Personalizing search in digital libraries

Bart Cornelis

Master's thesis CS 03-01

Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science of Knowledge Engineering
in the Faculty of General Sciences
of the Universiteit Maastricht

Thesis committee:
Prof. dr. H.J. van den Herik
C.A. van Dorp, M.Sc.
Dr. F.J. Wiesman
P. van Rosmalen, M.Sc.

Universiteit Maastricht
Institute for Knowledge and Agent Technology (IKAT)
Department of Computer Science
Maastricht, The Netherlands
January 2003

Personalizing search in digital libraries

Master's thesis by

Bart Cornelis (cobaco@linux.be)

affiliation: Universiteit of Maastricht

Aurus KTS

Abstract

Executing some simple WWW searches readily demonstrates that what search engines return is rough guess, often including completely unexpected and unintended results. As the amount and diversity of the information on the WorldWideWeb grows this becomes an increasingly bigger problem, more often than not resulting in information overload. Thus, it is only natural that ample research is directed to mitigating this “needle in the haystack” problem. One stream of thought is devoted to personalizing the search, which means taking into account the specific traits and background of a user when answering that user's query.

The thesis investigates this stream of thought from the context of digital libraries. First, the problem of personalizing search is explored from a general perspective and broken down into three subproblems. Each of these subproblems is investigated in turn, surveying and discussing the existing techniques for dealing with it. Following this exploration from a general perspective is a discussion of what measure to use when comparing different personalized search tools. Finally, the problem of personalizing search is discussed from the context of Greenstone, a specific piece of digital library software.

Table of contents

1 Research description....5
1.1 The problem at hand....5
1.2 Context of the thesis....6
1.3 Problem statement....6
1.4 Thesis outline....6
2 Personalizing the search....7
2.1 User profiling....7
2.1.1 Contents of the user profile....8
2.1.2 Methods to obtain information....9
2.1.3 User profile formats....15
2.2 Documents characterization....17
2.2.1 Metadata....17
2.2.2 The Resource Description Framework (RDF)....17
2.2.3 The Dublin Core....19
2.2.4 Generating metadata....21
2.3 Matching user and document....21
2.3.1 Considerations....21
2.3.2 Collaborative filtering....23
2.3.3 Content-based filtering....26
2.3.4 Demographic filtering....26
2.3.5 Economic filtering....27
2.4 Combining subproblem solutions....27
2.5 Chapter conclusions....30
3 Evaluating personalized search....31
3.1 Traditional measures from IR....31
3.2 Rank-ordered-list measures....31
3.3 Mean average error....33
3.4 Chapter conclusions....33
4 Greenstone....35
4.1 Introduction to Greenstone....35
4.2 Obtaining Greenstone....35
4.3 Greenstone overview....36
4.3.1 Creating a collection....36
4.3.2 The run-time system....38
4.4 Relevant details....39
4.4.1 Searching a collection....39
4.4.2 How Greenstone obtains metadata....40
4.4.3 Authentication within Greenstone....40
4.5 Chapter summary and conclusions....40
5 Personalizing search in Greenstone....43
5.1 Matching user and document....43
5.1.1 Considerations....43
5.1.2 What kind of filtering to use....44
5.2 Document characterization....46
5.3 User profiling....46

5.4 Implementation design....47
5.4.1 Document characterization....47
5.4.2 User profiling....47
5.4.3 Matching user and document....48
5.4.4 Putting it all together....49
5.5 Chapter conclusions....49
6 Conclusions and further research....51

Table of figures....53
Table of boxes....54
References....55
Web References....57

Chapter 1

Research description

Phenomenons like the WorldWideWeb and Usenet have given us access to a gigantic and growing amount of information. The amount of information available to us has, already, grown too large to retain an oversight of it. This creates a problem: if we are not aware of what knowledge we have access to, then how do we make use of that knowledge, how do we find the proverbial needle in the haystack? A solution is to create tools that help us find the information we need, so that we do not need to search that entire haystack manually.

In section 1.1 the general problem of personalizing a search is introduced. Section 1.2 then presents the context in which we study the general problem introduced in section 1.1. Next, in section 1.3, the specific problem statement for the thesis is put forth, and section 1.4 concludes this chapter by stating the outline for the rest of the thesis.

1.1 The problem at hand

A tool to help us finding information is confronted with a twofold problem. The first part is whether we have access to any information about the requested subject at all and if so which information this is. The second part is how to take into account the context of the request. Let me clarify this with an example: Assume there is a user of the tool who specifies the subject as “Grand Canyon”. If that user is a geologist the information he[1] wants may be something along the lines of “How did the Grand Canyon come into existence”, and “What kind of rocks can be found in the Grand Canyon?” However if that user is a tourist planning his vacation the information he wants is more likely to be ” How do I get to the Grand Canyon?”, “Where can I get a guide?”, or “Where are the nicest spots?” Obviously, the tool will somehow have to distinguish between these users if it wants to return relevant information. Furthermore, if the tool really wants to be accurate it will also have to know for what purpose the request was made: if our geologist is preparing his vacation the information he will need would again be different from when he is preparing a lecture.

One can get a long way by just considering the first issue. However as the amount and diversity of the available information grows, one inevitably encounters a point where the amount of returned information not directly irrelevant to the user becomes too large. It still transcends the willingness and the ability of the user to cope with it. The user still feels drowned in information and the information overload remains. To avoid that, one will have to take into consideration the context of the user's request for information. Which, as explained above, means that the search response needs to be differentiated according to who the user is, and according to what the purpose of the request is. Which brings us to the focus of the thesis: *“How to differentiate the search response with respect to who and what the user is, i.e., how to personalize the search?”*

1 Both here and in the remainder of the thesis the pronoun *he* is used, this is not intended as a gender bias (i.e., feel free to substitute she, it, (s)he, Jane Doe, etc. as preferred).

1.2 Context of the thesis

We describe and investigate the stated problem of personalizing search in the context of digital libraries. For the purpose of the thesis a *digital library* is taken to be *a service that offers access to a large number of digital documents through a uniform interface*. To substantiate our research even more we selected the Greenstone digital library software, a tool for creating and maintaining a digital library, for implementation study.

The reason for choosing Greenstone is as follows. First, Greenstone is the only available Open source digital library software encountered. Open source is called for, since using open source enables us to build on existing software instead of having to start from scratch. Second, Greenstone is tried and tested (as witnessed by the fact that it is used, among others, by project Gutenberg[2] and by certain sections of the United Nations) and has proved to be a piece of very usable software.

1.3 Problem statement

As discussed above one problem that information-finding tools are confronted with, is how to take into consideration the specific traits of the user when responding to an information request. The thesis examines how to handle this problem in the context of digital libraries. Specifically we try to answer the following question: *How do we personalize search in the context of digital libraries in general, and Greenstone in particular?*

This overall question is broken down into three sub-questions:

1. Which techniques and ideas concerning personalizing search already exist, and what are the problems and challenges in using them?
2. What measures are available for comparing different personalized search tools?
3. How would these personalized search techniques fit into a digital library software package, like Greenstone?
4. How would we go about augmenting Greenstone to incorporate personalization?

1.4 Thesis outline

In chapter 2 the techniques and ideas related to personalizing search encountered in the relevant literature are surveyed. Also the problems and challenges associated with the presented techniques are discussed there. Next, in the chapter 3, several evaluation measures are presented and discussed. Following that, in chapter 4, there is a discussion of the Greenstone software package for digital libraries: what it does, and how it is currently structured. Then, in the chapter 5, adding personalized search to the Greenstone software is outlined: what kind of personalization would be meaningful and why, and what are the changes that need to be made to the Greenstone software in order to implement this personalization. Finally, in chapter 6, the conclusions of the thesis are presented. Also chapter 6 identifies the areas that are in need of further research.

2 Project Gutenberg provides public domain versions of books of which the copyright has expired [PG]

Chapter 2

Personalizing the search

The problem statement for this thesis - how to personalize search in the context of digital libraries - was broken down into four subquestions: “What methods and techniques currently exist?”, “What measures are available for comparing search tools using these methods”, “How do those methods translate to digital libraries, like Greenstone?”, and “How exactly could Greenstone be augmented with personalized search?” This chapter focuses on the first question. Therefore each method for personalizing search found in the explored literature is described along with the problems and requirements inherent to that method. Also all the encountered methods are ordered into a conceptual framework.

Considering personalization of search, one recognizes three sub-problems: (1) building the user profile, (2) characterizing the documents, and (3) matching documents with the user profile. To understand this breakdown consider the following.
If one is going to take into account the specific traits of a user when answering a search query, one will need to know what those traits are, which means there needs to be some description of those user-specific traits. In other words, a user profile needs to be constructed.
Next, consider that a personalized search will need to distinguish between documents relevant to the user and documents that are irrelevant to him. If a rational decision - regarding which documents are interesting and which are not – is to be made, the characteristics that identify a document as relevant will need to be determined.
Finally, given a user profile and a characterized document a way is needed to determine whether the document matches the user interest, thus we need a matching method.

The structure of this chapter follows the breakdown explained above: In section 2.1 we discuss the user profile. Next in section 2.2 we turn our attention to the document characterization. Following that, in section 2.3 we discuss the different matching methods found in the literature and how they tie in with the user profile and document characterization. Then in section 2.4 we discuss how the three subproblems mentioned above interact. Finally, in section 2.5 we summarize the findings of this chapter and present the conclusions drawn from those findings.

2.1 User profiling

The overall problem of personalizing search was divided into three sub-problems above. This section examines the first of those sub-problems: the problem of building the user profile.

In order to build a user profile several questions pop up waiting to be answered: The first of which is what to put into the user profile, what user traits are relevant for determining the user's interests? Once this question is answered, the next problem is readily apparent: “How will this information be obtained?”. The final question, when building the user profile, is how to represent and store the user profile data in our program.

Each of the questions identified above will be explored in its own subsection. Subsection 2.1.1 ponders the content of the user profile. Next, subsection 2.1.2 goes on to discuss how that information can be obtained. Finally, in subsection 2.1.3 the focus shifts to the format the user profile will implore.

2.1.1 Contents of the user profile

When deciding what the contents of the user profile should be, the intention for the user profile needs to be taken into account: “What is the user profile going to be used for? ”
In the context of personalizing search this is clear: the intention of the user profile is to capture the user-specific information needed to decide whether a given document is interesting to the user. In other words, the user profile for a personalized search contains information from which the user's interests can be inferred.

The user profiles found in the literature for personalizing search can be broken down into several categories (as shown in Figure 2.1). This breakdown starts by recognizing that there are two general classes of user profiles.
The user profiles in the first class *contain knowledge about the user's past interests*. Starting from such a user profile *induction* is used to classify a given document as interesting or not. This overall class of profiles has a further binary subdivision into memory-based and model-based profiles. A memory-based user profile simply consists of a dictionary of past user interests (see Collaborative filtering in subsection 2.3.2 for approaches using this kind of profile). In contrast model-based user profiles store knowledge about past user interests indirectly into a model of what constitutes an interesting document (see Content-based filtering in subsection 2.3.3 for approaches using this kind

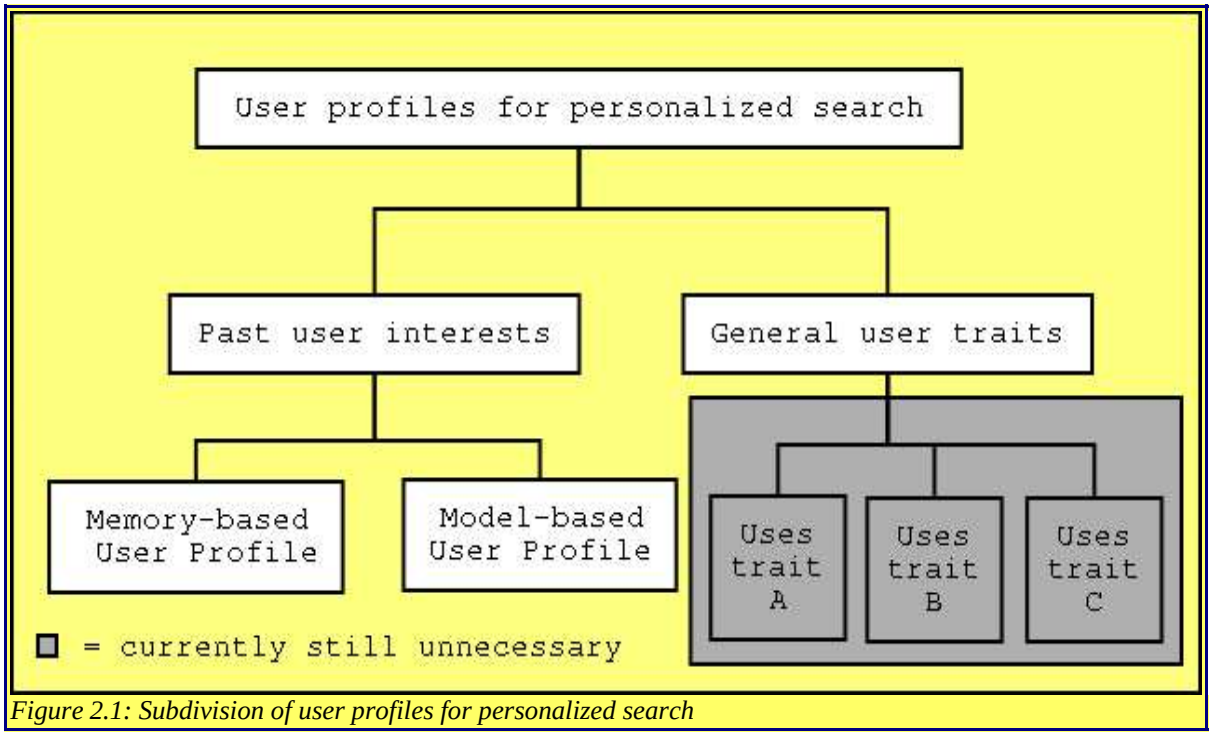


Figure 2.1: Subdivision of user profiles for personalized search

of profile).
The second overall class of user profiles does not collect knowledge about past user interests, but instead *contains knowledge about general user traits*. Starting from this general knowledge about the user *deduction* is used to classify documents as interesting or not. The overall class of user

profiles using this approach could be further divided into subclasses according to which kind of general user traits they contain. However since there seems to be only one kind of user profile using this kind of user profile (Demographic filtering described in subsection 2.3.4) such a subdivision is currently unnecessary.

2.1.2 Methods to obtain information

Subsection 2.1.1 discussed what kind of knowledge the user profile will contain. Having answered that question, another one pops up immediately: “How will this knowledge for the user profile be obtained?” In this subsection possible answers to this question are explored.

The structure of this subsection follows the first level of the subdivision introduced in subsection 2.1.1. In the first subsection, the discussion is about Gathering knowledge on user interests. Then, in the next subsection, the attention turns to Gathering data about general user traits. Finally this section is concluded by a subsection on Bootstrapping the user profile.

A. Gathering knowledge on user interests

The basic idea for gathering information about the user's past interests is very simple: record for each document the user encounters whether he was interested. While this information about the user's past interest is gathered, it needs to be translated into an actual user profile. For the first subclass of user profiles, those using the dictionary approach, this translation is trivial. The translation for the second subclass, those containing a model of past user interests, is more complex. There the gathered information is used as the input for a learning method, in other words it is used as the training set when creating the model.

The interesting problem in all this is how to determine whether the user is interested in a document he encounters. The explored literature offers many methods all of which fall into one of two categories: *Those that feature obtrusive interaction* with the user and those that merely observe the user as he acts, drawing conclusion from his actions, i.e., *those that feature unobtrusive observation*.

The rest of this subsection examines the different methods encountered in the explored literature. An exploration of methods featuring obtrusive interaction is followed subsequently by a discussion of those making use of unobtrusive observation.

a. obtrusive interaction

The methods introduced in this section all have one thing in common: they require the user to do something explicitly, thus indicating his interest or disinterest in the encountered document. First there is a discussion of four separate methods for obtrusive interaction. Then, to conclude the exploration of methods featuring obtrusive interaction there is a discussion of the general problem with the methods in this category, making apparent the reason for the decline of this approach.

Relevance feedback is the name of the first method we introduce. This method consists of having the user explicitly rate each encountered document on some scale. A binary scale was most often used (the user indicates his rating by clicking a thumbs up or thumbs down, see Figure 2.2 below for an example) . Examples of this approach can be found in, amongst others [Sheth, 1994], and [Jon, 1993]).

The second method is a generalization of relevance feedback. Relevance feedback asks the user one specific question: “How would you rate this document?” The generalization lays in also asking the user other questions. (e.g., “Are you interested in the topic characterized by the keywords X, Y,

Z?"). An example of this more general approach can be found in [Jon, 1993]. The system described there uses mail to ask questions to the user (thus allowing the user to respond when and if he chooses so).

A third method is called programming by demonstration . The idea here is to have the user present the system with a set of documents and document ratings. In other words the user comes in and says to the personalization tool: "I am interested in documents like this one", and "I am not interested in documents like this other one". This is very similar to relevance feedback, the difference being where the initiative lays. With relevance feedback the system selects the documents and presents them to the user for rating, whereas with programming by demonstration it is the user who selects the documents. An example using this approach can be found in [Sheth, 1994].

A fourth method is to provide the user with a way to change his profile by hand. This enables user control of the personalization component. Examples of this approach can be found in, amongst others [Sheth, 1994] (Figure 2.3 below gives a screenshot of that system in action), and [Jon, 1993].

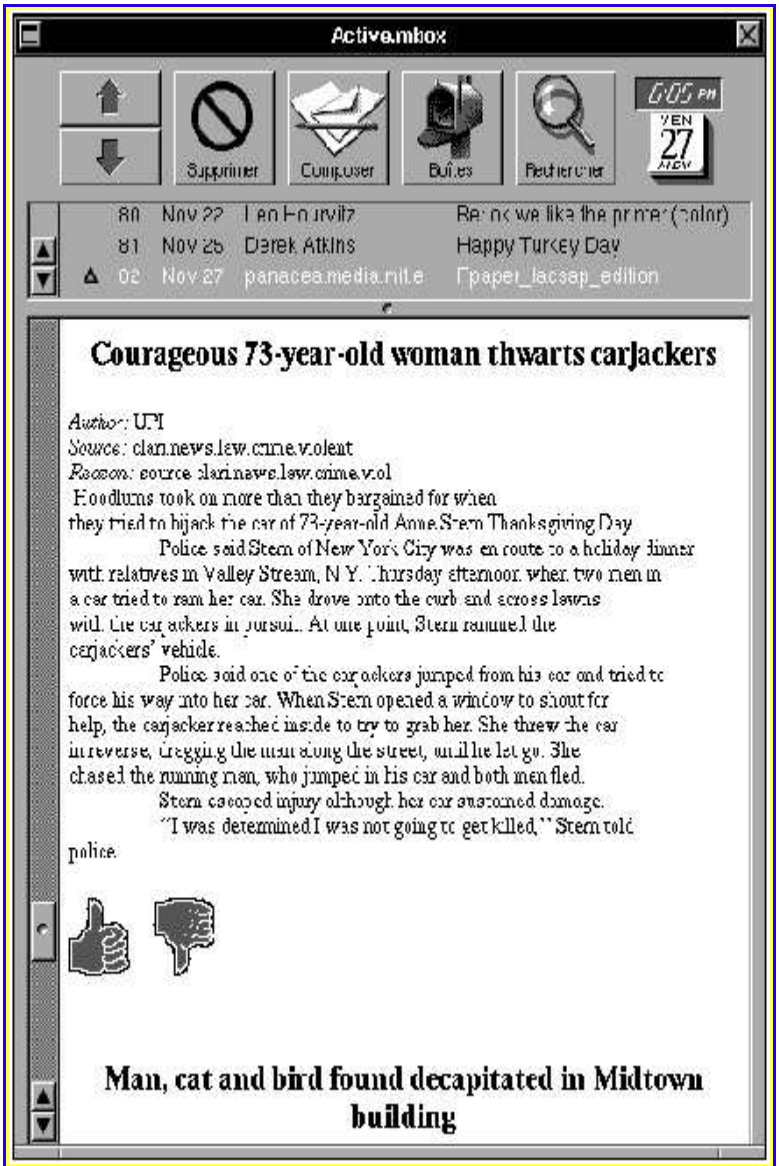


Figure 2.2: Example of relevance feedback found in [Jon, 1993]

The big problem with obtrusive methods is that they burden the user with extra actions, interrupting the normal work flow. Users are lazy, and resent this. Even more so, if the effect of those actions is not immediately apparent (as is the case with, for instance, relevance feedback) . The flip side is the unambiguity of these methods: there is no doubt whether the user was interested in a document. Experience has time and again shown that users do not want this, they tend either to skip the extra work or to move to alternatives systems if that is not possible. As a result obtrusive methods, most notably relevance feedback, have waned in popularity to the point where they are hardly used at all. Those systems that do engage in explicit interaction are well advised to make this interaction configurable, that way the user can turn it off when he is not up to it.

b. unobtrusive observation

We concluded the previous subsection with the observation that the extra work necessary for obtrusive methods is resented by users, leading these methods to all but disappear.
The alternative is to observe the user unobtrusively, drawing conclusions from the information inherent in the normal interaction of the user with the application.
Using this approach the design of the user interface is a major issue: a cleverly designed interface will yield more information without burdening the user (see for instance the slider interface described in [Balabanovic, 1998])

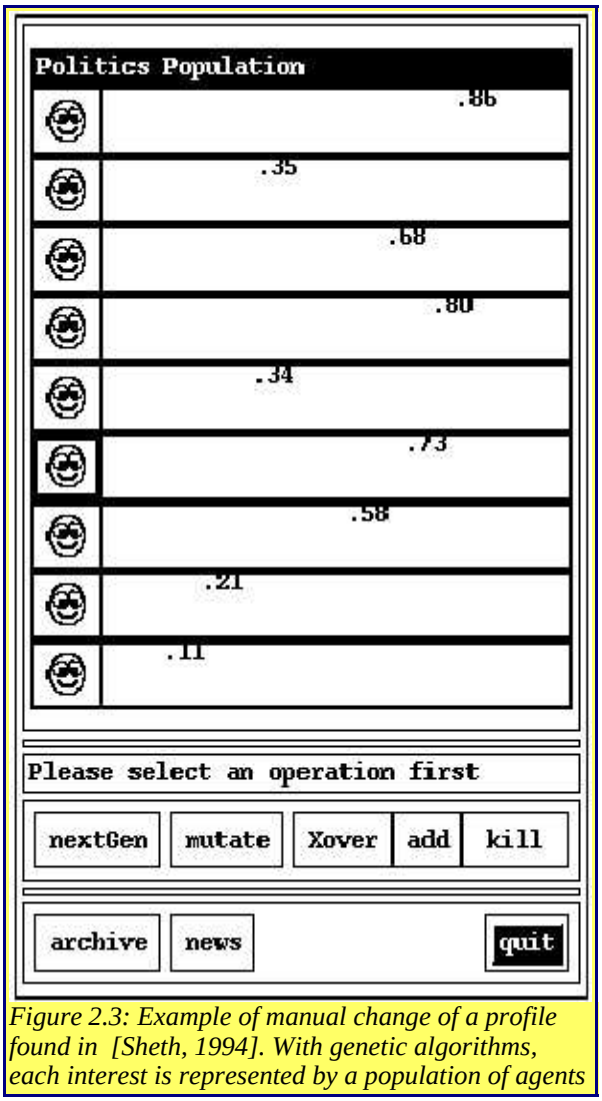


Figure 2.3: Example of manual change of a profile found in [Sheth, 1994]. With genetic algorithms, each interest is represented by a population of agents

That said, there are three issues that have to be taken into consideration when engaging in unobtrusive observation: what to observe, where to observe it, and how to interpret those observations. The rest of this section will discuss each of these three concerns consecutively.

The first issue is the question of what user behavior to observe. [Oard and Kim, 1998] studied this question and concluded that there are three types of user behavior potentially useful for deriving implicit feedback about the user's appreciation of a document: examination, retention, and reference. Below we take a closer look to each of these types individually. Box 2.1 summarizes this discussion.

Examination is the first type of user behavior useful for determining whether the user likes a certain information source. The user behaviors that fall into this category are those where the user examines the document in some way. Some example behaviors of this type are: Selecting a web page for viewing from the list returned by a search engine. Scrolling down a web page to read more. Expanding a thread in a newsgroup. Purchasing the right to access/get some document. Some points worth considering for this subtype are the following. First on this list is the duration, or how long the user examines the document. Repetition, or whether the user examines a document more than once is a second item worth taking into account. A third point that could be taken into account is edit wear, which generalizes repetition to take into account the use of a document by others. This measure simulates the uneven wear physical materials accumulate through usage.

The second type of user behavior useful for deriving the user's appreciation of a document is called *Retention*. This behavior type groups those behaviors that suggest whether the user has the intention to make future use of the object. Some example behaviors that belong to this type are: Bookmarking a web page, which generalizes to saving a reference to the document (another example of saving a reference is creating a symbolic link in a file system). Saving the document in some way is an obvious alternative to saving a reference to a document (e.g., making a local copy of a web page, printing out a document). Deleting a thread from a newsgroup is an example of a behavior indicating negative feedback. As noted in [Oard&Kim, 1998] there are a couple of points worth

further consideration when saving a reference to a document, or the actual document. The first of which is whether the reference is annotated or not, i.e., does the user associate some metadata to it. If so, this possibly indicates a stronger interest. To illustrate this point consider a user bookmarking a page to a user who not only bookmarks the page but who also renames the created bookmark to something descriptive. The second consideration when saving a reference to a document (or a copy of the actual document) is whether this is done in an organized fashion. Consider again a user who bookmarks web pages. Now consider two created bookmarks, one created in the default way, and one saved explicitly to a certain point in the user's bookmark hierarchy. The latter is probably of more interest, as indicated by the fact that the user went through the extra effort of placing the bookmark into his bookmark hierarchy.

Category	**Observable behavior**
Examination	Selection -> Duration -> Repetition -> Edit wear Expanding a thread Purchasing
Retention	Save a reference or save an object -> With or without annotation -> With or without organization Print Delete
Reference	Object->Object (forward, reply, post follow up) Portion->Object (hypertext link, citation) Object->Portion (cut & paste, quotation)

Box 2.1: Categories and their observable behaviors, see [Oard&Kim, 1998] for a detailed discussion

Reference is the name of the last behavior type useful for judging the user's like/dislike of a document. Behaviors of this type have the effect of creating some form of link between two objects. This type can be further subdivided into three subtypes. Some examples of the first subtype are forwarding an email, replying to an email, and posting a follow up to something on a web forum or a newsgroup. All behaviors of this subtype creates a link from the whole of a document to the whole of a second document. The behaviors in the second subtype create a link from a portion of a document to the whole of another document. Examples of this subtype include citing papers supporting some statement or including a hyperlink to some web page with further information. The third and last subtype consists of those behaviors creating a link between one document and a portion of some other document. Some example behaviors are cut-and-pasting some part of a document, and quoting some passage of another document.

The second issue concerning unobtrusive observation revolves is the question of where to observe the user: The simplest, and most common answer to this question is to observe the user when he is interacting with the application doing the personalization. [Mladenic, 1996] offers an example exercising this approach. Alternatively the scope of the observation can be broadened so as to observe the user in a larger part of his computing environment. In other words we also observe actions of the user in other applications he is using. An illustration of how this could be useful is represented in Box 2.2. An system exploiting this approach is discussed in [Budzik and Hammond, 2000]. The third and final answer to the question of where to observe the user broadens the scope of observation beyond the user's computing environment and into the user's real life environment. [Jon, 1993] describes a system that brings into play this approach. The answer to the question of

where to observe the user can be a deciding factor for the reliability of the observations. Consider the observation of examination duration: If a user has a document open for 10 minutes, does it then follow that he spent 10 minutes examining the document? Maybe the user got up to get a cup of coffee, or maybe he was interrupted by an IM message popping up, and spend time this time on the resulting conversation, and maybe he did spend 10 minutes examining the document.

The final issue with regard to unobtrusive observation is the question of how to interpret the observed behavior. This is where heuristics and the assumptions they are based on come into play. Below we look at heuristics for the three behavior types examination, retention, and reference defined above. First thing to note is that negative feedback is hard to obtain unambiguously: the only case when we can do this is when there is default retention (as is the case in, for instance, a newsreader which by default creates a reference to all messages), and this default retention is then killed off by the user (as happens when the user of a newsreader deletes a thread, or a message without reading it). Behavior like not selecting a returned search result for viewing do not unambiguously indicate negative feedback. Maybe the user already found what he was looking for and did not look further for more. Or maybe he tried a more promising looking result first and then forgot about the link he passed over. Or maybe he ran out of time. In contrast, positive feedback is easy, the basic heuristic is that engaging in a behavior falling into one of the types of examination, retention, or reference (other than deleting) indicates positive feedback for the relevant document. Further heuristics then refine this to indicate the strength of the positive feedback. Example heuristics of the latter kind include: “The longer you examine a particular document relative to the length of the document the more interesting it is”, “Organizing bookmarks into an explicit hierarchy indicates a stronger interest than letting it remain in the default spot of the bookmark hierarchy”.

A reporter working on an article interrupts his writing to conduct a web search. He does not click through to any of the returned documents, but he does add a hyperlink to one of the search results in his article. Obviously the hyperlinked document was of interest, however the search application would have no way of knowing this unless it was observing the reporter's behavior while writing

Box 2.2: An example illustrating the usefulness of observing the user in a larger part of his computing environment

B. Gathering data about general user traits

The previous subsection discussed gathering information about user interests. In this subsection the attention turns to gathering information on general user traits. More specifically, the focus is gathering information on demographic information about the user. Demographic information is the class of user traits used by the one method exploiting the general user-trait approach encountered in the explored literature (that would be Demographic filtering, see below).

There really is only one, very simple approach used: questionnaires. Each user is presented with a questionnaire the first time he starts the application. For an example see [Pazanni, 1999].

C. Bootstrapping the user profile

The discussion above focused on gathering information about user interests and general user traits to fill the user profile. Once built the user profile will be subsequently used to personalize the search. The big problem is that since the personalization system has no prior knowledge about the users, the initial user models tend to be inaccurate. This inaccuracy then trickles through to the

personalized search results. This, of course, is undesirable. Thus we need ways to deal with this initial inaccuracy, in order to limit its impact. The rest of this subsection discusses how to deal with this problem.

The first way to deal with this problem is to take the easy way out: simply state to the user that it will take the tool some time to get to know him, and consequently the initial results will be off. While this approach keeps the user from bailing out our tool immediately - on the account of the tool not working properly from the getgo - it does not deal with the initial inaccuracy. It is merely a band-aid, not a cure for the underlying problem.

To solve the problem of initial inaccuracy of the user profile we need a method that allows us to start not from an empty user profile but from a fairly accurate one. In other words, we need to bootstrap our user profile. The rest of this subsection introduces several methods for doing just this.

- The first method for bootstrapping the user profile is to start the user off with a stereotype, where a stereotype is a standard predefined profile. The problem that immediately pops up is how to assign a suitable stereotype to the user. One solution to this last problem is to let the user pick from the set of predefined profiles. Alternatively, one could provide a questionnaire to the user exploiting the answers to match the user with a predefined profile.
- A twist to the previous approach of using stereotypes is proposed in [Jon, 1993]. The basic idea is that instead of having the developer create a set of predefined stereotypes the system clusters the existing users of the system, and then uses an average member of each cluster as a stereotype. The readily apparent disadvantage of this approach is that it does not work until there is a sufficient number of users already in the system.
- Another variation on the stereotype approach is put forth in [Jon, 1993]: have each new user compare himself to existing users he knows (I am interested in the same things as user x). The new user profile is subsequently created as being equal to the profile of the alike user. There are two problems with this method. First, it only works if the new user can identify an existing user with similar interests. Second, this approach raises privacy concerns: using the user profile of another user as a starting point could reveal sensitive information about that existing user to the new user.
- Programming by demonstration, discussed above, is a fourth method for bootstrapping the user profile (see also [Sheth, 1994]).
- A fifth method for bootstrapping the user profile proposed in [Muller, 2000] is to use the user's bookmarks and homepage to generate the initial profile. (As noted in [Muller, 2000] adding a meta-search for documents referencing the user's homepage, might provide a more solid base for the generated user profile). One problem to look out for is the possibility of the user's homepage only reflecting part of the user's interests.
- This last method is an example of a class of methods that exploit information gathered by other applications, or a third party, about the new user to jumpstart his user profile. The idea is to collaborate with other applications. The big roadblock for this approach is the lack of standards for exchanging this kind of data. A proposed standard using P3P[3], XML + RDF[4], and XML-QL[5] is discussed in [Cingil et al., 2000a] and [Cingil et al., 2000b].

3 P3P, or platform for privacy preferences is a proposed standard from the W3C
4 Resource description framework, described later on in this document, is another proposed W3C standard
5 A query language for XML documents the W3C is working on

- Lastly, it is possible to combine two or more methods for bootstrapping the user profile. For example one could combine the stereotype approach with the approach using the user's bookmarks and homepage, eliminating the need for a questionnaire in order to assign a suitable user profile.

2.1.3 User profile formats

Subsection 2.1.1 focused on what to put into a user profile for personalizing search, and subsection 2.1.2 discussed ways of obtaining that information. This subsection investigates different formats for storing that information. The format used will be different for the different kinds of user profiles we introduced earlier (see Figure 2.1: Subdivision of user profiles for personalized search).

First, consider a *memory-based user profile*. These user profiles consist of a dictionary of past user interests. A dictionary entry for a particular document is simply a rating on some scale indicating the degree of interest the user had in that particular document. The rating is usually either binary (liked/disliked) or ordinal (a scale with five values seems common). This dictionary can be stored as as a simple hash table, a full blown database, ... Additionally, when using unobtrusive observation, the actual observations might be stored for each document also.

Now consider a *model-based user profile*. These user profiles contain a model of what constitutes an interesting document for the user. These models are learned either online (i.e., as new observations about documents the user liked/disliked are made, the model is adapted directly) or off-line (using a training set, consisting out of observations made so far). In the explored literature a number of different models were encountered. These will be discussed shortly, but first we turn our attention to several considerations that can be made in general for all model-based user profiles for personalized search.

- The first consideration is the fact that *user interests tend to change over time*, consequently the user profile will need the ability to adapt to the user's changing interests. In other words, some learning mechanism needs to be in place. Furthermore some aspects of the user's interests change rapidly while others change only over extended periods of time. Thus, when personalizing search we need to consider whether we want to model the short-term or the long-term interests of that user since each will require a different kind of learning mechanism.
- A second observation revolves around the fact that *the interests of the user will differ for different topics*. To reflect this, most user profiles used for personalized search actually consist out of a set of profiles, one for each topic the user is interested in. For examples, see for instance [Pazanni and Billsus, 1997], and [Balabanovic, 1998].
- A third observation is that *sometimes it is more practical to describe what is not interesting rather than what is*. Exploiting this fact is for instance [Widyantoro et al., 2001], there a three-pronged approach is used, each profile is made up out of three sub profiles: one modeling the short-term interest of the user for the topic, one modeling the short term disinterest of the user for the topic, and finally one modeling the longterm interest of the user for the topic.

Ideally, the model of user interest we use will take into account all three considerations. Keeping in mind these abstract considerations let us go over the actual model implementations found in the explored literature. The discussion below will briefly introduce model implementations using vector spaces, concept hierarchies, neural nets, and genetic algorithms respectively.

- The *vector space model* from information retrieval (IR): In the vector space approach each document is mapped to a vector $\{(t_1,w_1), (t_2,w_2), ..., (t_n,w_n)\}$ where each t_i is a term (keyword), and each w_i is a weight. The user profile thus consists of a vector that is somehow representative

of the interesting documents encountered so far (for an off-line learning scheme, this might simply be the average). The vector space representation of newly encountered documents is then compared to the profile vector using a similarity measure (e.g., the cosine similarity from IR), if the two are sufficiently similar the new document is considered to be of interest. For examples see [Widyantoro et al., 2001], and [Pazanni&Billsus, 1997].

- A second kind of model makes use of *concept hierarchies*: describing a user's interests as a set of nodes in the concept hierarchy. In the context of libraries DDC (Dewey Decimal Code[6]) and UDC (Universal Decimal Code[7]) are two aptly appropriate concept hierarchies. A newly encountered document is classified into the concept hierarchy and matched with the user profile (Note that the information implicit in the hierarchy structure will be used here: e.g., interest in a parent node likely implies interest in child concepts of that node.) [Muller, 2000]and [Pretschner, 1998] provide some examples of this approach.
- A third kind of model consists of a *neural or Bayesian network.* The network is trained with a set of the user's past liked/disliked documents. The document characterization of a newly encountered document is used as input, and the output is an estimate of the user interest in the newly encountered document. [Breese et al., 1998] offers an example of this approach.
- As noted above user interests change over time. To account for this [Sheth, 1994] used *genetic algorithms*. Doing this a user profile becomes in actuality a population of user profiles. To train the genetic algorithm a set of the user's past liked/disliked documents is again used. Assuming a vector space representation for a profile cross-over operator for the genetic algorithm is readily defined (e.g., the first10 terms from the first parent-vector and the rest from the second parent-vector), and a mutation operator can be defined just as readily (e.g., randomly replace a term in the parent-vector with some other term).

Above we discussed the user profile format for memory-based and model-based user profiles individually. Below we end this subsection with a discussion on three issues that should be considered regardless of the specific profile implementation used.

- The first general remark has to do with *control.* The purpose of a personalized search tool is to empower the user so that he can better deal with the information overflow he encounters. In order to achieve this, users need to feel in control (a tool that leaves the user feeling powerless is not doing a good job at empowering said user). For personalized applications, who return different results for different states of the user model, this means that our user has to be able to reason out way our application does something. There are two basic methods to achieve this. The first is to build in an explanation component, which, upon request, explains why something happened. The second is to make the user profile comprehensible and accessible to the user. This also provides another gain for power users, for they can now help the system by adapting the user profile manually to better fit his needs. The user could, for instance, delete a topic profile of a topic he is no longer interested in, or bootstrap the profile of a topic he just became interested in, etc.
- The second issue, related to the first is that of *trust.* Users need to trust our personalized search tool to give them good results, if they do not then they will refrain from using it. Again it is a big help if the user model is comprehensible, a black box performing magic does not inspire trust (especially when the results it yields can be substantially off).

6 See also [Scorpion], a project attempting to classify documents using DDC automatically.
7 See also [Gerhard], a project attempting to classify documents using UDC. automatically

- The last issue is that of *privacy*. Apart from the legal issues involved, there is the simple fact that people tend to be uncomfortable with having big brother watching their every move. Thus the place and format of the user profile needs to be considered carefully. If we make our user profiles anonymous the big brother issue largely disappears. If we store the profile server side instead of on the clients on machine we will need to define and enforce a privacy policy, with all the procedural machinery that entails. Mishandling privacy will yield big trust problems, preventing or at least limiting adoption of the personalized search tool in question.

2.2 Documents characterization

In order to personalize a search, documents of interest to the user need to be distinguishable of non-interesting documents. Thus we need a method to characterize the objects we are examining.
One obvious fact is that computers are severely lacking in their ability to determine what a particular source of information is about. For text documents a computer's ability in that regard is not totally inexistent, the field of information retrieval has introduced several ways for computers to differentiate between documents of different topics. But when information is contained in images or sound streams, computers are, currently, totally lost.
Since computers are so limited in their ability to grasp what a certain piece of information is about, we need to give them some pointers. The solution proposed is to use metadata describing the objects to be examined, thus giving the computer a characterization to work with.

In this section the discussion will be about metadata: First in subsection 2.2.1, metadata in general is discussed briefly. Next, in subsection 2.2.2, the discussion turns to The Resource Description Framework (RDF), a particular metadata standard. Then, in subsection 2.2.3, another, complementary metadata standard, named The Dublin Core is described. Finally, subsection 2.2.4 explores how to generate metadata.

2.2.1 Metadata

Metadata can be thought of as data about other data. The distinction between data and metadata is not an absolute one: what is metadata for one purpose might be ordinary data for another purpose. Metadata is the kind of data that librarians put in catalogs, it is a set of elements describing some resource. As shown by the library catalog we mentioned earlier the concept of metadata predates the Web and even the Internet. However it is the information overload created by these new media that has sparked new, intensified, and world-wide interest in metadata and metadata standards.
The use of wide-spread metadata standards, it is thought, will dramatically improve the ability for automatic retrieval of information. Several standards are currently being created, in the following subsections we will discuss two fairly well established and promising ones.

2.2.2 The Resource Description Framework (RDF)

The resource description framework, or RDF for short, is currently a [W3C] recommendation. It sprung fort from the W3C Semantic Web effort. The full text of this recommendation can be found in [RDF-M&S, 1999]. As the name indicates RDF is a framework for describing resources, in other words it provides a mechanism for creating metadata about resources in a standard way. The resources can be almost anything. The only thing RDF requires is that they can be represented by a URI (Uniform Resource Identifier, see [URI] for a description).

The overall goal of RDF is to provide a mechanism for describing resources in a way that makes no assumptions about a particular application domain, nor defines the semantics of any application domain. The definition of this mechanism is domain-neutral, yet suitable for any domain.

In this subsection we will examine the different aspects of RDF: first is a discussion of The RDF data model, next is an overview of RDF syntax, and concluding this subsection is an introduction to RDF schemas.

A. The RDF data model

The foundation of RDF is the data model it employs for representing named properties and property values. This data-model is a syntax-neutral way of representing RDF expressions. This basic data model consists of three object types: resources, properties, and statements.

A *resource* is basically anything that is being described by an RDF expression. Resources are always named by URIs (plus an optional anchor id). Since just about anything can have a URI, this means we can describe almost everything with RDF.

The second object type in the data model is called *property*. A property is a specific aspect, characteristic, attribute, or relation used to describe a resource. Each property defines its possible values, the types of resources it can describe, the specific meaning it conveys, and the relation it has with other properties. How the characteristics of properties are expressed is explained in [RDF-schema, 2000], and we will discuss this later on.

As the final object type we have *statements*. A statement has tree parts. The first part is the subject, which is just the resource concerning which the statement is made. The second part is the predicate, this is the named property from the statement. The last part is the object, which is the value of the predicate for the subject, it can be another resource or a literal (in RDF context, a literal may have content that is XML markup but is not further evaluated by the RDF processor).

B. RDF syntax

The W3C RDF work group has defined two syntactically different ways to represent RDF expressions (an example of each can be found in Figure 2.4):

1. The use of XML (eXtensible Markup Language) results in the first proposed syntax. RDF is complementary to XML in that RDF is a model of metadata and XML addresses the encoding issues brought fort by transportation and file storage (such as internationalization, character sets, etc.) for that metadata.
2. The second proposed syntax is called “Nodes and arcs diagrams”. These are simply a labeled, directed graph representation of RDF expressions. Resources are

Consider the phrase: “Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila”.

Translated to RDF/XML this becomes:

```
<rdf:RDF>
  <rdf:Descriptionabout="http://www.w3.org/Home/Lassila"
>
    <s:Creator>Ora Lassila</s:Creator>
  </rdf:Description>
</rdf:RDF>
```

and the corresponding nodes and arcs diagram is:

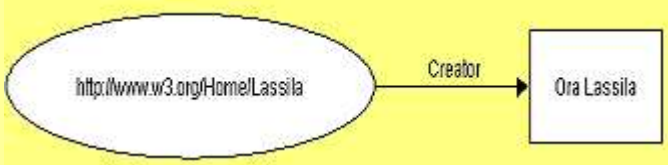


Figure 2.4: Example of RDF syntax: RDF/XML and Nodes & Arcs

represented by oval nodes, literals are represented by rectangular nodes. An arc represents a named property. Arches start at the subject of a statement, point to the object of the statement, and are labeled with the predicate.

C. RDF schemas

The RDF data model presented above defines a simple model for describing interrelationships among resources in terms of named properties and values. It does not, however, provide a mechanism for defining the relationships between these properties and other resources.

Different communities use different kinds of resources or use the same kind of resource in different ways. These different groups each have a need for a vocabulary of descriptive elements to describe the resources they use. The declaration of such a vocabulary of descriptive elements is known in the RDF world as an RDF schema.

In [RDF-schema, 2000] the W3C RDF working group defines a machine-understandable system for defining schemas for descriptive vocabularies. This system allows designers to specify classes of resource types and properties to convey descriptions of those classes, relationships between those properties and classes, and constraints on the allowed combinations of classes, properties, and values. The core schema vocabulary is presented by the W3C as an XML namespace, and can be found at [RDF-SCHEMA-NS]. The elements presented in this namespace, together with those presented in the core RDF namespace (which can be found at [RDF-NS]) make up the foundation on which application specific schemas are built.

By allowing the creation of statements which draw upon specialized schemas from various domains (through the use of XML namespaces), RDF makes it possible for diverse communities of expertise to contribute to a decentralized web of machine-readable vocabularies.

RDF schemas may be contrasted with XML document type Definitions (DTDs), and XML schemas. XML DTDs and XML schemas define specific constraints on the structure of an XML document. An RDF schema provides information about the interpretation of the statements given in an RDF data model. In other words, XML schemas provide syntactic rules, whereas RDF schemas provide semantic rules. Work on combining both in a simple way is on the way, see for instance [SCHEMA-ARCH].

The RDF Schema specification intentionally left unspecified a set of primitive datatypes. The idea is that since RDF uses XML for its interchange encoding, the work on data typing in XML itself [XML-DATATYPES] should provide these.

2.2.3 The Dublin Core

In this subsection we examine the Dublin Core which is a set of expandable metadata standards developed by the Dublin Core Metadata Initiative [DCMI] (The DCMI is an open organization dedicated to promoting the widespread adoption of inter-operable metadata standards and developing specialized metadata vocabularies for describing resources that enable more intelligent information discovery systems). First, the The Dublin Core Metadata Element Set (DCMES) is introduced. Next, the discussion turns to the The Dublin Core Qualifiers which offer a way to specify the right interpretation of DCMES elements.

A. The Dublin Core Metadata Element Set (DCMES)

The DCMES is a simple yet effective element set for describing a wide range of networked resources. The DCMES comprises fifteen elements, (e.g., title, creator, subject[8]), the semantics of which have been established through consensus by an international, cross-disciplinary group of professionals from librarianship, computer science, text encoding, the museum community, and other related fields of scholarship.

The DCMES has as it goals:

- Simplicity of creation and maintenance; non-specialists should be able to use the DCMES to create simple descriptive records for information resources easily and inexpensively.
- Commonly understood semantics; the common set of elements set fourth by the DCMES should be readily understandable for non-specialists.
- International scope; versions of the DCMES are being created in many languages yet all these versions are linked together, and convey the same meaning.
- Extensibility; this is done by ensuring that the DCMES can be used together with other metadata sets, the idea is that the DCMES provides the core descriptive information that is globally usable, while additional metadata sets make sense within a more limited arena.

The DCMES only provides the metadata element set, it does not provide a syntax for these elements. A syntax recommended by the DCMI is the RDF/XML introduced previously in subsection 2.2.2.

B. The Dublin Core Qualifiers

Qualifiers are mechanisms that define in more detail the interpretation of an element from the DCMES. There are two kinds of qualifiers.

The first kind of qualifier, called an element refinement, narrows down the meaning of an element. Since an element qualified in this way shares the meaning of the unqualified element, with a more restricted scope (e.g., *abstract* is an element refinement for the description element), any client that does not understand a specific element refinement term should be able to ignore the qualifier and treat the metadata as if it were unqualified.

The second kind of qualifier, called an encoding scheme, aids in the interpretation of an element value, by defining a controlled vocabulary, formal notation, or parsing rule for the element values (e.g., DDC and UDC are encoding schemes for the subject element). The advantage of a controlled vocabulary is twofold. On the one hand it can dramatically improve search results (because computers are good at matching words character by character, but weak at understanding synonyms). On the other hand it may reduce the likelihood of spelling errors (because the input could, for example, be given by choosing an entry from a selection list). Disadvantages of controlled vocabularies are that an administrative body is needed to define and maintain the vocabulary, and that people might need to be trained to know which entry to use or look for.

All qualifiers used should be publicly available and obey the dumb-down principle. Obeying the dumb-down principle means that an application that does not recognize the qualifier in question, should be able to ignore the qualifier and fall back on the broader meaning of the unqualified element.

8 See [DCMES, 1999] for a listing and detailed description of these elements

The Dublin Core Qualifiers are a set of qualifiers approved and recommended by the DCMI. Their description can be found in [DCQ, 2000].

2.2.4 Generating metadata

In the previous three subsections the discussion was about metadata in general and two particular and complementary metadata schemes, the Resource Description Framework and the Dublin Core. In this subsection the attention turns to the different ways of obtaining metadata. There are two approaches to obtaining metadata. The first one is to get a human to assign the metadata. The second one is to have it generated automatically by a computer. The rest of this subsection discusses each approach in turn starting with having computers generate metadata.

The ability of computers to generate metadata is currently very limited, though not unexisting. One way of doing this is to have computer tools insert easily inferred metadata (such as the creator, creation date, source URI, document language) as documents are edited or created. Another item of interest here is the emerging field of text classification. Text classification attempts to identify the subject of a text document in relation to some ontology. Two ongoing projects in this area are: [Scorpion] , which attempts to classify documents into the DDC ontology; and [Gerhard], which uses UDC as its target ontology.

Given the current limited ability of computers to generate metadata, human-added metadata is currently a necessity. Again, most users do not like to be bothered any more than absolutely necessary. Thus in open environments, like the Internet, this approach is not very practical. However in more controlled environments the situation is different (e.g., in a digital library the librarian could add the metadata, much like analog libraries where librarians construct various indexes. Within a corporation a policy could be enforced that all documents must be created with metadata attached). A collaborative, community-based approach, comparable to the way the CDDB[9] was built, would be another way to facilitate human-generated metadata. One possible problem with cooperatively developed metadata is the quality of contributions. While this is something to take into consideration it is not - as demonstrated by, for example, NASA's clickworker project - an unsurmountable problem.

2.3 Matching user and document

In the beginning of this chapter the overall problem of personalizing search was divided into three subproblems: building the user profile, characterizing the documents, and the matching of documents with the user profile. This section focuses on the last subproblem. In subsection 2.3.2 to 2.3.5 the four different matching methods found in the explored literature are introduced, these are respectively: Collaborative filtering, Content-based filtering, Demographic filtering, and Economic filtering. But first subsection 2.3.1 discusses several considerations regarding the matching process in general.

2.3.1 Considerations

This subsection discusses several considerations regarding the matching process that are independent of the specific matching methods used.

9 CDDB or CD-database is the name of a database with song titles and track information of CD's, this database was filled by contributions of users on the Internet. The original CDDB has since been made proprietary by the company that initiated it, what resulted in the creation of the [FREEDB] a community-maintained database using the CDDB protocol.

The first consideration to be made is where to do the matching: client side or server side (see for instance [Lawrence, 2000], [Bradley et al. ,2000]). As noted in subsection 2.1.3 the decision made in this regard has consequences for the whole privacy issue: server side matching means the user profiles have to be stored on or transmitted to the server, yielding the (legal) need for a well-defined privacy policy and the security measures necessary to enforce it. Also a choice for server side matching has consequences for the amount of storage capacity and processing power needed. If the matching is done server side the server will have to do all the work. If the profiles are also stored on the server side we will need to make sure sufficient storage capacity is present. server side matching does have the advantage of needing less bandwidth (since only the information matching the user profile needs to be transmitted to the client. A second advantage of server side matching is that collaborative approaches (see subsection 2.3.2) are easier to implement since all user profiles are stored at the server (a collaborative approach with client side matching is possible but would require peer to peer (P2P) system).

The second consideration is whether to use filtering in, or filtering out matching process. This consideration deals with the question of when to personalize the search process (see Figure 2.5). *Filtering in* means we first match all documents with the user profile, and then do the regular search on those that remain. *Filtering out* is when we first execute a regular search and then match the returned documents with the user profile. The main issue to keep in mind when choosing one over the other is the relative speed of the regular search and the matching process. If the regular search is faster than the matching, filtering out will be the most efficient approach. Otherwise filtering in will be more efficient.

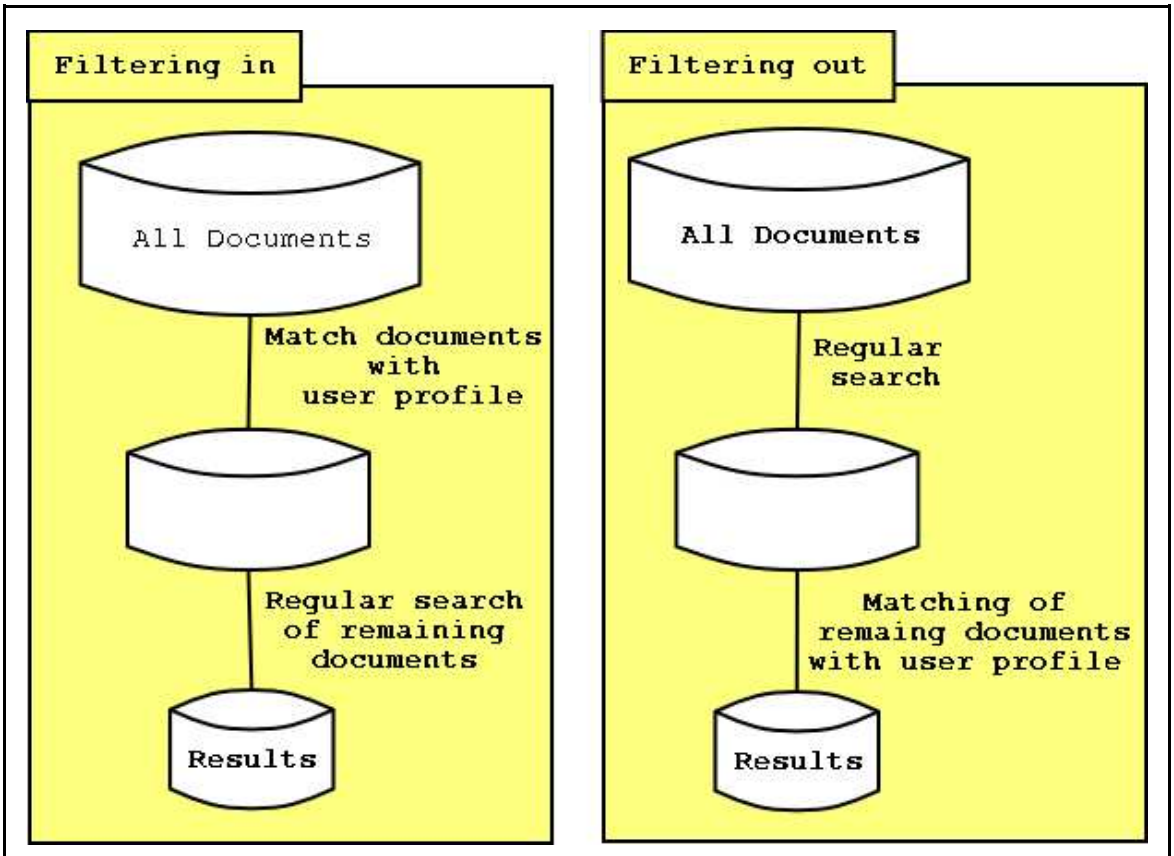


Figure 2.5: Visualization filtering in and filtering out

The third consideration is where to lay the balance between exploitation and exploration. *Exploitation* is taking advantage of what we already know about the user: if we know the user likes a certain document we give him many documents like it. *Exploration* is taking a chance and taking a new direction: it is presenting the user with something unlike previously encountered documents. If we put too much emphasis on exploitation the user gets stuck in a limited world, always encountering the same kind of information, never encountering something completely novel. If exploration is emphasized to strongly then our knowledge about the user remains unused.

The fourth and last consideration concerns what to do with the documents not matching the user profile. Do we *hide* them from the user *or* do we *re-rank* the results so that the least matching documents are presented last. Re-ranking gives the user the possibility to spot documents that were wrongly classified as not matching. Hiding non-matching documents to the user becomes acceptable as the number of matching documents grows: when the user is already presented with a high number of relevant documents the wrongly classified documents only contribute to information overload, the user would not look at the low ranked documents anyway. It is trivial to transform either a hide approach into a re-rank approach or a re-rank approach into a hide approach (see Box 2.3), Thus the solution seems to be to make this configurable, so that the user can choose which approach he likes best.

We can easily transform a personalization approach that filters to one that re-ranks, and the other way around:

hide->re-rank: just add the filtered out objects at the bottom of the list

re-rank->hide: leave out all objects below a suitable threshold

So whether we use one or the other is purely a practical concern

Box 2.3: Transforming personalization approaches

2.3.2 Collaborative filtering

This subsection discusses collaborative filtering, a matching technique that automatizes word of mouth recommendation. Collaborative filtering in general works as follows. Take a set of users and a set of documents. As users interact with the system they rate items (obtrusively or unobtrusively). These ratings are collected in a two-dimensional roster (the first dimension being the set of users and the second dimension being the set of documents). Not yet encountered documents are rated by computing a neighborhood of the point for the rating being predicted and combining the ratings in that neighborhood somehow.

This subsection is structured as follows. First, there is a discussion on *neighborhood dimensions* and how they give rise to two different ways of doing collaborative filtering. Then a decision should be reached regarding along which dimension to compute the neighborhood. The next question is which of the *similarity measures* will be used to compute this neighborhood. The next part of the discussion in this subsection surveys the available measures. Next, there is an exploration of the different methods of *combining the neighborhood ratings* into a predicted rating. Finally, this subsection is concluded with some thoughts on how to use collaborative filtering with model-based user profiles instead of memory-based ones.

A. neighborhood dimensions

There are two different approaches within collaborative filtering as to how to combine the ratings of the different users into predicted ratings for unseen items. These two approaches are called user-based collaborative filtering and item-based collaborative filtering, respectively.

User-based collaborative filtering first computes a neighborhood for the active user (i.e., the user for whom the prediction is made). This neighborhood consists of users which are similar to the active user and have rated the active item. The ratings of the users in this neighborhood for the active item are then combined into a predicted rating.

Item-based collaborative filtering ([Sarwar et al., 2001], [Shardanand&Maes, 1995]) is conceptually very similar: we just switch the dimensions. So instead of computing a neighborhood of users, we compute a neighborhood of items similar to the active item (i.e., the item about which the prediction is made). Then we combine the ratings of the active user for the items in this neighborhood into a predicted rating. NOTE: the collaboration, that is the use of the other user's profiles, is located in the similarity computation

Looking at the explanations of the two approaches above, we observe that both utilize the same basic idea but with switched dimensions (the dimensions being users and items). Thus the issues needing consideration for both approaches are the same. There are two issues to look at: The first is how to decide which users or items are similar. The second issue is how to combine the ratings in the neighborhood into a prediction. The next two subsections deals with each of these issues respectively.

Both approaches to collaborative filtering are normally used with memory-based user profiles. The individual user profiles are combined into the so-called *ratings matrix*, which is an *mxn* matrix *M* where m is the number of users, n is the number of items, and M_{ij} is the rating of user *i* for item *j*.

B. similarity measures

As explained above the question of how to decide which users or items are similar is central to collaborative filtering. In practice this is done by using a similarity measure on the user/item vectors (a user vector is made up of the row for that user in the ratings matrix, an item vector is made up of the column for that item in the ratings matrix). This subsection introduces several similarity measures that are in common use.

- Cosine similarity ([Sarwar et al., 2001], [Breese et al., 1998]):
  Here the similarity is measured as the angle between the two vectors. Formally, in the m x n ratings matrix, similarity between user/items *i* and *j* is given by:

  $$\text{sim(i,j)} = \cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{\left\|\vec{i}\right\| * \left\|\vec{j}\right\|}$$

- Correlation similarity ([Sarwar et al., 2001], [Shardanand&Maes, 1995], [Breese et al., 1998]):
  Here the similarity is obtained by computing the Pearson-r correlation. In order to make the correlation computation accurate we must first isolate the co-rated cases (i.e., those cases where both users rated item *i* and *j* by user-based collaborative filtering, and both items *i* and *j* were rated by item-based collaborative filtering). Let us denote the set of co-rated cases by C. Then the correlation similarity between *i* and *j* is given by:

$$\text{sim}(i,j) = \text{cor}_{i,j} = \sum_{c \in C} \frac{(\overline{R_{c,i}} - \overline{R_i}) \cdot (\overline{R_{u,j}} - \overline{R_j})}{\sqrt{\sum_{c \in C} (\overline{R_{c,i}} - \overline{R_i})^2} \cdot \sqrt{\sum_{c \in C} (\overline{R_{c,j}} - \overline{R_j})^2}}$$

where R is the ratings matrix and $\overline{R_i}$ is the average rating of vector i.

- Adjusted cosine similarity ([Sarwar et al., 2001]):
  Using cosine similarity for item-based collaborative filtering has one drawback: the difference in rating scale between the different users are not taken into account. Adjusted cosine similarity sets aside this drawback by subtracting the corresponding user average from each co-rated pair. Formally, the adjusted cosine similarity between item i and j is given by:

  $$\text{sim}(i,j) = \sum_{c \in C} \frac{(\overline{R_{c,i}} - \overline{R_c}) \cdot (\overline{R_{c,j}} - \overline{R_c})}{\sqrt{\sum_{c \in C} (\overline{R_{c,i}} - \overline{R_c})^2} \cdot \sqrt{\sum_{c \in C} (\overline{R_{c,j}} - \overline{R_c})^2}}$$

  where $\overline{R_c}$ is the average of the c-th user's rating.

C. combining the neighborhood ratings

As explained above the second central issue regarding collaborative filtering is how to combine the the ratings in a neighborhood of similar users/items into a predicted rating. This subsection introduces several methods for creating this prediction:

- K-nearest neighbor: With this technique we take a neighborhood with k members. We then take the rating that is most common amongst the members of that neighborhood.
- Weighted sum ([Sarwar et al., 2001], [Breese et al., 1998]): As the name says this is simply a weighted sum of the ratings in the neighborhood. We use the similarity between the two users/items as the weight for each rating. Formally, if N is the neighborhood then

  $$Pred(Ia) = \sum_{n \in N} \frac{R_{n,i} * sim(i,n)}{\sum n \in N} |sim(i,n)|$$ where Ia is the active item and R is the rating matrix and sim(i,j) is the similarity between i and j.
- Regression ([Sarwar et al., 2001]): This approach is similar to the weighted sum method except that instead of using the ratings directly we use an approximation of the ratings based on the linear regression model. The reason for not using the raw ratings is that the distance measure may be misleading in that two vectors that are similar may be distant. In which case using that rating results in very poor prediction.

D. model-based profiles

Collaborative filtering as described thus far has a memory-based user model. We simply store all observed user ratings into a matrix and generate predictions from that matrix. In [Breese et al., 1998] a class of collaborative filtering algorithms is introduced that does not uses a model-based user profile. These model-based methods use machine-learning methods to learn a model of user ratings. The observed ratings are used as training examples.

Model-based collaborative filtering takes a probabilistic view on collaborative filtering: the process is envisioned as computing the expected value of a user prediction given his or her ratings on other items.

Several model-based collaborative filtering algorithms have been proposed.

- Clustering treats collaborative filtering as a classification problem and works by clustering similar users in the same class and estimating the probability that a particular user is in a particular class, and from there computes the conditional probability of ratings ([Breese et al., 1998], [Basu et al.,1998], [Ungar&Foster, 1998]).
- The Bayesian network formulates a probabilistic model for collaborative filtering ([Breese et al., 1998]).
- The rule-based approach applies association-rule discovery algorithms to find associations between co-rated items and then generates item recommendations based on the strength of the association between items ([Sarwar et al., 2000]).

2.3.3 Content-based filtering

This subsection discusses *Content-based filtering* a second method for personalizing search. The basic idea of content-based filtering is to view the personalization as a classification task, where objects need to be classified into several classes (e.g., interesting and uninteresting) based upon their characterization (i.e., based upon the metadata and/or some IR-representation of the object).

The distinctive element of a content-based filtering method is the object characterization. The remainder of this subsection briefly introduces two content-based methods.

- A first content-based approach ([Mostafa et al., 1997]) uses a vector space representation in the topic profiles. Recommendations are then made by first calculating the distance between the topic profile and the object. Then the calculated distance is used to classify the object (e.g., if the distance < 0.2 then the object is interesting).
- A second approach uses statements about metadata in the topic profiles([Kurki et al., 1999]). (for instance, if Dublin core element author equals ”RMS” then the we can say with 82 percent certainty that the item is interesting, or if the Dublin core element language equals ”jp” then we can say with 100 percent certainty that the document is not interesting.)

2.3.4 Demographic filtering

This subsection discusses the oldest personalization approach: *Demographic filtering*, also called *rule-based filtering* (see for instance [Pazanni, 1999]). The basic idea of which is to use demographic data about the user in a simple rule-based approach. Let us clarify this with an example. Assume we have a user who is looking for information on football. If this user is European he most likely wants information about soccer. However if the user is American he would be looking for information about American football. This amounts to the following rules: “if the user is European then football means soccer” and “if the user is American then football means American football”.

The underlying assumption of this approach is that people with the same demographic background will also share similar viewpoints, and a similar cultural bias. And it is the failing of this assumption that explains the fact that this technique is deemed too limited and therefore is hardly ever used anymore. To understand way this assumption is failing consider phenomenon like the worldwide web and Usenet. In today's society the online world is increasingly important in determining the viewpoint and cultural bias of people. Thus the assumption that people with the same demographic background have the same cultural background, and thus similar views on what information is valuable, is increasingly untrue.

The basic idea of demographic filtering, namely that a person's informational needs are influenced by the viewpoints and cultural bias of that person, is of course correct. However, we need something that is a better indicator of this than a person's demographic background[10].

2.3.5 Economic filtering

This subsection introduces the last of the personalization methods named *economic filtering*. Here documents are classified based upon cost factors. These cost factors can be related to either creating, providing or using the information. Economic filtering tries to balance these cost factors with the expected worth of the obtained information. The big problem here is how to estimate the expected value of the information.
One example of this approach found in [Gokhale, 1999] is the following. When reading news we give a lower priority to messages that are cross-posted to different newsgroups. The rationale behind this is that they have a lower production cost per addressee and are thus less worthwhile.

2.4 Combining subproblem solutions

In the beginning of this chapter the overall problem of personalizing search was divided into three subproblems. Above in sections 2.1 to 2.3 we discussed each of those subproblems individually.

However, when thinking about how to combine the different approaches to each of the subproblems identified above it soon becomes apparent that the three sub-problems are intertwined and that a solution to one problem will go hand in hand with matching solutions to the other two sub-problems. If a particular kind of user profile and document characterization are chosen only a complementary matching method is applicable. Alternatively, each specific matching method will influence the features of the user profile and the document characterization that can be used.

In subsections 2.3.2 to 2.3.5 we introduced four distinct filtering techniques. Having several techniques to choose from raises the obvious question of when to use which techniques. In this section we will give an analysis of the requirements and problems of the different techniques, and personalized search in general. For the most part this section explicitly states assumptions made implicitly in the discussion above and combines them with already stated items providing a general overview of assumptions, problems and requirements as summarized in Box 2.4.

First for ***personalized search in general*** there are two issues:

- Obviously we need *the ability to recognize which user is interacting with our system*, otherwise we will be unable to update and use the right user profile. For client side filtering this is mostly a non-issue: we can essentially outsource the authentication problem to the OS (all modern OSs are multi-user, all we need to do is store the profile with the rest of the user's settings). For server side personalization an authentication component is needed.
- *Privacy* is also an issue. We need to ensure the user's information is kept save. Again for client side filtering, where the user profile never leaves the user's control this is an non-issue because we can outsource this responsibility to the user. Still, it is prudent to help the user along by setting the default file permissions on the file containing the profile settings so that only the user can read it. Also a hook could be provided for encryption software like GPG[11] allowing the stored profile to be encrypted. For server side filtering we need to ensure that: The information on the server is anonymous, i.e., if the server is compromised, the information found can not be

10 Obviously the name demographic filtering is then no longer appropriate and a new term is called for, maybe something like world-view-based or culture-based filtering.

11 Gnu Privacy Guard, a set of tools for encryption of files, and securing of communications developed by the Free Software Foundation.

linked back to the user. Or we need to encrypt the information on the server. Of course the combination of those is also possible. Apart, from ensuring privacy is guaranteed, we also need to have a clear privacy policy. Clients can then be sure that the information collected will not be sold, or used inappropriately (from the standpoint of the user). This is a very important issue because people like their privacy, when it is not respected they will back off.

In the remainder of this section the discussion will be about the the requirements and problems of the individual techniques.

- ***Collaborative filtering.***
  The great strength and appeal of collaborative filtering is that our software does not need to understand the objects searched over. Thus, no metadata needs to be available, and no machine-understandable formats need to be used.
  The great limiting factor for collaborative filtering is that a collaborative filtering system *can not say anything about an object unless that object has been rated by someone* (actually collaborative filtering can not say anything unless the object has been rated by someone who has rated at least one item, and preferably more in common with you). But the recommended objects may be very different from objects the user has seen in the past, thus collaborative filtering tends to allow a fair amount of exploration as well as exploitation (the focus being determined by what the other users have rated compared to you).
  Out of the above limiting factor grows the *ratings-sparsity problem*: the system is almost useless when there are to little users. The same limited usefulness applies for a new user who has rated no or very few items so far.
  Also since for collaborative filtering you need access to other user's profiles *privacy concerns are increased* for two reasons: First if you have access to other user's profiles that might compromise the other user's privacy. Second, this probably means you will have to do the filtering server side(a P2P[12] approach would be the other possibility).
  There are also some serious *scalability concerns.* One needs to compare to all users/objects available, which grows in to a major pain when there is a couple of million users/objects (think a search engine like Google). Especially so since the profiles of those users(which are probably memory-based) will probably need to be stored server side (unless the P2P approach is taken).
  A further problem with collaborative filtering is the *gray-sheep problem.* A user not fitting in with any groups of alike users, would find the system nearly useless, because he has no or nearly no peers recommending objects for him. Note that this problem is actually the ratings-sparsity problem in disguise.
- ***Content-based filtering.***
  There is one limiting factor with content-based filtering: it is *limited to recommending/filtering out items similar to items you have seen in the past* (e.g., if you liked books from I. Asimov in the past it will recommend more of those, but it would not recommend books from G. Bear which you would probably also like). This naturally means that content-based filtering focuses on exploitation, limiting exploration to similar items.
  The second problem with content-based filtering is that one of the following two statements needs to be true. Either the *objects need to be machine understandable* (e.g., HTML, using IR-techniques) in which case our system will also need a component that understands the format, or we *need to have metadata attached to the objects*, so the computer can make sense of them.

12 P2P means peer to peer.

Method	Problems and requirements
Personalized search in general	• Authentication is required (non-issue for client side personalization) • Privacy needs to be guaranteed ->only an issue with server side personalization
Collaborative filtering	• No need for machine-understandable objects or metadata • Ratings-sparsity problem • Needs access to other peoples profiles -> increased privacy concerns • scalability concerns • gray-sheep problem
Content-based filtering	• Limits exploration ability • Objects need to be machine-understandable or have metadata attached
Economic filtering	• Needs a limited resource related to user interest • does not focus on the need of the user, but focusses on the limitations of the user -> seems the wrong focus.
Demographic filtering	• Needs Demographic information about the user -> hard to get in unobtrusive way -> is non-anonymous, thus brings up privacy-issue • Needs information about properties of demographic groups -> probably needs statistical/social research -> some sort of dynamic stereotypes may be possible also, eliminating the above research need.

Box 2.4: Filtering methods overview of problems and requirements

- ***Demographic filtering.***
  This approach attempts to relate a person's cultural background to his informational needs.
  The big problem with this approach as it has been implemented is that the underlying assumption "a person's demographic background indicates a person's cultural background" is increasingly untrue (as explained in subsection 2.3.4).
  A second problem with this approach is that even when we have determined a person's cultural group we need to know how this translates in informational needs. For this we will need statistical and/or social research.

- ***Economic filtering.***
  This approach attempts to take into account the resources needed to obtain information, and balance them with the interests the user poses in that kind of information.
  In order for this approach to be useful there needs to be some kind of limited resource (e.g., money, or bandwidth) involved. Furthermore we need a way to relate this resource needs to the expected worth of obtaining that information. This is a big problem, and is usually solved by using a resource that has a direct relation to the user's interest (e.g., an item the user buys is considered interesting; a streamed-only broadcast needing 126kbs when only 56kbs is available to the user would be uninteresting).

2.5 Chapter conclusions

From the discussion in this chapter the following conclusions can be drawn.

- For user profiling the most promising approach is to observe the user's interests unobtrusively. The observed interests can then be stored either directly or used as training data for learning a model of the user interests. Obtrusive interaction is inadvisable, users wants a tool that reduces their workload, not one that gives them extra tasks to perform.
  Capturing the world-view and cultural bias of a user using general user traits is an appealing approach, but, since the question of which general user traits offer a good indication of this world-view is essentially unanswered this is currently not an option.
- When observing the user unobtrusively the interesting challenges are creating a good user interface and good heuristics. A well-designed user interface will both empower the user and also give our profiler lots of good and preferably unambiguous information, that may be used in our heuristics.
  The reliability of the observed information will depend largely on how much of the user's world is observed and on how accurately we can observe it (consider a document opened for 3 minutes, was our user looking at it, or was he answering the telephone?)
  Positive feedback is a lot easier to observe unambiguously then negative feedback, but maybe a novel user-interface approach could help?
- In order to be useful the user profile needs to be filled as good as possible. Thus ways to bootstrap the user profiling process are important. Since one way to bootstrap a user profile would be to use the information gathered by other applications, standardization is an issue that needs to be addressed.
- Computers are horribly unequipped to infer the semantics of documents. Thus human-supplied and machine-understandable metadata should be added in all possible ways if content-based filtering is to be used.
  In a controlled environment, such as a digital library, the adding of metadata can be made a requirement. RDF/XML coupled with the DCMES provides a good metadata base. This base can be extended with domain-specific metadata element standards (which for most domains will still have to be developed).
- Of the four matching methods only two are generally applicable for searches. Demographic filtering is flawed because its basic assumption (that demographic user traits are a good indication of user interest) is increasingly untrue. Economic filtering is only useful when there is a clear relation between some limited resource and level of interest, as is the case when, for instance, one is looking for a restaurant (people have different opinions of whether to prefer cheap over fancy). Collaborative filtering needs numerous users (to keep the ratings-sparsity problem under control), while content-based filtering needs to have metadata available, which often proves problematic.
- Privacy is a key issue. Without privacy user trust will dissipate as user profiles are (ab)used by spammers and other agents of questionable conduct. Also a lack of adequate privacy might make the Orwellian nightmare a reality (if you think that is far fetched consider the fact that an agency could the Information Awareness Office (IAW) has recently been called into existence by the US).

Chapter 3

Evaluating personalized search

In the previous chapter the overall problem of personalizing search was broken down into three subproblems. Each of those subproblems was discussed, and for each problem several methods for dealing with the problem were introduced. This chapter explores the second research question of this thesis, how to measure and compare the effectiveness of personalization in personalized search systems. Since a consensus about what measure to use has not currently been reached, this chapter is limited to offering an overview of the proposed measures found in the literature.

The internal structure of this chapter is as follows: First, in section 3.1 the focus is on the traditional measures form Information Retrieval (IR), and why they are not the answer in the context of personalized search. Next, in section 3.2, our attention turns to a class of measures with the name rank-ordered-list measures. Section 3.3 then introduces the mean average error measure. Finally, in section 3.4 the conclusions of this chapter are presented.

3.1 Traditional measures from IR

Since personalized search is a kind of search the first evaluation measures coming to mind are the two well-established measures from the field of Information Retrieval, namely:

- *Recall,* which is simply the number of relevant items retrieved divided by the number of relevant items in the collection.
- *Precision,*which is the number of relevant items retrieved divided by the number of items retrieved.

While these measures offer a good indication of the overall search performance, they do not say anything about how well our tool is able to predict user interest, that is they measure the search performance, not the personalization performance. Thus a different kind of measure will have to be devised.

3.2 Rank-ordered-list measures

In order to understand the reasoning behind this class of measures you need to realize that each personalized search system can be transformed to one that re-ranks the search results (this was explained in the last paragraph of subsection 2.3.1 and Box 2.3). The idea behind this class of measures is then to compare the rank-ordered-list produced by the personalized search with the optimal rank-ordered-list. Here the optimal rank-ordered-list is the ranked list that the user would produce if he re-ranked manually.

[Pretschner, 1998] introduces two such measures: the eleven point average, and the normalized distance-based performance measure. The remainder of this section explains each of these methods briefly.

- *Eleven point average.* The basic idea is to partition the set of retrieved documents into relevant and non-relevant documents (within each class the ordering does not matter). Eleven point average then yields a set of values reflecting how many relevant documents were highly rank ordered.
  In particular, we calculate for each relevant document d_r in the rank ordered list the following measures:
  - recallPoint(d_r) = #relevant items up to d_r (d_r included)/ #relevant items retrieved
  - precision'(d_r) = #relevant items up to d_r (d_r included)/ #items in list up to d_r (d_r included)

  In order to be able to compare a system with respect to different queries or to other systems we need a uniform recall scale. To this end we divide the interval [0;1] into 10 intervals of equal length, yielding 11 interval boundary points (hence the name 11 point average). We now calculate precision' for each boundary point. Of course since the actual recall points probably do not coincide with the boundary points so we need to interpolate the values: the interpolated precision' at a recall cutoff *R* is defined to be the minimum precision for all recallPoints >= *R*. By averaging the obtained vectors of length 11 component-wise for different queries we obtain an average vector for the personalization system that can be used to compare against other systems.
- *Normalized distance-based performance measure (n-dpm)*, (see also Box 3.1, and Box 3.2). If we have two rankings of objects, then for each pair of objects three situations are possible.
  1. Both rankings agree on which document is more important, we say the distance between the two rankings is 0.
  2. One ranking does not exhibit any ordering between the documents (i.e., they are indifferent), but the other does, we say there is a slight distance (=1) between the two rankings.
  3. The preference relationship in one ranking is the inverted from the other ranking, we say there is a large distance (=2) between the rankings.

  The total distance between two rankings is then the sum of the distances between all possible pairs of the two rankings.
  There is one *problem*: a performance improvement which reduces the distance from 300 to 100 for one query is considered the same as the improvement that reduces the distance from

As noted in [Pretschner, 1998] the motivation for this measure is the observation that while users are unable to reproduce a cardinal ranking, they are able to reproduce an ordinal ranking.

Box 3.1: Motivation of n-dpm

We can use n-dpm with either the “perfect criterion” or the “acceptable ranking criterion”:

- The *perfect criterion*: then we use the optimal ranking(i.e., the one the user would create manually) as the ranking to compare to.
- The *acceptable ranking criterion*: Here we compare to a ranking that only requires user-preferred documents to be ranked higher than non-preferred ones (i.e., we do not penalize a ranking that imposes a preference between objects the user finds indifferent, we only require the user preferences to be uphold).

Box 3.2: Different criteria used with n-dpm

3 to 1 for 100 queries. The *solution* is to normalize the distance measure. We achieve this by scaling the actual distance with respect to the worst distance (i.e., the ranking that inverses all preferences from the ranking you are comparing to). This comes down to dividing the distance by 2 times the number of preferences in the ranking.

3.3 Mean average error

Mean average error (MAE) is the name of another proposed measure. It simply computes the mean average error between predictions and ratings ([Sarwar et al., 2001], [Breese et al., 1998]). In other words it sums the absolute errors between the rating-prediction pairs and divides it by the number of rating-prediction pairs.

3.4 Chapter conclusions

In this chapter we discussed how to evaluate a personalized search, the most important conclusions are given below.

- If one wants to measure the effectiveness of the overall search, then the traditional measures from IR, i.e., recall and precision, are perfect. However, since we would like to compare different personalization approaches, we need a measure for evaluating the effectiveness of the personalization.
- Several measures have been proposed that put a number on the effectiveness of the personalization. However, currently there is no consensus as to what measure should be used. In my opinion the rank-ordered-list measures seem to form the most promising approach.

Chapter 4

Greenstone

In this chapter we introduce Greenstone, a software package for creating and maintaining digital libraries. As such this chapter does not have any direct relevance with respect to personalized search, Its relevance is indirect in providing a general understanding of the Greenstone software. This understanding is the base from which the next chapter will examine how personalized search, as discussed in chapter 2, can be incorporated into Greenstone.

The structure of this chapter is as follows: First, in section 4.1, Greenstone is introduced. Next, section 4.2 explains where to obtain Greenstone. Then in section 4.3 a general overview of the Greenstone design is given. Following that section 4.4 goes into more details on several points relevant for the discussion in the next chapter. Finally, section 4.5 reflects on the important points of this chapter.

4.1 Introduction to Greenstone

Greenstone is a software package for digital libraries. As such it provides a way to access a repository of digital documents, of different formats, through a uniform interface. Greenstone also provides a way to create that repository and easily maintain it.

Within Greenstone a set of digital documents is called a collection. Greenstone provides the infrastructure needed to search and browse such a collection in an organized manner. The exact nature in which a collection can be browsed through and searched upon is specified when the collection is built.

Greenstone is open source[13] and can be run on both Unix and Windows platforms. Greenstone is written in a combination of Perl (the plugins) and C++ (for everything else) and makes liberal use of other open source programs.

4.2 Obtaining Greenstone

You can find the Greenstone project at its homepage, [Greenstone]. Apart from the information there, you can find example collections, a user manual [GUM, 2001] and a developers manual [GDM, 2001] at the home page [NZDL], of the New Zealand digital library project (from which Greenstone sprung forth).

13 Greenstone is licensed under the [GNU-GPL]

4.3 Greenstone overview

Looking at how the Greenstone software works, one notices that it should be divided into two distinct parts. First there is the part responsible for building the collections. Second, there is also the runtime system, which is responsible for the interaction with the collections once they are built and put online. The structure of this section respects this dichotomy, discussing each part in turn: subsection 4.3.1 examines Greenstone from the perspective of building a collection, and subsection 4.3.2 explains the Greenstone runtime-system.

4.3.1 Creating a collection

As explained in section 4.1 a collection is a set of digital documents accessible through Greenstone. Creating such a collection happens in three stages. First, we import the documents into Greenstone, then we build the collection and its associated structures, and finally we activate the collection making it accessible in our digital library. The remainder of this subsection discusses each of these steps respectively and in its own subsection.

A. Importing documents

The goal of this stage is to convert all documents in the collection to Greenstone Markup language (GML). GML is an XML-compliant format that standardizes the document format and the way metadata is specified for those documents.

This conversion process is executed by a plugin-mechanism allowing for easy addition of new document formats. Currently plugins are available for a variety of formats including text, HTML, Ms-Word, PDF, postscript, email, e-text[14], FoxBase, and HTML with special UN-tags[15]. It is the responsibility of the plugins to extract the metadata implicit in the original document format (for instance, the title as set by the HTML title-tag), and expressing this metadata explicitly in GML.

ConvertToPlug:

This is a base class for writing plugins that use external conversion utilities to convert a format to text or HTML, which is then processed by Greenstone.
The reason this class exists is that it is Greenstone policy to use external and open source conversion utilities for proprietary formats. This is done to avoid having to support these formats directly, which avoids the tricky nature of these formats for whom there is often no, or very little documentation available, and which are subject to change without notice (just think of the MS-Word format).

Box 4.1: A base class for writing conversion plugins

14 The format used by the Gutenberg project, which you can find at [PG]
15 HTML-pages from created by the UN use a number of extra markups specific to the UN

B. Building the collection

The building stage does three distinct things: (1) the text is compressed, (2) the indexes are created (each index corresponds to one way of searching the collection), and (3) the browsing structures (called classifiers) are created, as specified in the configuration file.

Indexes can be created on either the full text, or on the metadata available (e.g., one index on the full text, a second index on the titles, and a third on the authors). Which indexes are created is again specified at building time.

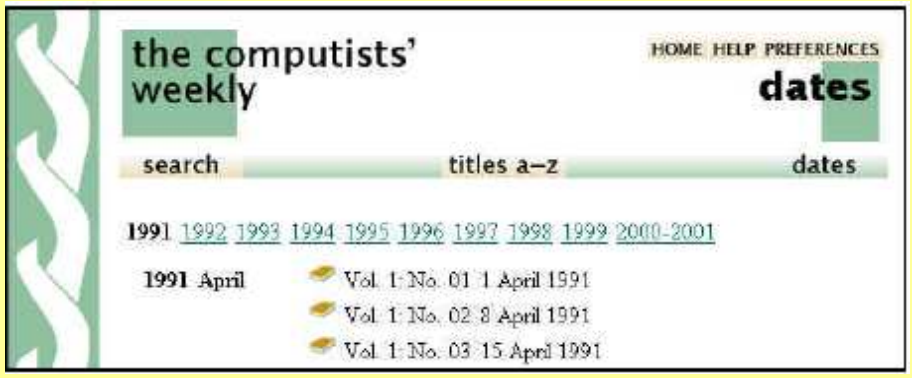


Figure 4.1: A screenshot showing the DateList Classifier

Every classifier gives one specific way to browse the collection in an organized manner. Like the format plugins these classifiers are Perl objects and Greenstone includes a number of classifiers out of the box (see Figure 4.1 and 4.2 for two examples). Classifiers organize a collection into some structure by means of the metadata available.



Figure 4.2: A screenshot showing of the Hierarchy Classifier

Classifiers tend to have parameters (e.g., the hierarchy classifier, shown in Figure 4.2, expects you to specify what metadata element to use on each level). Again creating new classifiers is as easy as creating a Perl object and specifying its use to the building system.

C. Activating the collection

This last stage of the collection creation process is very simple. The files created in the building stage are simply moved to the online directory. If you wonder why these files are not put in the online directly consider the following two facts.

- The building process can take a long time, especially for larger collections
- During the building process the collection is not guaranteed to be in a workable state

It is undesirable to take collections off-line for maintenance, thus while building a new version of a collection, we leave the previous version online, and intact (adding new files, indexes or classifiers all warrants a rebuild). Thus the output of the building process is directed to a temporary location, and then when the building process is completed the newly-built version is moved online, activating the collection.

4.3.2 The run-time system

Once a collection is online all interaction with that collection is handled by the Greenstone run-time system, which utilizes a client-server architecture. Below in the first subsection we discuss the setup of the client-server architecture. Then in the next two subsections we will discuss the client and server side respectively.

A. Client-server architecture

The Greenstone run-time system is split into a client-server architecture (illustrated in Figure 4.3). The clients, named receptionists, are responsible for the user interface. At the server side collection servers act as an abstract mechanism that handle the content of the collection.

The simplest configuration is that with a single collection server and a single receptionist running on the same computer. The current implementation corresponds to this situation and combines the receptionist and the collection server into a single executable. Thus, while the design has a client-server architecture, the current implementation appears to the user as completely server side.

To make more sophisticated setups possible, the Greenstone project is currently working on a CORBA implementation of the protocol[16]. This new protocol implementation will allow more sophisticated setups.

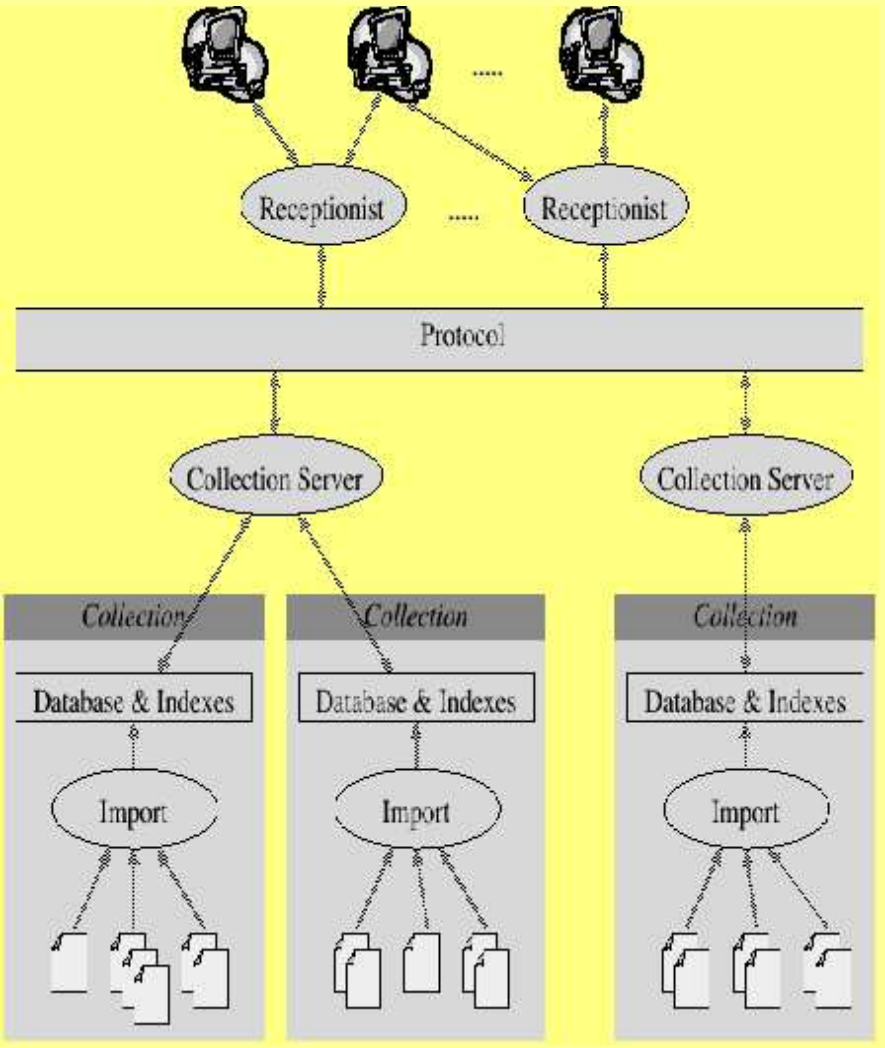


Figure 4.3: The Greenstone client-server architecture

B. The receptionist

As mentioned earlier the receptionist is responsible for the user interface.

The default Greenstone installation provides a web-interface. Here the user interacts with the Greenstone run-time system through a web interface. Commands are given clicking links that call the Greenstone-binary through the cgi-mechanism. The receptionist parses the arguments received

16 The current protocol implementation simply uses function calls (which is possible since the receptionist and server are combined into a single executable). This protocol implementation is called the null protocol.

and contacts the collection server accordingly. The collection server then returns the appropriate content in the form of bare HTML, which is then dressed up by the receptionist as specified by the appropriate macros, and displayed in the user's web-browser.

It is this use of macros that results in the flexibility of the user interface. All you need to do to adapt the interface, is adapt the macros. No recompilation is necessary because the web-pages are generated on the fly. With only some knowledge of HTML and the ,relatively simple, Greenstone macro language you can thus adapt the user interface to your desires. This mechanism is used in the standard Greenstone setup to provide the interface in different languages (Note: it is only the interface that is translated not the documents themselves). Every collection has a default language, but the user can change the default to any of the available languages. New languages can be added by providing an additional language macro.

In the created web pages there are, again, interface elements (links for browsing, text-field with button for search, and so on) who call the receptionist with the appropriate cgi-arguments when used. When the receptionist is called the whole process starts over again.

C. The collection server

As mentioned earlier the collection server acts as an abstract mechanism to handle the content of a collection. The collection server itself consists of three kinds of objects: filters, search objects and source objects.

Filters provide a way to select a subset of the information in a collection. There are two kind of filters: query-filters, who call upon the search objects, and browsing filters, who are, indirectly called by the classifiers.

Search objects supply an abstract interface to the search mechanism, Greenstone currently implements this abstract interface through the use of the managing gigabytes search tool [MG].

Source objects supply the abstract interface to get the text and metadata from the documents in the collection. Currently, Greenstone implements this through the use of managing gigabytes and GDBM(the Gnu database manager, a tool implementing a flat text DB, see [GDBM]).

These objects are implemented in C++ with virtual inheritance. Because of this it is relatively easy to adapt Greenstone. For instance to replace MG by an alternative search-tool, one only has to create a new sub-class of the search class, and adapt the statements at the point of object construction to use the new class. Everything else is handled automatically by the virtual inheritance mechanism.

4.4 Relevant details

The previous section offered a general overview of the Greenstone design. This section examines several parts of the Greenstone design in more detail. The parts examined more closely here are relevant to the discussion in the chapter 5 which examines how we could adapt Greenstone to use the methods for personalized search as discussed in chapter 2. We examine more closely the following aspects of Greenstone: First, In subsection 4.4.1 the focus is on how Greenstone executes a search. Next, in subsection 4.4.2 the attention turns to metadata and Greenstone. Finally, in subsection 4.4.3 the subject is the way in which Greenstone handles authentication.

4.4.1 Searching a collection

Like any other action in Greenstone a search is initiated through the cgi-mechanism. The series of events evolves as given below:

- User specifies a query and hits “Begin Search”.
- This executes a query action through the cgi-mechanism.
- The query action sets the filter type to “queryfilter”, passes search preferences, and calls the designated collections filter object through the protocol.
- The filter object decodes the options and makes a call to the search object.
- The search object abstracts the search, and translates the data generated by the search in a standard format (basically a list of OIDs, a number identifying an object, with coupled rankings).
- The search results are transferred by the filter-object to the receptionist who displays the results.

The interesting part to note for the purpose of the thesis is that search results are returned in a standard format, independent on how the search is carried out.

4.4.2 How Greenstone obtains metadata

As explained Greenstone converts all documents to GML during the import process, and this import process is done using plugins. The first way that Greenstone can obtain metadata is by using plugins that recognize metadata inherent in the document format (e.g., the title tag of a HTML page can be taken to correspond to the Dublin core title element). The second way in which Greenstone can obtain metadata is by using a plugin that associates metadata it is given, to documents being imported. Greenstone provides two such plugins. The first, named IndexPlug, works by adding metadata, specified in a manually-created spreadsheet-like text file, to documents. The second one, RecPlug, adds metadata, specified in multiple XML-files (one per document), to the documents.

4.4.3 Authentication within Greenstone

The Greenstone web interface, besides facilitating access to collections, also facilitates building collections, and doing common administrative tasks such as adding users, and browsing logs. This means security is necessary, after all we do not want everybody to be able to create collections or change settings. To this end, Greenstone has implemented a simple login and password authentication scheme.

This simple scheme provides some protection but, in its current implementation, has two problems. First, the Greenstone web interface is accessed using the HTTP protocol. Since HTTP is unencrypted this means that any communications, including any passwords, can be intercepted. Second , Greenstone stores the passwords in an encrypted form in a file. The problem here is that this file is world-readable and thus vulnerable to a dictionary-attack.

4.5 Chapter summary and conclusions

This chapter deals with Greenstone, a software package for digital libraries. Important to remember is the following.

- The Greenstone software consists out of two separate parts: the building system, and the run-time system.
- The building system is responsible for converting documents to GML, extracting and adding metadata, compressing the information, and creating the necessary indexes. New document formats and browsing structures can be created by constructing a corresponding Perl object and specifying the created object for use to the building system.

- The run-time system is responsible for making a built collection accessible to the user. The run-time system has a two-tier client-server architecture. The server tier consists of *collection servers* which provide an abstract mechanism to handle the content of the collections. The client tier consists of *receptionists* who provide the user-interface to the collections.
- Greenstone abstracts the search engine used, and has results of the search converted to a standard format
- Metadata is specified to Greenstone at building time through the use of plugins. Metadata can come from two sources: either it is implicit in the used document formats, or it is specified using an additional plugin, written for this purpose.
- Greenstone currently implements a simple authentication scheme.

Chapter 5

Personalizing search in Greenstone

Chapter 4 facilitated an understanding of Greenstone by giving a high-level overview of Greenstone and more detailed explanations of several Greenstone aspects. Using that understanding as a basis this chapter explores how the personalization techniques discussed in chapter 2 could be incorporated in Greenstone.

In chapter two we divided the overall problem of personalizing search into three interrelated subproblems: Matching user and document (discussed in section 2.3), profiling the user (discussed in section 2.1), and characterizing the document (discussed in section 2.2). Section 5.1 through 5.3 follow this division exploring each subproblem from within the context of Greenstone. Then in section 5.4 the discussion turns to how to augment the Greenstone design to incorporate the personalization methods found applicable in sections 5.1 through 5.3. The chapter then finishes with the conclusions in section 5.5.

5.1 Matching user and documenthttp://www.kde.org/

This section explores the problem of matching user and document introduced in section 2.3 within the specific context of Greenstone. The structure of this section matches that of section 2.3. Thus in subsection 5.1.1 we first discuss the considerations introduced in subsection 2.3.1. Then in subsection 5.1.2 we focus on each of the methods introduced in subsections 2.3.2 through 2.3.5 individually.

5.1.1 Considerations

When choosing a filtering approach we need to consider the issues introduced in subsection 2.3.1: “Do we execute the personalization client side or server side?”, “Do we use filtering out, or filteringhttp://www.kde.org/ in?” and “Do we focus on exploration or exploitation?” In this subsection each of these issues willhttp://www.kde.org/ be examined form the Greenstone perspective.http://www.kde.org/

- Regarding the *client side vs. server side* issue the following can be said.
  The first observation is that while Greenstone has a client-server architecture from a design perspective, this is not necessarily true from a user's perspective. If we have a Greenstone setup using the null-protocol (as opposed to one using the upcoming CORBA-protocol), then from the user's perspective there is only a server, with a browser as the client. Thus using the null-protocol client side personalization is not an option within Greenstone. Having said that, the remainder of this chapter will assume the CORBA protocol because such a setup unites the developer's and user's perspective of client side and server side.
  The second observation is that server side personalization requires the protocol to be expanded.

The cause is clear, server side personalization needs the user profiles to be available on the server, which requires either the profiles to be at the server, or the server being able to go and look at the profiles. Note that both solutions will require the protocol to be extended (either to allow user actions/observations about the user to be transmitted, or to allow the user profile to be examined by the server). In contrast, client side personalization does not require the protocol to be expanded. True, the personalization component will need to get metadata about documents from the server, and the current protocol already supports that by the use of browse filters. Thus client side personalization is the choice we make.

- Examining the *filtering out vs. Filtering in* issue shows that filtering in poses a problem while filtering out is readily added. Thus we choose filtering out. This demands a clarification. Relating to filtering in, the first thing to notice is that Greenstone currently implements search using MG. Also note that MG expects indexes to be available on the data to be searched. Also MG expects that data to be compressed (both compression and index building are taken care of during the Greenstone building process). We also know that, for filtering in, the search needs to be conducted on the output of the matching process. This poses two problems. First, it is difficult to have the matching process output something that MG can use. Second, if we supply output that MG can use, then the output of the matching process is tied to MG, and thus changing the search engine would also require the matching routines to be rewritten.
  When we examine adding filtering out to Greenstone the picture looks much better. Greenstone returns the search result in a standard format, unrelated to the search engine used. Thus filtering out can be implemented without a dependency on the search-engine used. Furthermore the standard format used conceptually can be looked at as a ranked list of OIDS(object identifiers, a value uniquely identifying an object from the collection to Greenstone), which is an easily manipulated format.
- Regarding *exploitation vs. exploration* we can not really give any additional thoughts. Whether one or the other is preferred is dependent on the digital library being examined. For each library the information domain covered and the way the library will be used will need to be considered in order to determine where the balance should be. Depending on where the balance is desired, the matching process will need to be implemented to favor one or the other. Ideally this should be tunable (for instance through run-time insertion of different personalization modules, and/or tweaking of module parameters).

5.1.2 What kind of filtering to use

In this subsection the discussion turns to what kind of filtering to use: what filtering method is suited for a digital library in general and Greenstone in particular.

A main difference between the different filtering methods is the kind of knowledge they use to filter out information. Collaborative filtering makes use of the relations between past choices made by the user and past choices made by other users. Content-based filtering utilizes knowledge about the characteristics of the documents and the choices the user made in the past. Demographic filtering tries to capture the point of view of the user through the use of demographic data. Finally, economic filtering takes into account the resources needed to obtain a piece of information and tries to balance that with the estimated worth of that information. Ideally we would like a method that combines all four angles stated. So we obtain the most complete picture possible at any time. However the question of how to combine the different approaches has yet to be answered. Hence, in the mean-while we will have to pick the most applicable approach. So let us examine the four approaches in the context of digital libraries.

- *Collaborative filtering.* As discussed collaborative filtering automates word-of-mouth recommendation. The greatest advantage of this technique is that the metadata it needs (namely user X liked item A, but did not like item B, whereas user Y liked both item A and B, ...) can be obtained in an automated fashion. A first disadvantage is that it needs access to other user's profiles which raises privacy concerns. While techniques for solving these privacy concerns exist implementing them does add a lot of complexity to the implementation. Also consider in this regard that there is a constant arms race between the good and bad guys on everything security related, which means that the privacy guarding features will need to be evaluated, and adapted continuously as techniques for breaking and ensuring security evolve. A second problem is the ratings-sparsity problem introduced in section 2.4 in order for the system to be able to say something about a document there must be at least one other user who has already rated the document in question and has also rated at least one item in common with the user for whom the filtering is made. Exactly how big a problem this is depends on how many users there are (if there are a lot of users, chances are you will find somebody with similar interests, who has already rated the document in question) and how many documents there are (if there are a lot of documents changes of somebody similar already having rated a random document will be smaller).
  Also since we decided above to not use server side filtering (see subsection 5.1.1), we would need to either reverse that decision (and thus extend the Greenstone protocol), or design a P2P system. Implementing a P2P system is a non-trivial task, and would need to be done for each client that wish to use the personalized search.
  Conclusion, generally speaking collaborative filtering is a definite option, though one has to consider that there will be an overhead of implementing and keeping up to date privacy guarding features and that depending of the contents and public of the library in question the ratings-sparsity problem might make this filtering technique of limited applicability. However taking into account the existing Greenstone architecture, adding collaborative filtering would mean extending the Greenstone Client-Server protocol. Since we decided against doing this, collaborative filtering is out of the running.
- *Content-based filtering.* The one big disadvantage of content-based filtering is that it needs metadata. Thus the objects being filtered need to be either machine-understandable or have metadata assigned to them. However this becomes a relatively small issue in the context of digital libraries. Here is why. A digital library is a controlled environment. Its content is controlled entirely by the librarian(s). Also consider the fact that librarians are used to creating/adding metadata to libraries for ages (in the form of indexes and catalogs). Thus in the context of digital libraries it is entirely feasible to assume metadata to be available, by making that the responsibility of the librarians.
- *Demographic filtering.* As discussed before demographic properties of an individual are an increasingly bad indicator for the world-view and cultural bias of individuals. This explains the limited usefulness of demographic filtering in its current form. Since a way to successfully apply the underlying principle of demographic filtering still needs to be established demographic filtering is also out of the running for the time being.
- *Economic filtering.* In the context of digital libraries I recognize two resources that could be used for economic filtering, namely bandwidth and time. To clarify this, consider the following. First consider bandwidth: if the user has a slow connection (because his own Internet link is slow, or because the server is overloaded, or because the route to the server is congested) it makes more sense to prefer smaller documents over large ones (e.g., if the user only has a 3600 baud connection a 2 MB document will have to be deemed very interesting for the user to download

it). Thus the available bandwidth to the server at any given time could be used for economic filtering. Next consider time: obviously if the user has little time to spend documents that are smaller, and thus read faster, will be preferred (e.g., if the user only has half an hour to spare a 200-page document is not a good choice). We already explained that economic filtering tries to balance the resources needed to obtain and use a piece of information with the expected worth of that information. However, in order to estimate effectively the worth of a piece of information we would probably need to combine economic filtering with content-based filtering. Since a method for combining economic and content-based filtering has yet to be devised, economic filtering is currently not a good choice for implementation in the context of digital libraries.

Given the above discussion content-based filtering is the approach to use for augmenting Greenstone. Collaborative filtering is possible, but would require extensions to the client-server protocol. Also consider that the big problem of content-based filtering, namely the availability of metadata, can be considered a relative non-issue in the context of Digital libraries, while the big problem of collaborative filtering, namely the ratings-sparsity problem, remains. Thus content-based filtering is the most natural choice both for Greenstone and other Digital library packages.

5.2 Document characterization

As argued in the previous section content-based filtering seems the most promising approach for personalizing search in the context of digital libraries. Thus the question arises which metadata to use.

The Dublin core was created as a set of generally applicable metadata. Furthermore the metadata elements of the Dublin core correspond directly to the metadata librarians have been maintaining for ages in the form of catalogs and indexes. Thus the Dublin core metadata set constitutes an obvious choice for minimal metadata.

Next to the minimal metadata laid out in the Dublin core domain-specific metadata schemes should be created (preferably using RDF, and in collaboration with the domain's experts), and applied where applicable to supplement the Dublin core elements.

5.3 User profiling

As explained in section 2.1 we need to answer three questions regarding user profiles: “What do we put into the user profiles?”, “How do we obtain the information needed for the user profiles?”, and “What format do we use for the user profiles?”

We also showed, in subsection 2.1.1, that user profiles for personalized search try to capture user interest. They do this by simply storing the observed interests (memory-based profiles), or by modeling what constitutes an interesting document for various interests topics (model-based profiles). We further noted that the choice of whether to use memory-based or model-based user profiles is determined by the matching method used. We decided, in subsection 5.1.2, on using content-based filtering. Hence we will be using a model-based profile.

In subsection 2.1.2 we discussed obtaining the information needed for the user profile. We argued that one can either use obtrusive or unobtrusive methods. We further argued that one should prefer unobtrusive methods, and that obtrusive methods should be used only at the user's discretion. Thus we now propose the following. First, we use unobtrusive observation for getting the needed observation. Second, we give the user the possibility to give reference feedback at his own discretion. Third, when the user chooses to use reference feedback we use that feedback to tune the interpretation of the observations we get unobtrusively. Let me clarify this as follows. Assume we observe the reading time of documents to get information about user interests. Obviously a

document we spent a lot of time reading will be interesting. However, different people will have different reading speeds. Thus, even though we can provide a suitable default value, the decision of after how much time a document should be considered interesting ideally should be user dependent. By examining the reading times of documents for which reference feedback was given we can tune the cutoff value used in interpreting reading time.
As far as deciding what user actions to observe is concerned we note that with the current Greenstone implementation our options are limited. This is because the only communication we have with the user is in the form of cgi-arguments. Thus we are limited to using the examination behaviors selection, reading time, and repetition. Also note that reading time can be inaccurate due to factors of varying network congestion and latency. However the upcoming CORBA-based protocol implementation will make new receptionists with broader capabilities possible.

5.4 Implementation design

In this section the discussion turns to the specific context of Greenstone. Discussed here are the adaptations needed to the Greenstone design in order to add personalized search to Greenstone. The breakdown of the overall personalized search problem introduced in chapter 2 is followed. So in subsection 5.4.1 the discussion is about document characterization. Next, in subsection 5.4.2 adding user profiling to Greenstone is discussed. Then in subsection 5.4.3 the attention turns to adding the matching process to a Greenstone search.

5.4.1 Document characterization

As explained in chapter 2 metadata can either be added manually or extracted by computers. From the Greenstone viewpoint metadata is added during the import phase of the building process. The muscle of this import phase is provided by the plugins used, and using plugins both the manual adding of metadata, and automated extraction of it are catered for. In fact, to some extent Greenstone already provides both: automatic extraction (in a limited form) is provided by the format conversion plugins (e.g., HTMLPlug extracts the title as found in the title tag), and manual addition is catered for by both IndexPlug(add metadata in a spreadsheet like file) and Recplug (add metadata for foobar file in foobar metadata using simple XML-format). One can also easily envision a plugin that carries out text classification, or that fetches metadata from the librarian using a nice graphical user interface, or maybe one that fetches metadata out of some available database[17].

Given the fact that a digital library is an environment controlled by librarians, who are used to adding/creating metadata (in the form of indexes and catalogs). A policy requiring, for instance, the Dublin core elements, possibly combined with elements from some domain-dependent metadata scheme, could be facilitated using a plugin requesting the missing elements from the librarian.

5.4.2 User profiling

To deal with the problem of user profiling we propose a framework with 3 kinds of objects, Observers, Judges, and Profilers (as illustrated in Figure 5.1).
Observers deal with the details of obtaining the information we need to build the user profile.
The information obtained is then interpreted by the judge leading to a verdict of whether the document is deemed interesting to the user.
The profiler takes the judges' verdicts and uses them to refine/adapt the user profile.

17 Maybe collaboratively maintained DB of BibTeX entries?

This framework is geared for unobtrusive methods of collecting information for the profile. However, obtrusive methods fit in also, even though in that case the distinction between the observer and judge is pointless.

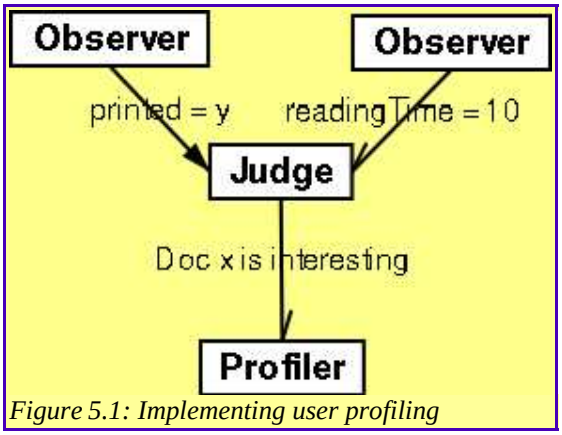


Figure 5.1: Implementing user profiling

The current Greenstone implementation limits us in our behavior of the user since we are only able to observe the user through the cgi-calls he makes. Still the examination behaviors of selection, repetition and reading time are easily implemented by adding observers to the code processing the cgi-arguments. Again, the upcoming CORBA-based protocol will allow us to observe the user more extensively and accurately. Let me illustrate this using the example of reading time: using the current implementation, reading time could be observed by measuring the time between the cgi-call requesting a document and the next cgi-call. However this measurement is inaccurate because of two reasons. First, there is the network connection, whose variable congestion makes any recorded times inaccurate. Second, there is the fact that we do not know whether the user is busy reading the document or doing something else (say sending an e-mail). A receptionist using the upcoming CORBA-based protocol implementation could avoid the network factor and lessen the second concern (we still do not know whether the user is making a phone-call, or has gone to the bathroom, but we do have the ability to observe whether the receptionist is the foreground process, or some other program is being used).

5.4.3 Matching user and document

For a discussion on what filtering method to use in the context of digital libraries we refer to subsection 5.1.2. In the current subsection the discussion is about how a filtering method (irrespective which one) fits into the Greenstone design.

As explained in subsection 4.3.2 Greenstone has objects named filters that provide a way to select a subset of the information in a collection. Currently Greenstone provides two kinds of filters: *queryFilters* and *browseFilters*. The design adaptation proposed here is to add a third kind of filter, namely a *personalizationFilter*. Such a *personalizationFilter* would intercept the results of a *queryFilter* and re-rank them according to one of the matching methods we discussed.

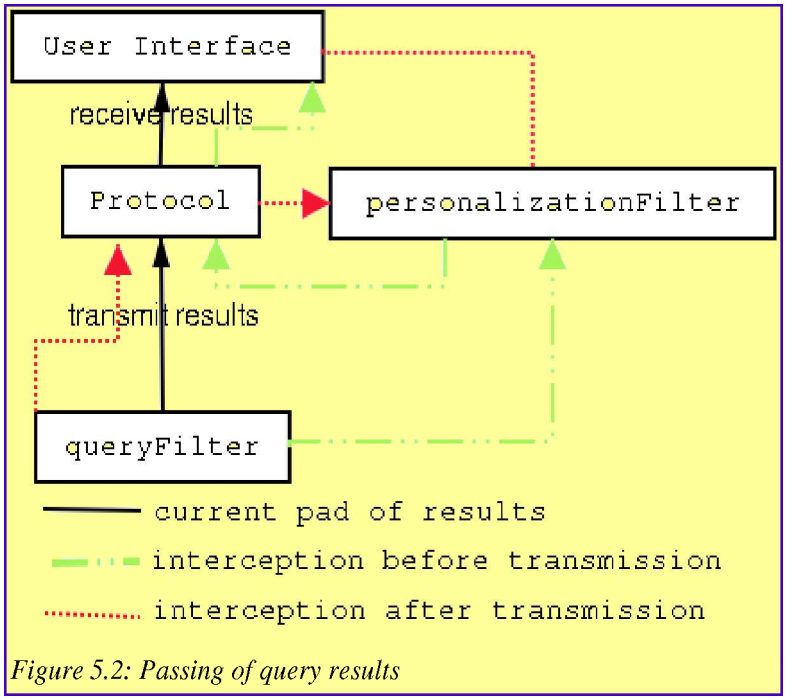


Figure 5.2: Passing of query results

Let us examine the possible filtering implementations, regarding personalization, this makes possible. First, note that this is a filtering-out approach to personalization since it re-ranks the results of the ordinary search process. Further consideration reveals that we can implement both

server side and client side personalization, we just need to decide whether we intercept the results on the client side or the server side of the protocol. In other words, do we intercept them on transmission (server side) or reception (client side) (see Figure 5.2)? Earlier (subsection 5.1.1) we decided to do client side personalization

A virtual base class should make up *personalizationFilter*, including a virtual method *getRating()* that takes a document and returns a rating. This allows the re-ranking to be implemented generically in terms of *getRating()*. Thus a specific filtering method only needs to create a base class specifying an implementation of the *getRating()* method. The implementation of the *getRating()* method ties back into the user profile maintained by the profiler.

5.4.4 Putting it all together

If we put the above together we see that this proposal ties into the Greenstone run-time system at two different places, as illustrated in Figure 5.3. The observers hook into the user interface, and the *personalizationFilter* sits in between the *queryFilter* and the user interface. Then there is the Greenstone building system which can be used as is, although adding new plugins to extract metadata is preferable.

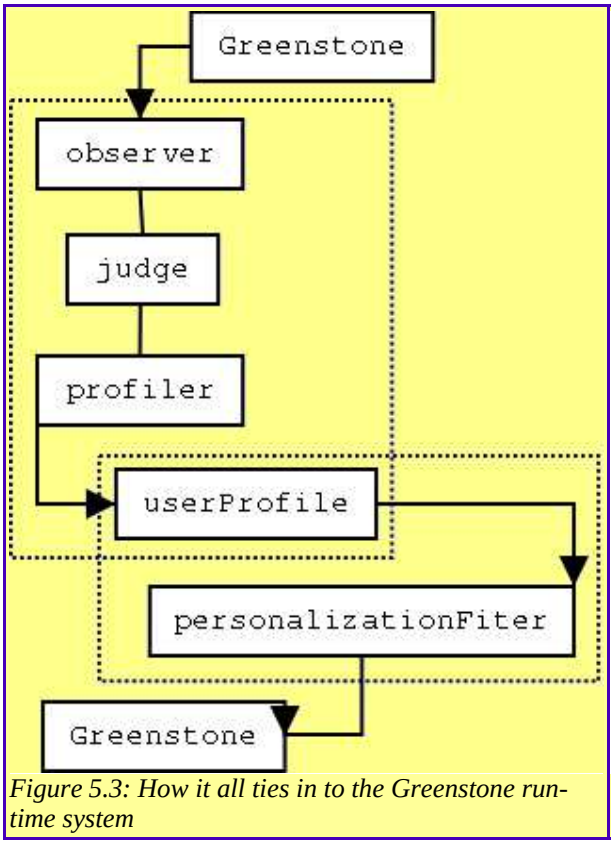


Figure 5.3: How it all ties in to the Greenstone run-time system

5.5 Chapter conclusions

In this chapter we discussed adding personalized search to Greenstone. We arrived at the following conclusions.

- Greenstone features a server-client architecture. However the current implementation is completely server side, though the upcoming CORBA-based protocol implementation will fix this.
- We showed that filtering out is readily added to Greenstone whereas adding filtering in is a problem.
- Regarding the balance between exploration and exploitation we said that the ideal balance is dependent on the use and contents of the library in question. We further noted that the implementation of the matching process determines where the balance lies.
- We concluded that in the context of digital libraries content-based filtering is the best choice. Collaborative filtering would also work, but is a less natural fit (its main deficiency, the ratings-sparsity problem remains, whereas the main problem of content-based filtering, namely metadata-availability, is a relative non-issue in this context). Unfortunately, demographic and economic filtering are currently unsuited in this context.
- We argued for using the Dublin core for the metadata. Furthermore, we argued that for collections dealing with a particular domain, domain-specific metadata sets should be created and used.
- Since we opted for content-based filtering, we will use a model-based user profile

- We opted to obtain information for the user profile using unobtrusive methods. However, if the user opts to use relevance feedback at a certain time, that information will be used both to build the user profile and to tune the interpretation of the unobtrusively observed user behavior.
- Metadata will be added at building time. Plugins will extract metadata automatically if possible and obtain it from the librarians otherwise.
- To deal with the user profiling we will use a framework consisting of observers, judges and profilers. The behaviors we can observe using the current Greenstone implementation are selection, repetition, and reading time.
- The matching process will be implemented as a personalized filter and will intercept and rearrange the normal search results, i.e., we use filtering out (because it can be added without a dependency on MG), we do this client side (to avoid having to extend the client-server protocol), and we re-rank the results instead of hiding them (by default, this would be changeable as a user preference, as converting one approach to the other is easy).

Chapter 6

Conclusions and further research

In chapter 1 four research questions were put forth for examination.

1. Which techniques and ideas concerning personalizing search already exist, and what are the problems and challenges in using them?
2. What measures are available for comparing different personalized search tools?
3. How would these personalized search techniques fit into a digital library software package, like Greenstone?
4. How would we go about augmenting Greenstone to incorporate personalization?

Below the conclusions regarding each of these questions are put forth.

The first research question was addressed by the discussion in *chapter 2*. There we showed that the overall problem of personalizing search can be broken down into three intertwined subproblems: (1) profiling the user, (2) characterizing the document, and (3) matching the user profile with the document characterization.

The second research question was addressed by the discussion in *chapter 3*. Two conclusions were drawn with respect to comparing personalized search tools. The first conclusion was that the traditional measures from IR do not really measure what we want to know. What we really want to know is how well our personalized search tool is able to predict the user interests. The second conclusion is that, while several proposals have been made, there is currently no consensus as how to measure this, though in my opinion the rank-ordered-list measures seem to form the most promising approach.

The third research question was addressed in *subsections 5.1 through 5.3*. There we discovered that, within the context of personalized search, content-based filtering is a natural fit. This is due, in part, to the fact that a digital library is a controlled environment, which makes it possible to ensure metadata is available for all documents. It is also due to fact that librarians have been adding metadata to libraries for ages, in the form of catalogs and indexes, that corresponds to a large degree with the Dublin Core. These two facts effectually render the big problem of content-based filtering, of how to obtain the metadata, a relative non-issue.
Collaborative filtering is also a possibility for personalized search in digital libraries in general, though the ratings-sparsity problem of this approach remains. This could be a problem for highly specialized libraries.
Demographic filtering is currently not a good option, since its basic assumption, that demographic user traits are a good indication of the cultural bias and hence user-world-view is flawed. The basic idea of this approach remains valid, but the open question here is which general user traits are a good indication of cultural bias. Until this question receives a better answer this approach will

remain of little value.
Economic filtering is not a a good approach for personalized search in digital libraries, because there is no clear way to link a limited resource, like bandwidth with user interests. So, unless such a link is found this approach is not applicable.
With regard to obtaining the data for the user profile we concluded that obtrusive interaction is a bad approach, users resent the extra work. Unobtrusive interaction is the way to go. Key here is the interface design. The interface must allow our tool to infer unambiguously as much information about the user's actions as possible. There is a lot of room for improvement and experimentation in this area, an ideal interface would avoid the information overload problem by empowering the user (so he can handle all the available data himself).
Regardless of which approach is taken, it is essential to keep in mind that the idea is to empower the user, extending his ability to deal with the enormous amount of available information. We must keep this in mind so that our tool does not constrict the user any more than absolutely necessarily. Thus we need ensure that our tool does not keep the user constantly in the same little sandbox (be exploitation/exploration balance too much to one side). Also we need to make sure that user's privacy is ensured lest an Orwellian society[18] is created.
All in all, the potential of personalization is quite large. Furthermore this potential will only increase since computers permeate our daily lives more everyday (a development which shows no sings of stopping anytime soon). This translates into an increasing ability to engage in unobtrusive observation of users, thus allowing personalization approaches to perform better. Another potential increasing factor is the development of techniques, like text mining and image recognition. As this class of techniques improve, the ability of computers to assign metadata automatically will rise also, again leading to more opportunities for personalization software.

The fourth research question was addressed in *subsection 5.4*. We showed that content-based personalization can be added to Greenstone relatively easily. Collaborative filtering could also be added but would require more substantial changes.

Several interesting opportunities are apparent for further research, below we offer four of them.

- A first direction for further research is finding a way to combine the various personalization approaches, in order to create an approach showing a synergetic effect.
- A second direction is finding a set of general user traits that is a good indication of the user's cultural bias, such as world-view; thus revalidating the core idea of demographic filtering, opening a whole additional point of entry to the personalization problem.
- A third direction is interface design, more particularly the invention of novel interfaces that empower users so they can handle larger loads of information more effectively.
- A fourth possible direction is using collaborative filtering techniques - not to find interesting information - but to find people with shared interests, thus helping users to expand their social network.

18 That is a totalitarian society as sketched by G. Orwell in his famous novel (imagine what it would have been like if Nazi-Germany had access to the information gathered by a widespread and powerful personalization method).

Table of figures

Figure 2.1: Subdivision of user profiles for personalized search 8
Figure 2.2: Example of relevance feedback found in [Jon, 1993] 10
Figure 2.3: Example of manual change of a profile found in [Sheth, 1994]. With genetic algorithms, each interest is represented by a population of agents 11
Figure 2.4: Example of RDF syntax: RDF/XML and Nodes & Arcs 18
Figure 2.5: Visualization filtering in and filtering out 22
Figure 4.1: A screenshot showing the DateList Classifier 37
Figure 4.2: A screenshot showing of the Hierarchy Classifier 37
Figure 4.3: The Greenstone client-server architecture 38
Figure 5.1: Implementing user profiling 47
Figure 5.2: Passing of query results 48
Figure 5.3: How it all ties in to the Greenstone run-time system 49

Table of boxes

Box 2.1: Categories and their observable behaviors, see [Oard&Kim, 1998] for a detailed discussion 12
Box 2.2: An example illustrating the usefulness of observing the user in a larger part of his computing environment 13
Box 2.3: Transforming personalization approaches 23
Box 2.4: Filtering methods overview of problems and requirements 29
Box 3.1: Motivation of n-dpm 32
Box 3.2: Different criteria used with n-dpm 32
Box 4.1: A base class for writing conversion plugins 36

References

[Balabanovic, 1998] M. Balabanovic (1998).an interface for learning multi-topic user profiles from implicit feedback. *Submitted to 21fst International {ACM}/{SIGIR} Conference on research and development in information retrieval*. URL: http://citeseer.nj.nec.com/balabanovic98interface.html

[**Basu et al.,1998**] C. Basu and H. Hirsh and W. Cohen (1998). Recommendation as Classification: Using Social and Content-Based Information in Recommendation. *Proceedings of the Fifteenth National Conference on Artificial Intelligence, Madison, WI* p. 714-720. URL:http://citeseer.nj.nec.com/basu98recommendation.html

[**Bradley et al. ,2000**] K. Bradley and R. Rafter and B. Smyth (2000). Case-Based User Profiling for Content Personalization. . URL:citeseer.nj.nec.com/386480.html

[**Breese et al., 1998**] J.S. Breese and D. Heckerman and C. Kadie (1998). Empirical Analysis of Predictive Algorithms for Collaborative Filtering. *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-98). San Francisco, CA: Morgan Kaufmann* p. 43-52. URL:http://citeseer.nj.nec.com/breese98empirical.html

[**Budzik and Hammond, 2000**] J. Budzik and K.J. Hammond (2000). Case-based user profiling for content personalization. . URL:http://citeseer.nj.nec.com/budzik00user.html

[Cingil et al., 2000a] I. Cingil, A. Dogac, and A.Azgin (2000).A broader approach to personalization. *Communications of the ACM* nr. 8 pp. 136-141. URL: http://www.srdc.metu.edu.tr/publications.html

[**Cingil et al., 2000b**] I? Cingil, A. Dogac, and A. Azgin (2000). A broader approach to personalization technical rapport. . URL:http://www.srdc.metu.edu.tr/pubications.html

[**DCMES, 1999**] DCMI (1999). *Dublin core metadata element set, Version 1.1.: reference description*. URL:htp://dublincore.org/documents/dces

[**DCQ, 2000**] DCMI (2000). *Dublin core qualifiers*. URL:http://dublincore.org/documents/dcmes-qualifiers

[**GDM, 2001**] David Bainbridge, Dana McKay, Ian H. witten (2001). *Greenstone digital library developers`s guide*. URL:http://download.sourceforge.net/greenstone/Developer.pdf

[**Gokhale, 1999**] A. Gokhale(1999). *Improvements to Collaborative Filtering Algorithms*. URL:http://citeseer.nj.nec.com/389112.html

[**GUM, 2001**] Ian H. Witten and Stefan Boddie (2001). *The Greenstone digital library software manual*. URL:http://download.sourceforge.net/greenstone/User.pdf

[**Jon, 1993**] O. Jon(1993). *Doppelganger goes to school: machine learning for user modeling*. URL:http://citeseer.nj.nec.com/orwant93doppelgnger.html

[**Kurki et al., 1999**] T. Kurki and S. Jokela and R. Sulonen and M. Turpeinen (1999). Agents in Delivering Personalized Content Based on Semantic Metadata . *In Proc. 1999 AAAI Spring Symposium Workshop on Intelligent Agents in Cyberspace* p. 84--93. URL:citeseer.nj.nec.com/431401.html

[**Lawrence, 2000**] S. Lawrence (2000). Context in Web Search. *IEEE Data Engineering Bulletin* Vol. 23 nr. 3 pp. 25-32. URL:citeseer.nj.nec.com/lawrence00context.html

[**Mladenic, 1996**] D. Mladeni'c (1996). Personal webwatcher: implementation and design. . URL:http://citeseer.nj.nec.com/5572.html

[**Mostafa et al., 1997**] J. Mostafa, S. Mukhopadhyay, W. Lam and M. Palakal (1997). A Multilevel Approach to Intelligent Information Filtering: Model, System, and Evaluation. *Information Systems* Vol. 15 nr. 4 p. 368-399. URL:citeseer.nj.nec.com/Mostafa et al., 1997multilevel.html

[**Muller, 2000**] M.E. Muller (2000). Learning comprehensible conceptual user models for user adaptive web search. . URL:http://citeseer.nj.nec.com/391515.html

[**Oard&Kim, 1998**] D. Oard, and J. Kim (1998). Implicit feedback for recommender systems. . URL:http://www.glue.umd.edu/~oard/research.html
[Pazanni&Billsus, 1997] M. Pazanni and D. Billsus (1997).Learning and revising user profiles: the identification of interesting web sites. *Machine learning* nr. 3 pp. 313-331. URL: htp:citeseer.nj.nec.com/pazanni97learning.html
[Pazanni, 1999] M. Pazanni (1999).A framework for collaborative, content-based and demographic filtering. *Artificial intelligence review*. URL: htp://citeseer.nj.nec.com/pazanni99framework.html
[Pazanni and Billsus, 1997] M. Pazanni, D. Billsus (1997).Learning and revising user profiles: the identification of interesting web sites. *Machine learning* nr. 3 pp. 313-331. URL: htp:citeseer.nj.nec.com/pazanni97learning.html
[**Pretschner, 1998**] A. Pretschner(1998). *Ontology Based Personalized Search.* URL:citeseer.nj.nec.com/pretschner99ontology.html
[**RDF-M&S, 1999**] W3C RDF workgroup (1999). *Resource Description Framework (RDF) Model and Syntax Specification*. URL:http://www.w3.org/TR/REC-rdf-syntax/
[**RDF-schema, 2000**] W3C RDF workgroup (2000). *Resource Description Framework (RDF) Schema Specification 1.0*. URL:http://www.w3.org/TR/rdf-schema/
[**Sarwar et al., 2000**] B. Sarwar and G. Karypis and J. Konstan and J. Riedl (2000). Analysis of recommendation algorithms for e-commerce. *ACM Conference on Electronic Commerce* p. 158-167. URL:http://citeseer.nj.nec.com/sarwar00analysis.html
[**Sarwar et al., 2001**] Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. (2001). Item-based Collaborative Filtering Recommender Algorithms. *the WWW10 Conference. May, 2001*. URL:http://citeseer.nj.nec.com/386002.html
[**Shardanand&Maes, 1995**] U. Shardanand and P. Maes (1995). Social Information Filtering: Algorithms for Automating ``Word of Mouth. *Proceedings of ACM CHI '95 Conference on Human Factors in Computing Systems* Vol. 1 p. 210-217. URL:citeseer.nj.nec.com/upendra95social.html
[**Sheth, 1994**] Beerud D. Sheth(1994). *A learning approach to personalized information filtering.* URL:http://citeseer.nj.nec.com/112107
[**Ungar&Foster, 1998**] L. Ungar and D. Foster (1998). Clustering Methods For Collaborative Filtering. *Proceedings of the Workshop on Recommendation Systems. AAAI Press, Menlo Park California*. URL:http://citeseer.nj.nec.com/ungar98clustering.html
[**Widyantoro et al., 2001**] D.H. Widyantoro, T.R. Ioerger, and J. Yen (2001). Learning user interest dynamics with a three-descriptor representation. . URL:

Web References

[**XML-DATATYPES**] W3C XML workgroup (). *XML Schema Part 2: Datatypes - W3C Recommendation* . URL:http://www.w3.org/TR/xmlschema-2/
[**DCMI**] *Dublin Core Metadata Initiative*, URL:http://dublincore.org/
[**FREEDB**] *freedb project - homepage*, URL:http://www.freedb.org/
[**GDBM**] *Gnu Database Manager - home page*, URL:http://www.gnu.org/directory/gdbm.html
[**Gerhard**] *Gerhard project - homepage*, URL:http://www.gerhard.de
[**GNU-GPL**] *GNU General Public License*, URL:http://www.gnu.org/copyleft/gpl.html
[**Greenstone**] *Greenstone Project - home page*, URL:http://greenstone.sourceforge.org/
[**MG**] *Managing Giga Bytes - home page*, URL:http://www.cs.mu.oz.au/mg
[**NZDL**] *New Zealand Digital Library Project - home page*, URL:http://www.nzdl.org/
[**PG**] *Project Gutenberg - home page*, URL:http://promo.net/pg/
[**RDF-NS**] *core RDF namespace*, URL:http://www.w3.org/1999/02/22-rdf-syntax-ns#
[**RDF-SCHEMA-NS**] *rdf-schema namespace*, URL:http://www.w3.org/2000/01/rdf-schema#
[**SCHEMA-ARCH**] *The Cambridge Communiqué*, URL:http://www.w3.org/TR/schema-arch
[**Scorpion**] *Scorpion project - homepage*, URL:http://orc.rsch.oclc.org:6109/
[**URI**] *W3C page - Naming and Addressing: URIs, URLs, ...*, URL:http://www.w3.org/Addressing/
[**W3C**] *W3C -home page*, URL:http://www.w3.org/