dk/WRS/categories/%U#this" (in
_CATEGORY_NAME varchar not null) .

create iri class WRS:rating "http:// wrs.imm.dtu.dk /WRS/ratings/%d#this" (in _ID integer
not null) . ;  (in language varchar not null, in article_name varchar not null) .

create iri class WRS:recommendation
"http://wrs.imm.dtu.dk/WRS/recommendations/%U_%U_%U#this" (in _RECOMMENDER varchar not
null,in _REVIEWER varchar not null,in _CATEGORY_NAME varchar not null) .

create iri class WRS:recommendation_list
"http://wrs.imm.dtu.dk/WRS/recommendations/%U#this" (in _REVIEWER varchar not null) . ;
```

The RDFS schema for the whole WRS database and data mappings file are given in Appendix.

The result of this conversion is that each database table row is converted into resource with unique identifier such as http://wrs.imm.dtu.dk/article/en/Albert_Einstein, http://wrs.imm.dtu.dk/categories/Science, http://wrs.imm.dtu.dk/recommendations/bob (representing the list of bob's recommendations), http://wrs.imm.dtu.dk/recommendations/alice_bob_Science (representing alice's recommendation given to bob in category Science) etc. These resources form a graph of linked data where each eclipse presents the subject, each arc presents the *predicate* or *subject's attribute* and each *rectangle* presents the *object* or *attribute's value*.

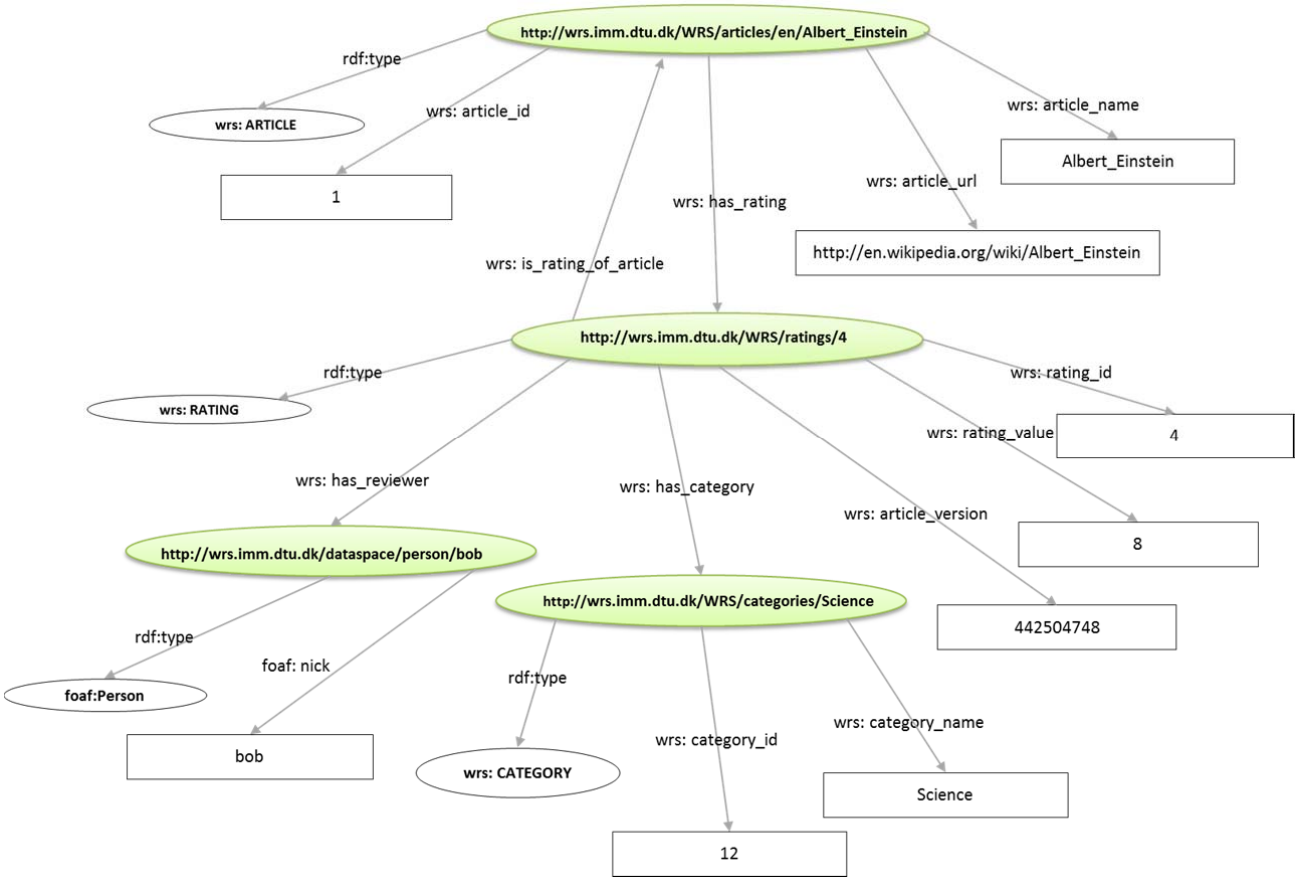



**Figure 6.2.1 - RDF Graph presenting the data about Wikipedia's article about Albert Einstein**

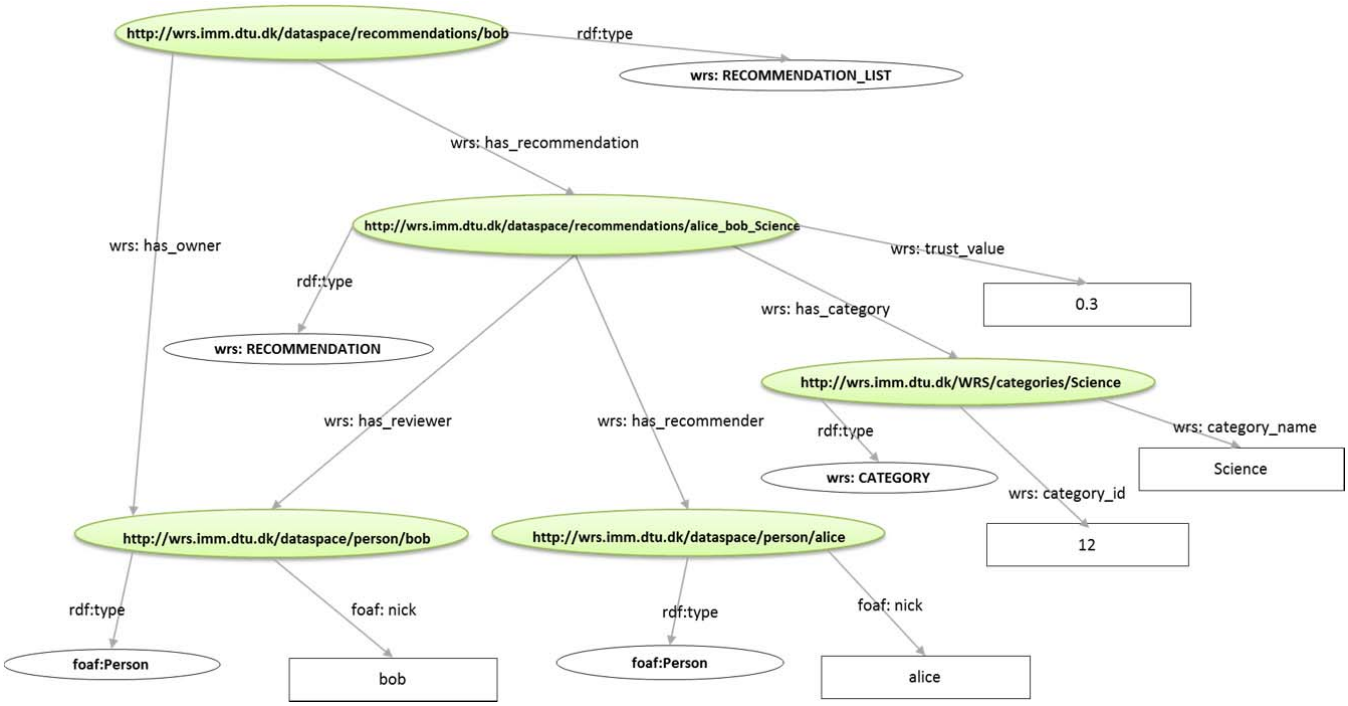



Figure 6.2.2 - RDF Graph presenting the recommendations of user bob

-

## 6.3 Recommendation Submission GUI

Submitting of the users' recommendations in other users expressed in the form of trust value in user as a good reviewer in the certain category is an extension to the previous version of WRS. In order to let users to submit their recommendations, GUI is developed to support this functionality.


Figure 6.3.1 - GUI used to submit users' recommendation in other users

The input parameter is trust threshold value used to determine which trust connections will be published as recommendations. In the above example the system will search for all trust connections with the trust value greater than 0.1 in certain category and publish them as recommendations given to trusted user in certain category.

## 6.4 WRS Configuration

The previous *passwords.txt* file, used to store user's username and password of the keystore is replaced with *wrs.properties* file. Using the *.properties file is more Java-like application configuration file, as `java.util.Properties` class can be used to read and write to the file.

```
Properties configFile = new Properties();

configFile.load(this.getClass().getClassLoader().getResourceAsStream

("wrs.properties"));

configFile.getProperty("keystore");
```

wrs.properties file is used to store path and password of the default Java keystore and other application properties such as trust propagation horizon and trust threshold value used in selecting propagation paths. The format of the stored values is <key>=<value>. Lines representing comments start with "#".

```
# Key store settings

keystore =

C:\\Program Files\\Java\\jdk1.6.0_18\\jre\\lib\\security\\cacerts

keystore_password = changeit


# Trust propagation settings

horizon = 3

threshold = 0.1
```

Key store settings data are important as WRS Plugin module is calling WRS Service over HTTPS, so the certificate of the WRS Server needs to be stored in the keystore for invocation of the web service to work properly. Default Java keystore for storing trusted certifiates is located in `%JAVA_HOME%/lib/security/cacerts`.

Trust propagation parameters are used in calculation of propagated trust value in unknown user. *Horizon* is used to determine the length of the propagation path and *threshold* is used to determine the minimum acceptable trust value on the path.

The format of the file enables easy editing of the file - the user can self-configure the value using the simple text editor such as Notepad.

## 6.4 Summary

In this chapter most interesting part of implementation of the WRS system are presented. As the system is built on top of the current WRS system implementation, detail description of the classes and modules is avoided. The chapter includes a description of the WRS database, example of data stored in the database, description of the mechanisms used to convert data stored in the database into RDF graphs, graphical presentation of the generated RDF graphs, illustration of the GUI used to submit users' recommendation in other users and explanation of the configuration file used to preserve predefined setting for the WRS client application.

The rest of the data belonging to Implementation chapter such as WSDL of the WRS Service, Ontology file used to convert relational data to RDF and RDF mapping of relational data into RDF views is given in the Appendix.

# Chapter 7: Evaluation

This chapter describes difficulties which authors can face while trying to evaluate their recommender system. The evaluation of the recommender system is a complex procedure which does not only include the confirmation of the accuracy of its predictions. Key problem in evaluating the current WRS are presented together with proposal for future evaluation. Functionality of the system is evaluated using a concrete scenario.

## 7.1 Key decisions in evaluating recommender system

One of the most quoted works in the area of evaluation of the recommender systems is the study by Herlocker et all. [34] in which authors review the key decisions in evaluating collaborative filtering recommender system. They claim that evaluating recommender systems and their algorithms is inherently difficult for several reasons:

1) Different algorithms may be better or worse on different data sets. Many collaborative filtering algorithms have been designed specifically for data sets where there are many more users than items (e.g., the MovieLens data set has 65,000 users and 5,000 movies). Such algorithms may be entirely inappropriate in a domain where there are many more items than users (e.g., the Epinions data set has 132,000 users and 1,560,144 articles). Similar differences exist for ratings density, ratings scale, and other properties of data sets.

2) Goals for which an evaluation is performed may differ. Some evaluations focus on accuracy of collaborative filtering algorithm in predicting ratings, while others find more valuable to measure how often the system leads its users to wrong choices. Some researchers find that more important than accuracy is user satisfaction etc.

3) Finally, there is a challenge in deciding what combination of measures to use in comparative evaluation. Authors noticed a recent trend that many researchers find that their newest algorithms yield a mean absolute error (MAE) of 0.73 (on a five-point rating scale) on movie ratings datasets. Though the new algorithms often appear to do better than the older algorithms they are compared to, authors claim that, when each algorithm is tuned to its optimum, they all produce similar measures of quality.

Authors propose that the best way to evaluate the recommender system is to perform both on-line and off-line evaluation. On-line evaluation is a live testing of the system by its users, and off-line testing is performed by researchers on the dataset gained by the on-line testing.

## 7.2 Evaluation of the WRS

Trust propagation algorithm used to fight the cold start problem has already been tested on the large real-life dataset. Authors (Massa and Avesani) performed the evaluation on the Epinions dataset, and their results are well documented in [26]. Rating scale used to rate the articles in the Epinions dataset is in rage 0 to 5. This is a similar rating concept to WRS with the difference that WRS uses scale from 1 to 9. The second, more significant reason why most datasets are not usable for WRS testing is classification schema. WRS uses classification schema in addition to the rating to determine both the credibility of the article and the users understanding of the article. Therefore, Epinions dataset as well as other (MovieLans, Last.fm etc.) cannot be applied in evaluation of WRS.

WRS dataset has not been formed yet. Previous versions of WRS used Wikipedia as storage for ratings. Extraction of the ratings would take considerable time as it is unknown which articles have been rated and which not. WRS is a distributed system with the localized trust profiles which means that all trust data is stored on users' own machines, and the only way of extracting data would be to ask users to share it.

This is the first time that users' recommendations of other users are introduced to the system. It is also the first time that the system is keeping track of the rated articles, ratings and recommendations and stores them in one centralized storage, independent from Wikipedia. The best way of evaluating the system would be to expose it publicly and allow on-line testing for certain period of time, until enough annotation is collected to form a complete dataset of WRS test data. Such a dataset could be later used for off-line evaluation of trust function, classification schema, trust propagation function etc.
The on-line evaluation of the system could also include usability test, to determine the overall user satisfaction of the system as well as system performance. It is worth to mention that proper evaluation does not only include the testing of the accuracy of the predicted data, but the WRS as recommender system. Here issues such as user satisfaction, performance and intuitiveness of the graphical user interface play much bigger role than accuracy.

The time required to conduct a throughout evaluation exceeds the time limit of master thesis and can be a good foundation for future work on the system.

## 7.3 A concrete scenario

Concerning what has been previously said, the most intuitive way to evaluate that the system is reducing the cold start problem is to perform a concrete scenario which clearly demonstrates such a situation.

I must state that conducting a test scenario would be best with group of 20 real users or simulation of such a group. However, as has been previously said many times: WRS is a distributed system where each user keep his own trust profile (web of trust) on his own machine. Simulation of 20 users on the single machine is exhaustive as it is not possible to run 20 instances of the WRS system on the same machine. Performing behavior of each user at the time is time consuming and brings issues such as maintenance of the 20 different trust profiles (source code of the system would have to be changed to accommodate this). Therefore, I have decided to perform a simple scenario with 3 users which illustrates in the best possible manner how the system behaves.

For the requirements of the scenario, 3 WRS users have been created with the following properties:

| **WRS User** | Username | Password |
|---|---|---|
| jane | jane | wrsjane |
| bob | bob | wrsbob |
| alice | alice | wrsalice |

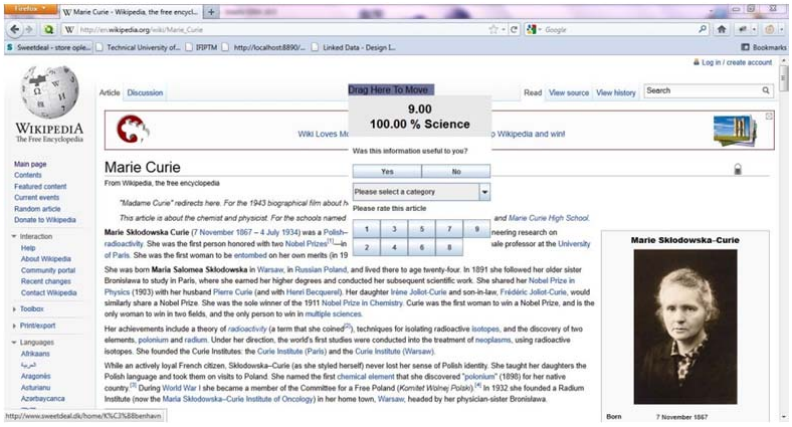
Jane is interested in science and she often evaluates articles about famous scientists. She has evaluated articles about Albert Einstein, Thomas Edison, Nikola Tesla and Marie Curie. She classified them all in category Science and assigned ratings 7, 6, 8 and 9 respectively.

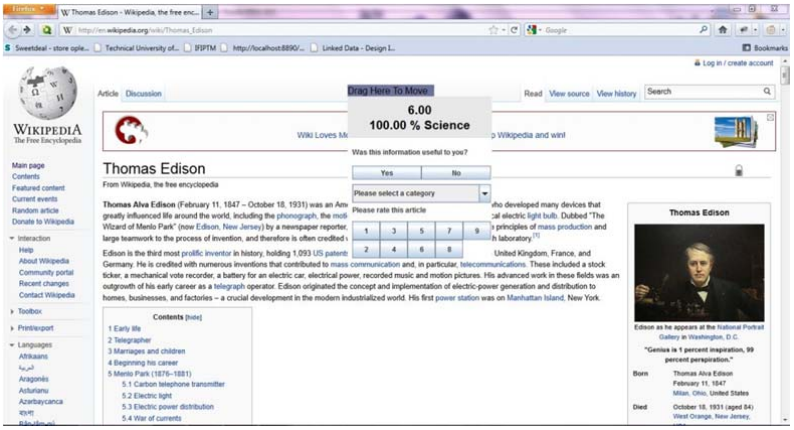
| **User** | **Article** | **Category** | **Rating** |
|---|---|---|---|
| jane | http://en.wikipedia.org/wiki/Albert_Einstein | Science | 7 |
| jane | http://en.wikipedia.org/wiki/Thomas_Edison | Science | 6 |
| jane | http://en.wikipedia.org/wiki/Nikola_Tesla | Science | 8 |
| jane | http://en.wikipedia.org/wiki/Marie_Curie | Science | 9 |

Bob is an ordinary WRS user who has recently read an article about Albert Einstein and submitted his evaluation of the article. He has classified the article in category Science and assigned it rating of value 8.

As Bob has rated the article about Einstein similar to Jane, trust connection between Bob and Jane is established and Jane is added to Bob's web of trust with the trust value of 0.0999 in category Science.

Bob decides to read articles about Thomas Edison and Marie Curie. Jane has already evaluated these two articles and, as trust connection between her and Bob already exists, Bob is able to see her ratings.





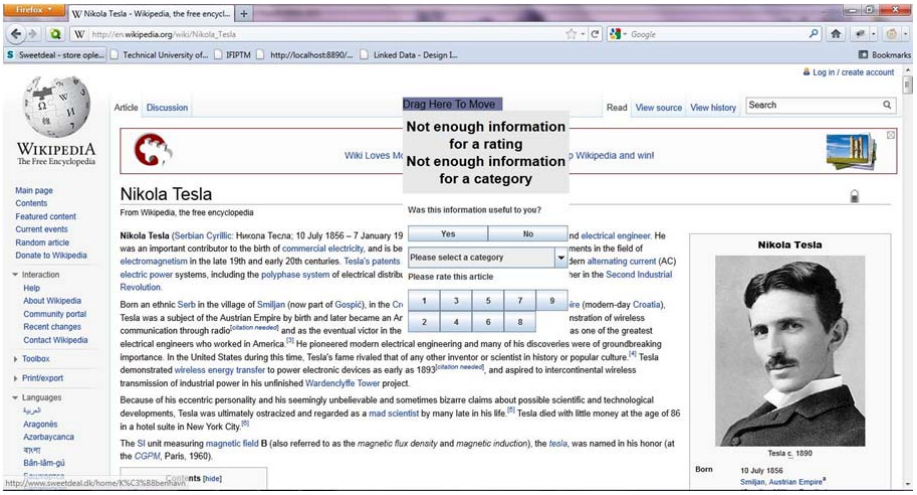

Now Bob want to submit evaluation of the articles. He agrees with Jane about the category of the articles, but gives different ratings. He agrees with the rating for the article about Edison, but gives 8 to the article about Marie Curie. Bob has generally agreed with Jane. The classification category is the same and there is a great similarity among the ratings. Therefore, trust between Bob and Jane is increased and WRS calculates that trust value is 4.151.

Bob has also evaluated the article about Electricity. Bob's rating session can be summarized in the table below.

| **User** | **Article** | **Category** | **Rating** |
|---|---|---|---|
| bob | http://en.wikipedia.org/wiki/Albert_Einstein | Science | 8 |
| bob | http://en.wikipedia.org/wiki/Thomas_Edison | Science | 6 |
| bob | http://en.wikipedia.org/wiki/Marie_Curie | Science | 8 |
| bob | http://en.wikipedia.org/wiki/Electricity | Science | 7 |

Alice is a cold start user. She is new to the system and does not have any interactions yet. Evaluations of the previous articles are not available to her until she rates at least one article and establishes the positive or negative trust connection with at least one of the reviewers. She decides to evaluate the article about Electricity. She classified the article as Science with rating 8. Her rating is similar to Bob's rating, so the trust connection between Alice and Bob is established and Bob is added to Alice's web of trust with trust value 0.0999. After this initial connection, Alice is able to see all article evaluations which Bob has submitted.

Further, Alice wants to know the credibility of the article about Nikola Tesla. As Bob is her only connection and Bob has not evaluated this article, Alice is facing the cold start problem again.

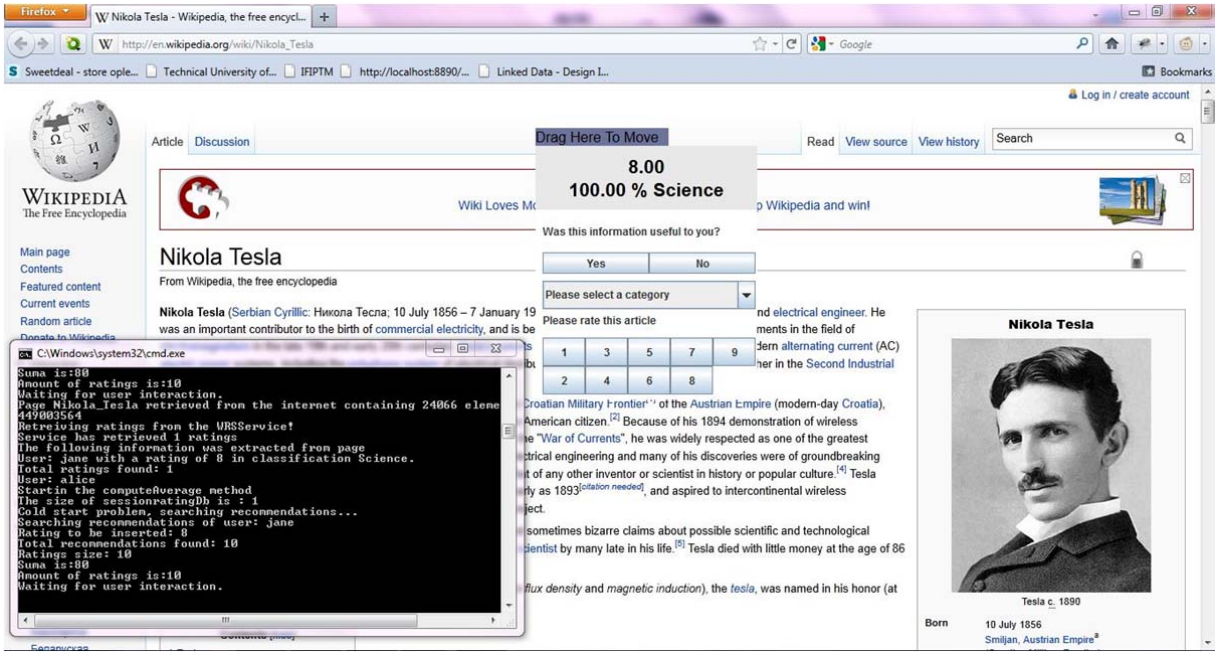



This is how the previous version of WRS is functioning. Alice would have to rate the article first before she would be able to see other peoples evaluation. Alice would have to rate many articles before she gets sufficient number of trusted connections, until the cold start issue becomes reduced.

With the proposed solution for reducing the cold start problem, discussed in this thesis, Alice avoids the cold start in the following way:

After having number of interactions with different users, Bob decides to share his positive interactions with other WRS users. He uses WRS to issue recommendations to people he trusts in certain category. Jane is one of them. Trust between Bob and Jane, developed by classifying and rating articles in the similar manner, reached value of 0.415 in category Science (on the scale –1 to 1). This makes Jane a trustworthy reviewer with high trust value in category Science.

Alice, using the new WRS and reaching for the article about Nikola Tesla will get the following feedback:




The small black console window in the left is used to follow the output from the WRS. It demonstrates how the system reports the cold start problem, searches for recommendations of users who have rated the article (in this concrete case, user is Jane) and matches recommendations with Alice's web of trust, to find recommendation given by people she trusts.

Bob's trust in Jane as trusted reviewer in category Science is propagated to Alice. As Alice trusts Bob on Science, Bob recommends Jane on Science and Jane has evaluated the article about Tesla - Alice is able to see Jane's evaluation. The same applies for all Jane's evaluations of articles classified in category Science, with exception that Bob has not rated them, as the advantage is always given to direct trust connections. In this concrete case, Alice did not have the cold start problem as the same has been reduced by having the recommendation given by somebody that Alice's trust to one of the reviewers of the article.

## 7.4 Summary

First difficulty is concerning of having a suitable dataset for testing. Properties of dataset such as rating scale, rating density etc. can make testing hard for algorithms that are not specifically designed for such a dataset. Evaluation goals can differ as some evaluation focus on accuracy of predicted recommendations while others can focus more on user satisfaction. Finally, there is a challenge of selecting the wright combination of measures for comparative evaluation as all systems perform the best when tuned to the optimum.

Evaluation of WRS is hard as there is no previously generated dataset and simulation of the pre-created dataset on the single machine can be challenging due to the system properties (such as that system is distributed, client side used embedded proxy server etc.). The proposed evaluation strategy is to let the

user perform on-line testing of the system for certain period of time, until the sufficient dataset is created to allow the off-line analysis of the accuracy of predictions. On-line testing can contribute to evaluation of non-functional properties of the system such as user experience and satisfaction.

Finally, a concrete scenario is performed to show interaction of 3 users where one would face the cold start problem with the previous version of WRS. With the trust propagation solution proposed in this thesis, and implemented in the new version of WRS, the concrete user is avoiding the cold start problem as the trust in reviewer of the article is propagated to her, by recommendation given to one of the reviewers from someone she trusts.

# Chapter 8: Future work

During the development of this project, several thoughts crossed my mind which could be considered as future work on the WRS project.

The system needs proper evaluation. The best evaluation of the system would be to conduct an on-line testing of the system for certain period of time. Such a testing can generate a valuable dataset which can be used for off-line analysis of the trust function, classification, propagation algorithm etc. On-line testing can contribute in determining other properties of the system than accuracy of the calculation, such as performance, user satisfaction, intuitiveness of use and valuable input to possible new features.

Concerning security, the analysis of the system's vulnerability and possible attacks would be beneficial. Attacks such as Sybil attack, where an attacker subverts the system by creating large number of identities in order to gain a disproportionately large influence on the trust network, are very common in reputation systems. Analysis of the possible attacks can tell a lot about trustworthiness of the examined recommender system.

It would be beneficial to conduct usability study of the system. The current solution as part of Scone is quite confusing as several windows are opened, one for http server, one for rmi registry and one for the Scone proxy. Previous authors pointed out the loading time of the article when WRS applet interface is injected to the HTML page. The conciseness of the messages provided by WRS calculator interface could be examined too. The performed usability study could lead to reengineering of the client side of the application.

Concerning the cold start problem, some sort of data mining of the submitted users' recommendations in other users would be beneficial in order to see: who are the top recommended reviewers per category, who is the most recommended reviewer of the article, etc. WRS is a decentralized system, but part of it is centralized as storage. Having the centralized unit that would perform such calculations would not compromise the core of the system as trust profile still stays distributed the same as trust calculations. But it would help cold start with no previous interactions with other users to choose people who seem trustworthy and add them to their web of trust as initial connections. This leads to another possible extension of the current system, to allow adding trusted people to the web of trust even if they do not have previous interaction based on similarity of the ratings. Other creative solutions such as the use of WikiTrust[7] are welcome too.

In the end, privacy risk analysis sounds like a good idea. The current press to secure people's privacy in social media is significant indicator to take into consideration privacy issues when designing a recommender system. Recommender system do store and publish sensitive trust data, and mistaken privacy risks together with security leeks are serious threats to a recommender system.

[7] http://www.wikitrust.net/

# Chapter 9: Conclusion

In this thesis a profound analysis of the current WRS solution is performed. Design of the WRS system is questioned and it led to abandoning of the Wikipedia as the storage for ratings and recommendations. The completely new design of the system is introduced where all WRS data is stored on the separate server. WRS users do not have to be registered Wikipedia users any more as WRS is maintaining their profiles. The system also enables them to manage their profiles via available web application.

The core values of the WRS are preserved. The system is still distributed, localized and open-source. Open source nature of the system is kept by introducing semantic web to the system and publishing data as LinkedData which are easily read and processed by other applications. To the open source nature of the system contributes, also, the fact that system is distributed and all calculations are done on the client side, and localized so that trust information is kept by the user. This contributes to the future improvements to the system accuracy as researchers can use the data to create more precise trust calculations.

Improvement to the way how user credentials were sent over the network is implemented. Previously, unencrypted user login information was sent over the network. Currently, all exchange between the client and the server part of the system is done over the HTTPS which ensures secure transmission of the data.

Possible solutions to the cold start problem are presented, discussed and one of them, implemented. The implemented solution reduces the cold start problem introducing the trust propagation and letting users to propagate part of their trust in other users by issuing recommendations. Positive experiences which one user had with other users are expressed as recommendations and published to the community. They are the most beneficial to cold start users, which could occupy up to 50% of the whole community. The prerequisite to such solution is users' willingness to share information about whom they trust with other users, and willingness of cold start users to trust submitted recommendations. It had been shown in literature and practice that people are willing to share if given something in return, and that people trust more trust statement issued by someone they trust than those generated by the recommender system.

It is important to note that implemented solution reduces the cold start problem but does not remove it entirely. It is best suited to users who have few connections, and not those who are using the system for the first time. They will still face the cold start problem. The user needs to have at least one connection for the solution to work. The same time a user rates an article for the first time, connections based on the similarity are established and his chance of facing the cold start problem again is significantly reduced.

The conclusion can be made that as soon as that one connection is made, user can reach significantly more article evaluation than with the previous versions of WRS. User's web of trust will grow, possibly exponentially, as more trusted users can be reached via trust propagation.

An example of the growth of the web of trust of the user "Alice" is shown in the Figure 9.1.

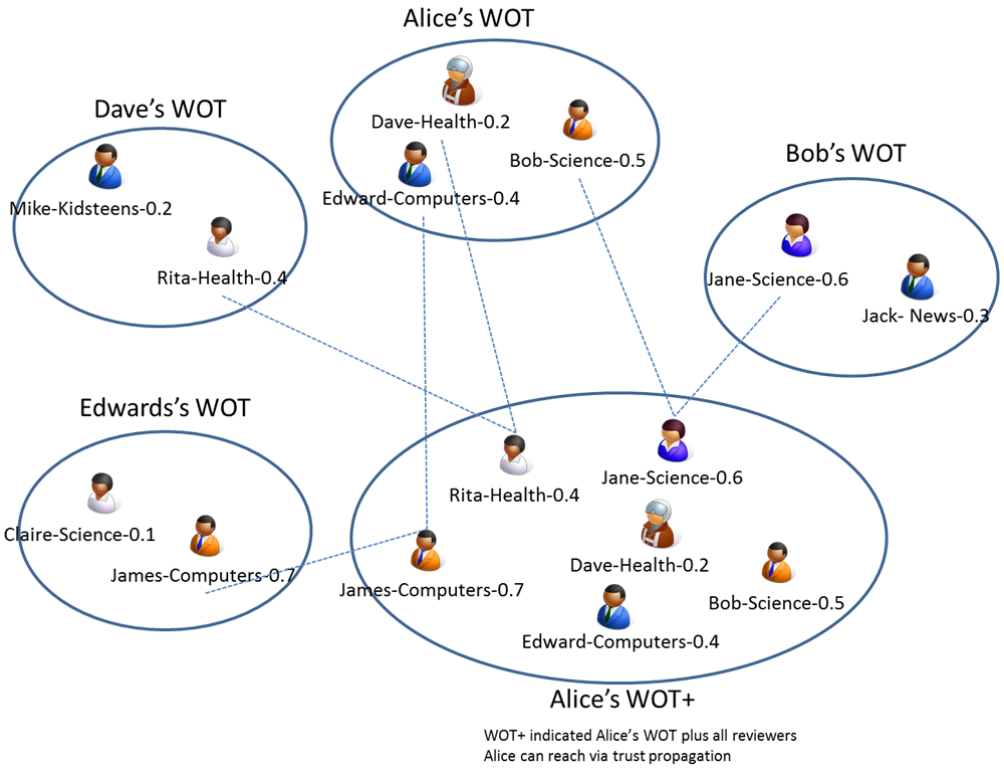



**Figure 9.1 – Growth of the user's web of trust enabled by trust propagation**

In the end, several ideas are given in the Future work chapter which are worth of consideration. The most important one is concerning the proper evaluation of the system. Not less valuable are ones that suggest the risk analysis of the privacy issues and security vulnerabilities. Distributed recommender systems such as WRS are very sensitive to those issues.

# References


Robert McHenry, “The Faith-Based Encyclopedia,” in TCSDaily.com, 15 November 2004, visited 14 March 2009.

Peter Denning, Jim Horning, David Parnas and Lauren Weinstein, “Wikipeadia Risks,” in Inside Risks 186. Communications of the ACM, 48(12), 2005.

John Seigenthaler, “A false Wikipedia biography,” editorial in USA TODAY, 29 November 2005.

Andrew Orlowski, “Avoid Wikipeadia, warns Wikipedia chief, It can seriously damage your grades.” In the Register, 15th June 2006 .

Thomas Rune Korsgaard, “Improving Trust in the Wikipedia.” M.Sc. Thesis, Department of Informatics and Mathematical Modeling, Technical University of Denmark, 2007.

Thomas Rune Korsgaard and Christian Damsgaard Jensen, “Reengineering the Wikipedia for Reputation,” In Proceeding of the 4th International Workshop on Security and Trust Management (STM 08), Trondheim, Norway, 2008, pp. 71-84.

International Workshop on Security and Trust Management (STM 08), Trondheim, Norway, 2008, pp. 71-84.

Christian Damsgaard Jensen and Thomas Rune Korsgaard, “Dynamics of Trust Evolution: Auto-configuration of dispositional trust dynamics,” In Proceeding of the International Conference on Security and Chryptography (SECRYPT 2008),” Porto, Portugal, 2008.

Scone website http://www.scone.de, visited 15 March 2009

John R. Douceur, “The Sybil Attack,” In Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)

Paolo Avesani, Paolo Massa, Roberto Tiella, “Moleskiing.it: A Trust-Aware Recommender System for Ski Mounteneering,” 2005.

Thomas Lefevre, ”Extending the Wikipedia Recommender System”, Assessing Expertese of Recomenders,” M.Sc. Thesis, Department of Informatics and Mathematical Modeling, Technical Univercity of Denmark, 2009

Wikipedia, www.wikipedia.org

Patricia Victor, Chris Cornelis, Ankur M. Teredesai, Martine De Cock, “Whom Should I Trust? The Impact of Key Figures on Cold Start Recommendations, ” Proceeding of the 23rd Annual ACM Symposium on Applied Computing (SAC08), p 2014-2018, 2008

Paolo Avesani, Paolo Massa, Roberto Tiella, “A Trust-enhanced Recommender System application: Moleskiing, ” Proceedings of the 2005 ACm symposium on Applied computing, Santa Fe, New Mexico, 2005

Jennifer Golbeck, Bijan Parsia, James Hendler, “Trust Networks on the Semantic Web,” In Proceeding of Cooperative Intelligent Agents

The friend of a Friend (FOAF) project, http://foaf-project.org

John Hebeler, “Semantic Web Programming”, Wiley, 2009

Shelly Powers, “Practical RDF”, O’Reilly, 2003

http://www.w3.org/Submission/SPARQL-Update/

Jena – A Semantic Web Framework for JAVA, http://jena.sourceforge.net/

http://en.wikipedia.org/wiki/Cold_start

The Guardian, “Wikipedia growth 'slowing' as it reaches 3 million articles”, http://www.telegraph.co.uk/technology/wikipedia/6020775/Wikipedia-growth-slowing-as-it-reaches-3-million-articles.html

Wiki-Java, “A lightweight bot framework”, http://code.google.com/p/wiki-java/

Patricia Victor, “Trust networks for Recommender Systems”, Dissertation submitted to the Faculty of Sciences of Ghent University in fulfillment of the requirements for the degree of Doctor of Computer Science, June 2010

Paolo Massa, Paolo Avesani, “Trust Metrics in Recommender Systems”, in Computing with Social Trust, J.Golbeck (Ed.), Springer. ISBN: 978-1-84800-355-2

Jeniffer Golbeck, James Hendler, “Filmtrust: movie recommendations using trust in web-based social networks”. In Proceedings of the 3rd IEEE Consumer Communications and Networking Conference (2006), pp. 282–286

Povilas Pilauskas, Expertise classification of recommenders in the Wikipedia Recommender System, MSc thesis, DTU, 2010

Christian Damsgaard Jensen, “Supporting Multi-Agent Reputation Calculation in the Wikipedia Recommender System”, Informatics and Mathematical Modeling, Technical University of Denmark

FilmTrust, http://trust.mindswap.org/FilmTrust/

Moleskiing, http://www.moleskiing.it/mski/home.do

Epinions, http://www.epinions.com/

Open Link Data Spaces, http://ods.openlinksw.com/wiki/main/Main/OdsIndex

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

Time Berners-Lee, "Linked Data", http://www.w3.org/DesignIssues/LinkedData.html, 2006

Deploying Linked Data, http://virtuoso.openlinksw.com/whitepapers/VirtDeployingLinkedDataGuide_Introduction.html#mozTocId883331

# Appendix

## A1. WDSL of the WRS Service

```
<?xml version="1.0" encoding="UTF-8"?>

<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:ns1="http://org.apache.axis2/xsd" xmlns:ns="http://service"
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:ax23="http://rating/xsd" xmlns:ax21="http://service/xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
targetNamespace="http://service">

    <wsdl:documentation>

                Please Type your service description here

        </wsdl:documentation>

    <wsdl:types>

        <xs:schema attributeFormDefault="qualified"
elementFormDefault="qualified" targetNamespace="http://service/xsd">

            <xs:complexType name="WRSServiceException">

                <xs:sequence>

                    <xs:element minOccurs="0" name="message" nillable="true"
type="xs:string"/>

                </xs:sequence>

            </xs:complexType>

        </xs:schema>

        <xs:schema attributeFormDefault="qualified"
elementFormDefault="qualified" targetNamespace="http://rating/xsd">

            <xs:complexType name="Recommendation">

                <xs:sequence>

                    <xs:element minOccurs="0" name="category" nillable="true"
type="xs:string"/>
```

```
                    <xs:element minOccurs="0" name="recommender" nillable="true"
type="xs:string"/>

                    <xs:element minOccurs="0" name="reviewer" nillable="true"
type="xs:string"/>

                    <xs:element minOccurs="0" name="trustValue"
type="xs:double"/>

                </xs:sequence>

            </xs:complexType>

            <xs:complexType name="Rating">

                <xs:sequence>

                    <xs:element minOccurs="0" name="articleName" nillable="true"
type="xs:string"/>

                    <xs:element minOccurs="0" name="articleURL" nillable="true"
type="xs:string"/>

                    <xs:element minOccurs="0" name="category" nillable="true"
type="xs:string"/>

                    <xs:element minOccurs="0" name="date" nillable="true"
type="xs:date"/>

                    <xs:element minOccurs="0" name="rating" type="xs:int"/>

                    <xs:element minOccurs="0" name="userName" nillable="true"
type="xs:string"/>

                    <xs:element minOccurs="0" name="version" nillable="true"
type="xs:string"/>

                </xs:sequence>

            </xs:complexType>

        </xs:schema>

        <xs:schema xmlns:ax24="http://rating/xsd"
xmlns:ax22="http://service/xsd" attributeFormDefault="qualified"
elementFormDefault="qualified" targetNamespace="http://service">

            <xs:import namespace="http://service/xsd"/>

            <xs:import namespace="http://rating/xsd"/>

            <xs:element name="WrsServiceException">

                <xs:complexType>

                    <xs:sequence>
```

```
                        <xs:element minOccurs="0" name="WrsServiceException"
nillable="true" type="ns:Exception"/>

                    </xs:sequence>

                </xs:complexType>

            </xs:element>

            <xs:complexType name="Exception">

                <xs:sequence>

                    <xs:element minOccurs="0" name="Message" nillable="true"
type="xs:string"/>

                </xs:sequence>

            </xs:complexType>

            <xs:element name="setRating">

                <xs:complexType>

                    <xs:sequence>

                        <xs:element minOccurs="0" name="sessionId"
nillable="true" type="xs:string"/>

                        <xs:element minOccurs="0" name="articleName"
nillable="true" type="xs:string"/>

                        <xs:element minOccurs="0" name="articleUrl"
nillable="true" type="xs:string"/>

                        <xs:element minOccurs="0" name="category"
nillable="true" type="xs:string"/>

                        <xs:element minOccurs="0" name="version" nillable="true"
type="xs:string"/>

                        <xs:element minOccurs="0" name="rating" type="xs:int"/>

                    </xs:sequence>

                </xs:complexType>

            </xs:element>

            <xs:element name="setRatingResponse">

                <xs:complexType>

                    <xs:sequence>

                        <xs:element minOccurs="0" name="return" nillable="true"
type="xs:string"/>
```

```
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="WrsServiceWRSServiceException">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="WRSServiceException"
nillable="true" type="ax21:WRSServiceException"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="register">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="username"
nillable="true" type="xs:string"/>
                        <xs:element minOccurs="0" name="password"
nillable="true" type="xs:string"/>
                        <xs:element minOccurs="0" name="email" nillable="true"
type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="registerResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true"
type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
```

```
            <xs:element name="recommend">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="sessionId"
nillable="true" type="xs:string"/>
                        <xs:element maxOccurs="unbounded" minOccurs="0"
name="recommendations" nillable="true" type="ax23:Recommendation"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="recommendResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true"
type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="logout">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="sessionId"
nillable="true" type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="logoutResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true"
type="xs:string"/>
```

```
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="login">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="username"
nillable="true" type="xs:string"/>
                        <xs:element minOccurs="0" name="password"
nillable="true" type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="loginResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true"
type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="getRecommendations">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="sessionId"
nillable="true" type="xs:string"/>
                        <xs:element minOccurs="0" name="reviewer"
nillable="true" type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
```

```
            <xs:element name="getRecommendationsResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element maxOccurs="unbounded" minOccurs="0"
name="return" nillable="true" type="ax23:Recommendation"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="getRatings">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="articleUrl"
nillable="true" type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="getRatingsResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element maxOccurs="unbounded" minOccurs="0"
name="return" nillable="true" type="ax23:Rating"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:schema>
    </wsdl:types>
    <wsdl:message name="logoutRequest">
        <wsdl:part name="parameters" element="ns:logout"/>
    </wsdl:message>
    <wsdl:message name="logoutResponse">
```

```
        <wsdl:part name="parameters" element="ns:logoutResponse"/>
    </wsdl:message>
    <wsdl:message name="WrsServiceWRSServiceException">
        <wsdl:part name="parameters"
element="ns:WrsServiceWRSServiceException"/>
    </wsdl:message>
    <wsdl:message name="recommendRequest">
        <wsdl:part name="parameters" element="ns:recommend"/>
    </wsdl:message>
    <wsdl:message name="recommendResponse">
        <wsdl:part name="parameters" element="ns:recommendResponse"/>
    </wsdl:message>
    <wsdl:message name="getRatingsRequest">
        <wsdl:part name="parameters" element="ns:getRatings"/>
    </wsdl:message>
    <wsdl:message name="getRatingsResponse">
        <wsdl:part name="parameters" element="ns:getRatingsResponse"/>
    </wsdl:message>
    <wsdl:message name="loginRequest">
        <wsdl:part name="parameters" element="ns:login"/>
    </wsdl:message>
    <wsdl:message name="loginResponse">
        <wsdl:part name="parameters" element="ns:loginResponse"/>
    </wsdl:message>
    <wsdl:message name="getRecommendationsRequest">
        <wsdl:part name="parameters" element="ns:getRecommendations"/>
    </wsdl:message>
    <wsdl:message name="getRecommendationsResponse">
        <wsdl:part name="parameters" element="ns:getRecommendationsResponse"/>
    </wsdl:message>
```

```
    <wsdl:message name="WrsServiceException">
        <wsdl:part name="parameters" element="ns:WrsServiceException"/>
    </wsdl:message>
    <wsdl:message name="setRatingRequest">
        <wsdl:part name="parameters" element="ns:setRating"/>
    </wsdl:message>
    <wsdl:message name="setRatingResponse">
        <wsdl:part name="parameters" element="ns:setRatingResponse"/>
    </wsdl:message>
    <wsdl:message name="registerRequest">
        <wsdl:part name="parameters" element="ns:register"/>
    </wsdl:message>
    <wsdl:message name="registerResponse">
        <wsdl:part name="parameters" element="ns:registerResponse"/>
    </wsdl:message>
    <wsdl:portType name="WrsServicePortType">
        <wsdl:operation name="logout">
            <wsdl:input message="ns:logoutRequest" wsaw:Action="urn:logout"/>
            <wsdl:output message="ns:logoutResponse"
wsaw:Action="urn:logoutResponse"/>
            <wsdl:fault message="ns:WrsServiceWRSServiceException"
name="WrsServiceWRSServiceException"
wsaw:Action="urn:logoutWrsServiceWRSServiceException"/>
        </wsdl:operation>
        <wsdl:operation name="recommend">
            <wsdl:input message="ns:recommendRequest"
wsaw:Action="urn:recommend"/>
            <wsdl:output message="ns:recommendResponse"
wsaw:Action="urn:recommendResponse"/>
            <wsdl:fault message="ns:WrsServiceWRSServiceException"
name="WrsServiceWRSServiceException"
wsaw:Action="urn:recommendWrsServiceWRSServiceException"/>
```

```
        </wsdl:operation>

        <wsdl:operation name="getRatings">

            <wsdl:input message="ns:getRatingsRequest"
wsaw:Action="urn:getRatings"/>

            <wsdl:output message="ns:getRatingsResponse"
wsaw:Action="urn:getRatingsResponse"/>

            <wsdl:fault message="ns:WrsServiceWRSServiceException"
name="WrsServiceWRSServiceException"
wsaw:Action="urn:getRatingsWrsServiceWRSServiceException"/>

        </wsdl:operation>

        <wsdl:operation name="login">

            <wsdl:input message="ns:loginRequest" wsaw:Action="urn:login"/>

            <wsdl:output message="ns:loginResponse"
wsaw:Action="urn:loginResponse"/>

            <wsdl:fault message="ns:WrsServiceWRSServiceException"
name="WrsServiceWRSServiceException"
wsaw:Action="urn:loginWrsServiceWRSServiceException"/>

        </wsdl:operation>

        <wsdl:operation name="getRecommendations">

            <wsdl:input message="ns:getRecommendationsRequest"
wsaw:Action="urn:getRecommendations"/>

            <wsdl:output message="ns:getRecommendationsResponse"
wsaw:Action="urn:getRecommendationsResponse"/>

            <wsdl:fault message="ns:WrsServiceException"
name="WrsServiceException"
wsaw:Action="urn:getRecommendationsWrsServiceException"/>

        </wsdl:operation>

        <wsdl:operation name="setRating">

            <wsdl:input message="ns:setRatingRequest"
wsaw:Action="urn:setRating"/>

            <wsdl:output message="ns:setRatingResponse"
wsaw:Action="urn:setRatingResponse"/>

            <wsdl:fault message="ns:WrsServiceException"
name="WrsServiceException" wsaw:Action="urn:setRatingWrsServiceException"/>

        </wsdl:operation>
```

```
        <wsdl:operation name="register">

            <wsdl:input message="ns:registerRequest"
wsaw:Action="urn:register"/>

            <wsdl:output message="ns:registerResponse"
wsaw:Action="urn:registerResponse"/>

            <wsdl:fault message="ns:WrsServiceWRSServiceException"
name="WrsServiceWRSServiceException"
wsaw:Action="urn:registerWrsServiceWRSServiceException"/>

        </wsdl:operation>

    </wsdl:portType>

    <wsdl:binding name="WrsServiceSoap11Binding" type="ns:WrsServicePortType">

        <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>

        <wsdl:operation name="logout">

            <soap:operation soapAction="urn:logout" style="document"/>

            <wsdl:input>

                <soap:body use="literal"/>

            </wsdl:input>

            <wsdl:output>

                <soap:body use="literal"/>

            </wsdl:output>

            <wsdl:fault name="WrsServiceWRSServiceException">

                <soap:fault use="literal" name="WrsServiceWRSServiceException"/>

            </wsdl:fault>

        </wsdl:operation>

        <wsdl:operation name="recommend">

            <soap:operation soapAction="urn:recommend" style="document"/>

            <wsdl:input>

                <soap:body use="literal"/>

            </wsdl:input>

            <wsdl:output>
```

```
            <soap:body use="literal"/>
        </wsdl:output>
        <wsdl:fault name="WrsServiceWRSServiceException">
            <soap:fault use="literal" name="WrsServiceWRSServiceException"/>
        </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="login">
        <soap:operation soapAction="urn:login" style="document"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
        <wsdl:fault name="WrsServiceWRSServiceException">
            <soap:fault use="literal" name="WrsServiceWRSServiceException"/>
        </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="getRatings">
        <soap:operation soapAction="urn:getRatings" style="document"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
        <wsdl:fault name="WrsServiceWRSServiceException">
            <soap:fault use="literal" name="WrsServiceWRSServiceException"/>
        </wsdl:fault>
```

```
        </wsdl:operation>
        <wsdl:operation name="getRecommendations">
            <soap:operation soapAction="urn:getRecommendations"
style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="WrsServiceException">
                <soap:fault use="literal" name="WrsServiceException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="setRating">
            <soap:operation soapAction="urn:setRating" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="WrsServiceException">
                <soap:fault use="literal" name="WrsServiceException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="register">
            <soap:operation soapAction="urn:register" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
```

```
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="WrsServiceWRSServiceException">
                <soap:fault use="literal" name="WrsServiceWRSServiceException"/>
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:binding name="WrsServiceSoap12Binding" type="ns:WrsServicePortType">
        <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
        <wsdl:operation name="logout">
            <soap12:operation soapAction="urn:logout" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="WrsServiceWRSServiceException">
                <soap12:fault use="literal"
name="WrsServiceWRSServiceException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="recommend">
            <soap12:operation soapAction="urn:recommend" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
```

```
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="WrsServiceWRSServiceException">
                <soap12:fault use="literal"
name="WrsServiceWRSServiceException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="login">
            <soap12:operation soapAction="urn:login" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="WrsServiceWRSServiceException">
                <soap12:fault use="literal"
name="WrsServiceWRSServiceException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="getRatings">
            <soap12:operation soapAction="urn:getRatings" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="WrsServiceWRSServiceException">
```

```
                <soap12:fault use="literal"
name="WrsServiceWRSServiceException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="getRecommendations">
            <soap12:operation soapAction="urn:getRecommendations"
style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="WrsServiceException">
                <soap12:fault use="literal" name="WrsServiceException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="setRating">
            <soap12:operation soapAction="urn:setRating" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="WrsServiceException">
                <soap12:fault use="literal" name="WrsServiceException"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="register">
```

```
            <soap12:operation soapAction="urn:register" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="WrsServiceWRSServiceException">
                <soap12:fault use="literal"
name="WrsServiceWRSServiceException"/>
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:binding name="WrsServiceHttpBinding" type="ns:WrsServicePortType">
        <http:binding verb="POST"/>
        <wsdl:operation name="logout">
            <http:operation location="logout"/>
            <wsdl:input>
                <mime:content type="text/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="text/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="recommend">
            <http:operation location="recommend"/>
            <wsdl:input>
                <mime:content type="text/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
```

```
        <mime:content type="text/xml" part="parameters"/>
    </wsdl:output>
</wsdl:operation>
<wsdl:operation name="login">
    <http:operation location="login"/>
    <wsdl:input>
        <mime:content type="text/xml" part="parameters"/>
    </wsdl:input>
    <wsdl:output>
        <mime:content type="text/xml" part="parameters"/>
    </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getRatings">
    <http:operation location="getRatings"/>
    <wsdl:input>
        <mime:content type="text/xml" part="parameters"/>
    </wsdl:input>
    <wsdl:output>
        <mime:content type="text/xml" part="parameters"/>
    </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getRecommendations">
    <http:operation location="getRecommendations"/>
    <wsdl:input>
        <mime:content type="text/xml" part="parameters"/>
    </wsdl:input>
    <wsdl:output>
        <mime:content type="text/xml" part="parameters"/>
    </wsdl:output>
```

```
        </wsdl:operation>
        <wsdl:operation name="setRating">
            <http:operation location="setRating"/>
            <wsdl:input>
                <mime:content type="text/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="text/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="register">
            <http:operation location="register"/>
            <wsdl:input>
                <mime:content type="text/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="text/xml" part="parameters"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="WrsService">
        <wsdl:port name="WrsServiceHttpsSoap11Endpoint"
binding="ns:WrsServiceSoap11Binding">
            <soap:address
location="https://wrs.imm.dtu.dk:8181/wrs/services/WrsService.WrsServiceHttpsSoa
p11Endpoint/"/>
        </wsdl:port>
        <wsdl:port name="WrsServiceHttpsSoap12Endpoint"
binding="ns:WrsServiceSoap12Binding">
            <soap12:address
location="https://wrs.imm.dtu.dk:8181/wrs/services/WrsService.WrsServiceHttpsSoa
p12Endpoint/"/>
```

```
        </wsdl:port>

        <wsdl:port name="WrsServiceHttpsEndpoint"
binding="ns:WrsServiceHttpBinding">

            <http:address
location="https://wrs.imm.dtu.dk:8181/wrs/services/WrsService.WrsServiceHttpsEnd
point/"/>

        </wsdl:port>

    </wsdl:service>

</wsdl:definitions>
```

## A2. Ontology used to convert relational data to RDF

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix aowl: <http://bblfish.net/work/atom-owl/2006-06-06/> .

@prefix foaf: <http://xmlns.com/foaf/0.1/> .

@prefix virtrdf: <http://www.openlinksw.com/schemas/virtrdf#> .

@prefix WRS: <http://localhost:8890/schemas/WRS/> .


WRS: a owl:Ontology .


# WRS.DBA.ARTICLE

WRS:ARTICLE a rdfs:Class .

WRS:ARTICLE rdfs:isDefinedBy WRS: .

WRS:ARTICLE rdfs:label "WRS.DBA.ARTICLE" .

WRS:article_id a owl:DatatypeProperty .

WRS:article_id rdfs:range xsd:int .
```

```
WRS:article_id rdfs:domain WRS:ARTICLE .

WRS:article_id rdfs:isDefinedBy WRS: .

WRS:article_id rdfs:label "ARTICLE_ID" .

WRS:article_name a owl:DatatypeProperty .

WRS:article_name rdfs:range xsd:string .

WRS:article_name rdfs:domain WRS:ARTICLE .

WRS:article_name rdfs:isDefinedBy WRS: .

WRS:article_name rdfs:label "ARTICLE_NAME" .

WRS:article_url a owl:DatatypeProperty .

WRS:article_url rdfs:range xsd:string .

WRS:article_url rdfs:domain WRS:ARTICLE .

WRS:article_url rdfs:isDefinedBy WRS: .

WRS:article_url rdfs:label "ARTICLE_URL" .

WRS:article_language a owl:DatatypeProperty .

WRS:article_language rdfs:range xsd:string .

WRS:article_language rdfs:domain WRS:ARTICLE .

WRS:article_language rdfs:isDefinedBy WRS: .

WRS:article_language rdfs:label "ARTICLE_LANGUAGE" .

WRS:has_rating a owl:ObjectProperty .

WRS:has_rating rdfs:range WRS:RATING .

WRS:has_rating rdfs:domain WRS:ARTICLE .

WRS:has_ratinge rdfs:isDefinedBy WRS: .

WRS:has_rating rdfs:label "ARTICLE_LANGUAGE" .


# WRS.DBA.CATEGORY

WRS:CATEGORY a rdfs:Class .

WRS:CATEGORY rdfs:isDefinedBy WRS: .

WRS:CATEGORY rdfs:label "WRS.DBA.CATEGORY" .

WRS:category_id a owl:DatatypeProperty .

WRS:category_id rdfs:range xsd:int .
```

```
WRS:category_id rdfs:domain WRS:CATEGORY .
WRS:category_id rdfs:isDefinedBy WRS: .
WRS:category_id rdfs:label "CATEGORY_ID" .
WRS:category_name a owl:DatatypeProperty .
WRS:category_name rdfs:range xsd:string .
WRS:category_name rdfs:domain WRS:CATEGORY .
WRS:category_name rdfs:isDefinedBy WRS: .
WRS:category_name rdfs:label "CATEGORY_NAME" .

# WRS.DBA.RATING
WRS:RATING a rdfs:Class .
WRS:RATING rdfs:isDefinedBy WRS: .
WRS:RATING rdfs:label "WRS.DBA.RATING" .
WRS:id a owl:DatatypeProperty .
WRS:id rdfs:range xsd:int .
WRS:id rdfs:domain WRS:RATING .
WRS:id rdfs:isDefinedBy WRS: .
WRS:id rdfs:label "ID" .
WRS:is_rating_of_article a owl:ObjectProperty .
WRS:is_rating_of_article rdfs:range WRS:ARTICLE .
WRS:is_rating_of_article rdfs:domain WRS:RATING .
WRS:is_rating_of_article rdfs:isDefinedBy WRS: .
WRS:is_rating_of_article rdfs:label "ARTICLE" .
WRS:article_version a owl:DatatypeProperty .
WRS:article_version rdfs:range xsd:string .
WRS:article_version rdfs:domain WRS:RATING .
WRS:article_version rdfs:isDefinedBy WRS: .
WRS:article_version rdfs:label "ARTICLE_VERSION" .
WRS:has_reviewer a owl:ObjectProperty .
WRS:has_reviewer rdfs:range foaf:Person .
```

```
WRS:has_reviewer rdfs:domain WRS:RATING .

WRS:has_reviewer rdfs:isDefinedBy WRS: .

WRS:has_reviewer rdfs:label "USERNAME" .

WRS:has_category a owl:ObjectProperty .

WRS:has_category rdfs:range WRS:CATEGORY .

WRS:has_category rdfs:domain WRS:RATING .

WRS:has_category rdfs:isDefinedBy WRS: .

WRS:has_category rdfs:label "CATEGORY" .

WRS:rating_value a owl:DatatypeProperty .

WRS:rating_value rdfs:range xsd:int .

WRS:rating_value rdfs:domain WRS:RATING .

WRS:rating_value rdfs:isDefinedBy WRS: .

WRS:rating_value rdfs:label "RATING_VALUE" .


# WRS.DBA.RECOMMENDATION

WRS:RECOMMENDATION a rdfs:Class .

WRS:RECOMMENDATION rdfs:isDefinedBy WRS: .

WRS:RECOMMENDATION rdfs:label "WRS.DBA.RECOMMENDATION" .

WRS:RECOMMENDATION_LIST a rdfs:Class .

WRS:RECOMMENDATION_LIST rdfs:isDefinedBy WRS: .

WRS:RECOMMENDATION_LIST rdfs:label "WRS.DBA.RECOMMENDATION" .

WRS:has_recommendation a owl:ObjectProperty .

WRS:has_recommendation rdfs:range WRS:RECOMMENDATION .

WRS:has_recommendation rdfs:domain WRS:RECOMMENDATION_LIST .

WRS:has_recommendation rdfs:isDefinedBy WRS: .

WRS:has_recommendation rdfs:label "RECOMMENDATION" .

WRS:has_recommender a owl:ObjectProperty .

WRS:has_recommender rdfs:range foaf:Person .

WRS:has_recommender rdfs:domain WRS:RECOMMENDATION .

WRS:has_recommender rdfs:isDefinedBy WRS: .
```

```
WRS:has_recommender rdfs:label "RECOMMENDER" .

WRS:has_reviewer a owl:ObjectProperty .

WRS:has_reviewer rdfs:range foaf:Person .

WRS:has_reviewer rdfs:domain WRS:RECOMMENDATION .

WRS:has_reviewer rdfs:isDefinedBy WRS: .

WRS:has_reviewer rdfs:label "REVIEWER" .

WRS:has_category a owl:ObjectProperty .

WRS:has_category rdfs:range WRS:CATEGORY .

WRS:has_category rdfs:domain WRS:RECOMMENDATION .

WRS:has_category rdfs:isDefinedBy WRS: .

WRS:has_category rdfs:label "CATEGORY" .

WRS:trust_value a owl:DatatypeProperty .

WRS:trust_value rdfs:range xsd:decimal .

WRS:trust_value rdfs:domain WRS:RECOMMENDATION .

WRS:trust_value rdfs:isDefinedBy WRS: .

WRS:trust_value rdfs:label "TRUST_VALUE" .
```

## A3. RDF Mappings of the relational data to Virtuoso RDF Views

```
SPARQL drop silent quad map <http://localhost:8890/schemas/WRS/qm-article> .;

SPARQL drop silent quad map <http://localhost:8890/schemas/WRS/qm-category> .;

SPARQL drop silent quad map <http://localhost:8890/schemas/WRS/qm-rating> .;

SPARQL drop silent quad map <http://localhost:8890/schemas/WRS/qm-recommendation> .;

grant select on "WRS"."DBA"."ARTICLE" to SPARQL_SELECT;

grant select on "WRS"."DBA"."CATEGORY" to SPARQL_SELECT;

grant select on "WRS"."DBA"."RATING" to SPARQL_SELECT;
```

```
grant select on "WRS"."DBA"."RECOMMENDATION" to SPARQL_SELECT;
```

SPARQL

```
prefix WRS: <http://localhost:8890/schemas/WRS/>

create iri class WRS:article "http://^{URIQADefaultHost}^/WRS/articles/%s/%s#this" (in _ARTICLE_LANGUAGE varchar not null,in _ARTICLE_NAME varchar not null) . ;
```

SPARQL

```
prefix WRS: <http://localhost:8890/schemas/WRS/>

create iri class WRS:category "http://^{URIQADefaultHost}^/WRS/categories/%U#this" (in _CATEGORY_NAME varchar not null) . ;
```

SPARQL

```
prefix WRS: <http://localhost:8890/schemas/WRS/>

create iri class WRS:rating "http://^{URIQADefaultHost}^/WRS/ratings/%d#this" (in _ID integer not null) . ;
```

SPARQL

```
prefix WRS: <http://localhost:8890/schemas/WRS/>

create iri class WRS:reviewer "http://^{URIQADefaultHost}^/dataspace/person/%U#this" (in _USERNAME varchar not null) . ;
```

SPARQL

```
prefix WRS: <http://localhost:8890/schemas/WRS/>

create iri class WRS:recommender "http://^{URIQADefaultHost}^/dataspace/person/%U#this" (in _USERNAME varchar not null) . ;
```

SPARQL

```
prefix WRS: <http://localhost:8890/schemas/WRS/>

create iri class WRS:recommendation "http://^{URIQADefaultHost}^/WRS/recommendations/%U_%U_%U#this" (in _RECOMMENDER varchar not null,in _REVIEWER varchar not null,in _CATEGORY_NAME varchar not null) . ;
```

SPARQL

```
prefix WRS: <http://localhost:8890/schemas/WRS/>
```

```
create iri class WRS:recommendation_list "http://^{URIQADefaultHost}^/WRS/recommendations/%U#this" (in _REVIEWER
varchar not null) . ;


SPARQL

prefix WRS: <http://localhost:8890/schemas/WRS/>

prefix aowl: <http://bblfish.net/work/atom-owl/2006-06-06/>

alter quad storage virtrdf:DefaultQuadStorage

 from "WRS"."DBA"."ARTICLE" as article_s

 from "WRS"."DBA"."RATING" as rating_s

 {

  create WRS:qm-article as graph iri ("http://^{URIQADefaultHost}^/WRS#")

   {

    # Maps from columns of "WRS.DBA.ARTICLE"

    WRS:article (article_s."ARTICLE_LANGUAGE",article_s."ARTICLE_NAME") a WRS:ARTICLE ;

    WRS:article_id article_s."ARTICLE_ID" as WRS:dba-article-article_id ;

    WRS:article_name article_s."ARTICLE_NAME" as WRS:dba-article-article_name ;

    WRS:article_url article_s."ARTICLE_URL" as WRS:dba-article-article_url ;

    WRS:article_language article_s."ARTICLE_LANGUAGE" as WRS:dba-article-article_language ;

    WRS:has_rating  WRS:rating (rating_s."ID")

     where (^{article_s.}^.ARTICLE_ID = ^{rating_s.}^.ARTICLE_ID) as virtrdf:dba-article-hasRating .

   }

 }


;


SPARQL

prefix WRS: <http://localhost:8890/schemas/WRS/>

prefix aowl: <http://bblfish.net/work/atom-owl/2006-06-06/>

alter quad storage virtrdf:DefaultQuadStorage

 from "WRS"."DBA"."CATEGORY" as category_s
```

```
{
  create WRS:qm-category as graph iri ("http://^{URIQADefaultHost}^/WRS#")
  {
    # Maps from columns of "WRS.DBA.CATEGORY"
    WRS:category (category_s."CATEGORY_NAME")  a WRS:CATEGORY ;
    WRS:category_id category_s."CATEGORY_ID" as WRS:dba-category-category_id ;
    WRS:category_name category_s."CATEGORY_NAME" as WRS:dba-category-category_name .

  }
}

;

SPARQL
prefix WRS: <http://localhost:8890/schemas/WRS/>
prefix aowl: <http://bblfish.net/work/atom-owl/2006-06-06/>
alter quad storage virtrdf:DefaultQuadStorage
 from "WRS"."DBA"."RATING" as rating_s
 from "WRS"."DBA"."ARTICLE" as article_s
 from "WRS"."DBA"."CATEGORY" as category_s
 {
  create WRS:qm-rating as graph iri ("http://^{URIQADefaultHost}^/WRS#")
  {
    # Maps from columns of "WRS.DBA.RATING"
    WRS:rating (rating_s."ID")  a WRS:RATING ;
    WRS:id rating_s."ID" as WRS:dba-rating-id ;
    WRS:article_version rating_s."ARTICLE_VERSION" as WRS:dba-rating-article_version ;
    WRS:has_reviewer WRS:reviewer(rating_s."USERNAME") as WRS:dba-rating-reviewer ;
    WRS:has_category  WRS:category (category_s."CATEGORY_NAME")
```

```
      where (^{category_s.}^.CATEGORY_NAME = ^{rating_s.}^.CATEGORY_NAME)
      as WRS:dba-rating-category ;
    WRS:rating_value rating_s."RATING_VALUE" as WRS:dba-rating-rating_value ;
    WRS:is_rating_of_article  WRS:article (article_s."ARTICLE_LANGUAGE",article_s."ARTICLE_NAME")
     where (^{article_s.}^.ARTICLE_ID = ^{rating_s.}^.ARTICLE_ID) as virtrdf:dba-article-isRatingOfArticle .
  }
}

;

SPARQL
prefix WRS: <http://localhost:8890/schemas/WRS/>
prefix aowl: <http://bblfish.net/work/atom-owl/2006-06-06/>
alter quad storage virtrdf:DefaultQuadStorage
 from "WRS"."DBA"."RECOMMENDATION" as recommendation_s
 from "DB"."DBA"."SYS_USERS" as user_s
 from "WRS"."DBA"."CATEGORY" as category_s
 where (^{user_s.}^.U_NAME = ^{recommendation_s.}^.REVIEWER)
 {
  create WRS:qm-recommendation as graph iri ("http://^{URIQADefaultHost}^/WRS#")
  {
    # Maps from columns of "WRS.DBA.RECOMMENDATION"
    WRS:recommendation_list(recommendation_s."REVIEWER") a WRS:RECOMMENDATION_LIST ;
    WRS:has_recommendation
WRS:recommendation(recommendation_s."RECOMMENDER",recommendation_s."REVIEWER",recommendation_s."CATEGORY
_NAME") as WRS:dba-recommendation-recommendation .

WRS:recommendation(recommendation_s."RECOMMENDER",recommendation_s."REVIEWER",recommendation_s."CATEGORY
_NAME")  a WRS:RECOMMENDATION ;
    WRS:has_recommender WRS:recommender(recommendation_s."RECOMMENDER") as WRS:dba-recommendation-
recommender ;
```

```
    WRS:has_reviewer WRS:reviewer(recommendation_s."REVIEWER") as WRS:dba-recommendation-reviewer ;

    WRS:has_category WRS:category(category_s."CATEGORY_NAME")

                    where (^{category_s.}^.CATEGORY_NAME = ^{recommendation_s.}^.CATEGORY_NAME)

                    as WRS:dba-recommendation-category ;

    WRS:trust_value recommendation_s."TRUST_VALUE" as WRS:dba-recommendation-trust_value .

  }

}


;




-- Virtual directories for instance data

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (

'wrs_rule2',

1,

'(/[^#]*)',

vector('path'),

1,

'/sparql?query=DESCRIBE+%%3Chttp%%3A//^{URIQADefaultHost}^%U%%23this%%3E+FROM+%%3Chttp%%3A//^{URIQADefaul
tHost}^/WRS%%23%%3E&format=%U',

vector('path', '*accept*'),

null,

'(text/rdf.n3)|(application/rdf.xml)|(text/n3)|(application/json)',

2,

null

);

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (

'wrs_rule4',
```

```
1,

'/WRS/stat([^#]*)',

vector('path'),

1,

'/sparql?query=DESCRIBE+%%3Chttp%%3A//^{URIQADefaultHost}^/WRS/stat%%23%%3E+%%3Fo+FROM+%%3Chttp%%3A//^{
URIQADefaultHost}^/WRS%%23%%3E+WHERE+{+%%3Chttp%%3A//^{URIQADefaultHost}^/WRS/stat%%23%%3E+%%3Fp+%%3
Fo+}&format=%U',

vector('*accept*'),

null,

'(text/rdf.n3)|(application/rdf.xml)|(text/n3)|(application/json)',

2,

null

);

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (

'wrs_rule6',

1,

'/WRS/objects/([^#]*)',

vector('path'),

1,

'/sparql?query=DESCRIBE+%%3Chttp%%3A//^{URIQADefaultHost}^/WRS/objects/%U%%3E+FROM+%%3Chttp%%3A//^{URIQA
DefaultHost}^/WRS%%23%%3E&format=%U',

vector('path', '*accept*'),

null,

'(text/rdf.n3)|(application/rdf.xml)|(text/n3)|(application/json)',

2,

null

);

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (

'wrs_rule1',

1,

'([^#]*)',
```

```
vector('path'),
1,
'/about/html/http://^{URIQADefaultHost}^%s',
vector('path'),
null,
null,
2,
303
);
DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
'wrs_rule7',
1,
'/WRS/stat([^#]*)',
vector('path'),
1,
'/about/html/http://^{URIQADefaultHost}^/WRS/stat%%01',
vector('path'),
null,
null,
2,
303
);
DB.DBA.URLREWRITE_CREATE_REGEX_RULE (
'wrs_rule5',
1,
'/WRS/objects/(.*)',
vector('path'),
1,
'/services/rdf/object.binary?path=%%2FWRS%%2Fobjects%%2F%U&accept=%U',
```

```
vector('path', '*accept*'),

null,

null,

2,

null

);

DB.DBA.URLREWRITE_CREATE_RULELIST ( 'wrs_rule_list1', 1, vector ( 'wrs_rule1', 'wrs_rule7', 'wrs_rule5', 'wrs_rule2',
'wrs_rule4', 'wrs_rule6'));

DB.DBA.VHOST_REMOVE (lpath=>'/WRS');

DB.DBA.VHOST_DEFINE (lpath=>'/WRS', ppath=>'/', vsp_user=>'dba', is_dav=>0,

is_brws=>0, opts=>vector ('url_rewrite', 'wrs_rule_list1')

);


-- Virtual directories for ontology

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (

'wrs_owl_rule2',

1,

'(/[^#]*)',

vector('path'),

1,

'/sparql?query=DESCRIBE+%%3Chttp%%3A//^{URIQADefaultHost}^%U%%3E+FROM+%%3Chttp%%3A//^{URIQADefaultHost}^/s
chemas/WRS%%23%%3E&format=%U',

vector('path', '*accept*'),

null,

'(text/rdf.n3)|(application/rdf.xml)|(text/n3)|(application/json)',

2,

null

);

DB.DBA.URLREWRITE_CREATE_REGEX_RULE (

'wrs_owl_rule1',
```

```
1,

'([^#]*)',

vector('path'),

1,

'/about/html/http://^{URIQADefaultHost}^%s',

vector('path'),

null,

null,

2,

303

);

DB.DBA.URLREWRITE_CREATE_RULELIST ( 'wrs_owl_rule_list1', 1, vector ( 'wrs_owl_rule1', 'wrs_owl_rule2'));

DB.DBA.VHOST_REMOVE (lpath=>'/schemas/WRS');

DB.DBA.VHOST_DEFINE (lpath=>'/schemas/WRS', ppath=>'/', vsp_user=>'dba', is_dav=>0,

is_brws=>0, opts=>vector ('url_rewrite', 'wrs_owl_rule_list1')

);
```