GMD Research Series

GMD –
Forschungszentrum
Informationstechnik
GmbH

Patrick Baudisch

Dynamic Information Filtering

N° 16/2001

GMD – Forschungszentrum Informationstechnik GmbH
Schloß Birlinghoven
D-53754 Sankt Augustin
Germany
Telefon +49 -2241 -14 -0
Telefax +49 -2241 -14 -2618
http://www.gmd.de

Anschrift des Verfassers/Address of the author:
Patrick Baudisch
Institut für Integrierte Publikations- und Informationssysteme
GMD – Forschungszentrum Informationstechnik GmbH
Dolivostraße 15
D-64293 Darmstadt

Die Deutsche Bibliothek - CIP-Einheitsaufnahme:

Baudisch, Patrick:
Dynamic Information Filtering / Patrick Baudisch.
GMD — Forschungszentrum Informationstechnik GmbH. - Sankt Augustin :
GMD — Forschungszentrum Informationstechnik, 2001
(GMD Research Series ; 2001, No. 16)
Zugl.: Darmstadt, Techn. Univ., Diss., 2001
ISBN 3-88457-399-3

ISSN 1435-2699
ISBN 3-88457-399-3

Dynamic Information filtering

Dem Fachbereich Informatik
der Technischen Universität Darmstadt
zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
vorgelegte

Dissertation

von
Diplom-Informatiker
Patrick Baudisch
geboren am 5 Dezember 1967 in Nürnberg

Referent: Prof. Dr. Erich J. Neuhold
Korreferent: Prof. Dr. Joseph A. Konstan

Einreichungstermin: 11. Februar 2001
Prüfungstermin: 26. März 2001
Hochschulkennziffer D 17

ABSTRACT

Keywords: Information Filtering, User Profile, Interest Shift, QuerySet Architecture, User Interface, Histogram-based Interface, Paintable Interface, Collaborative Filtering, TV Program Recommendation.

The goal of information filtering systems (IF systems) is to support users in finding relevant information from a dynamic base of data objects. IF systems base their relevance computations on so-called *user profiles* in which they maintain representations of the users' interests. Whenever users' interests change, user profiles become inaccurate and the prediction quality decreases. The more dynamic user interests are, the more important it is for IF systems to update user profiles rapidly to shorten this period of decreased prediction quality. A survey of existing systems, however, shows that they typically provide support only for selected types of interest changes, namely either for gradual *or* for abrupt interest changes.

The objective of this dissertation is to design an information filtering architecture capable of dealing with frequently changing user interests. To make profile updating more efficient, our *QuerySet* filtering architecture (QSA) uses user profiles of a modular structure. QSA profiles primarily consist of a set of queries that are each expected to have a high correspondence with one of the user's interests. Interest changes affecting only a subset of the user's interests can then be handled efficiently by updating only the queries representing the affected interests. Complementing the queries, QSA profiles contain an aggregation function that aggregates the ratings that the individual queries assign to an object. This resulting rating is used to rank filtered objects. The aggregation function allows for especially efficient profile updating whenever the relative relevance of individual queries changes.

To allow handling both gradual and abrupt interest changes, QSA profiles are provided with two distinct profile updating mechanisms. First, QSA systems can track gradual interest changes by learning from relevance feedback. Second, *user-aggregated relevance feedback* (URF) provides users with direct access to the aggregation function and allows them to manually handle major profile inaccuracies, as they occur during profile initialization and after major interest changes. URF is a rating that users assign not to a single object, but to a *set* of objects and that seamlessly integrates with relevance feedback in the profile updating process. We present a framework of specialized user interface families that allow entering URF. The individual interfaces are optimized for three different purposes, i.e. simplest learnability (*form-based interfaces*), maximum accuracy (*histogram-based interfaces*), and maximum efficiency (*paintable interfaces*). We present an experimental evaluation of the interfaces.

We demonstrate the QuerySet Architecture at the example of our TV program recommendation system *TV Scout*. The system contains multiple subsystems using different content-based and collaborative filtering techniques and allows users to combine these techniques in their profiles. Additional subsystems can be plugged in at any time to "assimilate" more query functionality. The TV Scout is currently in commercial, Internet-wide use. While the QuerySet Architecture proved very successful in the TV application area, we believe that it will prove equally useful for a number of other highly dynamic application areas, such as, for example, news, web pages, and Internet auctions.

KURZFASSUNG

Schlagworte: Informationsfilterung, Benutzerprofil, Interessensänderung, QuerySet Architektur, Benutzerschnittstelle, Histogrammbasierte Benutzerschnittstelle, Mal-basierte Benutzerschnittstelle, Kollaboratives Filtern, TV-Programmempfehlung.

Das Ziel von Informationsfilterungssystemen (IF-Systemen) ist es, Benutzer beim Finden relevanter Informationen aus einer dynamischen Informationsquelle zu unterstützen. IF-Systeme stützen ihre Relevanzberechnungen auf sogenannte Benutzerprofile, in denen Repräsentationen der Benutzerinteressen gespeichert werden. Wann immer sich die Interessen der Benutzer ändern, werden die Benutzerprofile unzutreffend und die Qualität der Filterergebnisse sinkt. Je veränderlicher Benutzerinteressen sind, umso wichtiger ist es für IF-Systeme, Benutzerprofile schnell zu aktualisieren, um so den Zeitabschnitt verminderter Filterqualität zu begrenzen. Eine Übersicht über existierende Systeme zeigt jedoch, dass IF-Systeme typischerweise nur eine Untermenge möglicher Interessensänderungen unterstützen, nämlich entweder nur graduelle *oder* nur abrupte Interessensänderungen.

Der Gegenstand dieser Dissertation ist eine Informationsfilterungsarchitektur, die für die Bewältigung häufiger Interessensänderungen ausgelegt ist. Um die Aktualisierung von Benutzerprofilen effizienter zu machen, benutzt die hier vorgestellte *QuerySet* Architektur (QSA) eine modulare Benutzerprofilstruktur. QSA-Benutzerprofile bestehen in erster Linie aus einer Menge von Suchanfragen, die jeweils genau ein Interesse des Benutzers repräsentieren. Interessensänderungen, die auf eine Untermenge der Benutzerinteressen beschränkt sind, können so besonders effizient behandelt werden, indem nur die betroffenen Suchanfragen aktualisiert werden.

Ergänzend zu den Suchanfragen enthalten QSA Profile eine Aggregierungsfunktion, die die Einzelbewertungen eines Objektes durch die einzelnen Suchanfragen zu einer einzigen Gesamtbewertung zusammenfasst. Die Gesamtbewertung dient dazu, die gefilterten Objekte in die Rangfolge zu bringen, in der die Objekte dem Benutzer präsentiert werden. Die Aggregierungsfunktion erlaubt eine besonders effiziente Aktualisierung von Benutzerprofilen, wann immer sich die relative Gewichtung der einzelnen Suchanfragen ändert.

Um sowohl graduelle als auch abrupte Interessensänderungen zu unterstützen, stellt die QuerySet Architektur zwei verschiedene Methoden zur Aktualisierung von Benutzerprofile zur Verfügung. Erstens können QSA Systeme graduelle Interessensänderungen mitverfolgen, in dem sie aus Relevanzrückmeldungen des Benutzers lernen. Zweitens wird Benutzern ein direkter Zugang zu ihren Profilen zur Verfügung gestellt. Durch Eingabe sogenannter benutzeraugregierter Relevanzrückmeldungen (BR) können Benutzer größere Profilabweichungen, wie sie während der Profilerstellung und nach gravierenden Interessensänderungen auftreten, manuell beheben. BR sind Bewertungen mit denen Benutzer nicht einzelne Objekte bewerten, sondern ganze Mengen von Objekten. In der vorliegenden Arbeit wird ein Baukasten von Werkzeugen für die Eingabe von BR vorgestellt. Die einzelnen Werkzeuge sind für jeweils einen von drei Zielsetzungen optimiert, und zwar Erlernbarkeit (formularbasierte Werkzeuge), Präzision (histogrammbasierte Werkzeuge) oder Effizienz (malbasierte Werkzeuge). Die Benutzbarkeit der Werkzeuge wurde experimentell verifiziert.

Die Anwendbarkeit der QuerySet Architektur wurde am Beispiel des TV Empfehlungsprogramms *TV Scout* überprüft. Der TV Scout enthält verschiedene inhalts- und kollaborationsbasierte Filtersubsysteme und erlaubt es Benutzern, diese Techniken in ihren Profilen zu kombinieren. Weitere Subsysteme können jederzeit hinzugefügt werden, so dass der TV Scout jederzeit um zusätzliche Filterfunktionalität erweitert werden kann. Der TV Scout ist zur Zeit in kommerzieller, internetweiter Benutzung. Während die QuerySet Architektur sich in dem Anwendungsgebiet Fernsehen bewährt hat, lassen ihre Fähigkeit zur Verfolgung von Interessensänderungen erwarten, dass sie sich auch in einer Reihe anderen Anwendungsgebiete, wie zum Beispiel Nachrichten, Webseiten, oder Internetauktionen, wertvoll erweisen wird.

ACKNOWLEDGEMENTS

I thank my advisors Erich J. Neuhold and Joseph A. Konstan for their support. I thank Dieter Böcker, Matthias Hemmje, and Ulrich Thiel for their valuable advice, as well as my colleagues at GMD Delite, especially Marcus Frühwein who pointed me to collaborative filtering. Thanks to Karl Aberer and Thomas Tesch for pointing my attention to utility theory, to Shirley Holst for her comments on the user interface experiments, and to Peter Spellucci for his comments on fitting algorithms.

I thank all my students from the TV Scout project team, especially Lars and Michael Brückner, Gerrit Voss, Nicole Freyler, Andreas Brügelmann, Claudia Perlich, Robert Werner, Matthias Eilers, Armin Ruhland, Christian Valentin, Martin Koehler, Thorsten Heller, Jérôme Dugué, Arnaud Gilbert, Christine Weidmann, Martina Bunte, Metin Goektay, Marlon Mansourian, Gundolf von Bachhaus, Marko Goerg, Christoph Felck, and Wolfgang Schmidt.

Thanks to all those researchers who gave valuable feedback during my talks, in particular John Riedl, Nathan Good, Ben Shneiderman, Doug Oard, Dagobert Soergel, Paul Moody, Arnd Steinmetz, Hannes Marais, Andreas Girgensohn, Gene Golovchinsky, Mark Stefik, and Jock MackInlay.

Finally, thanks also to the members of the TV-TODAY network team, above all Tom Stölting. Thanks to Maria Heckenbach, Ursula Bernhard, and Barbara Lutes for their support. Special thanks to Jenny, who helped me to remember to keep things small and simple.

Dedicated to E.H. Schubert.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

> We travel a narrow road toward our goal with a sea of seductive information to distract us on one side and a spiraling abyss of confusion and information overload on the other [Mar95].

Many people today suffer from information overload. There are more people and institutions; they engage in more formal relationships and exchange; increasingly more sources and larger volumes are available [Mar95]. There are often more research papers, books, movies, and even TV programs than people can handle effectively. With the Internet, information overload has reached a new dimension. The goal of information access systems is to support users in finding the information relevant to them and to free them from having to sift through masses of irrelevant information. Thereby, these systems help alleviate the user's information overload problem.

Historically, two analytical information seeking strategies (also called *information access* [GM99]) have been distinguished, namely *information retrieval* and *information filtering* ([Mar95], see Section 1.1.1.1). As shown in Figure 1, the two approaches can be considered special cases of information access. Information retrieval systems aim at supporting users with changing interests (high information need change rate), but under the assumption that the underlying object database is rather stable (low information source change rate)[1]. Information filtering systems, on the other hand, are designed to support information access to highly dynamic information sources (high information source change rate), but in this case the assumption is made that the user's interests are rather stable over time (low information need change rate).

Many information access situations fulfill the respective assumptions close enough to be handled successfully either by information retrieval or by information filtering systems. The user's interests in movies or music, for example, is often stable over years, so that information filtering systems can handle this information access problem appropriately. More task-related application areas, e.g. electronic dictionaries, are good application areas for information retrieval techniques.

---

[1] Alternatively, the information need may be of so short a duration that the information source dynamics is irrelevant.

Other application areas fit into neither of the two niches. These application areas have an information source change rate *and* an information need change rate that are both too high to be neglected. These application areas are especially difficult to handle and can therefore be considered the "grand challenge" in information access systems (see Figure 1, [OM96]). One such application area is, for example, the stock market, where both the market and the broker's interests change constantly. We will refer to systems dealing with applications areas characterized by highly dynamic information access as *dynamic information filtering systems*.

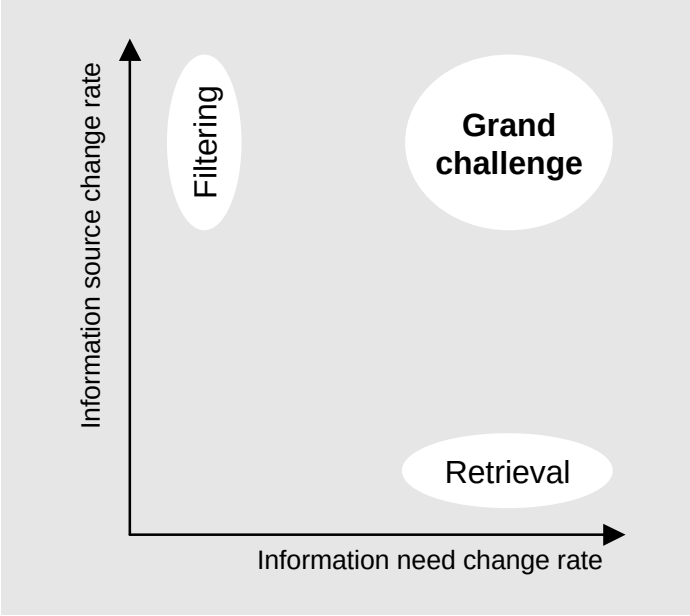


Figure 1: Dynamic information filtering—the grand challenge [OM96].

For such applications, maximum adaptation speed is crucial. Only rapid adaptation allows representing the user's interests accurately enough to make sufficiently accurate predictions before the user's next interest change invalidates the system's current representation of the user's interests (the *user profiles*, see Section 1.1.1.3). In this dissertation, we will tackle this challenge. Our goal will be to design an information filtering system architecture capable of dealing with rapidly changing user interests. The approach that we will present is based on the assumption that the adaptation capability of IF systems can be improved if user interests are not only tracked based on relevance feedback, but if users are also provided with a direct high-level access to their profiles via appropriate user interfaces.

Because of their generic nature, services providing news-related content, e.g. Usenet newsgroups, the Web, or TV, support an especially rich variety of interests and consequently, these application areas show a particularly rich variety of interest changes. We will therefore focus on these application areas. To be consistent with the TV Scout system presented in Chapter 5, we will place all our examples into the TV domain.

1.1 BASIC CONCEPTS

In the following, we give a brief introduction to information filtering. Additional information about this topic can be found in several surveys, e.g. [BC92, LT92, OM96, Oar97]. The related field of information retrieval is described in several books, e.g. [vRi79, Sal83, Bla90, Ing92, FB92, GF98]. A broader picture of information seeking in electronic environments can be found in [Mar95].

1.1.1 Information filtering

The term *information filtering* (IF) [Den82] has been used to describe a variety of processes involving the *delivery of information to users* [BC92, SM93]. Information filtering is an information seeking process in which documents are selected from a *stream of incoming data* to satisfy a relatively *stable and specific information need* [OK98]. The goal of an information filtering system is to sift through large volumes of dynamically generated information and to present users with information likely to satisfy their information requirement [OM96].

Information filtering is related to processes such as routing (with a heritage in message processing), categorization, and extraction [BC92], as well as “current awareness” [Leg75, PS79], as “data mining” [OM96], and as “Selective Dissemination of Information” (SDI) [Luh58, CSP69, Hou69, Sal68, Lan78, YG94, YG95]. The original SDI methods were manual alerting services that brought new information to the attention of users of research and special libraries [OM96]. SDI is sometimes used to imply that the profiles which describe the information need are constructed manually [OM96].

Information filtering systems have been referred to as time saving devices [Bac91]. If users had an infinite amount of time to handle all information intake information filtering would not be necessary. The goal of information filtering is the prioritizing of information, which assists the direction of human attention. Prioritizing can take the form of highlighting items of high importance or deleting items that are not considered relevant. Even though the term filtering has a literal connotation of leaving things out, we use it here in a more general sense that includes selecting things from a larger set of possibilities [MGT+87, p. 390].

1.1.1.1 Information filtering versus related research areas

IF is similar to information retrieval (IR) in several ways [Koc74, Sal83]. Both approaches involve the same basic components, the same flows of information, and can be accomplished with a very similar architecture [BC92]. The difference is that in IR a request is made to a system as a one-off occurrence and searched against the current collection of documents; in IF repeated searches, often with the same query, are made against successive additions to the collection, over a period of time [Rob81]. IF is typically concerned with *repeated uses* of the system, by a person or persons with *long-term goals or interests*, unlike a typical information retrieval system [SM93].

If one considers IF and IR from a very general viewpoint, IF is a special case in which the information space is very dynamic (high information source change rate). IR involves selec-

tion of relatively static information in response to relatively dynamic queries (high information need change rate). Therefore, IR can be viewed as the dual problem to IF [OM96] (see also Figure 1). The case where both information source change rate and information need change rate are high will be called *dynamic information filtering* and will be subject of this dissertation. We will use the term *Information Access* (IA) for referring to both IF and IR [GM99]. Information access is a special case of information seeking [Mar95], which not only includes the analytic strategies such as retrieval and filtering, but also opportunistic strategies, such as browsing [Hoh94, Mar95, CCG95].

IF and IR systems are information systems designed for unstructured or semi-structured data (e.g. text documents). This contrasts with a typical database application that involves highly structured data, such as employee records [BC92]. The distinguishing feature of the database retrieval process is that the output will *be* information, while in information filtering (or retrieval), the output is a set of entities (e.g. documents) which *contain* the information to be sought [Bla90]. Consequently, IF processing involves additional processes such as representation issues (see also Figure 2, page 7). The result of database searches may, however, be used as input to a filtering process [SM93].

Similar problems that IA systems face have been tackled in the context of utility theory (UT) [vNe47, KR93]. The basic question UT tries to answer is “Given a set of alternatives, which alternative should I choose?” A decision analysis proceeds by assessing the so-called *utility function* of a person, and by using that function to assign a *utility* to each alternative, thereby determining the decision. Since IA problems can also be understood as choosing between alternatives (with the alternatives being data objects, such as documents), and since UT also provides methods for dealing with uncertainty, IA systems can also be examined from the point of view of utility theory. Unlike IF, that is designed for the automatic filtering of masses of objects in a limited amount of time, the problems handled using utility theory are usually characterized by providing a limited number of alternatives (often only very few of them) and the availability of the decision maker for interactive questioning. Consequently, not all techniques of UT are directly applicable to IA. However, some authors have made attempts to apply concepts from UT for the creation of IF systems, e.g. the Decision Theoretic Video Advisor [NH98].

IF techniques are applied to a wide variety of data types, including multimedia objects [Ste99, CS94]. Therefore, we will use the more general term *object* instead of the term *document* when referring to the data objects delivered to the user.

1.1.1.2 Information needs and quality measures

IF and IR systems try to satisfy their users’ so-called *information needs* that are also called *interests*. Many suggestions have been made for motives of media access [All90]. Among the main factors which have been proposed are goal satisfaction [BGT87], the social importance of media [Ste67], as well as entertainment and play [KBG74]. Several models have been proposed defining information needs and their creation. In this dissertation, we adopt the model by Belkin [BC87]. In this model, a person with some goals and intentions related to, for instance, a work task, finds that these goals cannot be attained because the person’s resources or

knowledge are somehow inadequate. A characteristic of such a "problematic situation" [SL73] is an *anomalous state of knowledge* [BC87].

Different researchers have proposed slightly different classifications of information needs. Mizzaro distinguishes between four states of information needs [Miz96]. The initial phase, i.e. the anomalous state of knowledge, is called *problem*. A problem turns into an *information need* when the person becomes aware of it. Expressing the information need turns it into a *request*. Formalizing the request creates a *query*. These states are also referred to as real, perceived, expressed, and formalized information need [GM99]. Because of the temporal distance between the emergence of an information need and the satisfaction of that need, users can (and usually will) have several distinct information needs at the same time.

The performance of information access (IA) systems is difficult to measure. Since it is inherently difficult to measure in how far an information need has been satisfied by the delivered objects, performance is usually measured in how far delivered objects are *judged relevant by the user* [SM83]. Relevance [Fro94, SEN90, Sch94, Sar75] is one of the most debated and central concepts in information science. For an exhausting survey on relevance see [Miz96]. Classical IR research defines relevance mostly as topicality (e.g. the text retrieval conference competition [TREC, Har94]). Other researchers also take task-oriented utility (also called informativeness [Boy82]) into account [Boy82, Soe94, Har92, GL91, Coo73a, Coo73b, Coo78, Bar67]. Because of its subjective nature, task-oriented utility is harder to grasp and to express in a query. While topicality is often measured on a zero-to-one scale with zero and one denoting *non-relevant* and *fully relevant*, utility oriented relevance may use different, e.g. open-ended scales. See [GM99] for a detailed framework of relevance notions.

The goal of an IA system is to predict the relevance of objects, as the user would assign it, and to deliver only relevant objects to the user. Relevance predictions by the system were initially of Boolean data type (exact match methods), today many systems use gradual relevance values (best match methods, see Section 1.1.2). Probabilistic models acknowledge that there are degrees of relevance and are based on some estimated probability of document-query relevance [Mar95, p. 25]. Some authors have proposed the usage of multidimensional relevance, but these systems do not provide a total ordering of the retrieved documents and therefore cannot produce one single, meaningfully ranked output [GM99].

The most common measure for the performance of IA systems is the combination of precision and recall. Precision is defined as the number of relevant objects delivered divided by the total number of delivered objects. Recall is defined as the number of relevant objects delivered divided by the total number of relevant objects [SM83]. See [vRi79, Swe88] for other measures of retrieval quality.

1.1.1.3 User profiles

We will use the term *relevance function* to refer to the ensemble of all information needs of a user. The relevance function is a map from a space of objects to the space of real-valued relevance. If D denotes the space of objects, the relevance function is a map $r: D \rightarrow \mathbb{R}$, such that $r(x)$ corresponds to the relevance of object x. Users' relevance functions are not known in advance, and can also change over time [LMMP96].

In an IF system, the system's representation of users' relevance functions, i.e. of an individual user or a group of users, is commonly referred to as *user profile* or just *profile* [Tay68, BOB82, Ing82, Ing86]. The problem of IF may be described as the attempt to represent the user's relevance function as a user profile, i.e. to obtain a state of the user profile, such that it correctly represents the relevance of each object. If such a profile is given for all objects in the space of objects D, a finite set of objects can always be rank-ordered and presented in a prioritized fashion to the user [LMMP96].

SDI systems sometimes maintain multiple profile entities per user, with each of these entities representing only *one interest* of a user [OM96]. Within the context of SDI, these entities are also referred to as user profiles. Because this type of profile fills the same role as what is commonly called a *query* in information retrieval and database systems (with the difference that they are stored and repeatedly evaluated over longer time), we will use the term *query* instead of *profile* to refer to these entities (see also [OM96, GNOT92]). We will reserve the term *user profile* for the entity representing *all* information the system has collected about a user's interests. Consequently, in the context of SDI systems, we will use the term user profile to refer to the entire collection of all queries of a given user.

The term user profile, as used in this dissertation, is different from its usage in the context of the related research topic "query enhancement by user profiles" [Liu82, Kor84, Mya87]. The basic idea of this track of work is to enhance a query in an information retrieval situation with keywords collected during past uses of the system (the *user profile*) to help avoid unwanted additional meanings of the search terms in the queries. The main difference between this field of research and IF is that user profiles are not used to represent current information needs, but only to provide a context.

User profiles are also subject to the more general discipline of user modeling (e.g. [Ing92, p. 157-202]), where they are usually referred to as *user models*. It would not be technically correct to call IF user profiles a *user model* because a user model consists of both a *representation* of the user's interests and a method for *interpreting* that representation for making predictions. Whenever we use the term user profile in this dissertation, we will refer to the limited definition within IF.

1.1.2 Approaches to Information filtering

Information filtering techniques have been applied to several application areas including scientific publications [Luh58], technical memos and reports [FD92, FC97], Usenet news [FS91, Bac91, SK92, Ste92b, Bac92, JH92, SM93, Mae94, KHL+94, RIS+94, MS94, Lan95, YG95, Moc96, MRK97, MRK+97], electronic mail [Mye80, MGT+87, Pol88, GNOT92, Ter91, Ter93, LMM94], books [Ric79b, MR98], application program know how [LN98], the finding of experts [SW93, KSS96], Web pages [RM96, HT96, Bal97, THA+97, PB97, RP97, Bie98], classified ads [GGKS95], movies [Kay95, HRF95, AKK97, AKK98], music [Sha94, SM95, Loe92], and TV [EHWS96, DtHh98, Bau96]. In Chapter 2, we will discuss some of them in more detail.

Figure 2a shows the general architecture of IF systems, as proposed by Belkin and Croft [BC92]. The diagram consists of three blocks, consisting of four steps each. The three blocks

are surrogate creation (top left), profile creation (top right), and filtering/refinement process (bottom). Figure 2b to Figure 2d illustrate how profiles are maintained, i.e. profile creation (b), interest changes (c), and refinement of profiles (d).

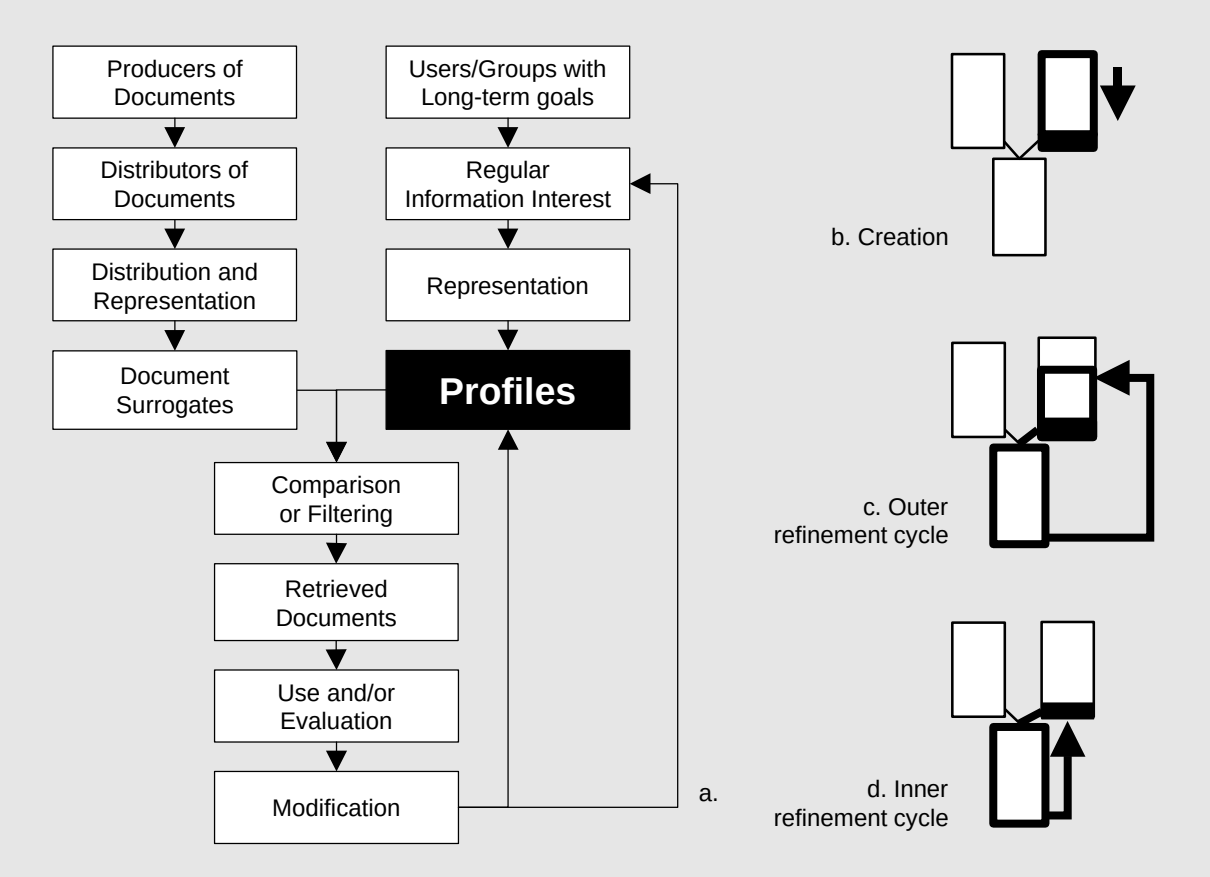


Figure 2: The information filtering model proposed by Belkin and Croft [BC92]

To make document surrogates and user profiles comparable, both are usually reduced to a common set of so-called *attributes*, sometimes also called *features*. Attributes map objects to a common representation providing a distance measure, typically real numbers. Different approaches to IF can be distinguished by the following properties. 1) Which attributes are used to compare surrogates and profile and how is similarity determined? 2) Who or what assigns these attributes to the objects (indexing)? 3) Who or what assigns these attributes to the users (profile generation)?

Malone [MGT+87] identified three approaches to IF. In cognitive filtering, also referred to as content-based filtering, attributes are usually extracted automatically from the objects. In social filtering, also called collaborative filtering, attributes are annotations and endorsements of objects by users. In economic filtering, attributes are additional incentives, such as credits attached to an object by its creator.

Economic filtering was developed in the application area of electronic mail. The basic idea behind it is to shift some of the costs of reading a message from the receiver to the senders

with the goal to avoid junk mail [Den82]. The economic filtering approach relies on various kinds of cost-benefit assessments and explicit or implicit pricing mechanisms [MGT+87]. Economic filtering concepts are implemented using content-based or collaborative filtering techniques, so that the main *technical* distinction is between content-based and collaborative filtering techniques. We will briefly compare these approaches in the following, as far as relevant to this thesis.

1.1.2.1 Content-based information filtering

Content-based approaches work by characterizing the contents of a message [MGT+87]. Representations in content-based filtering systems exploit only information that can be derived from the objects' contents [OM96].

The content-based approach to information filtering has its roots in the information retrieval (IR) community and employs many of its techniques [Bal97]. The most prominent example of content-based filtering is the filtering of text objects (e.g. mail messages, newsgroup postings, or Web pages) based on the words contained in their textual representations. Each object, here text documents, is assigned one or more index terms selected to represent the best meaning of the document. These index terms are then searched to locate documents related to queries expressed in words taken from the index language. The assumption underlying this form of filtering is that the "meanings" of objects and queries can be and are captured in specific words or phrases [Mar95].

The three most prominent retrieval models [BC92] are the Boolean (e.g. [Bla90]), the vector space [Sal68, Sal71, JF87], and the probabilistic retrieval models [TC90]. The first of these is based on what is called the "exact match" principle; the other two on the concept of "best match". For surveys on these and other retrieval models see [Sal83, BC87].

Boolean retrieval is based on the concept of an exact match of a query with one or more text surrogates. The term "Boolean" is used because the query specifications are expressed as words or phrases that are combined using the standard operators of Boolean logic. In this retrieval model, all surrogates, or generally, texts, containing the combination of words or phrases specified in the query are retrieved, and there is no distinction made between any of the retrieved documents. Thus, the result of the comparison operation in Boolean retrieval is a partition of the database into a set of retrieved documents, and a set of not-retrieved documents [BC92]. The name Boolean retrieval can be misleading. It is important to distinguish between the use of Boolean operators in queries (which does not imply an exact-match model) and the use of Boolean logic as the interpretation of those operators [TC92]. Below, we will present two other approaches that use Boolean syntax without being exact-match models. One historical application area of Boolean retrieval is library search, where the retrieved documents are, for example, presented in lexicographical order.

A major problem with the Boolean retrieval model is that is does not allow for any form of relevance ranking of the retrieved document set, although some objects are more likely to be relevant or are more relevant to an information need than others. Excluding documents that do not precisely match a query specification results in lower effectiveness [Sal83, TC91]. Therefore, the Boolean model is considered too weak for large text collections [TC92].

Best-match retrieval models have been proposed in response to the problems of exact-match retrieval. These systems are based on the assumption that presenting documents to the user in order of presumed relevance results in more effective and usable systems. If the system has produced a good rank ordering, the density of useful documents should be greatest near the top of the list. The probability ranking principle [Rob77] describes how to optimize this assumption. It states that if a retrieval system's response to each request is a ranking of the documents in the order of decreasing probability of relevance then precision and recall will be maximized at any cut-off. Best match methods give users control over the size of the output and thereby assist in managing large result sets [Mar95, Bla90].

1.1.2.1.1 The vector space model

The most widely known best-match retrieval model is the vector-space model [Sal83]. This model treats text representations of objects *and* queries as vectors in a multi-dimensional space, the dimensions of which are the words used to represent the texts. Queries and texts are compared by comparing the vectors, using, for example, the cosine similarity measure. The assumption is that the more similar a vector representing a text is to a query vector, the more likely it is that the text is relevant to that query. In this model, an important refinement over exact match retrieval models is that the terms (or dimensions) of a query, or text representation, can be weighted, to take account of their importance. These weights can be computed on the basis of the statistical distributions of the terms in the database, and in the texts (e.g. using *term frequency* * *inverse document frequency* weighting (TF-IDF) [Sal83]). Many web search engines use the vector space model or retrieval model derived from it [Fel98].

The absence of Boolean operators limits the vector space model. It does not provide a means to control how the *combination* of query terms maps to relevance. This combination of terms is, for example, different if two terms are synonyms compared to if they form a concept (queries using Boolean syntax may use **or** or **and** connectives to express the respective relations). Consequently, the *extended vector space model* introduces **and**(p) and **or**(p) to enhance the original vector space model [SM83]. Varying the value of p allows modeling Boolean connectives ($p=\infty$), the original vector space operator ($p=1$), and gradual transition between the above ($1<p<\infty$). Operators with different p-values can be mixed freely in a query. As in the original vector space model, term weights can be adjusted either manually or using TF-IDF weighting.

1.1.2.1.2 Probabilistic models / inference networks

Probabilistic information retrieval models are based on the probability ranking principle, i.e. they try to rank objects in the order of their probability of relevance to the query, given all the evidence available [Rob77, vRi77, vRi80, SW80, RMC82, FB90]. The most typical source of such evidence is the statistical distribution of terms in the database, and in relevant and non-relevant texts. The principle takes into account that the representation of both information need and text is uncertain, and the relevance relationship between them is also uncertain.

Inference networks are one possible implementation of probabilistic models [TC90, TC91, BC92]. Since inference networks can be used for implementing parts of the information filtering architecture that is the subject of this thesis (see Section **3.3.3**), we will present this approach in some additional detail. Figure 3 shows the structure of an inference network consist-

ing of four layers of nodes. In this network, each document is represented as a document node (d-nodes). During indexing, each document are assigned one or more concepts (r-nodes). Concepts can match one or more queries (q-nodes). Queries, finally, can represent the user's interest (I-node). Edges between nodes represent probabilistic dependencies. An edge from a node A to a node B is assigned the conditional probability p(B|A). An edge from a document node to a concept node, for example, represents the probability that the document contains information about this concept. A document's relevance is computed as the probability of this document to be relevant to the interest, i.e. $p(I|d_i)$.

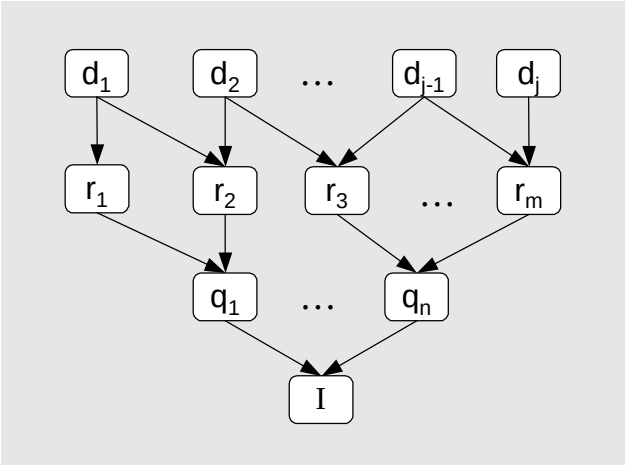


Figure 3: Example of an inference network consisting of 4 layers [TC92, p. 282]. d = document nodes, r = concept representation nodes, q = query representation nodes, I = Interest node.

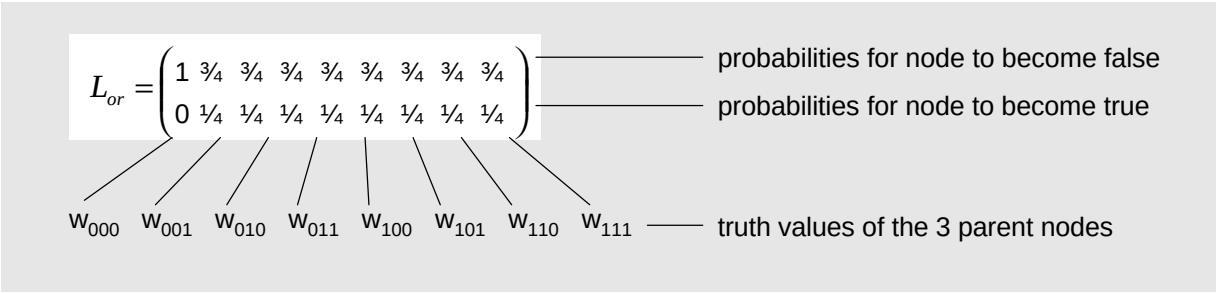


Figure 4: Link matrix implementing a weighted **or** connective. Values in the matrix are probabilities that the node takes the state determined by the row if parent nodes have the states determined by the column.

Since nodes are often connected to multiple parent nodes, they have to contain functions for computing the probability of the child node based on the probabilities of all these parent nodes. For this purpose, so-called *link matrices* can be used [TC92, p. 283]. Figure 4 shows an example of such a link matrix. The top row of a link matrix contains the probabilities that the node takes on the state *false* depending on the states of the parent nodes; the bottom row con-

tains the probabilities that the node takes on the state *true*. Columns enumerate all combinations of parent node states. A truth value of "011", i.e. the forth column of the shown matrix, for example, means that the first parent node is assumed to be false, while the other two parent nodes are true. The example shown in Figure 4 is the link matrix of a weighted **or** operator. This bottom row of the matrix says that the child node is assigned a probability of ¼ in the cases that any of the parents is true (w_{001} through w_{111}). Only in the case that all parent nodes are false (w_{000}), child node is known to take on the value false (single 1 in the top row).

Link matrices define node probabilities in dependency of parent states that are certain, i.e. parent node probabilities of 0 or 1, not on probabilities in between. Node states with uncertain parent states are defined using the definition of conditional probabilities, i.e. they are interpolated by overlaying functions defined by products of x_i, and $(1- x_i)$. The weighted **or** operator in Figure 4, for example, is thereby defined as P(Q = true) = ¼ (1-(1-a)(1-b)(1-c)), with parent probabilities *a*, *b*, and *c*.

Figure 5 illustrates the interpolation function by the example of two parent nodes. Figure 5a shows the link matrix of such a child node. The probabilities of the 2 parent nodes span a 2-dimensional space and the values w_{00} to w_{11} from the bottom row of the link matrix assign probability values to each of its four vertices (Figure 5a). Figure 5c and d use brightness to illustrate how the probabilities for the child node are interpolated (brightness values rounded to distinct color steps to show "equi-rating lines" as used by [SM83]). The overlaid functions shown in Figure 5c are $(1-x_1)(1-x_2)$, $x_1(1-x_2)$, $(1-x_1)x_2$, and x_1x_2, which assures a smooth interpolation between any combination of vertex probabilities. Figure 5d shows a selection of node functions that are obtained by weighting the interpolations with the respective factors from the link matrix w_{00} to w_{11} and summing them up. The shown node functions are identity 2, weighted sum, equality, tautology, and an arbitrary function.

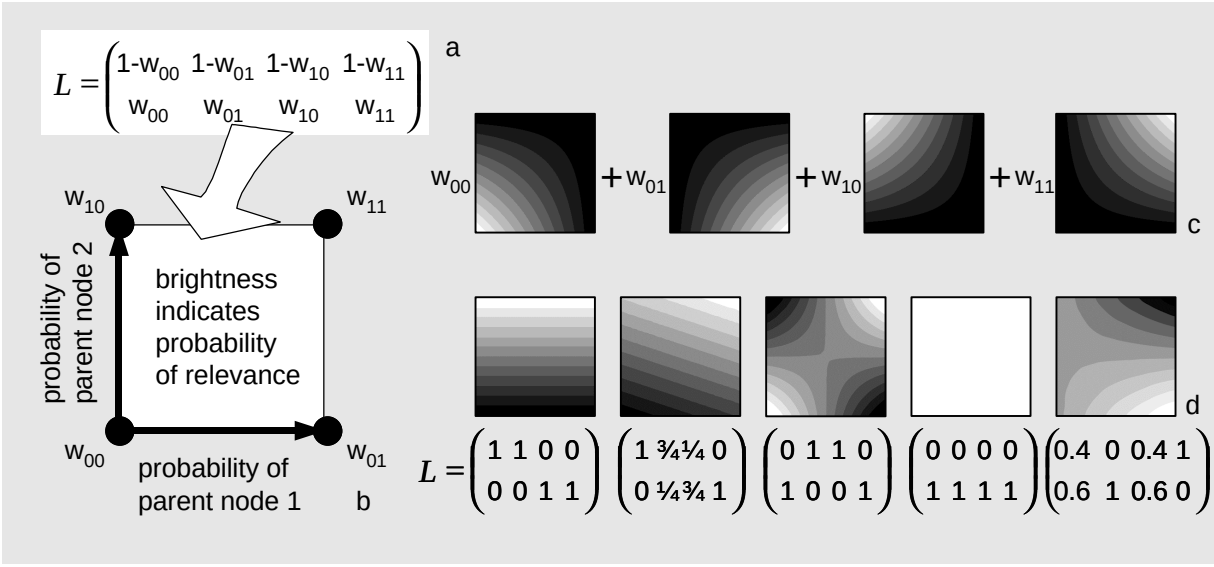


Figure 5: Link matrix for an inference network node with two parent nodes (a), the space spanned by the two parent's probabilities (b), interpolation of the child node probabilities for the cases where one link matrix values equals one (c), and interpolations for selected link matrices.

The usage of different link matrices implementing the network's nodes allows implementing features of exact-match, vector space, and probabilistic models. Therefore, inference networks can, for example, rank documents using Boolean query syntax [TC91]. The Boolean extension of probabilistic models showed equally good retrieval performance as the extended vector space model [GCT97]. Beside that, inference networks allow representing different retrieval models in parallel [TC90]. A retrieval system that implements the probabilistic retrieval model is *INQUERY* [CCH92, INQUERY92].

1.1.2.1.3 Query refinement through relevance feedback

A lot of research has been done on the simplification of query formulation. One of the most successful techniques is relevance feedback [SB90, FB90, HC93, Ols98]). In this approach, users provide the system with judgments about the relevance of examined objects. The actual computation depends on the underlying retrieval model, but typically a subset of the terms found in the objects judged relevant is added to the query or their term weights are increased. Relevance feedback simplifies the query formulation process, because the user's task of formulating queries is partially replaced by the task of criticizing suggestions made by the system.

1.1.2.2 Collaborative filtering

A purely content-based approach to information filtering is limited by the process of content analysis. In some domains the items are not amenable to any useful feature extraction with current technology (such as movies, music, restaurants). Even for text documents the representations capture only certain aspects of the content, and there are many others that would influence a user's experience, e.g. in how far it matches the user's taste [BS97].

Collaborative filtering (CF) is an approach to overcome this limitation. The basic concept of CF [GNOT92] is to automate social processes such as word of mouth. In everyday life, people rely on the recommendations from other people either by word of mouth, recommendation letters, movie and book reviews printed in newspapers, or general surveys such as *Zagat's* restaurant guide. Collaborative filtering systems assist and augment this process [RV97] and help people making decisions [BT99].

Different terms have been used to refer to CF. Initially it was termed *social filtering* or *collaborative filtering* [MGT+87, Sha94, SM95]. Some authors argued that in an automated system architecture recommenders may not necessarily collaborate explicitly with recipients, and may even be unknown to each other, and therefore prefer to use the more general term *recommender systems* [RV97]. But, as often, this term is not always used with an identical meaning. Some authors use it to refer to CF systems only, while other authors also include content-based techniques or systems mixing both approaches (e.g. [DI98]). We will use the term recommender system in this wider meaning.

In the following, we will give a brief introduction to this field. More information can be found in the ACM special issue about recommender systems [ACM99], a brief survey by Al Borchers [BHKR98], or in the proceedings of workshops on recommender systems and collaborative filtering [Rec96, CF98, Rec98, Rec99a, Rec99b].

1.1.2.2.1 Annotations and ratings

Collaborative filtering systems work by recording the reactions of users to data objects, such as documents or movies. These reactions are called *annotations* [GNOT92]. The system then aggregates these annotations and directs them to appropriate recipients. In some cases the primary transformation is in the aggregation; in others the system's value lies in the ability to make good matches between the recommenders and those seeking recommendation [RV97].

Referring to the filtering classification given at the beginning of Section 1.1.2, the attributes that CF systems use to match user profiles and objects are annotations. While content-based filtering systems consider two objects to be similar if their *content* matches the same attributes, two objects are considered similar in CF if they are liked or endorsed by the same users. Because of this special indexing technique, CF is also applicable when content-based indexing is impossible or difficult, e.g. when no computer-readable descriptions are available.

To allow effective aggregation, the annotations that users are allowed to enter are often restricted to being *ratings*. Because these ratings are entered manually, CF can adopt any type of relevance measure. Often hedges are used, such as “The best”, …, “I hate that” [SM95] or values from numerical scales, e.g. 1-5 [MRK97]. The actual relevance dimension implied by the rating may be defined explicitly (e.g. “quality of writing”, “suitability of the topic for the newsgroup”) or left to the user (“What score would you have liked GroupLens to predict for you for this article” [KRBH98]). Because providing ratings usually means effort to users, attempts to use multi-dimensional ratings have failed so far [KRBH98].

Ratings may be provided explicitly, e.g. by selecting a rating from a pull-down menu or by pressing a numeric key, or implicitly by monitoring user activities, e.g. reading time of news messages [MS94], or the saving of an object for later use [Nic97, AZ97, Bal98]. Similar information may also be gathered in an offline manner by mining objects containing usage data. Examples for such systems are the *Referral Web* [KSS97a, KSS97b, KS98] that mines publicly available documents to build a model of existing real-world relations between users, and the *Phoaks* system [THA+97] that mines newsgroup postings for recommendations of Web pages.

1.1.2.2.2 Active vs. automated collaborative filtering

Collaborative filtering systems may be distinguished as either using *active CF* or *automated CF*. In active CF, the referral network between recommenders and recipients is held in the heads of the users. Users may know other users personally or from messages that these users originated or annotated. Active CF may be classified as either *push active CF* or *pull active CF*, depending on whether recommenders selects recipients, or recipients select recommenders [Mal94, ME95].

In the simplest case, (push) active CF may be performed by simply forwarding objects to other users, that one expects to be interested in these objects. Consequently, most electronic messaging systems may be considered as rudimentary active CF systems [MGT+87]. Malone suggested allowing users to create weighted lists of people whose opinions on various topics they value. Messages could then be prioritized for a given receiver based on the number of endorsements received from people on the receiver’s endorsement list [MGT+87]. The first system actually implementing these concepts was the newsgroup filtering system *Tapestry*

[GNOT92]. A recent system using active CF is the *Knowledge Pump* [Gl98]. Pull active CF may be understood as an extension of the concept of a moderated newsgroup, so that users can select which other users they want to be their personal moderators [GNOT92].

The necessity to maintain the referral network in the user's heads limits the manageable size of active CF applications. The so-called *neighborhoods* of users are restricted to the amount of people they personally "know". Automated CF [RIS+94, SM95], also called passive CF, is an approach that overcomes this limitation. It automates the process of maintaining a network between users that may profit from exchanging recommendations. Since users are not required to know other users to be able to exchange recommendations with them, this approach makes CF applicable to much larger communities of users.

Implementations of automated CF (ACF) are conceptually based on a table like the one shown in Table 1. The ACF system gathers ratings by users about objects (here fictitious restaurants) and stores them in a single table maintained centrally. To give recommendations to user *X*, the system computes the *neighborhood* of that user, i.e. the subset of users that have a taste similar to that of user *X*. Similarity in taste is computed based on the similarity of ratings for objects that were rated by both users. The system then recommends objects that users in *X*'s neighborhood had indicated to like and that have not yet been rated by X.

Example 1 **(Automated collaborative filtering)** In the example shown in Table 1, Joe seeks recommendations for restaurants. He wants to know whether he should prefer Pizza H. or McD's. Comparison of the ratings in the table shows that Ellen and Ethan have agreed with Joe before (so that they form Joe's *neighborhoods*), while John and Joe constantly disagreed. Since Ellen and Ethan have liked Pizza H. and disliked McD's respectively, Joe is recommended to check out Pizza H., and to avoid McD's.

	Chez P.	Wally's	Beef-O	Veggio	Pizza H.	McD's
Joe	D	A	B	D		
John	A	F	D		F	
Brad	C	D		B		
Sue		C	C			C
Ben	A		A			A
Ellen	F	A				F
Ethan	D		A		A	

Table 1: Automated collaborative filtering systems base their computation on the relation *user* x *document*. Table cells contain ratings of the respective user about the respective object. (Example table by Joe Konstan (personal communication)).

The central problem of ACF is that predictions cannot be made unless the central rating table contains a sufficient amount of ratings to base the prediction upon. When new users enter the system, these users have not provided any ratings yet, so there is no basis to determine their neighborhoods from. A training period is required, during which the system can't effectively

filter for the user [ME95]. This problem is usually called *cold-start problem* [ME95] or *bootstrapping problem* [RIS+94]. We will discuss approaches to handling the cold-start problem and the related *first-rater problem* in Chapter 5.

With its network structure and the huge number of users, the Internet provides excellent support for ACF systems. A number of successful research prototypes such as the Ringo music recommender system [Sha94] and the GroupLens newsgroup filter [RIS+94] were able to attract enough users to reach the *critical mass* that is required to make predictions of sufficient quality that then attract even more users. Since these application areas are characterized by relatively stable databases *and* relatively stable user interests, ACF has been especially popular in entertainment-related application areas, such as movie, CD, or book recommendation (see also Chapter 2). In Chapter 5, we will investigate strategies to apply ACF to more dynamic application areas.

1.2 CONTRIBUTION OF THE DISSERTATION

This dissertation contributes in various aspects to the state of the art in information filtering. The three major contributions of this dissertation are (1) a new generic IF system architecture designed for the efficient handling of highly dynamic interests (the *QuerySet Architecture*, Chapter 3), (2) a new paradigm of high-level access to user profiles (*user-aggregated relevance feedback*), and (3) a framework of new user interface interaction styles providing users with this high-level access (Chapter 4).

Information filtering systems based on the *QuerySet Architecture* achieve high reactivity to interest changes by providing users with two distinct ways of updating their user profiles. First, QSA systems track gradual interest changes based on relevance feedback provided by the user. Second, they allow users to manually correct major profile inaccuracies, as they occur during profile initialization and after major interest shifts using so-called *user-aggregated relevance feedback* (URF, see Section 3.3.6). URF allows users to directly update those parts of user profiles that were affected by interest changes. Since URF is a generalized type of relevance feedback, it seamlessly integrates with relevance feedback in the profile learning process.

To achieve maximum performance, specialized user interfaces are used to enter URF. *Histogram-based interfaces* are optimized for maximum accuracy when updating individual queries in QSA profiles. *Paintable interfaces* are optimized for simultaneously updating multiple interests, which for example allows representing mood change.

We demonstrate these concepts at the example of a recommender system for TV programs (Chapter 5).

1.3 OUTLINE OF THE DISSERTATION

The remainder of this thesis is structured as follows. In Chapter 2, we state the requirements of dynamic information filtering—basically the rapid adaptation of user profiles to the user's relevance function. Since no formal classification about the nature of interest changes has yet

been published, we briefly survey interest changes recognized by different authors. Based on these interest changes, we assess the related work and point out shortcomings.

In Chapter 3, we introduce the QuerySet Architecture. We first motivate the design principles of our underlying conceptual model using an analogy between information filtering and computer animation. Then we present the conceptual structure of our dynamic filtering model and present and discuss possible implementations of its components. We introduce the concept of user-aggregated relevance feedback and show how it can be used to provide users with high-level control over their profiles. We conclude this chapter by briefly comparing the QuerySet Architecture with related work and with the requirements stated in Chapter 2.

Chapter 4 covers manual profile updating via user-aggregated relevance feedback. We present a framework of three different user interfaces, which are application-specific operationalizations of our model and that are implemented and optimized for different profile updating tasks. We discuss the application of histogram-based overviews, introduce a new interaction technique called *paintable interfaces*, and evaluate our designs through a detailed user study.

In Chapter 5, we present our TV program recommender system *TV Scout* that is based on the dynamic information filtering model and the QuerySet Architecture. After introducing the application area TV, we give a brief overview over the *TV Scout* system and its user interface before we focus on the dynamic filtering mechanism. We show how the TV Scout gathers user feedback implicitly and present the different query-execution subsystems that provide the queries that users can combine in their profiles. In Chapter 6, we conclude with a brief summary of the achievements of this dissertation and an outlook to future work.

CHAPTER 2 REQUIREMENTS ANALYSIS AND RELATED WORK

The grand challenge for information detection systems is to match rapidly changing information with highly variable interests [Oar97].

When filtering information, IF systems make predictions about the relevance of objects with respect to the current user. These predictions are made on the basis of the system's internal model of the user, i.e. the user profile. To maximize the quality of their predictions, IF systems try to adapt the user profile as closely as possible to the user's actual interests (the user's *relevance function*). Changes in the user's interests require the IF system to update the user profile accordingly. Since any delay in making these updates results in lowered prediction quality, designers of IF system strive for maximization of the adaptation speed.

In this chapter, we will begin by formulating requirements that will help us judging the appropriateness of IF systems in dynamic information filtering situations. The main part of the requirements will address the capability of the systems to rapidly adapt to selected types of interest changes. We will review existing work in information filtering, compare it to the requirements, and point out strengths and potential weaknesses.

2.1 REQUIREMENTS

A dynamic filtering system is supposed to be able to react rapidly to all kinds of interest changes. We will therefore begin our requirements definition by looking at what interest changes are mentioned in the related work.

2.1.1 Requirements 1-3: adaptation to interest changes

Numerous authors have recognized that users' information needs are constantly changing (e.g. [SK92, Moc96, All90]). Some authors have identified individual types of interest changes.

Gradual interest changes. In the domain of information filtering, even though interests are usually expected to be long-term, they are still widely recognized as *changing slowly and gradually* over time, e.g. as conditions, goals, and knowledge change [BC92, p. 31]. Gradual

changes happen as consequences of continuous processes, e.g. the user gaining experience or growing older. An example of a gradual interest change is a user's taste in music as it changes while the user gets older. Most authors who refer to interest changes only in general, implicitly refer to gradual interest changes (e.g. [Luh58, Bac91, Lan95]).

Abrupt interest changes, sometimes also referred to as interest shifts [LMMP96] have been recognized as happening induced by events [Mar95]. Such changes may occur, for example, in a professional environment due to a change in the job assignment or the initiation of a new job [LMMP96]. As a special, but very common case, abrupt interest changes may occur when new information needs arise. Marchionini distinguishes between internally (e.g. curiosity about the details of immediate thought) or externally (e.g., a teacher asking a question or making an assignment) motivated interest changes [Mar95]. The satisfaction of information needs may also be happen as an abrupt interest change [FC89].

Repetitive interest changes. Allen [All90] reports about users' interests switching randomly between different types of information and entertainment content in a news system. Nguyen observed that users have a basic taste in movies that is fairly stable over time, as well as preferences that change between uses of the system [NH98].

Loeb mentions two types of repetitive interest changes [Loe92]. The first type of change refers to repetitive but *predictable* interest changes, e.g. interest changes following the time of day. Users may, for example, have a general preference for light music in the evening hours. Since these changes may also be considered as long-term interests over the attribute "time of day", we will not consider this type of variation as interest *change*. In this dissertation, we will reserve the term repetitive interest change for *unpredictable* interest changes. This corresponds to the second type of repetitive interest changes recognized by Loeb, and to what he refers to as *mood*. A user's taste in music, for example, may vary depending on whether the user is a more outgoing or more contemplative mood. Since such moods occur repeatedly, so do the related interests.

Douglas Oard mentioned *all three types* of interest changes. "Users have *many interests*, and the lifetime of a particular interest is subject to wide variability. For example, a user may have an abiding interest in stock market quotations, while their interest in a news story may be considerably more transitory. Even when interests have long persistence, they may undergo significant changes over their lifetime. Furthermore, these *changes may occur either gradually or abruptly*. For example, the particular stocks for which a user desires quotations might change abruptly when the user sells all of their shares of that stock. A further complication is that the *intensity of an interest may vary over time* in a way that is difficult to predict. A user may have an abiding interest in restaurant reviews, for example, but that interest may only be manifest when planning a luncheon meeting with a colleague" [Oar94]. The terms *gradual changes* and *abrupt changes* are adopted from Oard's work. Table 2 relates the three types of interest changes found in the literature to each other.

	New interest state is *permanent*.	New interest state is *temporary*.
Change takes place *rapidly*, caused by an event.	Abrupt changes	Repetitive changes
Change takes place *slowly*, caused by a gradual process.	Gradual changes	

Table 2: Classification of interest change mentioned in the literature

The three types of interest changes from the related work are summarized in Table 2 that arranges them in the two dimensions. These three types of interest changes may not be comprehensive, i.e. there may be other dimensions worth distinguishing. The classification rapidly/ slowly and permanently/temporary, however, is of practical use, because it has an impact on IF system design.

- Gradual changes and abrupt changes differ in who will recognize a change first. A gradual change, caused by a slowly proceeding process that is potentially unconscious to the user, may first be observed by the system continuously monitoring the user's information intake behavior. An abrupt change, caused by an event, may first be recognized by the user, because the user may be aware of the event that caused the interest change. An IF system may profit from this distinction by allowing the user *and* the system to initiate profile updates, depending on who perceives the interest change first.
- The distinction between permanent and temporary is equally useful. A system may profit from this distinction by applying changes expected to be permanent permanently to the profile, thereby overwriting the old state of the user profile. Changes expected to be only temporary, on the other hand, may be represented in a way that allows former states of the user profile to be recalled.

Since a formal classification of interest changes is still outstanding, we will use the three interest changes summarized in Table 2 to assess existing IF work (Requirements 1-3).

2.1.2 Requirement 4: output styles

IF systems are supposed to support their users by pre-processing the stream of available objects so that users find relevant objects faster and that users can avoid irrelevant objects. To receive good grades in their function as "time-saving devices" [Bac91], IF systems should alleviate the user's task by covering the largest possible portion of the work involved in information intake. The price for this additional scope, however, is that more comprehensive IF systems have to maintain more information about the user, which also means that there is typically more information that may be affected by interest changes. Since this additional information may require updating during interest changes, more comprehensive systems may take more time to adapt to interest changes.

One aspect of an IF system's scope is reflected by the output style that the system is able to produce. At the limit, the user profile of a best match IF system defines a (transitive) preference relation between all filtered objects. Such a system is able to compute a *unique* output

ranking by sorting all objects according to the preference relation. To emphasize that all objects are combined into a single ranking, we will also use the term *single ranked output*. Only a single ranked output allows users to maximize the overall sum of relevance for any number of desired objects (see also Section 1.1). It can answer the request "Give me the *n* most relevant objects" for an arbitrary *n* by simply returning the *n* top-ranked objects. (Figure 6a)

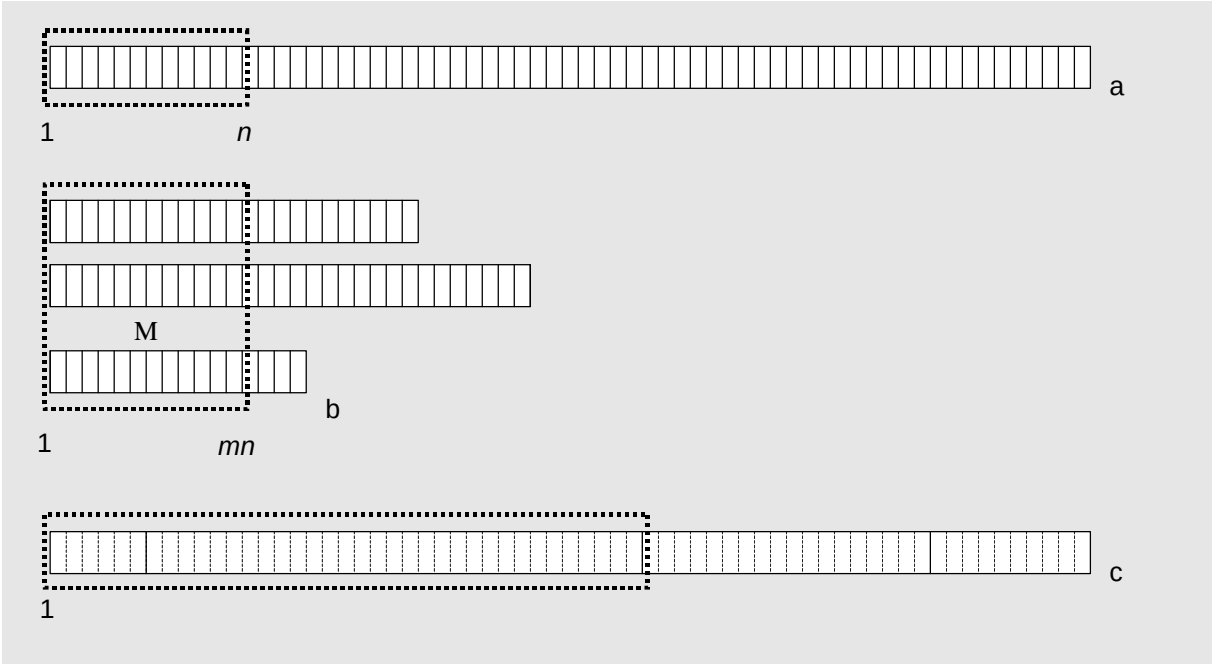


Figure 6: A system producing a single ranked output can return the *n* top-ranked objects (a). Users of systems producing multiple ranked outputs (b) or a ranking of clusters (c) have to check more objects.

Other systems represent only the relations between subsets of objects and can therefore only produce less structured output. These systems are not capable of producing a single ranked output because their user profiles leave part of the inter-object relations undefined. When users ask for the *n* most relevant objects[2], these systems can only approximate the sought set of objects.

- A *multiple ranked output system* is a system in which users define a number of interest categories, for each of which the system produces a rank-ordered list of objects. In the news filtering system *Newt*, such categories may for example be politics, business, sports, and computer [SM93]. Such an output style is for example useful if the user has dedicated timeslots assigned to each category, e.g. a private slot and a professional slot. If there are

[2] Beside their relevance criteria, users may have a set of hard constraints. A TV program, for example, might be irrelevant for a given user if it is not broadcast within the time interval where the users has time to watch. In this case, a query retrieving the *n* most relevant objects may have to be combined with an exact match filter that assures the satisfaction of the hard constraints. See Section 5.2.2 for a description of how the TV Scout combines exact match and best match retrieval in a single query.

no such dedicated timeslots, i.e. if users try to find out what the overall ranking of relevant objects is, this output style causes additional user effort. Multiple ranked output systems have no knowledge about the user's preferences across categories; they cannot determine the preference ordering of two objects, if these are from different categories. Consequently, to check out the *n* most relevant objects, users may have to check out the top *n* objects from *each* category (assuming that there are at least *n* objects in each category). This results in an overall number of up to *mn* objects, if *m* is the number of categories (Figure 6b).

- A *ranking of clusters system* is a system returning a single ranking, but multiple objects can be at the same rank (a *weak order*). Typical examples for systems producing this type of output are rule-based systems running a set of exact match rules that insert objects into categories or assign predefined ratings to objects (e.g. the so-called *appraisers* in the IF system *Tapestry* [GNOT92]). To find the *n* most relevant objects, users have to check out the smallest set of top-ranked clusters that together contain at least *n* objects (Figure 6c).

Table 3 relates these output styles to each other. The *single ranked output* is the most powerful output style, because it defines the relevance ordering between all objects. *Multiple ranked outputs* define only the order within each cluster/category, but not between clusters/categories. *Rankings of clusters* define only the relevance order between clusters, but not within clusters. In an *unranked output* no relevance order is defined at all.

	No order between clusters	Order between clusters
No order within cluster	(Exact match, unranked output)	Ranking of clusters
Order within cluster	Multiple ranked outputs	Single ranked output

Table 3: Classification of user profiles according to their output style.

More comprehensive user profiles can save user effort during every execution of the profile, but they also require more user input during initialization and maintenance. Systems producing a single ranked output, for example, are affected by interest changes that alter the preference order between *any* two objects. Systems using less comprehensive user profiles may therefore be easier to maintain. To not overestimate the value of IF systems producing less structured output, we will consider output styles in our comparison of IF systems.

2.1.3 Requirement 5: correctness of representation

The general correctness of IF methods is virtually impossible to judge and even the attempt would exceed the scope of this thesis. We will therefore restrict our discussion to a single criterion that we found worth taking into account, i.e. a system's capability to correctly represent multiple interests. Some of the reviewed systems are based on user profiles representing a linear combination of object attributes. These systems are able to correctly represent a single interest, but they may produce artifacts when trying to represent multiple interests in a single user profile. Mock illustrates this limitation with the following example ([Moc96], see Section 2.2.1). If users are not interested in the features "King" and "Queen", but are interested in

messages with the features “Sacramento” and “King” (the basketball team), then the profile learning method will be using the same accepted and rejected values for the word “king” and may be unable to classify these articles correctly. Systems using user profiles representing a linear combination model are not able to correctly represent this example and will generally not be able to learn combinations of any two features A and B, where (A and B), (¬A and ¬B) are desired, but (¬A and B), (A and ¬B) are not (Xor problem).

2.2 SUPPORT FOR INTEREST CHANGES IN RELATED WORK: AN OVERVIEW

In this section, we will survey the state of the art in information filtering. We will briefly present each IF system and discuss its particular features. In Section 2.3.3, we will give a brief summary of the systems, compare them to the requirements, analyze which features are best suited for fulfilling the requirements, and point out possible shortcomings.

The speed at which IF systems can react to interest changes depends on how much input they receive from the user and on how informative this input is for deducing interests and interest changes. Large amounts of relevant information about the user’s current interests allow faster profile updates, which can lead to a better representation of the user’s interests in the user profile. This, in turn, can lead to higher prediction quality, which is one central objective of IF systems. Another objective, however, is to require as little user effort as possible. Consequently, the design of an IF system has to trade off the maximization of adaptation speed (and thus prediction quality) and the minimization of usage effort.

Different researchers and system designers have assigned different priorities to these two design goals and have consequently come to different approaches to IF system design. Figure 7a shows a close-up of the inner refinement loop of the general IF model we discussed in Section 1.1.2 (see Figure 2). Based on the user profile, objects are selected and presented to the user (left-hand side of the diagram). IF systems have three options for gathering input about the user’s interests (right-hand side of the diagram). They can gather the user’s preference judgments with respect to attributes (in the case of a movie filtering system, users could express “I like movies featuring the actor *Harrison Ford”* or “I like movies described by the keyword *Comedy*”), ratings about objects (e.g. “I like the movie *Star Wars*”), or demographic data to match it with stereotypes (e.g. “I am a *male Student*”).

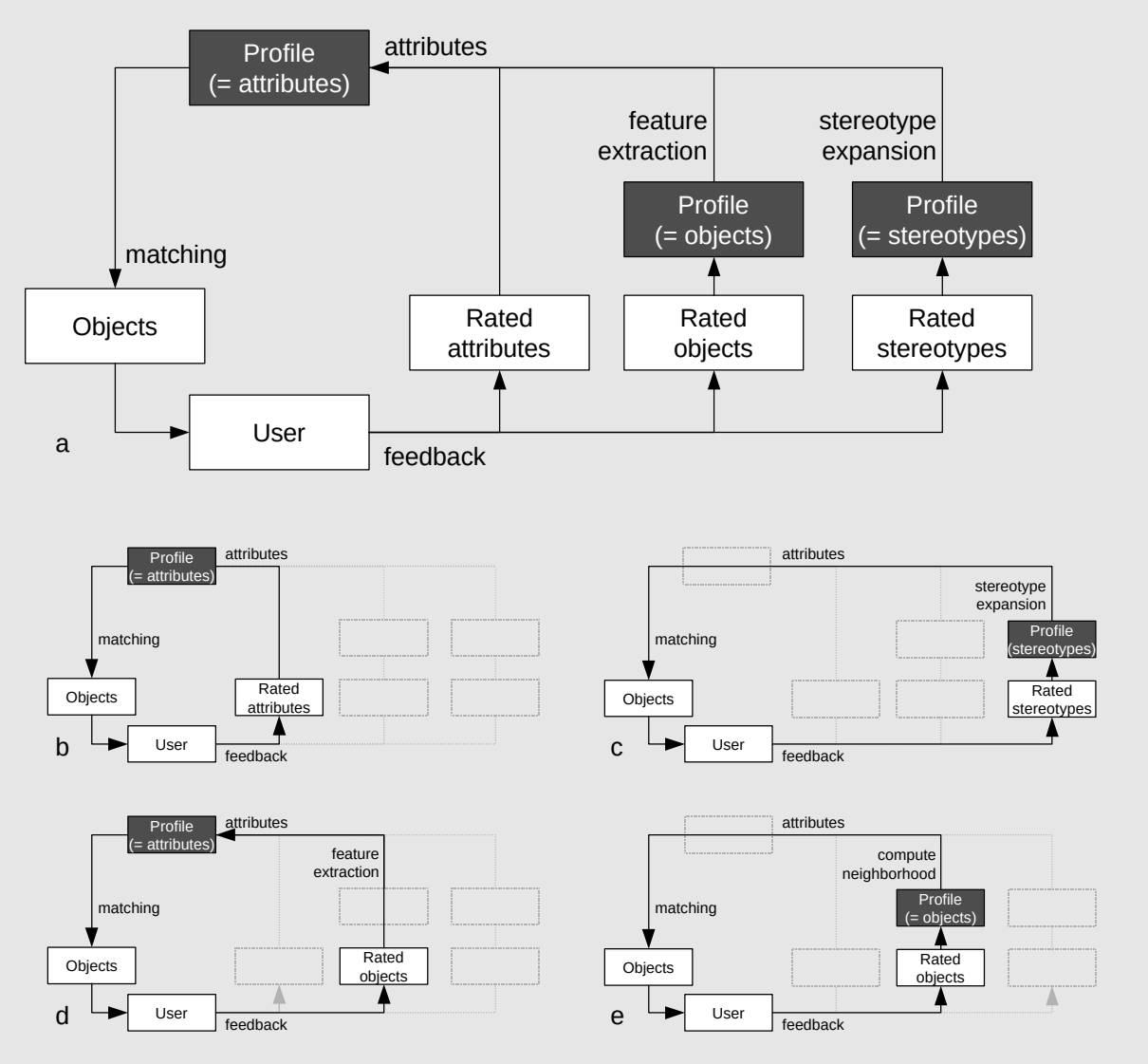


Figure 7: The profile manipulation cycle (a) of IF systems gathering input about attributes (b), stereotypes (c), object ratings (d), and of automated collaborative filtering systems (e).

The type of input that is easiest to process for IF systems is preference judgments about attributes (Figure 7b). We will understand this term in a wider sense that also includes all other data types mapping attributes to ratings, e.g. rules. User profiles may store this profile information using an explicit *attribute* × *rating* table, a neural network, so-called agents, or collections of queries or rules. Since the data obtained from the user (i.e. preference judgments about attributes) is already in the format to be stored in the profile, no conversion such as feature extraction is required here, so that profile maintenance becomes rather straightforward. At filtering time, when a rating for a new object is to be predicted, the respective object is disassembled into its attributes (indexing), these attributes are matched with the profile, and the associated ratings are aggregated to form the resulting object rating. Stereotype-based systems work simi-

larly, but require an additional conversion step from stereotypes to attributes (*stereotype expansion*, Figure 7c).

IF systems gathering object ratings as input (relevance feedback) have to perform additional computation. The fact that some type of *attribute* × *rating* relation is necessary for computing predictions requires the system to perform an "abstraction process" from objects to attributes, e.g. using (automatic) feature extraction. Usually, feature extraction is carried out before storing the profile, so that the profile stores the result of the feature extraction, e.g. *attribute* × *ratings* relations (Figure 7d). Automated collaborative filtering (ACF) systems don't proceed this way, because the abstraction process from object ratings to attributes depends on other users and their ratings. Since other users' ratings may change at any time, ACF systems usually store object ratings explicitly and carry out the abstraction process (computation of the neighborhood) at filtering time instead (Figure 7e).

Some IF systems specialize on gathering only one type of user feedback, but since the individual types of input are not mutually exclusive, many systems use multiple sources of input, which makes the classification of related work more complex. However, since IF systems usually *store* only one type of data in their profiles, we will use the profile data type as the primary classification criterion, to organize our survey.

2.2.1 IF systems with user profiles consisting of (attribute, rating) vectors

The first group of IF systems that we will look at are those that store *attribute* × *rating* relations (e.g. as collections of queries) in their user profiles. Systems of this group gather their input using content-based indexing, manual input of queries or similar data structures, or both.

The number of attributes that may be assigned to an unstructured object, such as an email message, is huge. Consequently, the indexing process has to be limited to only the most promising descriptors or descriptor combinations to avoid the "curse of dimensionality" [Lan95]. On the one hand, descriptors should bear as much meaning and expressiveness as possible. On the other hand, descriptors must not be too specific, so that they will match a sufficient number of objects to be able to contribute to a significant number of predictions. Often words or combinations of words are used as descriptors, so that the user profiles actually are *keyword* × *rating* vectors [DI98]. Depending on the actual system, descriptors can be simple descriptors (e.g. a keyword or the name of a specific email sender), or combinations of descriptors.

- Although a very early system, the *Business Intelligence System* by Luhn possesses interesting features supporting interest changes [Luh58]. This system belongs to a track of research that is usually referred to as *Selective Dissemination of Information* (SDI). While Luhn's system carries out many of its task using human help (e.g. based on microfiches), it has most of the features of fully computer-based systems. This system represents the information needs of users and user groups by using keywords vectors, called *patterns*. New documents are delivered to the user if they match at least one of the user's patterns.

In this system, users can manually update their profiles by creating new patterns, which gives users a means for communicating new interests and abrupt interest changes. Profiles are also updated automatically by the system based on the articles a user accepts (implicit relevance feedback). To take care of gradual interest changes, the system automatically removes patterns

gradually from the profile as time passes. Since patterns can be used to actively query the system in a query by example fashion, users have a rudimentary means to deal with repetitive interest changes. Luhn's system produces unranked output only.

- The filtering system for technical memos by Foltz and Dumais [FD92] also uses user profiles that consist of sets of keyword vectors that are matched against incoming articles using the cosine measure. Profiles are text-only and are created and modified manually by the users. Foltz and Dumais' main finding is that *Latent Semantic Indexing* (LSI) can enhance prediction quality. LSI is a technique for reducing the dimensionality of the space of index terms [Dum88, Bil98].

- The same basic architecture of multiple queries is used again in *SIFT*, the filtering system for Internet news articles [YG95]. Queries can be formulated either using the vector space model or as conjunctive Boolean queries. To control the number of articles delivered per vector space query, users are allowed to assign rating thresholds to each of these queries. In SIFT, user profiles can be adapted to interest changes in two ways. First, queries can be created, reformulated, or deleted manually, allowing users to represent abrupt interest changes. Rating thresholds can also be adjusted manually, so that users can control how many objects per query they will typically receive. The other way of updating each individual query in the profile is using relevance feedback. Since all queries of a user are completely independent from each other, the SIFT system provides multiple ranked output only.

- Although usually not considered an IF system, we want to include the *bookmarks* concept of World Wide Web browsers in this discussion. When a bookmark is used to refer to a dynamic page such as search engines [Fel98], the bookmark saves not only the address of the Web location (URL), but also the query parameters. Recalling the bookmark re-executes the query in the current state of the Web. Since the Web is in continuous change, the repeated executions of a bookmarked query may return different sets of pages when re-executed. The reuse of bookmarked search engine queries, e.g. in combination with an offline reader that executes the bookmarks automatically, may be used to stay up to date about a subject and a collection of such saved queries may be considered a user profile.

The properties of a bookmark collection are very similar to those of the SIFT system. Since bookmarks are stored individually, it is easy to recall individual queries when the respective interest comes up, i.e. in the case of repetitive interest changes. Abrupt interest changes can be handled manually by creating and bookmarking new queries or by removing those that represent interests that are gone. There is no gradual learning of interest changes, unless the search engine that the bookmark refers to supports relevance feedback and the user updates the bookmarks after performing a refinement operation. Similar to SIFT, bookmarks support multiple ranked outputs only. See Chapter 4 for a prototypical bookmark-based system that performs the aggregations required for creating a single ranked output.

- The *NewsWeeder* system is a news filtering system learning from explicit user feedback [Lan95]. This system tries to improve the performance of filtering systems by replacing TF-IDF weighting used for example in the SIFT system by the *Minimum Description Length* (MDL) measure. In a small-scale study, Lang found MDL to perform better than TF-IDF.

- *INFOS* [Moc96] is another filtering system using alternative measures instead of TF-IDF—for a different purpose, however. Realizing that the users' interests change quickly, user

profiles in the INFOS system are designed to be especially simple and therefore easier for users to modify. User profiles consist of a *single* vector of *term* × *rating* pairs, which are supposed to simplify the communication of abrupt interest changes. Messages are filtered based on a rating computes as the inner product of message features and profile. User profiles are updated automatically by the system based on relevance feedback provided by users. During this updating procedure, the counters associated with a term are increased if the respective term is found in a document judged relevant by the user; counters are decreased if the respective term is found in a document judged irrelevant. This learning method is referred to as *Global Hill Climbing* (GHC).

While the simplified user profile has advantages with respect to its modifiability, this format also has drawbacks. The learning mechanism is restricted by the fact that it linearly combines all input features based on the conditional independence assumption, as described in 2.1.3, so that this method cannot correctly represent multiple interests. Mock partially solves this problem by adding the online lexical reference system *WordNet* that helps to identify concepts in the examined messages.

- The Web page recommendation service *Fab* by Marko Balabanovic [BS97] uses user profiles similar to those used by INFOS. Profiles, here called *selection agents*, are simple keyword vectors and are generated by adding the weighted keyword vectors of the training examples. Consequently, Fab suffers from similar problems as the INFOS system when trying to learn multiple user interests (see the “King” and “Queen” example in Section 2.1.3).

- *LyricTime* [Loe92] is a music filtering system that aims at supporting casual users looking for entertainment music. Unlike the systems described above, the songs that LyricTime recommends are described by structured records. Indexing is therefore not required. The user profile consists of a single set of *attribute* × *rating offset* pairs that are learned through explicit relevance feedback. Because of this design, LyricTime is not able to learn the profiles of users liking only specific feature combinations, e.g. the soft songs by some band. This problem corresponds to the “King” and “Queen” example discussed earlier.

Two features distinguish LyricTime from the systems discussed above. One is that it delivers (i.e. it actually plays) songs repeatedly—a strategy that distinguishes the application area of music from application areas, such as news, where objects typically become irrelevant after being retrieved once. When compiling play lists, LyricTime does not repeat the best-liked songs all the time. Instead, it tries to optimize the *mix* by letting user preference determine play frequencies. The second distinguishing feature of LyricTime is that users have multiple profiles—one for each individual mood, e.g. a profile for soft music, a rock music profile, etc. Only one profile is active at a time. We will discuss this approach to handling repetitive interest changes in more detail in Chapter 4.

- The Decision-Theoretic Video Advisor *DIVA* helps users in finding movies they are likely to enjoy [NH98]. Like the LyricTime system, DIVA is based on *structured* descriptions of the filtered items, here movies, and predicts the value of a movie as a sum of the predicted values of its attributes. To reduce the complexity of the high-dimensional attribute space, DIVA restricts itself to the five attributes *runningTime*, *rating*, *casting*, *director*, and *genre*, which the authors expect to have the highest expressiveness. Making the assumption of preferential independence (see [KR93]), *runningTime* and *rating* are decomposed from the rest, so that only

the triple *(casting, director, genre)* is handled as an entity. This triple is not decomposed further, because the authors expect a strong preferential dependence between the three attributes; users liking James Cameron's dramas and romances may, for example, dislike his action movies.

Users can update their profiles by entering attribute ratings explicitly or by giving training examples of liked or dislikes movies. To provide additional support for repetitive interest changes, the DIVA system stores interest parameters expected to be rather stable in the profile, while the more dynamic ones are entered as exact match retrieval parameters during every use of the system. Using this mechanism it is, for example, possible to *exclude* action movies, but impossible to *reduce* the amount of recommended action movies.

- The personal Usenet news service *Browse* by Jennings and Higuchi [JH92] uses neural networks to represent user profiles. Profile networks are trained as follows. Repeated co-occurrence of keywords in documents judged relevant establishes a mutual activation relationship between these keywords. During the filtering process, such keywords can activate each other, so that also related keywords will contribute to a document's rating. This way, a thesaurus-like functionality is added to the filtering process that is expected to help improving coverage. On the other hand, the King and Queen example listed above cannot necessarily be learned successfully by this type of network, which leads to problems with the representation of multiple interests. The users of *Browse* are allowed to manually activate or deactivate keywords in the neural network to temporarily influence predictions. Since *Browse* uses an individual network for each subscribed Newsgroup, users receive multiple ranked outputs.

- The filtering system *SIFTER* by Lam et al. [LMMP96] has been applied to LISTSERV mails and to academic research reports in the domain of computer science. User profiles in SIFTER consist of two distinct layers that are trained independently. The lower layer is responsible for long-term interests. It uses a reinforcement learning algorithm to learn the user profile assuming unchanging user interests. The categories that form the lower layer are generated by running a clustering algorithm in an offline manner and cannot be customized by the user, so that the user's actual interests may not necessarily be accurately represented by the system.

At the higher level, a user interest tracking algorithm using Bayesian decision theory is employed to detect shifts in user interests. This level collects relevance feedback about documents from the user and compares it with the system's predictions. Whenever the difference exceeds a certain threshold, the system considers the user to have undergone an interest shift and reacts by reinitializing the lower level of the user profile. In an experiment, SIFTER required eight to twelve units of relevance feedback to be able to detect interest shifts. Although SIFTER may *detect* interest changes rather fast compared to other systems, it then requires additional input for performing the actual profile *adaptation*. Another aspect is that the experiment was carried out in the absence of interest variations ("noise"). The presence of such interest variations can be expected to decrease the performance of the interest detection mechanism by causing unjustified profile reinitializations. The output of the SIFTER system is ordered according to the categories, with no particular order within categories.

The news and mail filtering system by Paul Baclace [Bac91, Bac92] is based on genetic algorithms. Since Baclace's learning algorithm can be used as part of the information filtering

architecture that we will present in this thesis (see Section **3.3.3**), we will describe this filtering approach in a bit more detail.

In this filtering model, user profiles consist of a number of so-called *agents*. Agents can either be *simple agents* or *conjunction agents*. Initially, a user's profile contains only *simple agents* that each represents one elementary feature, such as a term. Each simple agent contains an internal value that Baclace calls "bid". The bid represents how strongly correlated the agent's feature is assumed to be with the relevance of an object, so the notion of "bids" is very similar to the notion of term weights. When filtering an object, all agents that represent features present in that object make their "bids". These bids are summed up to form the predicted rating for that object. This rating is used to rank objects when presented to the user. The user is then allowed to provide relevance feedback. Agents are then evaluated based on this relevance feedback, i.e. agents that had made successful predictions are rewarded with "money", which allows them to survive and to make larger bids next time.

As long as only simple agents are used, this algorithm computes only weighted sums, so it would not be able to correctly learn the "king and queen" example discussed earlier. To allow assigning different ratings to objects matching multiple agents, so-called *conjunction agents* are introduced. These agents represent conjunctions of multiple features and are only activated by objects matching *all* features. To correctly represent the "king and queen" example, two conjunction agents would be created. The "King AND Sacramento" agent would make positive bids whenever activated and the "King AND Queen" would make negative bids.

To avoid an exponential number of conjunction agents, Baclace's system uses an economically inspired selection model for keeping the agent population small enough to be handled effectively. First, conjunction agents are only created when a new feature combination was actually observed in an object that is to be filtered. Second, an economically inspired model is used that lets agents only survive, if their predictions are correct and substantially different from what the system would have predicted without them (see [Bac91] for details). Furthermore, the most effective agents are allowed to reproduce, e.g. to form new conjunction agents.

- Similarly, Sheth and Maes' filtering system *Newt* filters news articles from USENET newsgroups combining genetic algorithms [Gre85, DeJ80] and learning from feedback [SM93]. Similar to Baclace's system, agents holding feature combinations have to gain fitness to survive to the next generation. The number of articles retrieved by an agent is proportional to its fitness, so Newt provides some basic management of object amounts. One difference between the two systems is that the Newt system allows for a greater amount of user participation. To increase learning speed, users can manually indicate preference for particular keywords occurring in an article. Newt uses significantly more complex agents than Baclace's system. Newt's agents do not cooperate to make a recommendation, but produce recommendations individually. Newt allows users to have multiple agent populations, one for each set of newsgroups, so that the system produces multiple ranked outputs.

- The Web page filtering system *WebMate* by Chen and Sycara [CS98] uses a user profile that consists of the keyword vectors of positive training examples. During the filtering process, the keyword vectors of Web pages to be filtered are compared with the vectors in the profile using TF-IDF weighting. They are delivered to the user if the similarity of the new Web page to any vector in the profile is above a given threshold. To keep user profiles manageable,

WebMate holds the number of these keyword vectors constant by successively aggregating those keyword vectors that are most similar according to the TF-IDF measure. A profile of *n* keyword vectors can correctly represent up to *n* independent user interests.

2.2.2 Rule-based approaches

Rule-based systems employ user profiles consisting of rules. The exact format of rules varies between systems, but generally rules are of the form "if <*condition*> then <*action*>". During the filtering process, rule-based systems compare incoming objects with all rules in the user's profiles. Whenever an object matches the *condition* of one or more rules, some system-specific subset of the corresponding *actions* is executed on that object. Typical actions are displaying, filing, forwarding, or appraising of objects. In the following we will look at an arbitrary selection of such systems.

- In his ACM presidential letter 1982, Peter Denning suggested a filtering approach in which senders attach priority values to their mail messages, informing the receiver about how important the sender considers the respective message to be [Den82]. To avoid senders biasing the receiver's information intake by assigning unjustified priority values to their messages, Denning suggests to upgrade or downgrade sender ratings by applying a sender-specific *bias number*. Bias numbers would be computed automatically based on relevance feedback assigned to each message by the receiver after reading the message. Denning did not implement this concept (it may be actually implemented as a rule-based system or using a different technique), but the functionality he suggested can be found in several rule-based systems that were built in the following years.

- The *Information Lens* is one of these systems [MGT86, MGT+87, MMC+89]. User profiles in this mail filtering system contain so-called *production rules* of the type "If <*Boolean expression over object properties*> then <*action*>". User profiles consist of collections of such rules. Extending Denning's design, the Boolean expressions can also take the message content, e.g. keywords contained in the message, into account. The actions initiated by rules allow, for example, the storage of messages or the removal of messages from the incoming stream. A central aspect of the Information Lens system is that the conditionals of rules refer to the content of *fields* in the received email messages, so that the email messages are expected to be semi-structured. Therefore, senders are required to compose their mail messages based on predefined standardized message schemes. To communicate abrupt interest changes, users may manually update the rules that form their profiles. There is no support for gradual or repetitive interest changes. The Information Lens provides no ranking mechanisms on output, so that unranked output is delivered to the user.

- The *ISCREEN* system [Pol88] is very similar to the Information Lens, but supports the user's rule definition task by providing a simple rule editor. Rules in ISCREEN can include attributes of the message as well as the state of user-specified variables, such as "on vacation". Multiple conditions are interpreted as conjunctions. The ISCREEN system is provided with a conflict detection component that detects conflicts between rules and helps users resolve them, as well as an explanation component that explains the carried out actions to the user. ISCREEN delivers unranked output.

- The *MAFIA* system [LKH90] is another approach inspired by the Information Lens. Lutz et al. criticize that the Information Lens system requires active participation of the sender, which the sender may not generally be willing to provide (for more discussion on the assignment of work and benefit see [CR87, Gru87, Gru89]). To alleviate this problem, the MAFIA system shifts part of the user effort from sender to receiver. Instead of senders filling in predefined forms allowing the simple recognition of message types and structure, Lutz et al. use a wrapper program that tries to find the semantic structure of the unstructured messages. The classification accuracy, however, drops to 95% or even to 72%, which may not be sufficient for all types of users. So-called "prioritizers" allow users to produce a weakly ordered output.

- The *INFOSCOPE* system, following up on the same line of research, is a filtering system for Usenet newsgroup articles providing semi-automatic rule-creation support [FS91, Ste92a]. So-called *agents* monitor the user's behavior, automatically generate filter rules based on the observed behavior, and suggest them to the user. Users may accept, modify and accept, or drop suggested rules, thereby slipping into the role of a filter critique instead of a filter constructor. Rule suggestion provides some support for gradual interest changes by making users aware of these changes and by supporting the required profile updates. Rules are made accessible to the user as so-called *virtual newsgroups* that are unions of multiple real newsgroups plus selection rules. INFOSCOPE performs exact match filtering and therefore produces only unranked output.

- The *Tapestry* email/newsreader system [GNOT92, Ter93] has more similarities with a database system than with a rule-based system. Its user profile structure and the *appraiser* concept (see below), however, make it so similar to rule-based systems that we are listing it in this section. Tapestry allows users to manually construct profiles from queries that refer to both document content *and* annotations added to documents by other users. Tapestry is therefore considered to be the first (*active*) *collaborative filtering* system. Additional aspects of combining content-based filtering with collaborative filtering have been discussed in [Bau99b]. Tapestry's so-called *continuous queries* allow the consistent handling of temporal aspects, e.g. using queries, such as "select documents that are more than two weeks old and to which nobody has sent a reply".

Similar to rule-based systems, continuous queries can be updated manually whenever interest changes occur. There is no dedicated support for gradual interest changes. Continuous queries can also be executed individually as so-called *ad hoc queries* over the current state of the Tapestry repository [MD89, TGNO92], which allows users at least to "reuse" individual queries in the case of repetitive interest changes. To provide an ordering on and to categorize filtered messages, additional rules called *appraisers* can be used to assign ratings to matching objects. If an email message is appraised by multiple appraisers, then Tapestry uses the highest priority assigned by the appraisers. Using appraisers, Tapestry can provide a weak ordering of filtered messages.

- The Gnus newsreader provides two types of rule-based filtering functionality [Gnus]. So-called *kill files* allow the removal of unwanted articles matching a set of Boolean expressions from the news stream; the remaining articles are passed on to the user as an unranked set. *Score files* are an extension to kill files. Similar to appraisers, score file rules allow associating positive or negative rating offsets to article properties. Unlike the Tapestry appraisers, these rating offsets are summed up. All articles with a rating sum higher than a given thresh-

old are rank-ordered and delivered to the user, resulting in a weak ordering. Newer versions of score files provide support for automatic profile reformulation based on relevance feedback.

2.2.3 Stereotype-based approaches

- The groundbreaking book-recommender system *GRUNDY* by Elaine Rich [Ric79a, Ric79b, Ric83] uses user profiles based on stereotypes. To create a user profile in GRUNDY, users enter keywords describing *their personality*, not their information need. GRUNDY then associates terms used in the users' self-descriptions, e.g. "feminist", with pre-defined stereotypes. These stereotypes expand into *attribute* × *rating* pairs describing the users' information needs that are aggregated to form the object ratings when making predictions. GRUNDY produces a single ranked output. The personality traits that users list to describe themselves are inherently long-term. Consequently, GRUNDY does not provide any possibility for the users to directly update the representation of their information needs.

- The *um movie-advisor* [Kay95] is a filtering/recommendation system for movies that comes from the background of user modeling. User profiles are based on genres and other high-level movie attributes (e.g. violence, horror) that are annotated with the two dimensions level of importance and level of like and dislike. The movie advisor takes multiple sources into account when building user profiles. First, *attribute* × *importance* × *liking* triples may be entered and updated manually. Second, the user's personal attributes are taken into account by applying stereotypes. Third, ratings about individual movies are used to deduce these attributes (by a module called the *startup movie rater*).

The *um* system gives users access to their profiles and supports users by automatically updating them. Extracts of the current user profile content are displayed to the user whenever false predictions have been made. The so-called *trouble shouter module* allows users to manually resolve such conflicts. If users can find no error in the current content of their user models, they are allowed to manually add new *attribute* × *importance* × *liking* triples to fix the false prediction. Since the attributes in these triplets may be arbitrarily complex, e.g. a conjunction of simple attributes, the linear combination of attributes that *um* uses is still able to represent multiple interests. The um system provides no means for handling repetitive interest changes. It produces single ranked output.

2.2.4 IF systems with user profiles consisting of object rating vectors (automated collaborative filtering)

Automated collaborative filtering systems (e.g. [RV97, BHKR98], see also Chapter 1) recommend objects to a user U that have not yet been presented to U and that were judged relevant by other users that have repeatedly agreed with U in the past (*U*'s so-called *neighborhood*). Similar to systems learning from relevance feedback, ACF systems create user profiles automatically, based on a user's relevance feedback. Instead of performing feature extraction, however, ACF systems employ *users* as features. Objects are not classified by their content (e.g. a news article being characterized by the appearance of the keyword "Internet"), but are characterized by the group of users from the community judging them as relevant (e.g. a

movie being endorsed by the user <good@cs.umn.edu>). Since the ratings by other users may change at any time, ACF systems store non-aggregated object ratings in the user profiles, as already mentioned in Section 2.2. Using this data, ACF systems can recompute a user's neighborhood whenever necessary.

Because of the relatively high homogeneity between ACF systems, we will mention only two of the most prominent projects here. The early news recommender system *GroupLens* [RIS+94, KMM+97, MRK97, MRK+97, KRBH98, GroupLens] inspired many of the later ACF systems. The recommender system *Ringo* [Sha94, SM95] uses automated collaborative filtering to recommend music. The movie recommender system MovieLens that is run by the same project uses the same recommender system architecture to recommend movies (http://www.movielens.com). A commercial version of the GroupLens recommender engine is applied, for example, in the online bookstore Amazon.com (http://www.amazon.com). More ACF systems can be found in [Rec96, Rec98, Rec99a, Rec99b].

Because of the similar architectures, the characteristics of all currently available ACF systems' are relatively similar. The differences between individual ACF systems, e.g. how the subset of users that contribute to a user's predictions (the neighborhood) is selected and how these users' ratings are weighted (e.g. using the Pearson coefficient [JF87]), have little influence on their dynamic filtering properties. Since no aggregation of the ratings in the user profile is provided, entire user profiles cannot be manipulated effectively, so that ACF systems provide users with no means to instantly communicate their abrupt interest changes to the system. Users would have to re-rate large amounts of objects in their profiles to communicate interest shifts. For the same reason, ACF systems cannot provide efficient means for handling repetitive interest changes[3]. To provide at least some adaptability, some of these systems, e.g. the MovieLens system [GSK+ 99, SKB+98], allow pre-filtering the set of recommended objects with some content-based exact match filter. The MovieLens system, for example, allows users to restrict the recommendation by movie genre and publication decade. One possibility to support the system's adaptation to gradual interest changes is to let old rating data date out whenever rating new data comes in, which, however, requires a substantial rate of fresh ratings.

2.3 COMPARISON WITH REQUIREMENTS

The different approaches presented in the previous section have their specific strengths and weaknesses. In this section, we will discuss the dimension most relevant to the dynamic filtering requirements, i.e. the systems' ways of gathering information about users' interests. We will discuss the two most prominent styles "gathering object ratings" (relevance feedback) and "gathering preference judgments about attributes ratings", before we finally assess and compare the systems presented in the previous section.

[3] The small interest change rate of information needs is, together with the small information source rate, one of the primary reasons why movies and music CDs are successful applications of automated collaborative filtering. The fact that ACF systems cannot deal with interest changes is not much of a problem for these application areas, because the interest change rate is so small.

2.3.1 Advantages and limitations of learning from relevance feedback

Before we analyze the implications that the gathering of object ratings has on the fulfillment of the requirements, we will make some general observations about this approach. Learning from relevance feedback has two practical advantages. First, users are only required to interact with the objects themselves, not with any auxiliary data type, such as attributes or other internal representation of user profiles. The fact that users are familiar with the object domain—objects are the very reason why users use IF systems—makes these systems easy to learn. Consequently, being provided with appropriate interfaces, users have no difficulties in expressing their personal relevance judgments about objects (see [SB90, FB90, HC93, TKI96]). Second, in many systems, the required relevance feedback about objects may be gathered automatically and unobtrusively. This approach is called implicit feedback [HHWM92, HRS94, Nic97, AZ97]. When users deal with objects, e.g. when users examine, retain, or reference objects, the system may observe this user behavior and use this data to estimate how the user would rate the respective objects [OK98, p.81] (see also Section 5.5.3). Related work shows, for example, that the amount of time users spend on reading a news articles can be a good predictor for the relevance of that article [MS94]. Implicit feedback helps reducing the user effort and is therefore an attractive alternative to explicit ratings.

How far can IF systems that learn from relevance feedback adapt to interest changes? IF systems that gather their input about the user's interests *only* from relevance feedback are inherently limited in their ability to adapt to interest changes. Jennings and Higuchi summarize: "There is a delay in acquiring user interests, as they are determined by observation. Once a set of interests is established there is a tendency for these to persist" [JH92, p. 207]. Figure 8a visualizes this fact by the example of a gradual interest change. In this example, the user's interest changes are sketched by the bold line. Since the IF system has no *direct* way to recognize this interest change, the IF system has to incrementally gather evidence for the interest change before it can adapt the user profile. Consequently, the interest change cannot be detected before the user has (implicitly or explicitly) rated a sufficient number of objects that can serve as training examples. During this period, the system's predictions still refer to the state of the user's past interests and the system's prediction quality is reduced.

If interest changes take place slowly and gradually, the relative error caused by the learning delay is small. If the first derivative of the interest signal is close to zero, so is the resulting relative prediction error. Consequently, learning from relevance feedback is an appropriate approach for application areas where user interests are subject to gradual changes only. Abrupt interest changes, however, cause much larger errors if attempted to be learned from relevance feedback. The relative error caused by an abrupt change is necessarily bigger than the one caused by a gradual change of the same amplitude (Figure 8b)[4].

[4] An especially problematic case of interest change is the detection of new interests. Since systems learning from relevance feedback receive no explicit notification about new interests of their users, they have to probe such interests. See [SM93] for a description of an optimization process between *exploitation* (i.e. to profit from the current state of the profile) and *exploration* (i.e. to attempt to improve the profile and to adapt to changes). Exploration of new user interests, however, is a very costly strategy that may produce many false predictions to guess a single new user interest.

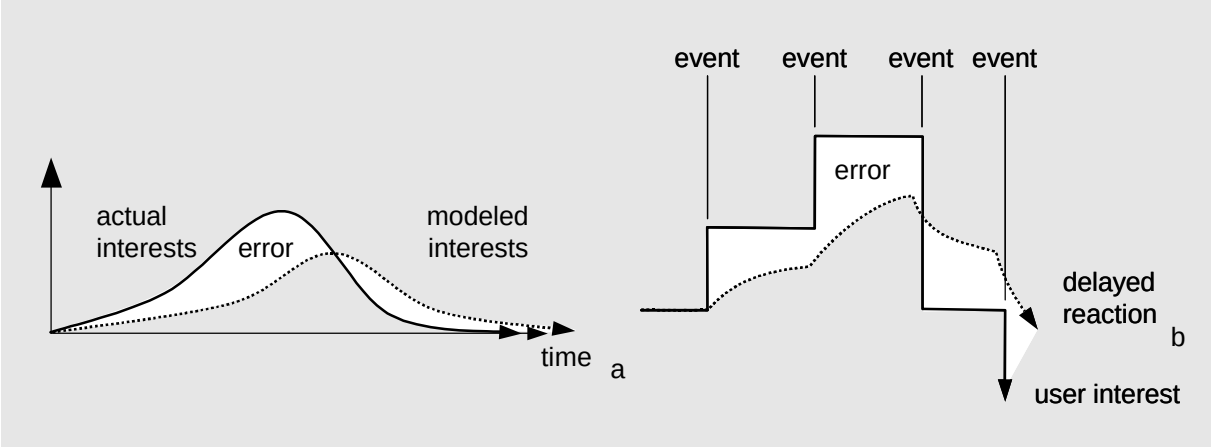


Figure 8: Inherent imperfection in interest tracking [JH92, p. 207][5]. The error caused by the delay is highlighted in white (a). Errors introduced by IF system learning from relevance feedback when trying to adapt to abrupt interest changes (b).

Some authors have suggested to reduce the prediction error that is caused by learning delays by maximizing the learning speed of IF systems [FC89, Bac91]. Increased adaptation speed, indeed, can reduce the error caused by the learning-delay. The SIFTER system is an interesting step in that direction ([LMMP96], see Section 2.2.1), although it's interest shift detection does not solve the problem that relevance feedback provides only little data for reinitializing the profile once an interest change has been detected.

Systems learning only from relevance feedback do not inherently provide support for handling repetitive interest changes. If a user of a news filtering system, for example, retrieves no articles about a given topic for a certain period, this may be because the user has lost interest in it (permanent change) or because other things are temporarily more important (temporary change). This distinction between permanent interest change and the temporary change cannot be made by a system (or human) only *observing* the user's information intake behavior; only the users themselves may have this information.

If IF system designers want to improve system performance by distinguishing between permanent and temporary interest changes, then users have to provide additional high-level input about this factor. In the related work, two such approaches have been proposed. The music recommendation system LyricTime system creates, stores, and maintains *multiple profiles*—one for each of the user's mood [Loe92]. This approach requires users to manually communicate their current mood, so that the learning algorithm can assign the current input to the corresponding profile. Another approach is to allow users to "tweak" the automatically generated profile, e.g. by adding additional keywords ("augmented keyword search" [JH92]). This approach requires users to enter the additional keywords.

[5] In the original diagram in [JH92] the peak of the modeled interest has the same height as the peak of the actual interest. This is not always realistic. Unless a learning method overcompensates to attempt to reduce the learning delay, the learned signal will usually follow the original signal at any time instead of overshooting it. We adapted the diagram accordingly, so that the modeled interest now has its peak at the intersection with the actual interest.

Systems maintaining multiple profiles consisting of categories/queries (and producing multiple ranked outputs), e.g. the Newt system [SM93], have some system-inherent capabilities of handling repetitive interest changes. Since the individual categories are polled individually by the user, users can let their current moods determine their category browsing strategy. Users may, for example, check out currently preferred categories first and omit currently disliked categories. The additional flexibility is paid with the additional effort that users have to sift through multiple categories every time they use the system (see Section 2.1.2).

To summarize, IF approaches that learn only from relevance feedback are generally appropriate for handling gradual interest changes. Abrupt interest changes can be learned (unless they occur too frequently), but the delay required to gather a sufficient amount of training examples causes a period of reduced prediction quality. Repetitive interest changes are not recognized automatically, so unless users provide additional high-level information, these IF systems perceive repetitive interest changes as sequences of (permanent) interest changes that are learned over and over again, resulting in a continuous prediction error.

2.3.2 Advantages and limitations of attribute-level interaction

Filtering systems that interact with users on the basis of high-level entities, such as queries or rules, have different characteristics than systems learning from relevance feedback. While systems learning from relevance feedback have to *deduce* high-level entities from the given examples automatically, systems interacting with users at the level of attributes leave this abstraction task to the user.

Before we discuss the impact that this difference has on the handling of interest changes, we will briefly look at two general aspects of these systems. Systems requiring manual high-level interaction, e.g. rule-based approaches, have been criticized for requiring active participation of their users and for the relative complexity of the user interaction. There has been a longer discussion in literature on whether or not users of rule-based IF systems are able and willing to build sufficiently complex rules (e.g. [Lan95]). Although some researchers were able to provide evidence for that even casual users can deal with rules [MMC+89], it is still widely accepted that writing rules is more complex than the interaction required by systems learning from relevance feedback. On the other hand, attribute level interaction allows users to better monitor the state of their profile, which may make such profiles preferable in some situations (*transparent user profiles* [All90]).

How far can systems interacting with users at the attribute level adapt to interest changes? These systems have inherent advantages in handling abrupt interest changes. Since abrupt changes are caused by (internal or external) events, users may become aware of these interest changes through being aware of the events that caused them. This awareness allows users to initiate a user profile update *immediately* when the interest change happens. The period of reduced prediction quality that systems learning from relevance feedback suffer from can be avoided.

How efficient manual profile updates are depends heavily on the representation of user profiles. If a user profile is designed in a way that the user's interests map directly to the internal profile representation, then interest changes can be handled very efficiently. If, for example,

each of the user's interests is expressed by a single entity in the profile, e.g. a single query or rule, abrupt interest changes can be handled by only updating the respective query or rule (see Section 3.3.6). Less dense profile representations, e.g. profiles consisting of big *keyword* × *weight* vectors with keywords automatically selected by a full-text indexer, will not allow such efficient updates.

Gradual changes are generally handled less efficiently by systems interacting with users at the attribute level. Continuous change requires multiple successive adjustments to keep the prediction error small (Figure 9). Unlike systems learning from relevance feedback, systems interacting with users using explicit attribute level interactions do not have a continuous stream of information about the user's current interest. The actual interest signal is only sampled during profile updating, so that a prediction error can never be completely avoided. Furthermore, the successive profile updates that are required for approximating gradual changes cause a significant amount of user effort. And finally, being caused by continuous processes, users may not be aware of their gradual changes currently taking place and therefore miss to make the required updates.

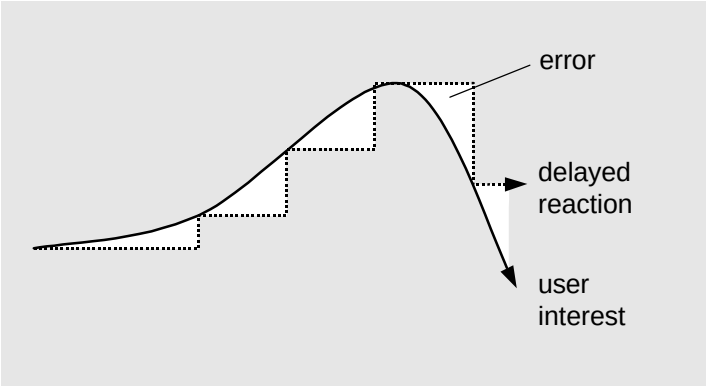


Figure 9: Prediction error introduced by IF systems using manual profile updating in the case of continuous interest change.

IF systems interacting with users at the attribute level can generally provide better support for handling repetitive interest changes. Assuming that past queries/rules/stereotypes are archived and made available, users can reuse these entities to rebuild past profile states or as ad hoc queries (e.g. Tapestry [GNOT92]).

Summarizing, the strength of IF systems interacting with users at the attribute level is that they support the handling of abrupt changes and also provide the general means required for supporting repetitive interest changes. On the other hand, gradual changes cause repeated user effort and prediction errors due to delayed profile updates. The overall user interaction, however, is more difficult for users to learn, because users have to deal not only with the objects themselves, but also with additional high-level entities, such as attributes, rules, or queries.

2.3.3 Comparison of related work with requirements and conclusions

Table 4 recalls the systems listed in Section 2.2 and summarizes which of the five requirements stated in Section 2.1 they fulfill. All systems provide support for at least some type of interest change.

We make the observations that most systems support *either* gradual changes *or* abrupt changes. Systems learning from relevance feedback usually support gradual changes, while systems interacting with users at the attribute level support abrupt changes. The systems that can deal with both types of interest changes are those that combine learning from relevance feedback with an attribute-level interaction method. In these systems, the relevance feedback method covers the gradual changes, while the high-level interaction method allows users to take care of abrupt changes. The *um movie advisor*, for example, allows users to manually modify their queries, but also to automatically update them based on relevance feedback. We can conclude, that a system supposed to support both types of interest changes has to combine relevance feedback interaction with some sort of high-level interaction.

Only very few systems provide support for handling repetitive interest changes. Those systems that are able to handle such variations have gained this capability through an additional feature that allows users either to add or remove attributes at the attribute level or to reuse a past user profile or user profile component.

These observations will provide us with a foundation for designing a new dynamic information filtering model and architecture that is optimized for the efficient handling of interest changes.

	User profile structure or filtering method	1. Learns gradual changes from rel. feedb.	2. Immediate handling of abrupt changes	3. Can Handle repetitive interest changes	4. Output style	5. Represents multiple interests correctly
Business Int. S. [Luh58]	SDI	3	3	-[3]	unranked	3
(*Memo filter*) [FD92]	LSI	(3)[1]	3	-	single	(3)[1]
SIFT (with RF) [YG95]	*n* queries	(3)[2]	3	-[3]	multiple	3
Bookmarks to search eng.	*n* queries	(3)[11]	3	-[3]	multiple	3
NewsWeeder [Lan95]	MDL	3	-	-	multiple	3
INFOS [Moc96]	*n* keyw.	3	3	-	single	-
Fab [BS97]	*n* keyw.	3	-	-	single	-
LyricTime [Loe92]	*n* attrib.	3	-	3[9]	single	(3)[5]
DIVA [NH98]	*n* attrib.	3	-	(3)[4]	single	(3)[5]
Browse [JH92]	neural n.	3	-	3[10]	multiple	-
SIFTER [LMMP96]	2 layers	3	(3)[6]	-	weak	3
(*News filter*) [Bac91]	agents	3	-	-	single	3
Newt [SM93]	agents	3	-	-[3]	multiple	3
WebMate [CS98]	*n* vectors	3	-	-	single	3
GroupLens [RIS+94],	ACF	3	-	-	single	3
MovieLens [GSK+99],	ACF	3	-	(3)[4]	single	3
Ringo [Sha94]	ACF	3	-	-	single	3
(*Mail filter*) [Den82]	undef.[7]	-	3	-	weak	3
Info. Lens [MGT+87]	rules	-	3	-	unranked	3
ISCREEN [Pol88]	rules	-	3	-	unranked	3
Mafia [LKH90]	rules	-	3	-	weak	3
INFOSCOPE [FS91]	rules	(3)[8]	3	-[3]	unranked	3
Tapestry [GNOT92],	contin. q.	-	3	-[3]	weak	3
Gnus score files [Gnus]	rules	(3)[8]	3	-	weak	3
GRUNDY [Ric79a]	stereoty.	-	(3)	-	single	-
Um movie advis. [Kay95]	hybrid	3	3	-	single	3

Table 4: Summary: comparison of reviewed systems with requirements (3 = supported, (3)= partly supported, **-** = not supported). [1] only in relevance feedback mode, [2] relevance feedback on individual queries only, [3] minimum representation of temporary variations by selecting a currently interesting category or ad hoc queries, [4] additional exact match filter that is used in an IR fashion, [5] restricted attribute set, [6] interest shift detection requires multiple relevance feedback units, [7] Denning's system has not been implemented (concept only), [8] recommends rule updates, [9] multiple profiles for individual moods, [10] users can manually activate keywords of profile network, [11] depends on refinement mechanism provided by of search engine.

CHAPTER 3 THE *QUERYSET* INFORMATION FILTERING ARCHITECTURE

In this chapter, we will present our approach to dynamic information filtering. We will begin by motivating our approach.

3.1 EXCURSUS TO COMPUTER ANIMATION

The central piece of our IF architecture is its user profile data structure that is designed to maximize the system's capability to adapt it to the user's interest changes, thereby maximizing the prediction quality during periods of interest change. To illustrate the underlying design principle, we will use an analogy between information filtering and computer animation. We will show that some of the ideas and strategies behind computer animation also apply to the IF context; the architecture that we will present in this thesis is directly motivated by the architecture of 2D-animation systems. This metaphor will also allow us to derive user interfaces concepts from user interfaces used in computer graphics applications (Chapter 4).

3.1.1 Similarities between information filtering and digital image processing

The basis for our analogy is the structural similarity between digital image processing (e.g. [Fol90]) and IF. In both application areas, users have a data structure in their minds, i.e. either an image or a *relevance function* (see Section 1.1.1.2). In both application areas, this data structure has to be entered into a computer to allow processing it with the help of the computer. The goal of system and user is to make the computer-based representation the closest possible approximation of the data structure that the user has in mind. Although the resulting representations, i.e. digital images and user profiles, are used for very different purposes, the problem of entering them and the solutions found for implementing the interface process show a number of similarities in both application areas (Table 5).

	Computer graphics	Information filtering
What is in the user's head…	The *image* in the user's head defines a *color value* to arbitrary *image coordinates*.	The *relevance function* in the user's head defines a *relevance value* to arbitrary *objects*.
…is modeled using a computer	*Graphics programs* allow users to model this mental *image* as a *digital image.*	*IF systems* allow users to model this mental *relevance function* as a *user profile.*
Sampling-oriented perspective	Users define the *digital image*, by entering *pixel colors* (e.g. using *painting*). The overall image is then interpolated (e.g. when zooming). (→Painting programs)	Users define the *user profile* by entering *object relevance values* (*relevance feedback*). The overall relevance function is then interpolated. (→ACF, feature extraction)
Object-oriented perspective	*Graphical objects* define the image in the user's head. Users enter the digital image by entering a *scene graph* consisting of *graphical primitives* and *transformations*. (→Drawing programs)	*Interests* define the relevance function in the user's head. Users enter the relevance function by entering a *rule base* consisting of *rules/stereotypes* and *priority rules* (→Rule/stereotype-based systems)

Table 5: Analogy between user profiles and digital images

Using this metaphor, many of the systems from related work (Chapter 2) can be described in terms of image processing systems. Some systems sample and shade (interpolate) the user's image (e.g. feature extractors), create phantom images by complementing the user's image with the images of other users (automated collaborative filtering), allow users to create images of overlapping flat-shaded shapes (rule-based systems), or allow building images from a selection of predefined textured shapes (stereotype-based systems)[6].

3.1.2 Interactive computer graphics

As discussed in Section 2.1, the user's relevance function may change over time. This means that the user's profiling task in IF does not correspond to entering a single image, but an image that changes over time. In analogy to the interest changes in IF that we had mentioned in Chapter 2, a mental image of a face, for example, may change *gradually* when the face ages, *abruptly* as a result of an surgical operation, or *repetitively*, e.g. when smiling. Since interest

[6] One difference between sample-based representations in image processing and in IF is that interpolation in images is rather straightforward, because the (Euclidean) topology is given. For IF systems, constructing this topology is an additional task that may require additional input from the user. This also means that interaction techniques applicable to images that are based on an image's topology, e.g. graphical overviews or painting, are not necessarily applicable to user profiles. The fact that images are two-dimensional is of no importance for our analogy and user profiles are not assumed to actually have such a topology. The two-dimensionality of images merely simplifies illustrations.

changes are not known in advance, the corresponding task in computer graphics corresponds to creating an *interactive* animation, as used for example in video games. Two of the main objectives for interactive graphics are high interactivity (rapid reaction to input) and graphical quality. These requirements correspond to the dynamic filtering requirements rapid adaptation to interest change and retrieval quality (see Section 2.1).

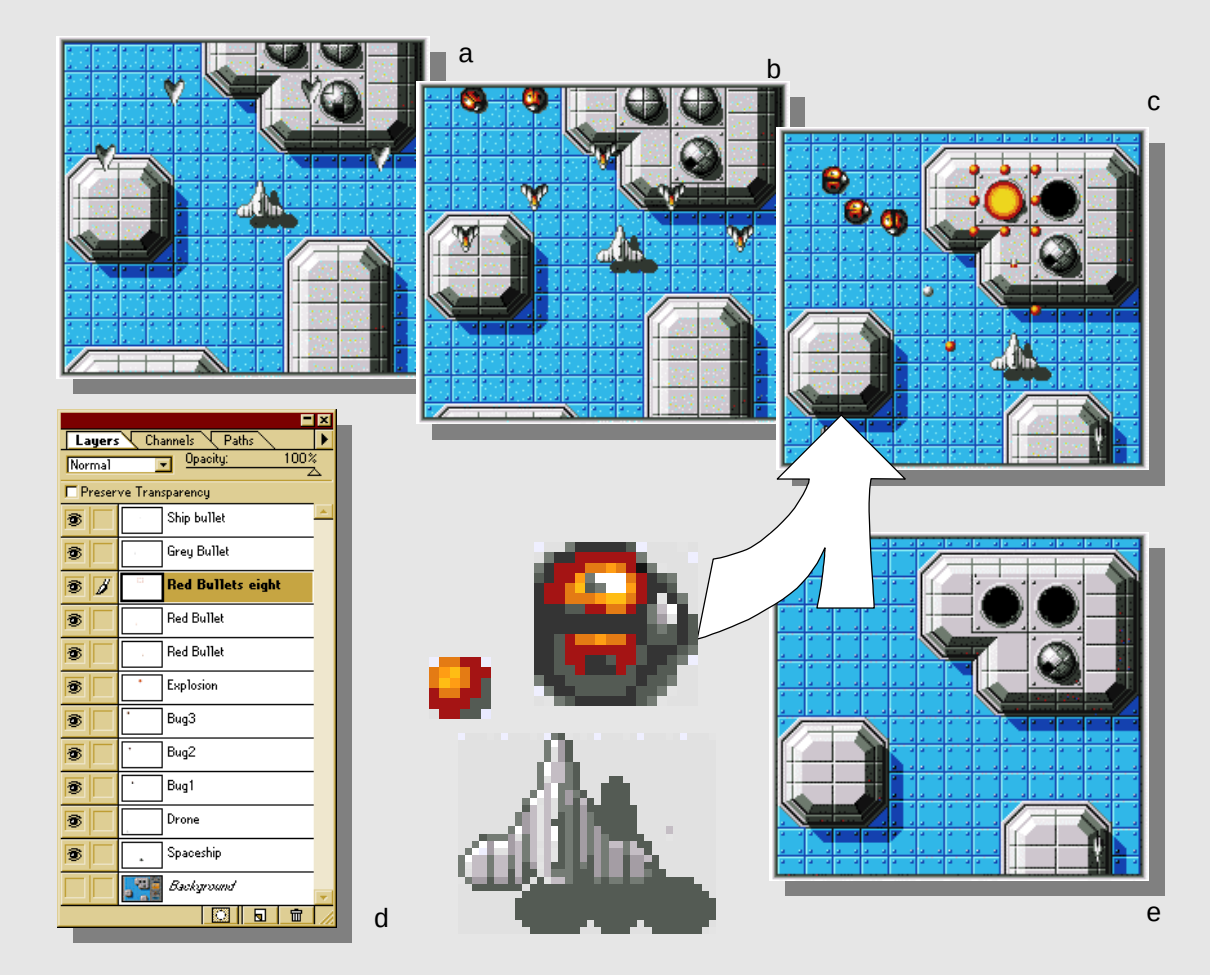


Figure 10: Fast animation and interactivity by decomposing an image into components that change as a whole (layers). Screenshots taken from the video game *Xenon* © The Bitmap Brothers 1988 (a-c, e) and from *Adobe Photoshop 4.0* [Adobe96] (d).

To satisfy these two requirements in computer graphics, hybrid systems have been built that combine sample-oriented and object-oriented perspectives. Purely sample-oriented graphics lack high-level structure and are therefore inefficient to modify; purely object-oriented graphics make it difficult to model realistic details. To combine the strengths, both approaches were extended, assimilating the characteristics of the other approach.

- Drawing programs were extended with so-called *texture maps* [Fol90]. Instead of modeling the full detail of an object using large amounts of geometric primitives, a rather rough geometrical model is used that is covered with a pixel-oriented image called texture map[7].
- Painting programs were extended with *layers* [Adobe96]. Images are represented as a set of depth-ordered layers that are combined with dedicated layer combination functions and that can have pixel-wise transparency. Layers are only combined into a single bitmap (rendering) when viewed or printed. The set of layers and their combination functions may be considered a simplified scene graph. They allow preserving some structure of the content shown in the image and therefore enhance the modifiability of the image.

In the resulting systems, the object-oriented aspect provides modifiability and interactivity; the sampling-oriented aspect provides the realistic details. This hybrid architecture is the basis for 2D-animation systems (e.g. Macromedia Director [SW00]) and for many interactive applications, e.g. the video game shown in Figure 10. The individual frames (Figure 10a to c) are composed by overlaying several 2D objects (Figure 10e) that are stored in individual layers (Figure 10d).

The user effort for creating the individual layers is usually bigger than the effort for painting a single frame, because overlapping regions are painted once for each layer. This additional effort, however, pays off when the scene is animated. Instead of repainting each individual animation frame, animations are created by updating only selected layers; layers not affected by the change require no updating. Furthermore, if the change can be represented at the level of the scene graph, layers can be updated very efficiently. Often it is possible to translate, fade in or out, or taint a layer, instead of entirely redoing it. To maximize the benefit of the layering concept, attention has to be paid to how a scene is decomposed. Those components should be grouped into the same layer that will change as a whole. In the scene shown in Figure 10a-c, for example, the space ship in the center of the scene and its shadow belong to the same object and are therefore represented as a single layer. The hostile spacecrafts move independently and should therefore be represented using multiple layers.

In the following section, we will transfer this concept of interactive computer graphics to information filtering. The decomposition of a scene into multiple layers and a scene graph will then correspond to a decomposition of the user profile into multiple queries and a so-called aggregation function.

3.2 THE QUERYSET MODEL

The filtering architecture we propose is called *QuerySet Architecture*[8] (QSA) and corresponds to the 2D-animation concept discussed in the previous section. QSA user profiles are decomposed into multiple queries and a so-called aggregation function to allow restricting profile updates to those components that actually were affected by interest change. QSA systems can

[7] This approach easily extends to 3D, but because of the lack of the regular topology of images, translations and 3D rendering are not applicable to IF. We will therefore focus on 2D.

[8] Credits to Joe Konstan for suggesting this name.

learn user profiles not only from relevance feedback (RF), but also from high-level input, so-called user-aggre*gated relevance feedback* (see Section 3.3.6). The latter allows users to update their profile directly and efficiently, thereby allowing users to effectively make major profile updates as they are useful during profile initialization and after interest changes.

3.2.1 Decomposition of user profiles into queries and aggregation function

To describe the data structure used to represent QSA profiles, we will use a graph structure as shown in Figure 11a. The figure shows a *general* model of the information filtering process, i.e. how IF systems in general compute object ratings. An edge from some nodes A_i to some other node B indicates that B is computed as a function of A_i. During indexing, objects (the d-nodes) are assigned low-level attributes and weights (r-nodes). The *user profile* takes the *attribute* × *weight* pairs as input and aggregates them into an *output rating*. In the typical case, output ratings are scalars, so that objects can be ranked according to their output rating.

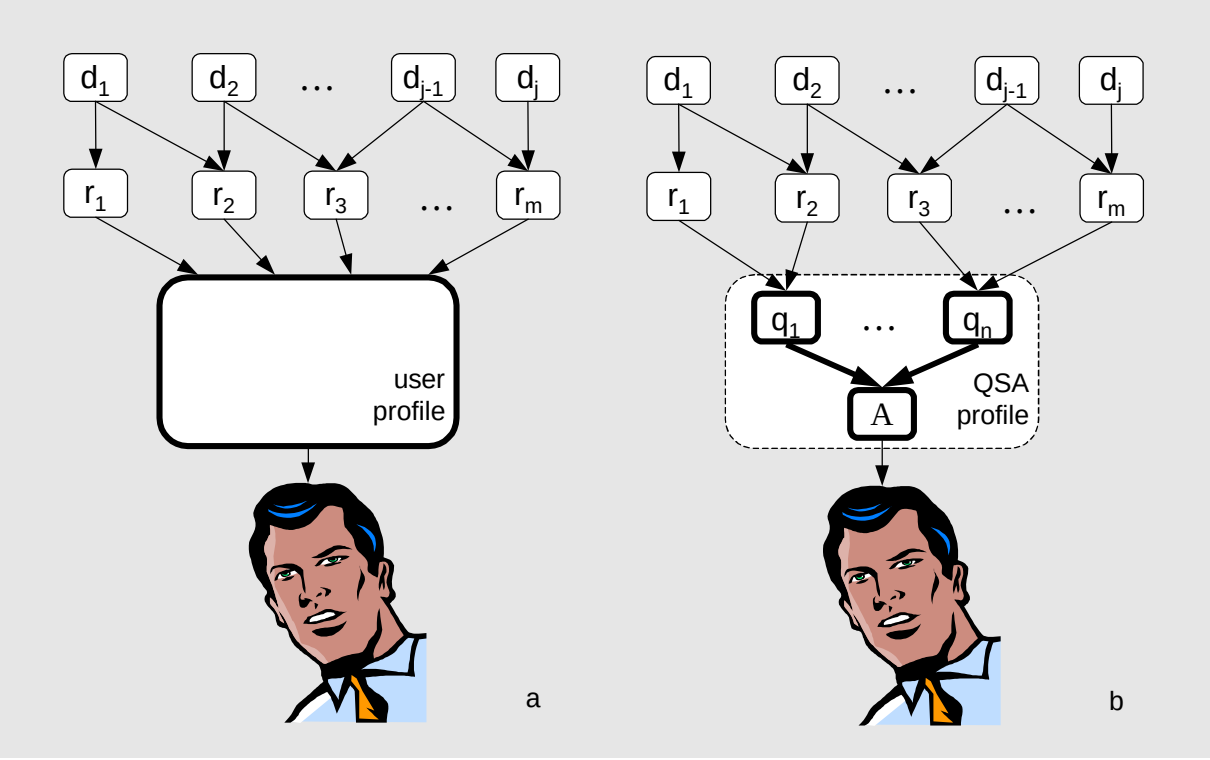


Figure 11: The general IF model (a) and the dynamic information filtering model of the QSA (b)

The diagram style we use in Figure 11 is also used for describing inference networks implementing probabilistic retrieval methods (e.g. [TC91], see also section 3.3.3). We have chosen this diagram style for its visual clarity, but do not want to suggest a specific type of computation. The dependency between nodes expressed by edges not necessarily has to be a probabilistic one; any other filtering technique may be applied equally well, e.g. the ones discussed in Chapter 2.

Figure 11b shows the structure of *QSA user profiles* that is obtained by decomposing the general profile. A QSA profile consists of a *collection of queries* plus an *aggregation function*. The term *query* is here used with a very general meaning denoting "a function that assigns ratings to objects". Each query takes object *attribute* × *weight* pairs as input and condenses them into a single *query rating*. The aggregation function aggregates all query ratings into a single *output rating* that is used to compute the output ranking.

Queries in a QSA profile are intended to represent *multiple* interests (see Section 3.2.3). This distinguishes them from query representation nodes that are used by some types of inference networks and that represent multiple representations of the *same* interest (see also Section 3.3.3). Queries are also different from *concepts* (or *facets* [MCBC89]) that are *part* of a query that represents an interest. By holding representations of multiple interests, a QSA profile is different from an IR query that represents a single interest only.

3.2.2 The objective of the QSA is to maximize adaptation speed

The goal of the user profile decomposition in the QuerySet Architecture is to *maximize the speed of updating operations*, very similar to the way the decomposition into layers in 2D-animation programs allows for high interactivity. Since QSA systems provide direct access to queries (see Section 3.3.2 and 3.3.3) *and* aggregation function (user-aggre*gated relevance feedback*, see Section 3.3.6), interest changes that affect only parts of the profile can be handled by updating only these affected parts; the required updating effort is reduced. There are two classes of interest change that can be handled especially efficiently.

1. **Only the aggregation function, but no individual query is affected.** If an interest change affects each query only as a whole, i.e. if the rank orderings of objects matched by the same query remain intact and only the relation *between* queries is affected, then it is sufficient to update the aggregation function. Typically, such cases are the complete or partial satisfaction of the interest represented by a query or an increased interest in it[9]. The aggregation function typically aggregates fewer dimensions (the number of queries) than the complete profile (the number of basic object properties, such as keywords). Updating the aggregation function therefore requires less input and can therefore be faster than learning a non-decomposed profile. If a canonical representation of the aggregation function can be found (see Section 3.3.1), the updating effort caused by an interest change may even be reduced to an effort that is linear in the number of affected queries [TC91, p. 281].
2. **Only a subset of queries is affected, in the optimum case only a single one[10].** If an interest has changed, only the queries representing that interest require updating. Queries not affected by the interest change are not updated, which saves updating effort and preserves the filtering quality of these queries during the adaptation process following interest changes.

---

[9] The corresponding case in computer graphics is when layers are only affected as a whole, so that all adjustments can be made in the scene graph. One or more objects may, for example, be faded in or out which could be represented by adjusting layer opacities.

[10] The corresponding case in computer graphics is when only some layers are affected. A sprite may for example be animated by flipping through a series of bitmap images.

To make interest changes fall into these two cases, the user's interests (that can then be represented as queries in the user's profile) are required to be at least partially mutually independent. The bigger the amount of mutual independence, the bigger the benefit of the decomposition.

3.2.3 Query selection: profile design for modifiability

To maximize the benefit of the QSA, the number of affected queries has to be minimized, i.e. the number of the cases fitting into the two categories described above has to be maximized. Therefore, the proper selection of queries is crucial. In general, the system's adaptation performance increases the better the correspondence between queries and interests is.

1. A query has to be reformulated if only a part of it is affected by change; if the entire query is affected uniformly, the query remains and it is sufficient to update the aggregation function. The more interests are overlaid in a query, the more likely the query is to be "ripped in two" by an interest change, causing the need to reformulate it. *Whenever possible, each query should therefore represent only a single interest.*
2. The more queries are related to the same interest, the more queries have to be updated when this interest changes. *Whenever possible, each interest should therefore be represented by only a single query.*

We will illustrate these query selection principles with some examples. We will begin with two negative examples, i.e., query selection strategies that are *not* appropriate for selecting QSA queries. (1) In *data fusion* (e.g. [BCCC93, BCB94, FS94, Lee97]), multiple queries redundantly represent the same interest with the objective to improve retrieval quality. If this interest changes, all queries are affected and have to be updated. (2) In automated collaborative filtering (see Sections 1.1.2.2.2 and 2.2.4), a user's neighborhood consists of users that are similar to this user. A single interest change may cause the user's entire neighborhood to change so that the profile has to be redone entirely.

A *good* correspondence between queries and interests is encouraged by the QSA user interface structure. Before users insert a query into their QSA profiles, they first run and refine the query interactively (see discussion of evolving usage in Section 3.4.1). Since ad hoc querying generally aims at a single interest at a time, this incremental profile construction procedure encourages an interest-oriented profile composition. At the example of a TV recommendation system, a QSA user profile may for example consist of the following queries (example taken from TV Scout): {genre(MartialArts), genre(ComedyMovies), textsearch("HarrisonFord"), textSearch("TheSimpsons")}.

The more mutually independent interests are represented in a QSA profile, the smaller the percentage of profile that is affected by an interest change and the bigger the benefit over a non-decomposed profile. This makes the QSA especially useful for filtering applications supporting a wide range of interests, e.g. in such generic application areas as TV, or the World Wide Web (see Section 5.1). When starting to use a QSA system, however, users start with a single interest, so that the user's QSA profile consists of only a single query. Is this situation, all interest change affects this query and all relevance feedback from the user is passed to the refinement mechanisms of that query (see Section 3.2.4). The additional component that

makes the system a QSA system, i.e. the aggregation function, is constrained to monotonic functions and therefore never has any effect on the output ranking of a single query. Since it requires no user input in the single query situation, the performance of QSA profiles does not drop below the performance of a system handling only the single query.

3.2.4 Query execution and refinement are handled by external subsystem

Queries and query formulation have been discussed in detail in the IR and IF literature (see Section 1.1). Since the QuerySet Architecture implies no particular requirements on queries that could not be satisfied by existing work, queries will *not* be the subject of this thesis. To minimize the development effort, query execution may be delegated to one or more subsystems capable of executing queries. Our wide definition (Section 3.2.1) of the term *query* allows including any retrieval or filtering subsystem, collaborative filtering system, databases, etc.

QSA systems delegate all query-related tasks to the respective subsystems. Subsystems are responsible for indexing, execution of queries, and are encouraged to provide a refinement method based on relevance feedback (RF). Query refinement from RF is a well-known technique in information retrieval (e.g. [Sal68, Sal71, SB90, FB90, HC93]), so we will not repeat the involved strategies. In Chapter 5, we will demonstrate how subsystems can be embedded into a QSA system and which user interfaces can support users in formulating queries, inserting queries into, and deleting queries from QSA profiles.

One of the strengths of the QuerySet Architecture is that it allows combining multiple retrieval/filtering techniques in a single QSA system. The TV Scout system, for example, simultaneously uses a database, a retrieval system, and an application-specific collaborative filtering subsystem (see Chapter 5). Users may combine queries that are executed on all these subsystems in a single QSA profile. See Section 3.3.4 for the rating conversion functions that make this possible. Since objects are handled as fully encapsulated (object, rating) pairs, query-executing subsystems may even deliver *different* object types. A user may, for example, subscribe to a stock information service, a TV recommendation system, and a news service. The QSA system then aggregates all these different sources into a single ranking.

3.3 THE AGGREGATION FUNCTION

The aggregation function compiles multiple query ratings into a single output rating to allow all objects to be ranked uniquely (see also [GCG96, GG98]). If a profile consists of, for example, three queries, the purpose of the aggregation function is to answer questions of the type “Is an object rated (0.3, 0, 0) more relevant than an object rated (0, 0.2, 0.4)?”

To simplify the implementation of QuerySet Architecture systems, it is desirable to also delegate the functionality of the aggregation function to an external subsystem. Since the aggregation function is structurally similar to a query insofar as it maps a larger number of input parameters to a single output parameter, best match IR or IF systems are potential candidates for implementing the aggregation function subsystem. Since the aggregation function takes query

ratings as input parameters, while a query takes basic attributes, such as term frequencies as input, rating conversion may be required (see Section 3.3.4 and Figure 14 on Page 53).

3.3.1 Classification of the relation between queries

If a QSA user profile contains *n* queries, each object is characterized by *n* query ratings. These query ratings span an *n*-dimensional *query rating space*. The purpose of the aggregation function is to define an output rating for each point in this space, so that objects with arbitrary query rating vectors are assigned an output rating.

How much input from the user is required for obtaining a representation of that space depends on how it is modeled. One approach is to sample the entire space and to interpolate output ratings between samples (Figure 12a), but due to the large number of required samples, learning that representation would require a huge amount of user input. Gathering this input might also take too long for handling interest changes appropriately.

Similar problems are also encountered by IF systems that use feature-extraction (see Section 2.2.1). Although the QSA aggregation function is different from these IF systems in that it attempts to learn a map from *query rating vectors* to output rating, while general IF systems attempt to learn a map from *feature combinations* to output ratings, the basic problem of sampling a large space is the same. We will therefore look at what strategies these systems use to deal with the complexity of the space to be learned. IF systems use different types of simplifications. The first simplification is that ome systems, such as such systems [TC91, Bac91] sample only the corners of the vectors space (see Figure 12b).

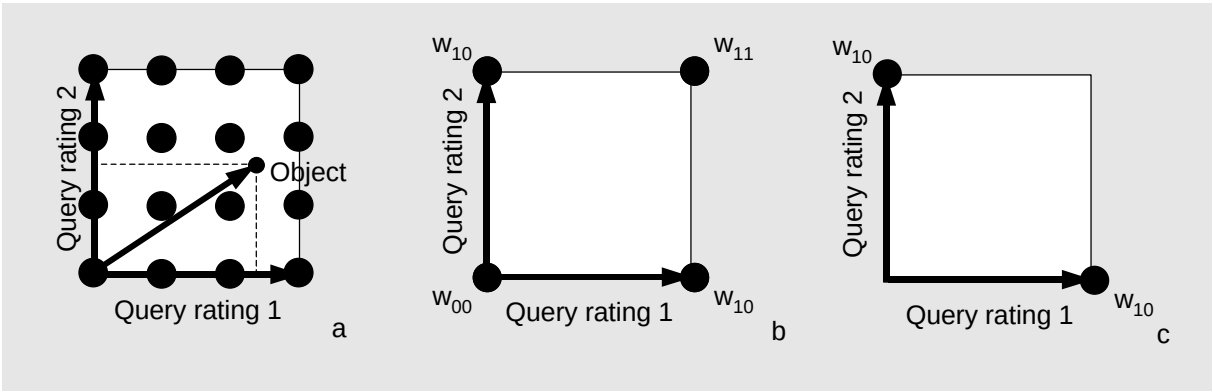


Figure 12: The aggregation function can be represented using different levels of detail. The *n* query ratings assigned to each object span an *n*-dimensional vector space; here *n*=2. Large black dots denote samples of query rating vectors that the aggregation functions assigns an output rating explicitly; values in between are to be interpolated. Sampling arbitrary query rating vectors (a), reducing the sampling rate to two samples per dimension (b), and using a canonical representation to reduce the sampling rate to a single sample per dimension (c).

Sampling the space spanned by *n* features (or queries, in the case of the QSA), however, still requires 2^n samples. To avoid requiring an exponential amount of user input, a second simplification is often made. Many systems assume the output ratings of objects containing multiple

features to be a predefined function, such as weighted sum, of the ratings assigned to objects matching only one of the features. Several *IF system*, such as [JH92, Moc96] use this approach (see the King and Queen problem discussed in Section 2.1.3), but also many *IR systems*, such as the vector space model (see Section 1.1.2.1.1). This leads to a linear number of samples (Figure 12c), but the price for this simplification is that the resulting model looses its arbitrary to assign different output ratings to *combinations* of query ratings.

To simplify the discussion of candidate representations for implementing aggregation functions, we will provide a rough three-level classification of the assumptions that IR/IF architectures imply if applied to the QSA. We will refer to the relation between queries as the *query interaction*.

1. **Mutual independence:** Users are assumed to profit from each relevance aspect of an object independently. It is assumed that there is no redundancy between queries so that ratings are additive and all ratings are treated uniformly. Since this assumption results in the simplest implementations, this model is very popular in both IR and IF systems. Depending on the underlying relevance model, ratings may be transformed before being summed. Some retrieval systems assume term frequencies or their logarithms to be directly additive, e.g. the weighted request and indexing retrieval model ([Bla90], see also Section 3.3.2). Also some filtering systems handle ratings this way (e.g. [Moc96, JH92, Bal97], see Chapter 2). Probabilistic retrieval systems making this assumption compute the resulting probability of relevance as the negative product of the parent node probabilities, i.e. $p(r) = 1-(1-r_1)\cdots(1-r_n)$ (e.g. [TC90]). This model is isomorphic to the additive model if ratings are transformed with $v(r) = -\ln(1-r)$ (see Section 3.3.4). In utility theory, the additivity of utilities is described as *mutual preferential independence* [KR93, p. 104]. While the mutual independence assumption is hardly ever realistic, this model can be useful as the respective queries *approach* mutual independence.
2. **Existence of redundancies**: The relevance that multiple queries assign to an object may be based on the same object properties, so that at least part of the object's rating is explained by redundancy between queries. In this situation, the relevance of an object having received ratings r_1 and r_2 may range between the additive rating (zero redundancy) and the maximum of both ratings (full redundancy, e.g. q_1 implies q_2). A retrieval method that allows representing such redundant relations is the extended vector space model [SM83]. By allowing p to vary, the **or**(p) connective becomes an extension of the **or** connective from fuzzy-set theory [Boo80, Boo81, WC79, BCH92, BG89] and can range from maximum (**or**(∞)) to sum (**or**(1))[11]. In QSA systems, redundancy between queries cannot always be avoided. Although QSA systems should support users in building QSA profiles such that queries and interests have a one-to-one match, this is not always possible. A query that would represent the user's interest exactly may, for example, not be available in the system. Users may therefore approximate the sought query with multiple overlapping queries.

[11] **And** connectives in general play no role for implementing the aggregation function. If a match with query1 were only useful in combination with a match with query2, then query1 would not represent an interest, but only a facet of an interest. In this case, query1 and query2 should both be removed from the QSA profile and replaced by a single new query that is defined as the conjunction of queries1 and 2.

If the retrieval model implementing the aggregation function is able to represent redundancies, such an approach can be handled correctly.

3. **Existence of implicit interests**: Combinations of query matches may have specific relevance to the user. A user having the two queries *action* and *comedy* in the profiles, for example, may actually *also* be a fan of *action comedies*, although there is no query *action comedy* in the profile that would make this interest explicit. We call this an *implicit interest*. Since the effect may also occur in the opposite direction, i.e. a user liking *action* and *comedy* but disliking *action comedies*, implicit interests extend the relevance range of objects matching multiple objects to arbitrary ratings. Some filtering methods allow learning arbitrary relations between terms, e.g. the agent-based filtering method by Baclace (see Section 3.3.3).

In the following, we will describe two possible implementations. The first one will assume mutual independence; the second will support redundancy and implicit interests.

3.3.2 Model 1: Implementation based on the weighted request and indexing retrieval model

For a simple, thus efficient implementation of the aggregation function, the *weighted request and indexing retrieval model* (WRIR) may be used [Bla90]. Using this model, the output rating for an object with the query rating vector $\mathbf{r}=(r_1,\ldots, r_n)$ is computed as $r_1w_1+\ldots+ r_nw_n$, with w_i denoting the *query weights* that are stored in the user profile[12]. If a QSA system uses probabilistic relevance to implement the aggregation function, the WRIR model can be adapted by using the negative product $1-(1-r_1)\ldots(1-r_n)$ instead of the sum and $1-(1-r)^w$ for weighting query ratings (see also Section 3.3.4). The weighted request and indexing retrieval model can be provided with a relevance feedback learning mechanism, i.e. by increasing the weights of those query ratings that contributed to the correct prediction of a relevant object [SB90, FB90, HC93, Ols98].

When a new query is inserted into a QSA profile, its query weight can be initialized according to Zipf's law [Sal83, p. 60]. Applied to the aggregation function, Zipf's law assigns higher query weights to more specific queries, i.e. queries matching fewer objects. This initialization keeps objects delivered by very specific queries from being buried among the objects delivered by less specific queries.

Since the WRIR model represents the relation between all queries uniformly, redundancies and implicit interests require special attention.

---

[12] The WRIR model is a predecessor of the vector space model [Sal68]. The difference of the WRIR compared to the vector space model is that resulting ratings are *not* post-normalized with $1/(|\boldsymbol{r}|_2\ |\boldsymbol{w}|_2)$. While a normalization with $1/|\boldsymbol{w}|_2$ has no impact on the output ranking (this factor is the same for all objects), normalizing with $1/|\boldsymbol{r}|_2$ will assign a higher output rating to an object with query ratings $r = (w_1,\ldots,w_n)$ than to an object $r = (w_1,\ldots,w_i+k,\ldots,w_n)$, $k>0$. The objective behind the cosine measure, i.e. to obtain a perfect *mixture* of matches, was designed for combining terms or concepts. For combining the ratings of queries representing different interests, however, an aggregation function assigning a higher output rating to the object with the query ratings $(w_1,\ldots,w_i+k,\ldots,w_n)$, $k>0$ is preferable.

- The WRIR represents query interactions incorrectly if there are redundancies between queries. Since the sum function is an upper bound for both maximum function and itself, predicted ratings will be higher than or equal to the correct rating. Objects matching multiple redundant queries may therefore appear earlier in the output ranking, but not later, so that the problem of judging objects matching multiple objects is implicitly passed to the user.

  If users have groups of mutually redundant queries in their profile (to increase the recall of one interest, i.e. a data fusion approach), the erroneous summation of redundant ratings may be avoided by combining these queries into a single disjunctive query (see [Bau99b]). Furthermore, if there is a certain level of redundancy between most queries in the profile, it may be possible to decrease the overall error by aggregating query ratings with a *p*-norm where $p>1$.

- The WRIR model is not able to correctly represent implicit interests. In case implicit interests occur, they can be handled by introducing an additional conjunctive query that represents the implicit interest, thereby making it explicit. In the example from Section 3.3.1, this would mean to add the query *action comedies* to the profile. This approach is basically the manual version of what methods capable of learning query interactions do automatically (see Section 3.3.3). To handle *negative* implicit interest, the two original queries may also have to be replaced by more restrictive queries that avoid matching the objects belonging to the implicit interest.

Since the WRIR model is a simple retrieval model that allows for a simple implementation, it is first choice, whenever the loss of prediction quality caused by inflexible representation of the query interaction remains small. This is the case if the prediction error of objects matching multiple objects is small or if the case simply occurs very seldom, i.e. if the number of objects matching multiple queries is small. In the case of the TV Scout system (Chapter 5), we were able to provide evidence for the latter case. Over 92% of all objects matched by a user profile were matched only by a single query within that profile (see also Section 0). Since most users' QSA profiles matched less than 200 programs per week, only about 16 programs would be ranked too high, which we considered too little to justify a more complex aggregation model. We consequently used the WRIR model to implement the aggregation function in the TV Scout system.

3.3.3 Model 2: Implementation based on an inference network

In cases where the portion of objects matching multiple queries is considerable, and where these objects have relevance values substantially different from the sums of their individual ratings, the handling of objects matching multiple queries may better be handled by a subsystem capable of representing query interactions more flexibly. One way of implementing this is by using an inference network [TC90, TC91] (see Section **1.1.2.1.2**). The overall structure of inference networks, i.e. a directed acyclic graph (Figure 13a), complies with a QSA implementation.

There are different ways of representing a QuerySet Architecture based on an inference network. The inference network structure proposed by Turtle and Croft contains a layer of query

representation nodes (the "q" nodes in Figure 13a) that are designed to each hold a different representation of the interest (see Section **1.1.2.1.2**). These nodes may be (mis)used to represent the QSA queries so that these nodes now represent *different* interests, instead of a different *representation* of the *same* interest. Alternatively, the functionality of the query representation nodes may be preserved to allow for additional grouping of queries representing the same interest. In this case, an additional layer of nodes has to be introduced as shown in Figure 13b. In the shown graph, the interest nodes I_1 to I_k represent the individual queries of a user's QSA profile and an additional node, i.e. the *aggregation node*, represents the output rating. Although nodes in these two models bear different names, they have the same functions (nodes with thick borders and nodes with dashed borders in Figure 13a and Figure 13b).

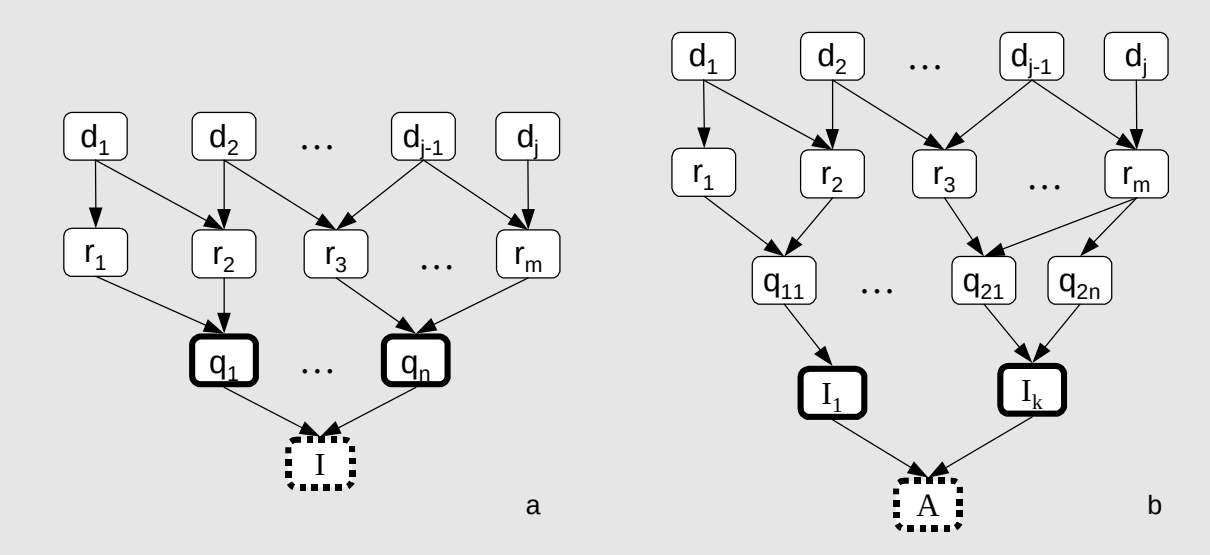


Figure 13: Inference networks (four-layer version, [TC92, p. 282]) may be used to support a QuerySet Architecture (a). QSA-enhanced inference network (b). Thick-bordered nodes implement QSA queries, dashed nodes the aggregation function. (d = document nodes, r = concept representation nodes, q = query representation nodes, I = Interest node. A = Aggregation node)

The aggregation node of a QSA inference network is substantially different from the interest node of the original inference network. The original interest nodes implement a data fusion function, i.e. they combine the evidence represented by the individual query representation nodes. This allows for a relatively simple implementation. In the related work, simple, predefined, symmetric functions, such as sum or max, have been successfully used for data fusion purposes (see [FS94] for a comparison of five such functions; see [Lee95] for a comparison based on normalized ratings). Since an aggregation node's parent nodes represent different interests, representing them with predefined, symmetric node functions will generally *not* be appropriate. The interests represented by different query nodes may imply different relevance to the user and, as discussed in Section 3.3.1, there may be different types of query interactions. Therefore, ratings of parent nodes may have to be weighted and different combinations of them may have to be assigned different relevance values. Since the aggregation node depends on user and queries, predefined node functions are *not* appropriate; the aggregation

node has to be defined for each individual user and it requires to be maintained through interest changes. With their capability to assign arbitrary ratings to all 2^n combinations of query ratings, however, link matrices (see Section 1.1.2.1.2) provide the functionality required for modeling QSA aggregation functions that include redundancy and implicit interests.

Since aggregation functions are query- and user-specific, they have to be created and maintained for each user individually. This brings up the question how to manage the exponential number of values in the link matrix. The agent-based IF architecture by Paul Baclace offers a solution for that problem ([Bac91, Bac92], see Section 2.2.1). The exponential learning effort is avoided by learning values only for a canonical representation in the first place; this representation is only enhanced where the predictions made on the basis of the canonical representation fail.

When used to implement a QSA aggregation function, the profile learning method is applied to query weights instead of basic object properties. The result is the following extension to the weighted request and indexing retrieval model. Beside the weights that are assigned to each query (the simple agents) the user profile contains additional weights that are assigned to *combinations* of query matches (the conjunction agents). Baclace's selection mechanism is used to only maintain those conjunctive agents that make predictions that are correct and substantially different from what the system would have predicted without them (see [Bac91] for details). Baclace showed that this learning method is able to learn arbitrary feature combinations including exclusive **or** functions [Bac91], which implies that the method is able to learn aggregation functions that include redundancy and implicit interests.

To combine this learning method with an inference network, two conversions are necessary. First, the two rating models have to be made compatible. Baclace's learning method uses an additive rating model (the partial ratings provided by the individual agents are added, sum ratings can reach arbitrary values), while inference networks use a probabilistic relevance notion (probabilities are normalized to the range from zero to one). See Section 3.3.4 for a description of how such a conversion can be constructed. Second, the representations of user profiles have to be made compatible. Inference networks use link matrices, i.e. one value *for each combination* of parent node truth values (see Section **1.1.2.1.2**); Baclace method uses a denser representation where the values for combinations of parent values are represented as a sum of all matching agents (see Section 2.2.1). To expand Baclace's representation into link matrix form, each entry of the link matrix has to be computed by summing up the bids of all simple and conjunction agents contributing to this feature combination.

3.3.4 Rating conversions

In the QuerySet Architecture design presented so far, three main entities are involved, i.e. the subsystem(s) executing the queries, the subsystem implementing the aggregation function, and the user[13]. All three entities may use their own relevance models. Unless two entities exchanging ratings use the same relevance model, exchanged ratings have to be converted (Figure 14).

[13] Additional components may exist if any of the subsystems consists of multiple sub-components, e.g. as described in Section 3.3.3.

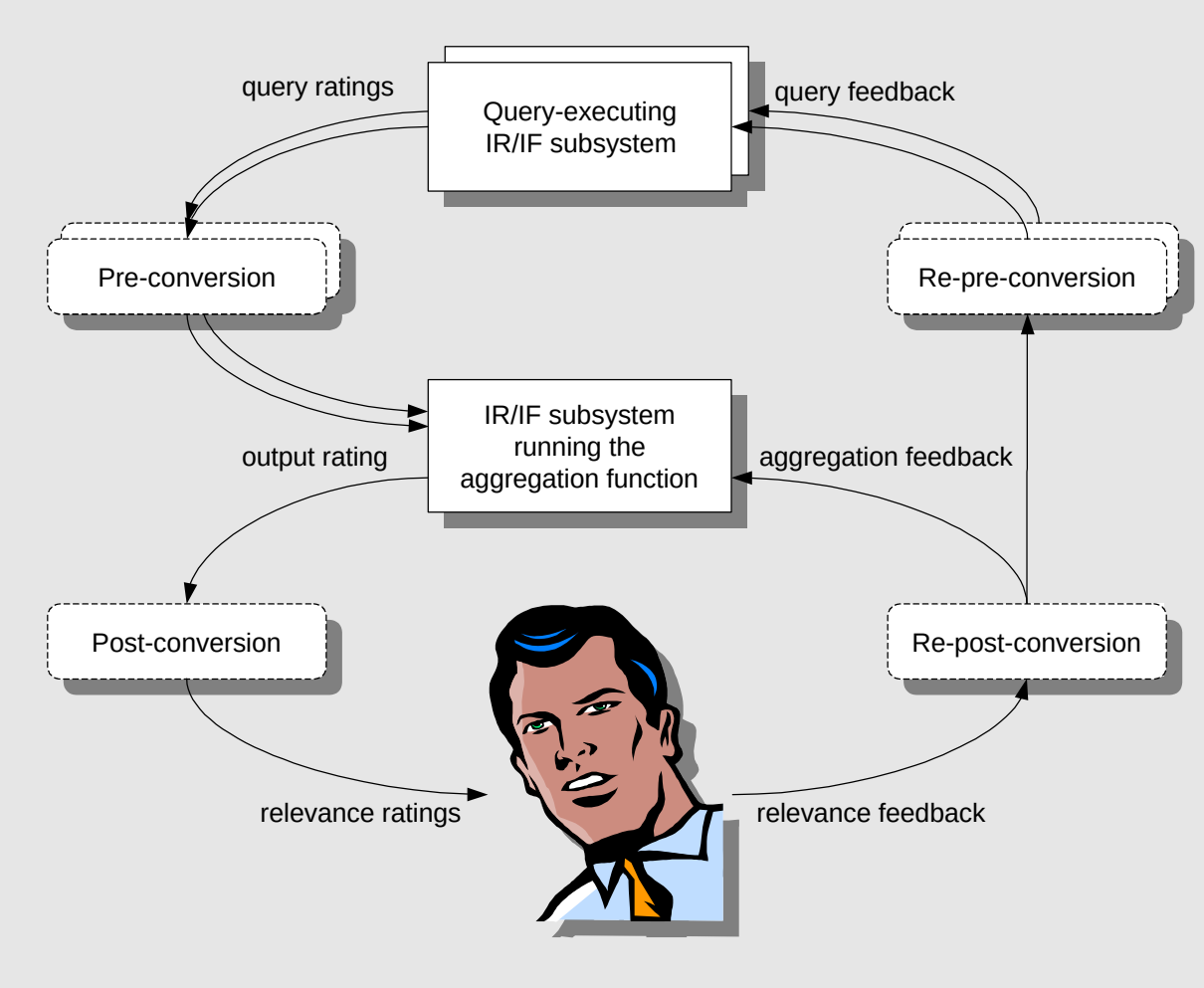


Figure 14: Combining different subsystems may require rating conversions.

The two back conversions (at the right hand side of Figure 14) allow the subsystems to correctly interpret the provided relevance feedback. If relevance feedback provided by the user is already of the rating type expected by the subsystems (e.g. Boolean RF), these conversion steps can be omitted and RF can be handed back directly. If a subsystem expects RF on the relevance scale of its own output ratings, the back conversions can be computed as the inverse functions of the respective forward conversion. Since forward conversions have to preserve query rankings, they are strictly monotonic growing functions, so that their inverse functions are unique.

Since ratings are already aggregated at this point, post-conversion has no impact on the output ranking. It mainly serves to allow users to understand ratings, i.e. to estimate *how* relevant an object is expected to be. See Section 4.2.5 for a discussion on this subject. If possible, rele-

vance feedback scale and output rating scale should use the same relevance model[14], which eliminates the need for a post conversion.

Pre-conversion, on the other hand, is executed *before* aggregation takes place and therefore has a substantial impact on the output ranking. Conversion functions are built as follows. (1) If query ratings are defined as symbolic values, such as {non-relevant, relevant} or {"I hate that", …, "I love that"}, these symbolic ratings have to be mapped to numerical ratings such that their original rank ordering is preserved (see [Zah73] for an extensive discussion on the quantification of hedges in natural language using fuzzy set theory). (2) If the range of query ratings differs from the desired range, query ratings have to be mapped to the goal range. Major non-linearities may be corrected using non-linear transformations. To convert from a probabilistic scale (as used by inference networks) to an additive scale (as used by the weighted request and indexing retrieval model), for example, a function f may be used (Figure 15), such that f maps the probabilistic aggregation function (negative multiply) to the additive aggregation function (sum).

$$f(1-(1-r_1)(1-r_2)) = f(1-r_1)+f(1-r_2) \quad \text{(Equation 1)}$$

The function

$$f(r) = -\ln(1-r) \quad \text{(Equation 2)}$$

solves the equation. Proof:

$$f(1-(1-r_1)(1-r_2)) = -\ln(1-(1-(1-r_1)(1-r_2))) = -\ln(1-r_1) -\ln(1-r_2) = f(r_1)+f(r) \quad \text{(Equation 3)}$$

This inverse function

$$f^{-1}(r) = 1-e^{-r} \quad \text{(Equation 4)}$$

allows converting ratings back.

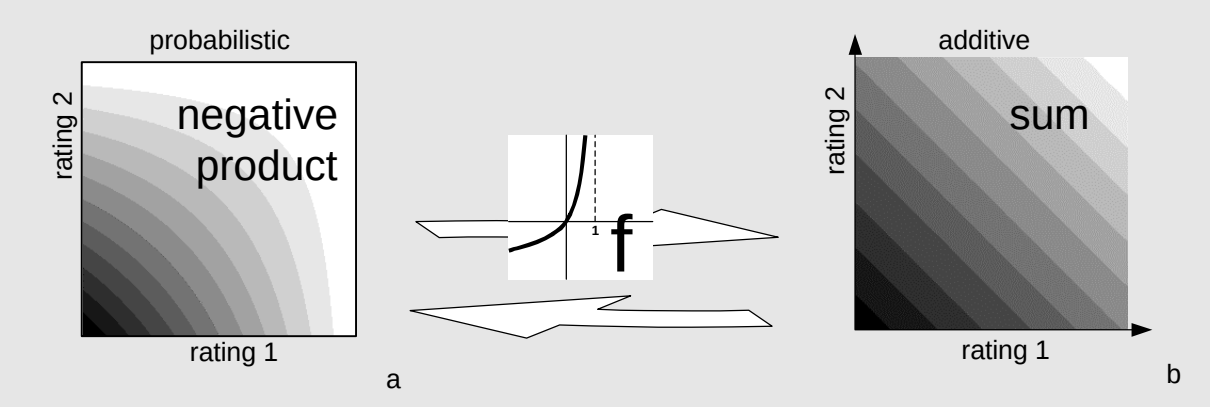


Figure 15: Conversion between probabilistic (a) and additive rating scales (b).

[14] Since objects having the same query rating vector may receive different RF, output ratings actually are probability distributions over the RF data type. If Boolean RF is provided, for example, output ratings are probabilistic ratings.

3.3.5 Enhanced query weighting

The two implementations of the aggregation function presented in Sections 3.3.2 and 3.3.3 do not allow assigning arbitrary output ratings to arbitrary query ratings. The weighted request and indexing retrieval (WRIR) model is limited to scaling query ratings linearly; the inference network interpolates ratings using a formula for conditional probabilities.

The actual relevance of objects is not necessarily correctly described by these interpolations. There may, for example, be sequence effects, such as redundancy within the result set of a query; a user may be highly interested in one or two news programs, but not in all hundred news programs available at one day. Such effects are query-specific and cannot necessarily be foreseen and corrected by rating pre-conversion. Since query weighting has an impact on the output ranking, query weighting functions with additional degrees of freedom can allow users to relate objects matched by different queries more precisely to each other. In the news programs example, users may use a query weighting function to assign a high relevance to the top-ranked news programs, but low relevance to all other news programs. This way, weighting functions allow users to control amounts of desired objects on a per-query basis.

To enhance the weighting mechanisms of the WRIR model, the normal linear weighting function is replaced with a more flexible function; inference networks are enhanced by replacing the weighting functions of the agents. For maximum flexibility, a spline with free knots can be used for the approximation. To preserve query rankings, a spline respecting monotonicity constraints is required, e.g. [Pet96]. Along with the weighting function, also the original relevance feedback learning method has to be replaced. Enhanced weighting is now learned by approximating the gathered (*query rating*, *relevance feedback*) pairs with the spline, as illustrated by Figure 16. The number of degrees of freedom can be handled dynamically. The default is a single degree of freedom, which corresponds to original query weighting. More degrees of freedom are introduced when more RF has been gathered. Typically, the number of degrees of freedom is chosen proportionally to the square root of gathered samples [Pet96].

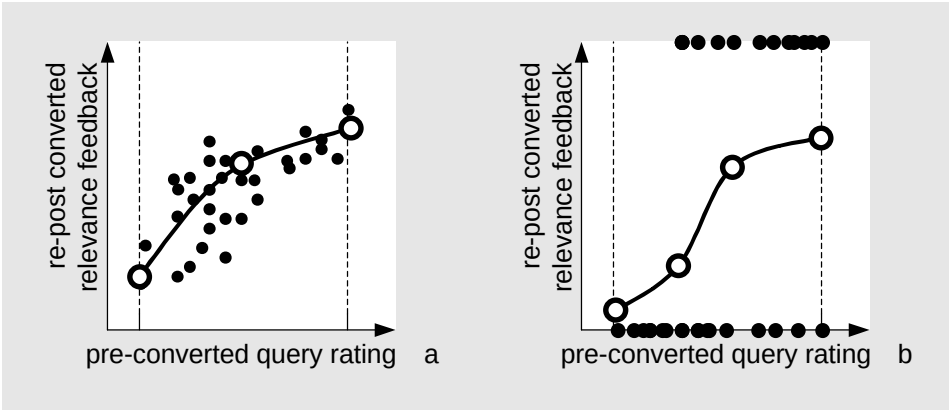


Figure 16: Enhanced weighting function. A spline with free knots approximates (query ratings, relevance feedback) tuples. Example with three free knots and real-valued relevance feedback (a) and with four knots and Boolean feedback (b).

3.3.6 User-aggregated relevance feedback

Learning query weights or enhanced query weighting functions from relevance feedback (RF) is subject to a delay. There is a certain variation in relevance feedback even among objects that have been assigned the same or similar query ratings. This variation is especially high if discrete RF is used, such as Boolean RF. Because of this variation, larger amounts of RF samples have to be aggregated when approximating weighting functions and query interactions. Gathering larger amounts of relevance feedback is time-consuming, which results in delays in the profile learning process, which in turn results in reduced prediction quality when interests change. To allow users to control their user profiles rapidly and effectively, a denser type of input is required. For this purpose, QuerySet Architecture systems allow users to enter *user-aggregated relevance feedback* (URF) via specialized user interfaces that we will present in Chapter 4.

URF is a rating that users assign not to a single object, but to a *set* of objects. Unlike RF, where users communicate the ratings of individual objects one by one to the system, URF input leaves the aggregation to the user. To update user profiles based on URF, users enter a *relevance value* to a set of objects (the URF) and the system aggregates the *query ratings* of all objects in the set. Together, the two ratings form the (*query rating vector*, *relevance*) pair that is required for the profile learning mechanism.

Since users communicate only the single result of the aggregation to the system, user-aggregated relevance feedback saves the major portion of the communication effort that the corresponding relevance feedback for all objects in the set would require. Since a single URF sample is an aggregate of multiple RF samples, it is generally closer to the approximation curve than the average RF sample and can therefore be assigned a higher weight than RF samples (confidence values, see Section 3.3.6.2). Therefore, URF can rapidly and effectively provide the amount of input required for initializing a user's aggregation function or for reinitializing it after an interest change. The price of URF compared to RF is that URF does not contain any object-specific information anymore and therefore cannot be used to refine queries; it only serves for updating the aggregation function. Furthermore, entering URF (see Chapter 4) may be more difficult for users than entering RF, because URF is requires users to think at a higher level of abstraction.

3.3.6.1 Using ranks as object sets allows users to profit from experience and expectations

To be able to assign a rating to a set of objects, users have to know which set is referred to and they have to know at least some objects within this set. One way to ascertain that is by displaying samples. To obtain information about the user's preference for movies starring the actor *Harrison Ford*, for example, the system could present the user with the titles of a selection of these movies. If this approach is taken, the benefit of URF over RF is that the effort for communicating individual ratings is saved; the effort for recognizing and mentally rating objects, however, remains basically the same.

The user's task of providing URF is accelerated if users know the set *without* looking at example objects. The sets defined by the individual queries in a QSA profile can be used for that

purpose. As a first iteration, the entire set of all matched objects may be rated by the user. If the user has previously used a query to search for movies starring the actor *Harrison Ford*, then the name of that query may be used when asking the user for URF.

If the system displays query results to the user in the form of a ranking, then the URF model can be refined to use selected ranks instead. Because of their prominent position within a ranking, users are most likely to recognize and remember the objects that were found at the top and at the bottom ranks. Using this observation, the user's task is to assign a relevance value to the set of all objects ranked top-*n*% and another rating to the set of all objects ranked bottom-*n*% (Figure 17a). If the rating variance within these ranks is sufficiently small, a specification of the rank intervals may be omitted so that the system may simply refer to *the* top-ranked and *the* bottom-ranked objects. In the case of the *Harrison Ford* query, users would be asked to rate the top-ranked movies starring the actor as well as those movies found at the end of the ranking.

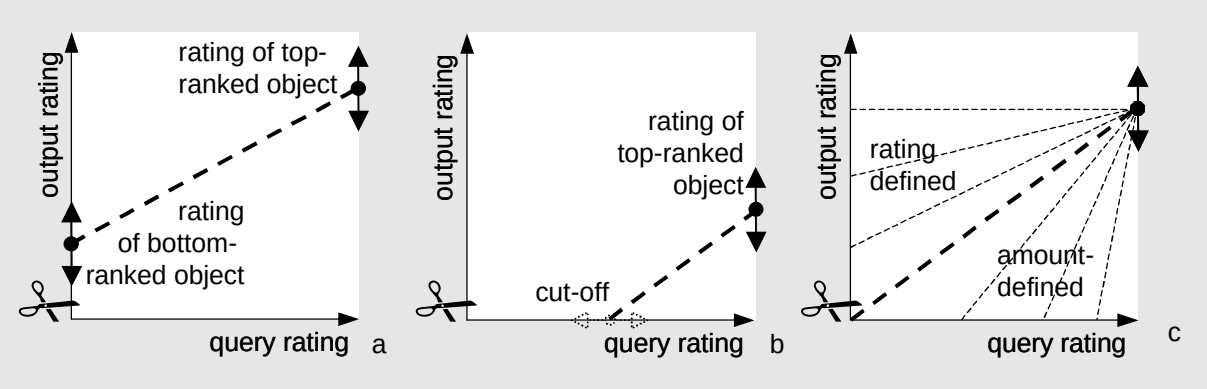


Figure 17: URF samples at top- and bottom-ranked objects (double arrows) (a) and URF samples if output ratings of the query are not entirely above the cut-off value (b). Depending on whether the query is entirely above the cut-off value, the bottom-ranked URF sample can be defined by entering a rating or an amount (c). (Normalization functions sketched as dashed lines, but are not necessarily linear functions).

Users may decide not to retrieve all objects matched by a query, so that there is a certain *cut-off* for each query. We will use the term *cut-off value* to refer to the query rating of the last retrieved object of a query[15]. If a query's bottom-ranked objects correspond to relevance values below the cut-off value (Figure 17b), judging their relevance becomes awkward; a correctly defined QSA profile does not deliver objects below the cut-off value to the user, so that

[15] In an offline situation, users have to specify the amount of objects they wish to retrieve in advance. This aspect has been discussed under the name *futility point* [Bla90]or *frustration point* [Cooper 73a, Cooper 73b]. In literature, two models are found that define different ways of limiting the number of objects. Foltz and Dumais allow users to receive a *fixed number of top-ranked documents* per update period [FD92]. Yan and Garcia-Molina [YG95] allow users to define a *relevance threshold*, which is the minimum similarity score that a document must have against the profile in order to be delivered. The QSA generally uses a relevance threshold. Since users may have difficulties entering such a value, it is entered as a fixed number of top-ranked documents instead and automatically *converted into* a relevance threshold.

users never see these objects and will therefore have difficulties judging *how* irrelevant they are exactly.

To allow users to enter bottom rank URF for cropped queries, we use a variation of URF that is the opposite approach to the URF samples used above. Bottom rank URF for cropped queries is based on the set of objects at the cut-off rank. The output rating at cut-off can be determined by the system. Since it is identical for all queries, this rating has no impact on the merged QSA ranking, so that an arbitrary value can be used. To form a complete (query rating, output rating) pair, the corresponding *query rating* at the cut-off rank has to be obtained from user input. For this purpose, *users enter which percentage of objects they want to retrieve* from that query. The query ratings of the cut-off rank can be obtained by running one or more test executions of the query and by aggregating the query ratings at the entered rank. The resulting pair (*query rating*, *relevance*) at the cut-off rank defines the second URF sample.

The benefit from using ranks as URF samples is that users can contribute their *experiences* with the query, their *expectations* about the query, and even their expectations about the interest that the query is expected to represent. In the case of the TV Scout, users having selected the query *Movies starring Harrison Ford* to be part of their QSA profiles have several means of estimating what it represents. Users, who have executed the query before, may come up with relevance estimates based on their experience with past rankings received from that query. Users who know and understand the query formulation may deduce relevance estimates based on their understanding of the query formulation, even without ever having seen the query working. And users who have seen neither nor may still profit from the expectations they have based on the interest they expect the query to represent. Although the user's expectations may be biased (users may for example expect a perfect query), these expectations can *accelerate* the perception of sample objects to be rated. At the limit, users may even be able to enter URF without being provided with samples.

3.3.6.2 Combining URF and RF using confidence values

Since User-aggregated relevance feedback is fully compatible with relevance feedback, URF is especially easy to integrate into the learning of query weighting functions. URF samples can simply be mixed with RF samples; the only difference is that URF samples are given higher confidence values, so that they have a higher impact on the weighting function. To further improve the approximation, the following properties may also be taken into account when assigning confidence values.

1. *Probability of being outdated.* Confidence values of RF and URF may be decremented proportionally to their age or proportionally to the amount of samples gathered since. Newer input to the same entity overwrites previous input. While this hardly happens for RF input, this is the usual case for rank-based URF samples. An additional interest shift detection mechanism may be used to decrement confidence in samples affected by interest change (e.g. [LMMP96]).
2. *Confidence in gathering method.* If samples do not unambiguously indicate the user's judgment, their confidence is reduced. This can, for example, be the case, if ratings are

gathered implicitly, i.e. by monitoring user behavior, and if there are other possible explanations for the observed behavior (see Section 5.3).

User-aggregated relevance feedback allows users to directly manipulate weighting functions. When used with the user interfaces presented in Chapter 4, URF not only allows users to provide input to their profile, but also to view the current state of it and to "tweak" it interactively. To show the current state of the weighting function to the user, the weighting function is converted to a set of URF samples that are then displayed and offered for manipulation in the profile editing tool (Figure 18). To make this possible, the knots of the spline representing the weighting function are selected such that they coincide with the desired URF samples. When a profile editor is invoked, all past RF/URF samples (Figure 18a) are approximated by a spline with free knots (Figure 18b), then samples are removed. The resulting knots define the URF samples that are displayed in the profile editor and offered to be manipulated (Figure 18c).

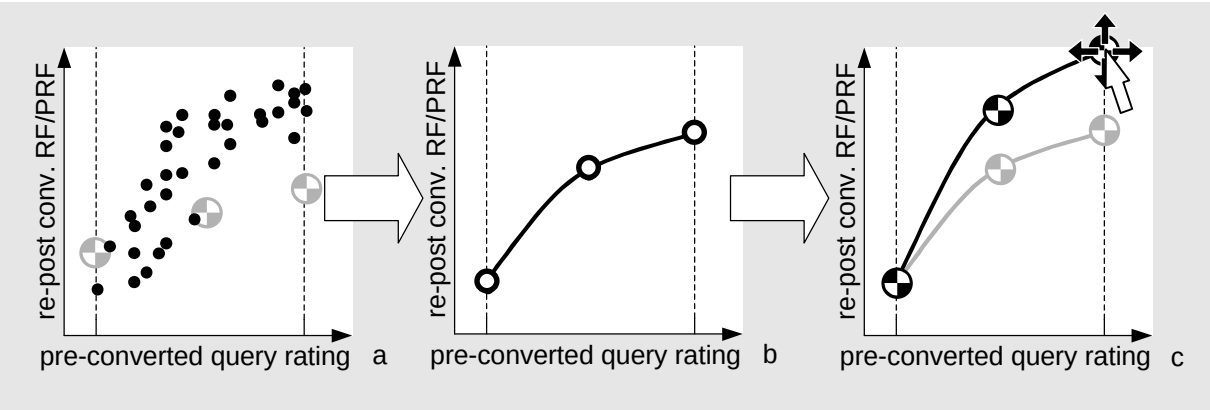


Figure 18: Allowing users to interactively update a weighting function using user-aggregated relevance feedback. URF samples (cross dots) and RF samples (black dots) define the current state of the weighting function (a). Samples are approximated by a spline with free knots (ring dots). Then samples are removed (b). The resulting knots define the URF samples that users can manipulate using specialized profile editing tools (c).

3.4 DISCUSSION

Concluding this chapter, we will discuss the intended usage of QuerySet Architecture systems, compare the QSA with related work, and finally discuss the strengths and limitations of our approach.

3.4.1 Evolving usage of QSA systems

The fact that QSA profiles are made of queries has implications on the overall system usage. The circumstance that queries can be used without being part of a QSA profile allows users to

incrementally explore the system's capabilities as shown in Figure 19. During their usage of a QSA system, users pass up to three stages.

1. **Query state (S1):** Users can formulate queries manually (U1) or can pick predefined queries, provided by the system (see Sections 5.2.2 and 5.4) (T1).
2. **Bookmark/reuse state (S2):** If the interest represented by the query is not permanently satisfied after executing the query, users may store the query for later reuse ("bookmarking") (U2). To support users in this process, the system may suggest bookmarking a query, e.g. when the same query has been entered repeatedly (T2). Once a query has been bookmarked, it can be executed efficiently by recalling it from the bookmark list.

   Profile creation (T∗): Bookmarked queries are stored in the QSA profile; the user's QSA profile is created automatically when the first query is bookmarked (T∗). Inserted queries are initialized based on Zipf's law (see Section 3.3.2).
3. **Profile state (S3):** Initially, the QSA profile serves to allow users to execute all their bookmarks simultaneously. Continuous supply of relevance feedback allows the system to improve the aggregation function that is part of the user's QSA profile and thereby to optimize the output ranking of the profile. Relevance feedback-based adaptation also allows the profile to adapt to gradual interest changes (T3). Alternatively, users can update their profile directly by providing user-aggregated relevance feedback. This way, users can accelerate the profile learning process or re-initialize the profile rapidly after substantial interest changes (U3).

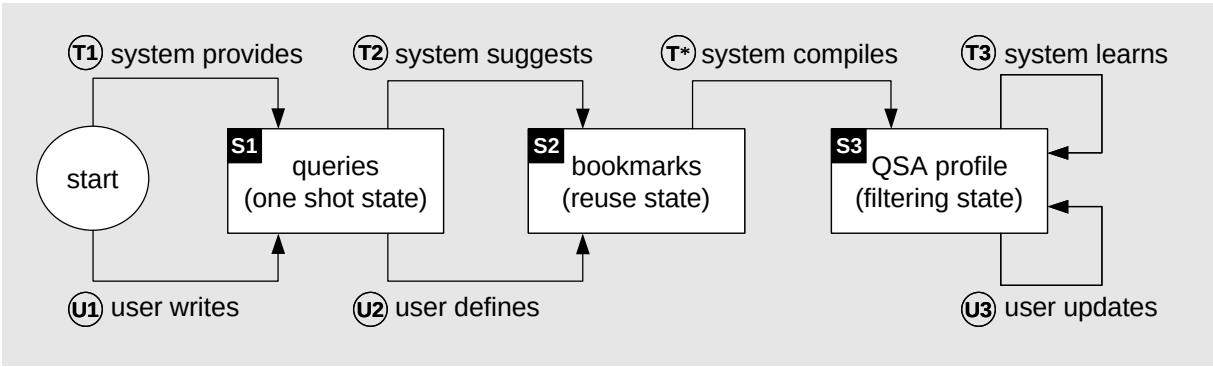


Figure 19: Evolving usage of a QSA system. Both, system (T1-T3) and user (U1-U3) may initiate progression.

An important strength of the QuerySet Architecture is that new users find themselves in an IR, *not* in an IF situation. This system design allows users to start using QSA systems in a non-planning, prototypical fashion that is only directed towards *immediate* benefit; long-term planning always remains optional. A QSA system may therefore replace both an IR and an IF system, additionally providing the option to skip back and forth between the two "modes" without loosing already provided profile information. In Chapter 1, we mentioned the two analytical information seeking strategies IR and IF and explained that IR aims at supporting users with changing interests while IF systems are designed to support information access in highly dynamic information sources. It is therefore not surprising that the QSA not only unites

the characteristics of both approaches, but also can be used as both. To help users save redundant effort (the purpose of IF systems is to be "time-saving devices" [Bac91]), the system will suggest advancing in the IF-oriented direction, but users are never forced to do that. Users may even decide not to build a profile at all and to settle with the initial IR or bookmarking functionality.

3.4.2 The QuerySet Architecture is a flexible architecture

Because of its generic structure that mainly consists of two stacked IF/IR systems, the QuerySet Architecture is unusually flexible. Replacing selected components allows adapting it to different requirements and application areas. To illustrate this statement, we will briefly show how selected IF models (see Chapter 2) can be implemented as special cases of QSA systems.

SDI systems (Selected Dissemination of Information, systems that allow users to run multiple continuous queries, see Sections 1.1.1 and 2.2.1) can be emulated by a QSA by simply omitting the aggregation function and returning the individual query rankings individually. *Rule-based systems* allow rules initiating arbitrary actions, such as filing or forwarding an object. The QSA cannot emulate the functionality of arbitrary rules, but can implement the functionality of appraiser rules (Section 2.2.2), by using exact match queries and an aggregation function that defines the respective aggregation. *Stereotype-based systems* can be emulated by using a predefining set of queries (the stereotypes) that are associated with character properties of the user. The aggregation function of the QSA can be used without modification. Beside their ability to "emulate" other architectures, QSA systems can "assimilate" any type of IF functionality by plugging it in as an additional query-executing subsystem.

3.4.3 Comparison with dynamic filtering requirements

With respect to the dynamic filtering requirements stated in Chapter 2, user-aggregated relevance feedback forms the central component of the QuerySet Architecture. User-aggregated relevance feedback (URF) provides users with direct access to the aggregation functions of their user profiles and therefore allows users to make very effective profile updates. This makes URF the method of choice for handling all major profile inaccuracies, as they occur during the initialization of a profile, when adding a new query, or after *abrupt interest changes*. The correspondence between queries in the QSA profile and interests allows users to make these profile updates in a straightforward fashion and to restrict the updating to what really requires updating.

Other IR/IF architectures that use hierarchical representations of the user's interests (e.g. inference networks, but also queries using Boolean syntax, e.g. [SM83]) also allow users to manipulate the representation at a high level, e.g. by manually adjusting weights, the values of *p*-norm connectives, or link matrix definitions. Since these entities, however, do not correspond to any real world entities, users may have difficulties to quantify them effectively. The QSA avoids this problem, by using URF. URF receives its meaning from being related to the relevance of objects, and therefore allows users to think prototypically. We will dedicate the entire Chapter 4 to this important concept and to how users can interact with QSA systems based on it.

The direct access to the aggregation function via URF also provides the QSA with capabilities for handling *repetitive interest variations*. Some of the systems surveyed in Chapter 2 that provide special support for repetitive interest changes do that by allowing users to save profiles and to reuse them when the represented interest state occurs again. The music recommendation system LyricTime, for example, allows users to have multiple profiles representing different moods ([Loe92], see also Section 2.2.1). A potential problem of the multiple profile approach is its limitation to a finite set of profiles. If there are very many states that the user can be in, e.g. if taste changes are characterized by a combination of several independent factors, it may occur that none of the saved profiles is satisfactorily close to the desired state.

The QSA allows extending the reuse concept. While users may still be allowed to save and reuse the profile as a whole (see the history approach in Section 4.4.5), QSA systems can also support users in "tweaking" their profiles in a more generic way. To do that, users may keep queries that are currently not of interest in the profile. Instead of removing the queries, they are only deactivated by setting their query weight to zero. When a past interest state occurs again, the profile can be reassembled rapidly. All queries are still there; only the aggregation function has to be reconstructed by adjusting the respective query weights using URF. Specialized user interfaces allow users to enter the desired profile state with minimum user effort, as we will demonstrate in Section 4.4.5.

Since QSA systems can also trace *gradual interest changes* by adjusting the profile based on relevance feedback, all three types of interest changes from Chapter 2 receive specific support in QSA systems. Whether this support meets the requirements of an actual usage situation requires looking at an actual application. We will therefore discuss these issues in Chapter 5 at the example of our QSA-based TV recommendation system *TV Scout*.

CHAPTER 4 TOOLS FOR MANUALLY UPDATING USER PROFILES

It is more effective to support users in making decisions than to try to decide for them
[Ing92]

In the previous chapter we presented the QuerySet Architecture (QSA) and a mechanism for updating QSA profiles based on relevance feedback and user-aggregated relevance feedback (URF). In this chapter, we will present user interfaces for entering user-aggregated relevance feedback. To cover a whole range of QSA applications, we will present three interface families that are optimized for three different purposes, i.e. learnability (*form-based interfaces*, Section 4.1), accuracy (*histogram-based interfaces*, Section 4.2), and efficiency (*paintable interfaces*, Section 4.4).

4.1 SIMPLE FORM-BASED EDITORS

Figure 20a shows a simple form-based user interface that is based on the URF concept presented in Section 3.3.6. Using this editor, users can inspect and update the aggregation function of their user profiles. In the shown state, the editor contains the three queries *news*, *sports*, and *comedy shows*. The editor shows the weights assigned to the individual queries and allows users to modify them. The shown editor uses two URF samples per query. The first defines the rating for top-ranked objects. The second defines either the rating of the bottom ranked objects or the cut-off value. The query *news*, for example, is very important to the user, i.e. both top and bottom ranks are set to "very important". The query *comedy shows* is set in a way that assigns an even higher rating to the top ranked comedy shows, but cuts off the bottom ranks, so that only the top 15 programs are delivered to the user.

To keep the interface as simple as possible, it uses pull-down menus with a rather rough set of values instead of sliders or such. This version allows users to enter URF in the form of symbolic ratings; other applications of this interface will use their respective rating scales. To implement the model shown in Figure 17c, this version uses dynamically updated pull-down menus for the bottom rank. Whenever the rating for the top-ranked objects is adjusted, the menu for the bottom-ranked objects is adapted accordingly, so that it always offers only same or lower ratings. To avoid the usage of the notion of ranks in the user interface, this interface

refers to ranks according to how they are computed; here it is popularity. In other application areas, different explanations for ranks may be used.

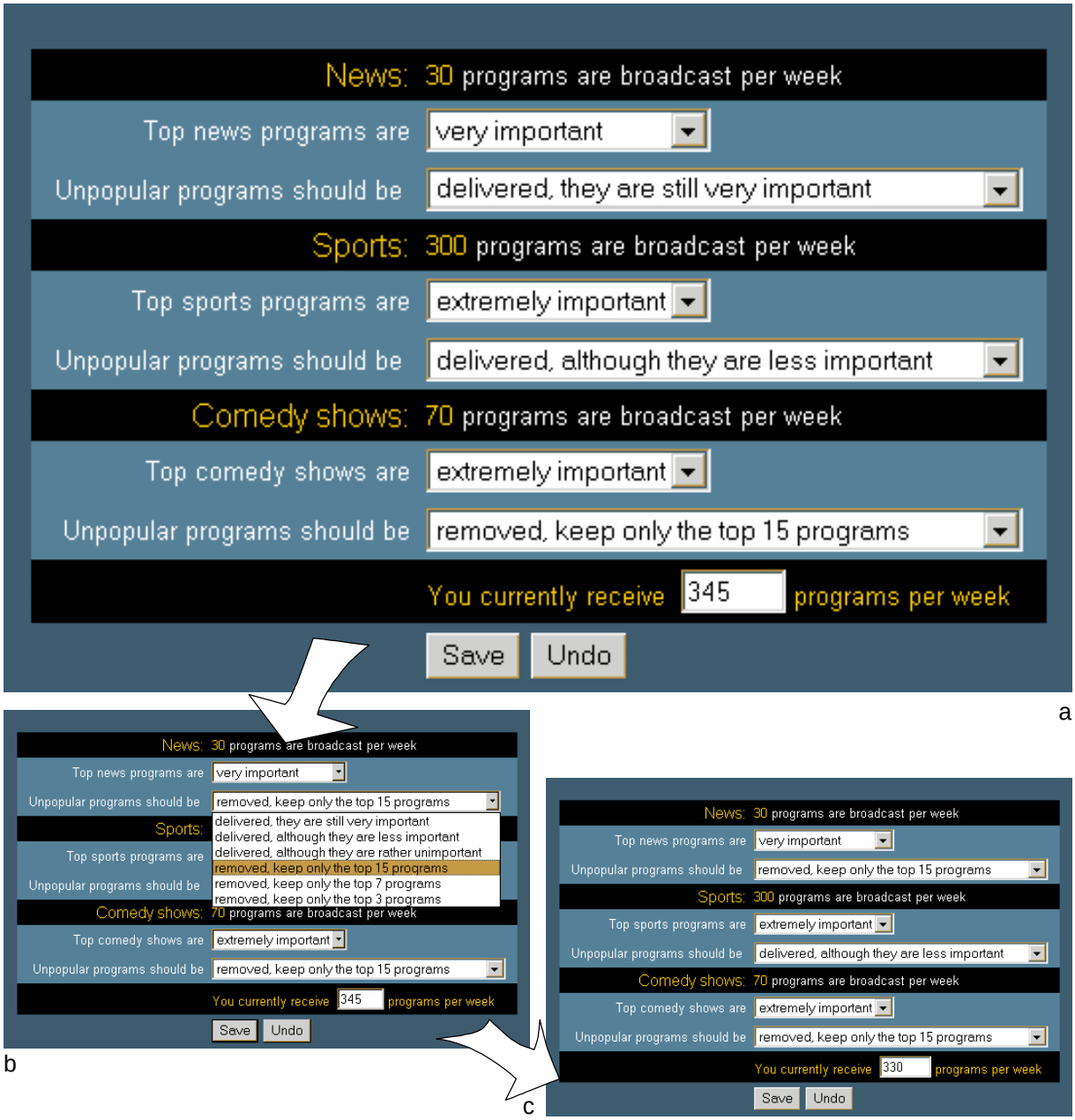


Figure 20: Form-based user interface for entering two URF samples per query. All programs delivered by the query *news* are considered very important (a). The user decides to reduce the amount of objects delivered per week by reducing the number of news programs (b). The overall counter at the bottom of the interface is updated accordingly (c).

To better correspond to the users' perception of their information intake (e.g. the number of TV programs they actually watch every week), absolute amounts of objects per week are used

instead of percentages. Amount options are organized as geometrical rows to equally support users aiming at a high recall as well as users "just trying to be informed" by receiving only few, top-ranked objects. An automatic sum field at the bottom of the interface informs users about the overall amount of objects they have currently selected. This value is updated automatically as the user updates the settings (Figure 20b and c).

For maximum simplicity, the user interface may be restricted to a single parameter per query, The form-based interface shown in Figure 21 offers only a single pull-down menu per query. The two URF samples cannot be manipulated independently anymore, but the interface still displays both parameters to allow users to select the best combination.

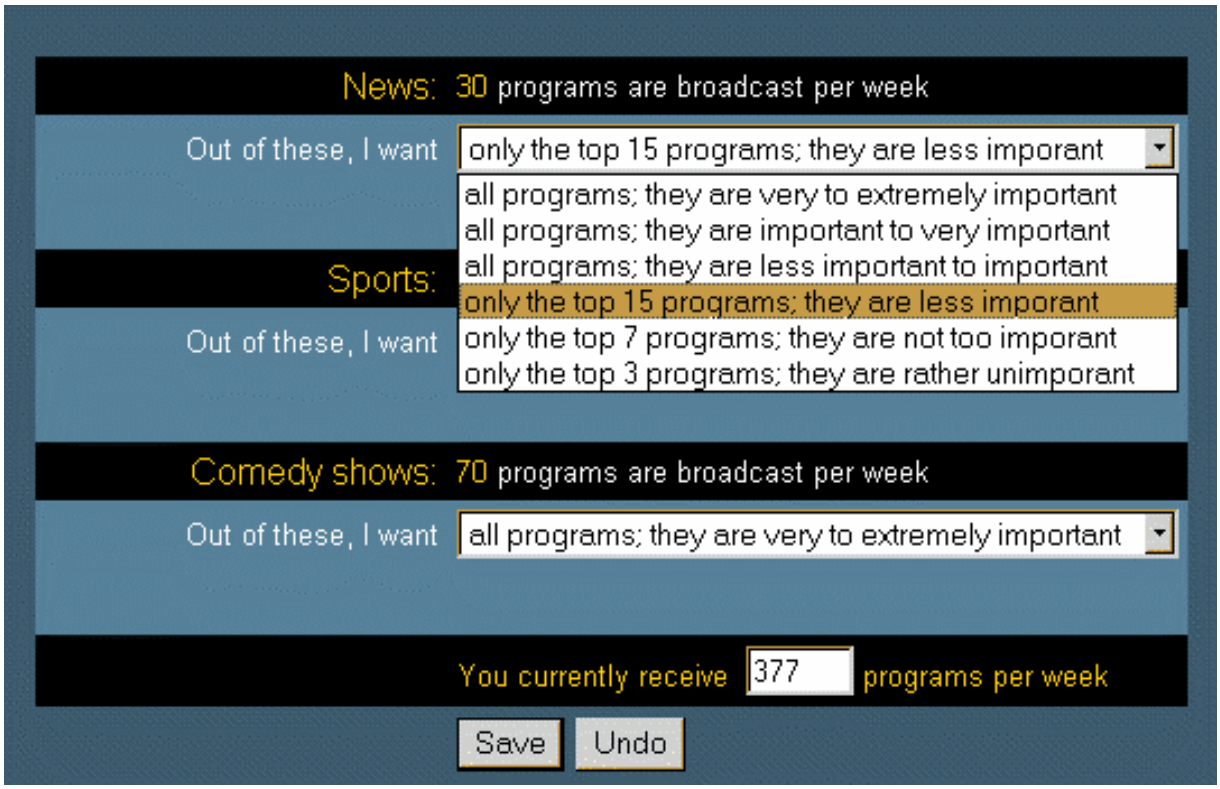


Figure 21: Form-based user interface for entering combinations of two URF samples per query (faucet interaction technique).

We termed this one-dimensional interaction technique *faucet interaction*, derived from a special type of faucet that works after the same principle. Figure 22 shows two pictures that I took in two different hotels in the United States. These faucets use a single revolving knob for both warm and cold water. When turning the knob, at first only the amount of cold water increases. When the maximum amount of water is reached, turning the handle further increases the water temperature, finally providing hot water. See [Nor88, p. 158-172] for a discussion on different faucet types.

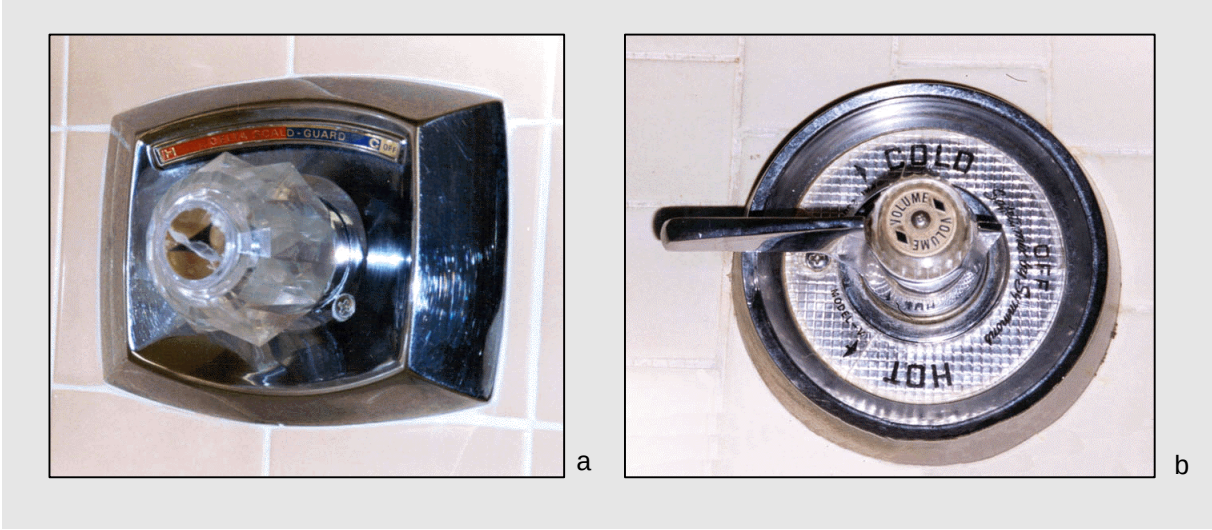


Figure 22: One-dimensional faucets. Turning the knob/handle first increases the amount of cold water, then increases the water temperature.

Form-based interfaces are designed for simplest usage. See Section 4.3 for an evaluation of this interface style that was implemented and tested as part of the TV Scout system (see Chapter 5).

4.2 ACCURATE HISTOGRAM-BASED EDITORS

Form-based interfaces allow users to enter up to two user-aggregated relevance feedback samples per query. In this section, we will generalize the URF concept to achieve higher accuracy and to allow users to manipulate more degrees of freedom of weighting functions. For this purpose, enhanced visualization techniques are required. In this section, we will explore the potential of histogram-based user interfaces for entering such user preferences.

The interaction technique we use, namely to relate multiple histograms to each other, is groundbreaking work. In the past, histograms have been used in the context of the optimization of databases [IP95, AC99], in image retrieval (as image surrogates for comparing images [SC96, PZM96]), and for various visualization purposes, e.g. for providing overviews over data obtained using statistical methods [WBW96]. The only work involving histogram displays in user interfaces published so far is the work by Lisa Tweedie et al. The *attribute explorer* [TSW94] and its follow-up the *influence explorer* [TSD95] combine histograms with interval sliders to allow users to select objects satisfying a number of hard constraints from a database. This interaction technique allows, for example, to search for houses with at least *n* rooms in a given price interval, so that this technique is strongly related to systems using Dynamic Queries, such as the *Dynamic HomeFinder* [WS92] (see [Twe97] for a comparison of the two interaction techniques) or the *Focus* system [SBB96]. The work presented in this chapter differs from *attribute explorer*-like systems with respect to the type of user preferences that can be entered. While the *attribute explorer* allows users only to select a subhypercube from the high-dimensional space of attributes (*exact match* filtering), the interfaces

presented in this section allow users to relate individual attributes to each other (attributes here being the object's individual query ratings), allowing users to define a preference ranking on the objects (*best match* filtering).

4.2.1 Reusing transformation dialogs from image processing

In Sections 3.1, we mentioned structural similarities between user profiles and queries on one hand and digital images on the other. Exploiting this similarity, we will check in how far user interfaces originally developed for normalizing the colors of digital images can be used to normalize query ratings.

When multiple image elements are combined, e.g. to create a collage or a scene in a 2D animation system (see Section 3.1), source image colors may have to be adjusted to match the other image elements. Even grayscale image elements can require numerous adjustments, e.g. if they are too dark, too light, if they have too little or too much contrast or if the main contrast is not in the luminance range where it is required. The manipulations performed when post-processing images include adjusting luminance levels, contrasts, and luminance distributions of the source images. Color images can be normalized using the same techniques by adjusting each color channel individually.

Figure 23 shows three transformation dialogs that are used to accomplish such adjustment tasks (screenshots taken from Adobe Photoshop [Adobe96]). They map the current luminance of each pixel in the image (the input luminance) to output luminance, thereby performing the same task that profile editors perform when allowing users to enter weighting functions mapping query ratings to output ratings.

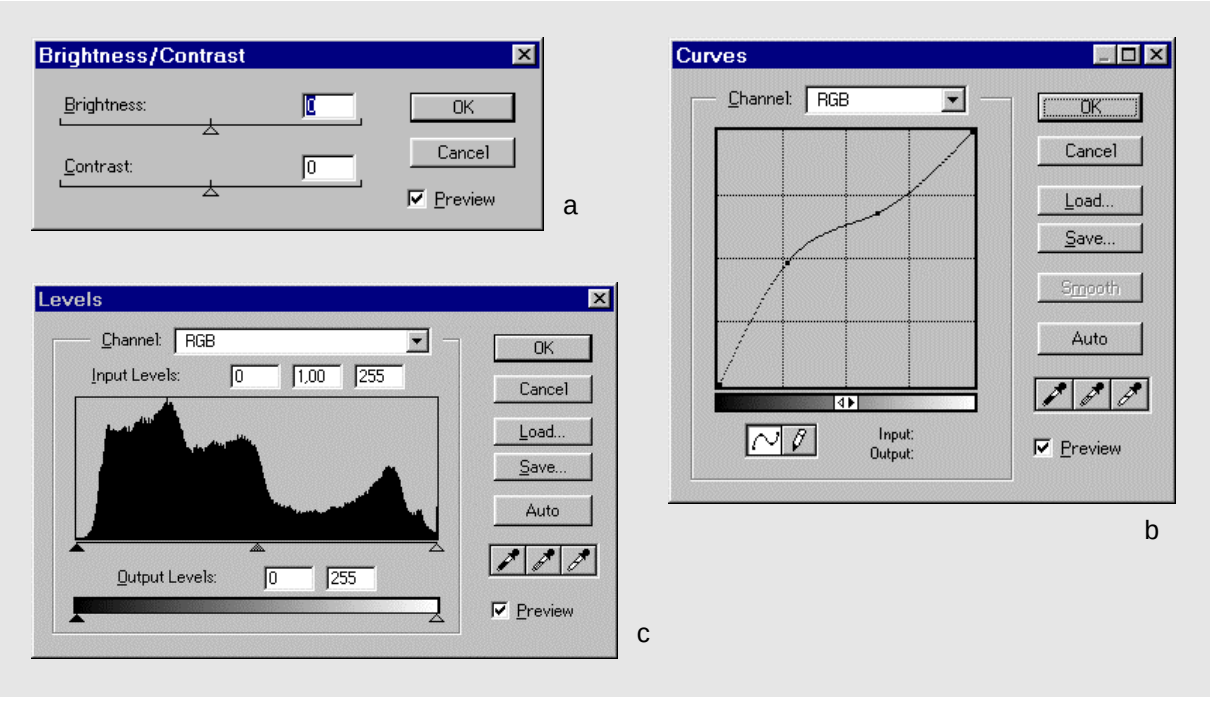


Figure 23: Brightness/contrast dialog (a), graduation curves dialog (b), and levels dialog

The three dialogs shown in Figure 23 take different sets of parameters. The *brightness/contrast dialog* shown in Figure 23a provides two degrees of freedom and allows defining similar functions as the two-dimensional form-based profile editor presented in Section 4.1. Its parameters brightness and contrast define luminance offset and factor (Figure 24a). The *graduation curves dialog* shown in Figure 23b allows defining arbitrary continuous transformations by allowing users to draw the transformation by inserting and positioning an arbitrary number of nodes (Figure 24b).

Since the nature of these parameters does not allow users to predict the effect that their adjustments will have on the picture (e.g. "what will the image look like if brightness is increased by 0.3?"), these interfaces use direct manipulation[16]. Users play with the controls until a satisfactory result is obtained; simultaneously, the continuously updated image itself or a smaller preview of it allows users to judge progress. Image processing is in the fortunate situation that the manipulated object type itself—images—are perceived so rapidly that no additional preview is required.

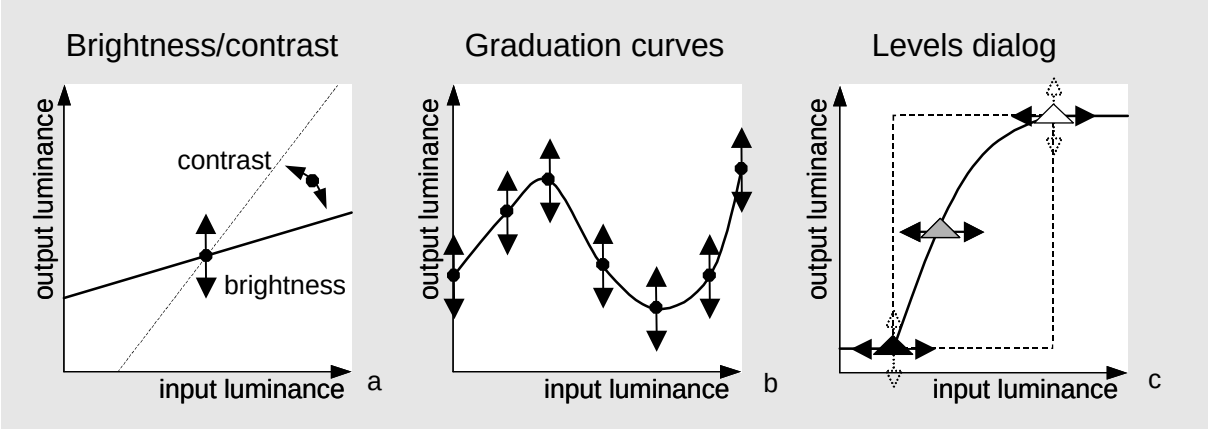


Figure 24: Degrees of freedom of the three image-processing dialogs shown in Figure 23 in the same order. All graphs have input values on the x-axis and output values on the y-axis.

How can we apply this user interface concept to the weighting of query ratings? Direct manipulation would be useful here, because meaningful names for URF samples other than the two rank-based URF samples presented in Section 3.3.6 are difficult to find. If additional nodes of an approximating spline are to be entered, the values of these parameters will not allow users to predict the effect that their adjustments will have on the results of their user profile, i.e. the output ranking. Users therefore find themselves in a similar situation as the users of the two image processing dialogs. To allow users to manipulate their weighting function by playing with parameters, a preview is required that informs users about the effect of their actions. This preview has to inform users about how objects will be assigned different output ratings and how the output ranking is affected.

[16] The criteria for direct manipulation are (1) continuous representation of the object of interest, (2) physical action or labeled button presses instead of complex syntax, and (3) rapid incremental reversible operations whose impact on the object of interest is immediately visible (e.g. [Shn82, Shn97, Shn98]).

As discussed in Section 3.3.6, it is possible to display a selection of sample objects for the purpose of a preview. However, the more degrees of freedom the weighting function to be entered has, the more such sample objects are required, and the bigger the user effort for identifying and rating objects becomes. Again, the question is whether it is possible to accelerate the perception process by using sets of objects that users can recognize without looking at the contained objects, i.e. whether it is possible to aggregate objects such that the resulting set can be labeled in a way meaningful to the user.

Unfortunately, because of the encapsulated nature of objects in the QuerySet Architecture, the only object property that is generally available to the QuerySet Architecture system is rating (which in turn includes derived properties, such as ranks). This renders all aggregation techniques but those based on ratings inapplicable. In turn, this also renders all visualization techniques that require more object properties than a single scalar value per object inapplicable, including star field displays [WS91, WS92]) and layouts created using multi-dimensional scaling (see Section 4.4.3, see [HMM00] for a survey on visualization techniques).

What remains, are aggregates that are based on ratings and/or ranks, and visualizations that are based on these aggregates. A visualization that maps ratings to luminance, for example, may inform users about average output ratings (image brightness), range of output ratings (contrast), and distribution of output ratings (brightness distribution). If similar visualizations of other queries are shown simultaneously, aggregated query properties may be visually compared to other queries, which may allow users to get an impression of how the queries in the user's profile currently relate to each other. Based on these observations, the *levels dialog* with its additional visualization (Figure 23c) becomes attractive for the purpose of visualizing ratings.

The shown dialog offers a larger amount of functionality (fields for entering values numerically, buttons for saving and loading states, output level sliders, and a menu for manipulating color images), but at this moment, only the histogram and its handles are of importance for our purpose. This histogram provides users with an especially compact and intelligible visualization of the luminance distribution of pixels within the image. The x-axis of this histogram represents pixel luminance; the y-axis represents the number of pixels having the respective luminance[17]. One may think of histograms as a heap of "particles". Each object is represented by such a particle; all particles are sorted according to a given property (here brightness) and piled up, thereby forming the histogram.

Figure 25 illustrates the functionality of the histogram-handle combination. The histogram visualizes the *current* state of the image. The x-axis represents luminance, so the left hand side of the histogram represents the amount of dark pixels, while the right hand side represents the amount of bright pixels. The shown histogram represents an image consisting only of rather dark pixels. The three handles represent the *desired* state of the image. The black, gray, and white triangular-shaped handles below the histogram represent 0%, 50%, and 100% luminance respectively. Letting a handle of luminance X point to a region in the histogram means that these pixels will be assigned luminance X. In the shown state, the white handle, for example, is used to assign 100% luminance to the brightest pixels in the image.

---

[17] A histogram may also be considered a special type of star field display; X-coordinates represent query rating intervals, y-coordinates represent object ranks within the respective rating interval.

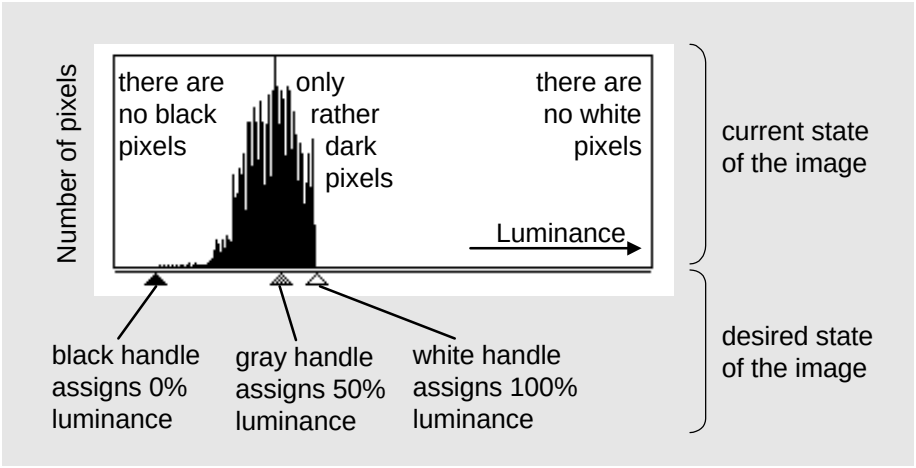


Figure 25: The histogram component that is part of the levels dialog visualizes the current luminance distribution of the image (top half) and allows assigning new luminance values to groups of pixels (bottom).

Handles can be dragged horizontally, which means to assign the respective output luminance to the part of the histogram that the triangles point to. Figure 26 shows how this dialog is used. The dim image (a) has an unbalanced luminance distribution (b). Adjusting the left and right slider increases contrast, also brightening the image. The center handle is coupled to the two outer handles and automatically stays in the middle between them (c). The unbalanced distribution is now corrected by dragging the gray handle to the median of the distribution, so that there is the same amount of pixels left and right of the handle. The histogram visualization makes it easy to find this position visually. This non-linear transformation is called *gamma correction* [Ber98, Poy98] (d). The resulting image is well-balanced (e). If the levels dialog is closed and reopened, the image's new luminance distribution is loaded into the histogram. The fact that the histogram now spans the whole luminance range plus the fact that the distribution it is almost symmetrical confirms that the image is well-balanced (f).

Figure 24c shows which degrees of freedom the *levels dialog* offers. The three triangle-shaped handles below the histogram offer the three horizontal degrees of freedom. More degrees of freedom could be provided by adding more handles, e.g. for 25% and 75% output luminance. Since the three triangle handles are restricted to point to positions *within* the histogram, they can only be used to stretch the distribution. To allow compressing the distribution, a second set of handles at the bottom of the dialog (Figure 23c) offers the corresponding two additional degrees of freedom, shown as vertical arrows in Figure 24c.

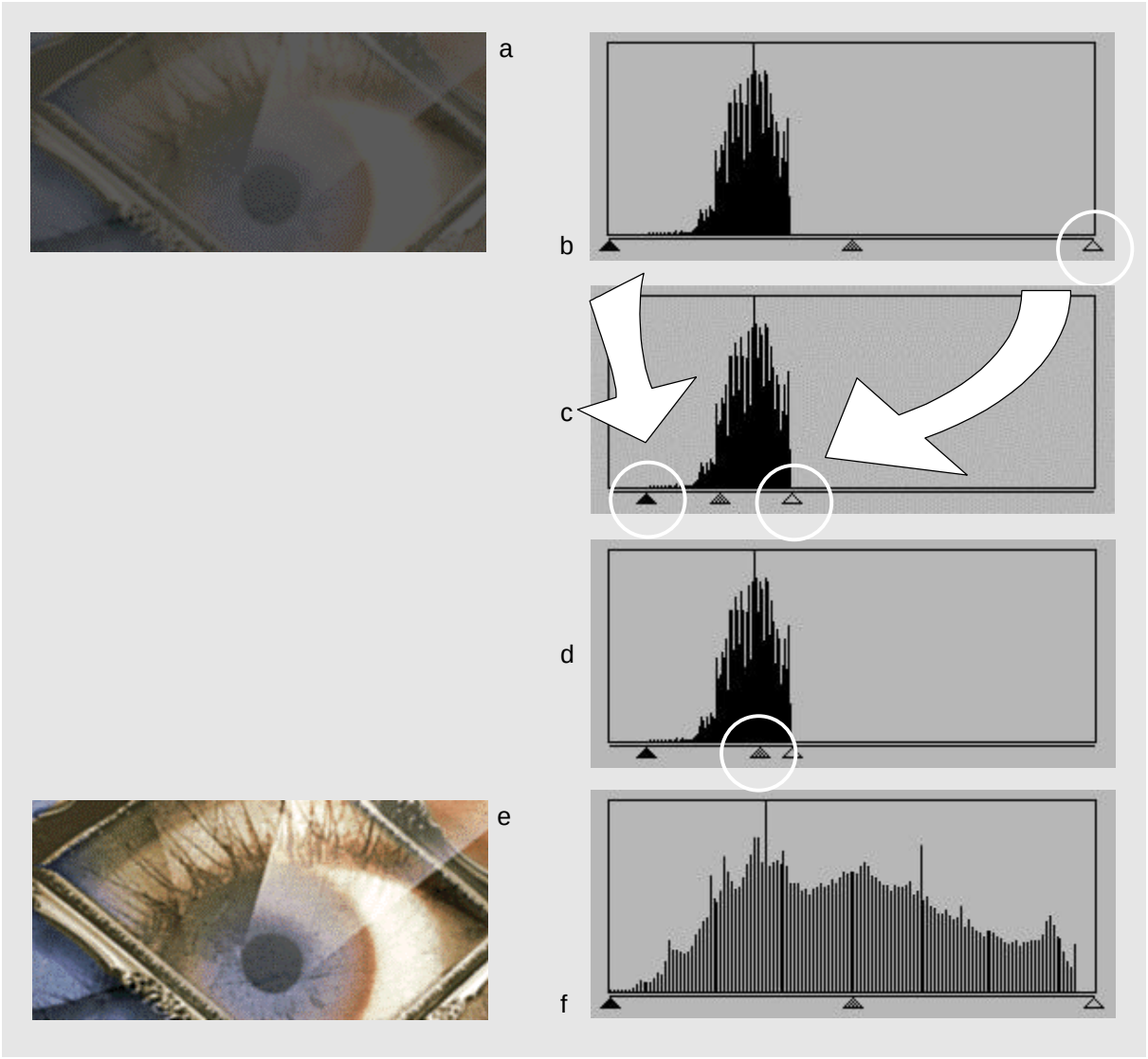


Figure 26: Adjusting luminance distributions using a histogram-based dialog.

4.2.2 Applying the levels dialog to the QuerySet Architecture

When applying the histogram-based transformation dialog to represent the weighting functions of a QSA query, *query ratings* form the x-axis of the histogram and the triangular handles represent *output ratings*. The container holding the histogram now defines the range of possible query ratings. Figure 27 shows an example. In the shown case, query ratings are real-valued and range from zero to one. The histogram shows that the majority of all objects have ratings ranging around 0.5, that there are some highly rated objects, but no objects with ratings below 0.2. Output ratings are defined as the symbolic ratings *vertical line*, *one star*, *two stars*, *three stars*, and *big star*. The vertical line represents the cut-off value, i.e. all objects with query ratings below that point are rejected and will not be delivered to the user. Star ratings

represent increasingly high relevance. In this example, we chose a symbolic output rating scale to clearly distinguish output ratings (stars) from query ratings (numeric ratings). As a matter of fact, both rating types are allowed to use arbitrary types of ratings, i.e. such editors could for example equally well be used to map symbolic query ratings to numeric output ratings.

The handles are used to assign output ratings; accordingly, they are labeled with output ratings. The two-star handle, for example, points to the query rating ¾ thereby expressing that all objects with a ¾ query rating should be assigned a two-star output rating. We arbitrarily chose to use five handles for this example to recall that this interface style works with any number of handles (of course, visual cluttering limits the number of handles that can be operated in a useful way). Dragging handles allows users to change to correspondence between query ratings and output ratings. Since histogram surfaces represent amounts of objects, the histogram visualization can help making positioning decisions by informing the user about which portion of all objects a given rating is assigned to. In the shown state, for example, only a small percentage of all objects is left of the cut-off line, so that almost all objects will be delivered to the user.

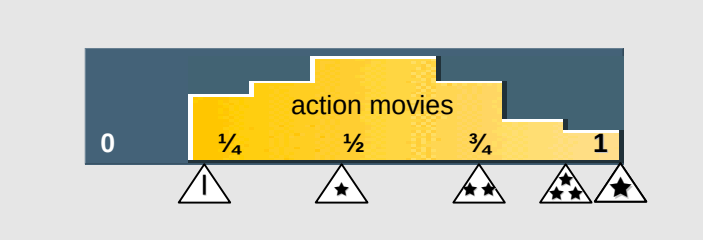


Figure 27: Applying the histogram-based transformation dialog to represent the weighting functions of a QSA query.

4.2.3 Relating query rating histograms to each other

So far, histogram-based interfaces allow users to define the correspondence between query ratings and output ratings for a single query. Since these interfaces can be used to define the rating transformation for each query, they technically provide users with all they need for defining their aggregation functions. Nonetheless, users still lack information for make the right adjustments. The reason is that the purpose of a QSA system is to define an output *ranking*. Defining how query ratings relate to output ratings is only an auxiliary construction; the real goal is to define how the query ratings of the individual queries relate to each other, because this defines the output ranking. For that purpose, a visualization of a single query is not sufficient; in how far a rating transformation has an impact on the output ranking can only be seen from simultaneously looking at multiple queries.

The interface shown in Figure 28 establishes a relation between two histogram components that each represent a different query. Since the triangle-shaped handles under the two histograms represent *the same* three output ratings, handles not only map query ratings to output

ratings, but also relate queries to each other. Each pair of same handles defines a correspondence between two ranks within the two queries. In Figure 28, dotted lines illustrate this correspondence. Dragging the three-star triangle of the *action movies* query to the shown position, for example, expresses “All top 10% *action movies* are preferred over any *comedy*” (the size of the top-ranked region, highlighted in Figure 28a, can be estimated based on its histogram surface, which is roughly 10% of the overall histogram surface).

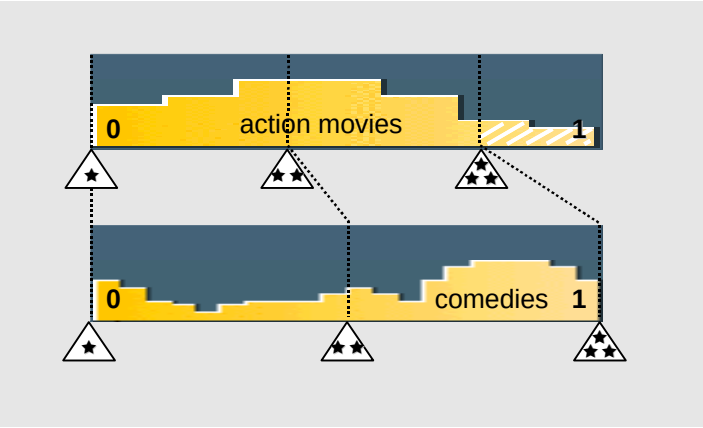


Figure 28: Relating histograms to each other by positioning handles representing output ratings. X-axes of the containers represent query ratings, here ranging from zero to one. The position of the three-star handles expresses a preference for top-ranked action movies.

Figure 29 continues the example with different handles and different positions. This time, the cut-off handle (the vertical line) is used to crop the two queries, such that approximately the bottom 8% and 10% of the *action movie* and the *comedy* queries are removed, respectively. The one-star handle has been used to assign average output rating to above-average comedies. Thereby, a non-linear correspondence between the two queries has been established.

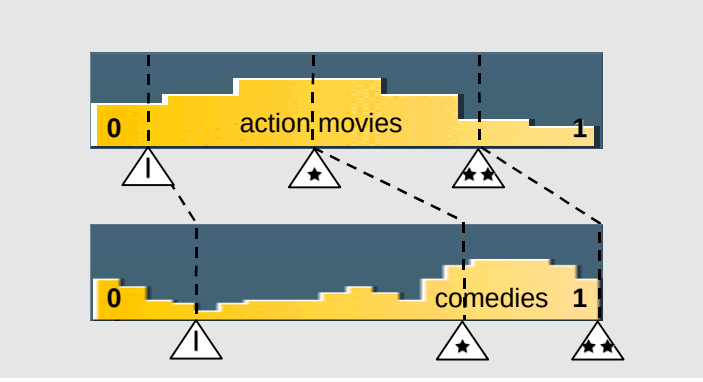


Figure 29: Both queries have been cropped using cut-off handles, i.e. the handle bearing the vertical line symbol. The second query was transformed non-linearly, assigning only average output ratings to above average comedy movies.

Before we look at how this type of user interface can help users entering weighting functions, we modify the interface design again to better highlight the correspondence between queries. For this purpose, we transform the space inside the containers to output rating space. Figure 30 illustrates that. Figure 30a and Figure 30b show the same profile state. In Figure 30a, the two containers represent the respective query rating scales and triangular handles assign output ratings to selected positions. In Figure 30b, containers and contents have been transformed to output rating space. As a consequence, all "handles" are now at fixed positions that correspond to their output ratings. In this interface, *histograms are deformed* to represent the correspondence between query ratings and output ratings. The shown state shows the same correspondence between query ratings and output ratings that Figure 30a did; all handles still point to the same histogram positions. To conserve the information about amounts and ranks that is provided by histograms, histograms are deformed in a surface-preserving fashion. Compared to Figure 30a, the histogram belonging to the *action movies* query in Figure 30b was horizontally compressed; this was compensated by anti-proportionally scaling its height. Since a non-linear transformation was applied to the *comedies* histogram, also histogram height was scaled non-linearly. While the part right of the one-star handle remained almost unchanged, the entire left part was compressed to about half its width. To preserve amount and ranking information contained in this histogram, the horizontally compressed part has become taller instead, so that the surface remained constant.

With this transformation, also the interaction style changes. The handles representing output ratings that we had used in Figure 30a have become purposeless in Figure 30b, because they cannot be moved anymore. These handles are abandoned. Instead, we now use a different set of handles that represent query ratings instead of output ratings. Since it is the histograms that represent query ratings, the new *square-shaped handles* that allow deforming the corresponding histogram are attached to the histograms (Figure 30c). In the new model, handles have to be moved into the opposite direction, to obtain the same effect. In the original model, assigning a one-star output rating to the ¾ *comedies* query rating, for example, required dragging the one-star handle towards the right (Figure 30a); in the transformed model, the ¾-handle is instead moved towards the left (Figure 30c).

Although the cut-off handle belongs to the first model, we preserved it in Figure 30c. Allowing users to assign the cut-off to an arbitrary output rating, permits users to rapidly adjust the overall number of objects to be delivered by their profiles.

The advantage of the transformed interface style is that it allows reading the correspondence between any two ranks in a more intuitive fashion. Output rankings are generated by merging the two weighted histograms in a zipper-like fashion (Figure 30d), so that ranks and ratings within two different queries can now be compared by simply comparing their horizontal position. If two ranks are located in the same column, they are assigned the same output rating indicating that they are equally relevant. Otherwise, the one that lies more to the right is preferable. The fact that top-ranked *action movies* are preferred over *comedies* now shows immediately.

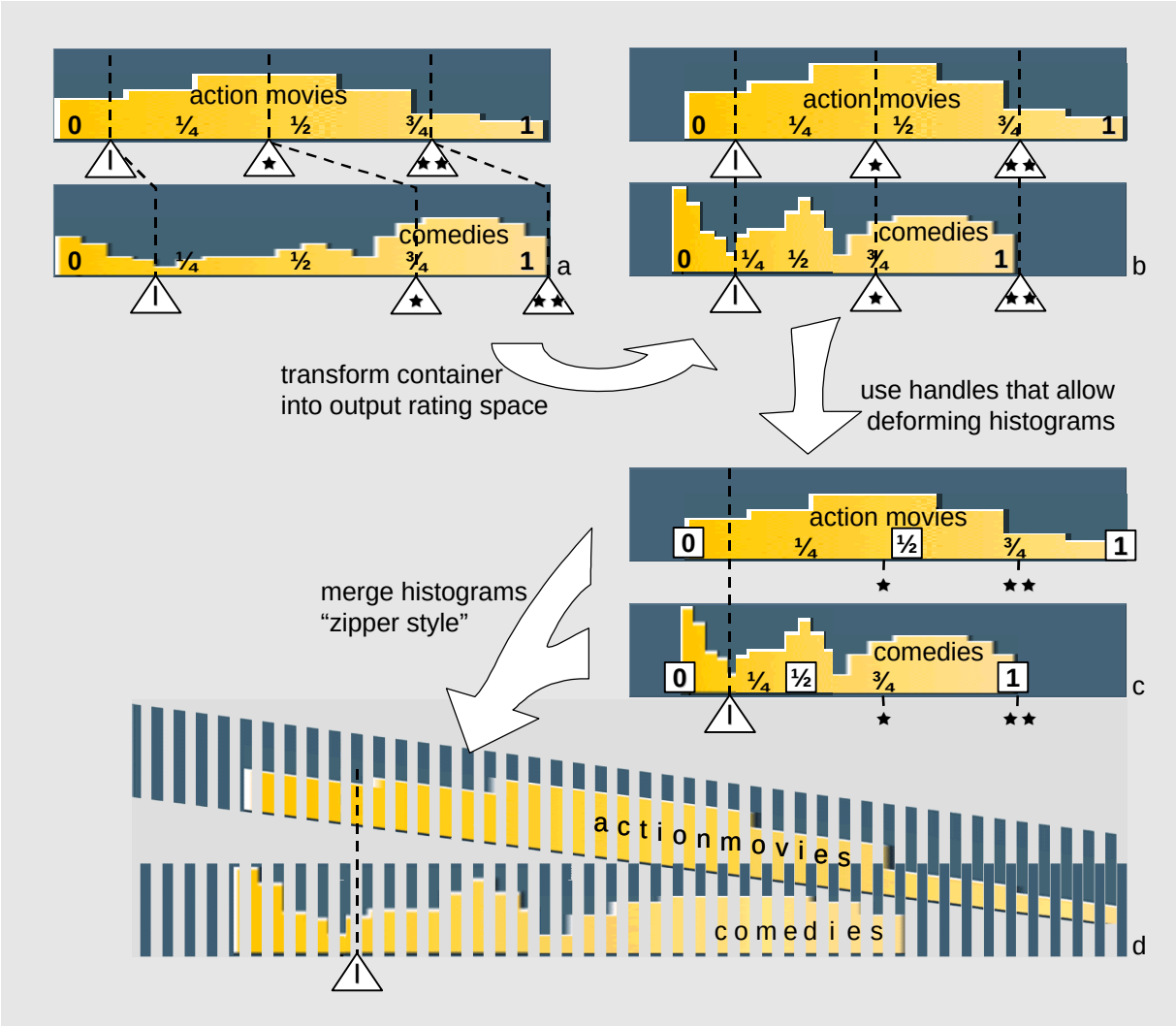


Figure 30: Transforming histograms (a) into output rating space (b) and application of histogram-attached handles (c). Since horizontal positions now represent the same output rating scale, the two weighted histograms can be merged visually (d).

4.2.4 Using histogram-based interfaces

One way of using histogram-based editors is by incrementally adjusting them with respect to actual and hypothetical cut-off values. First, all histograms are positioned with respect to the current position of the cut-off line to obtain the desired amounts of objects from the individual queries. This corresponds to the *amount of objects*-parameter of the form-based interfaces. In the example shown in Figure 31 the cut-off line is shown as a (left) dashed vertical line. The user has decided to retrieve the majority of all programs about *stock news*, all movies starring the actor *Harrison Ford*, and some *comedy shows*.

Now the profile is optimized with respect to additional “hypothetical” cut-off values. To not conflict with the settings regarding the current cut-off line, histograms are *deformed* this time to obtain the correct settings with respect to the hypothetical cut-off line. In the shown example, a hypothetical cut-off line is shown as a dot-dashed line at the right of the original cut-off line. In the shown state, the user has entered the additional information “If I had less time for watching TV, I would watch fewer stock news programs and stop watching comedy shows, but I would still not want to miss a single program starring Harrison Ford”.

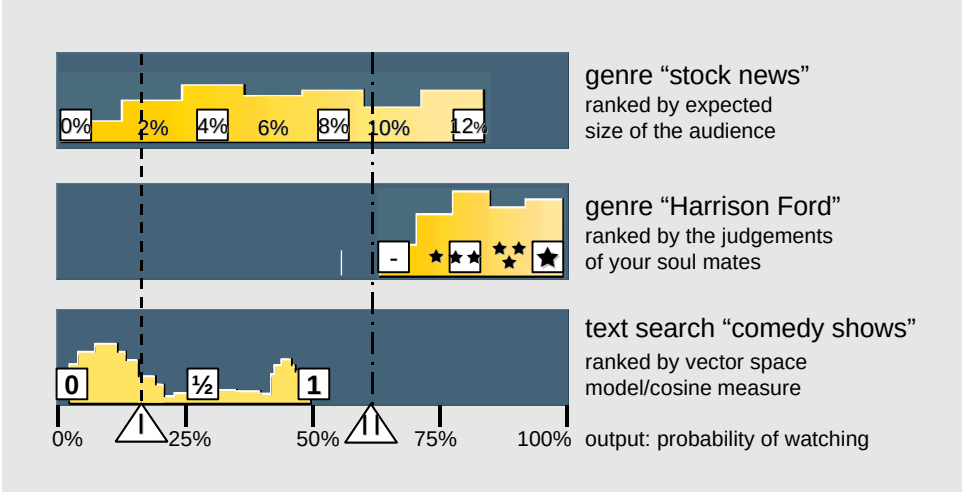


Figure 31: Histogram-based interfaces allow users to optimize the object mix for multiple cut-off values. This interface provides labels for query ratings and output ratings.

Continuing this process of defining amount information for arbitrary cut-off values results in an optimum output ranking. Once it has been achieved, the cut-off value can be changed arbitrarily, the output ranking always remains optimal. This gives users the benefit of *ranked output*, which is that it allows users to maximize precision and recall at *any* cut-off ([Rob77], see also Section 1.1.2.1).

Note that all three queries from Figure 31 are run on different query-execution architectures and thus use different rating scales. The handles attached to each histogram are labeled with different ratings from these different rating scales and even use different numbers of handles. They are used to associate each query rating with an output rating, which is on yet another different rating scale (a probabilistic scale representing the probability of the user watching the respective program). This example illustrates our earlier claim that QuerySet Architecture systems provide effective methods for combining different query models in a single system.

4.2.5 Benefit of histogram-based interfaces

The benefit of histograms in this process is that they allow users to *combine their knowledge* about ratings *and* ranks during profile editing. Figure 32 illustrates the user’s task and shows how histograms support it. The user’s goal is to enter the individual weighting functions forming the aggregation function, i.e. functions mapping query ratings to output ratings (arrow at

the bottom of illustration). Instead of working on *ratings*, users of histogram-based editors arrange histograms to optimize the *output ranking*. During this task, histogram-based editors allow users to combine their knowledge about *ranks*, *ratings*, and *objects* by relating all these properties to output ranks (the three horizontal pointing to output ranks in Figure 32). All three knowledge sources may be important.

1. *Knowledge about query ranks:* As discussed in Section 3.3.6, users may perceive ranking information when working with ranked outputs. Query ranks can have various types of impact on relevance. The relevance of lower-ranked objects, for example, may decrease by redundancy between objects or by saturation effects, e.g. in entertainment-related application areas (see, for example, [Boy82, Soe94, Har92, GL91, Coo73a, Coo73b, Coo78, Bar67])
2. *Knowledge about query ratings:* If queries generate human understandable ratings or if users have the chance to learn the rating scale by query or filtering output being labeled with query ratings, users may be able to associate query ratings with relevance. Users may, for example, remember that all comedy shows they had judged relevant in the past had at least a two-star rating. The interface shown in Figure 31 displays rating scales to allow users to profit from their knowledge about ratings. Query rating scales are displayed, to allow users to interpret the ratings (e.g. expected size of the audience and ratings provided by other users of the system). The third query uses a rating scale that non-expert users can extract very little information from (cosine measure); the display of query ratings is therefore omitted, but users may still profit from the distribution information shown by the histogram. The smaller the absolute value of higher derivatives of the transformation function, the more closely query ratings are related to relevance and the better the intuitive understanding of “more is better” works as expected. Outstanding ratings (Figure 31, the bulb at the right of the bottom histogram), for example, can be perceived to be outstanding even if the rating scale is unknown. See [TC91, p. 198] and [FB90] for a brief discussion about the advantages of displaying ratings.
3. *Knowledge about objects:* If objects are labeled individually (see, for example, the user interface shown in Figure 36 on page 82), users may use these interfaces to associate objects with relevance.

Fully labeled histogram-based editors visualize four properties of objects, i.e. query rating (scale attached to the histograms, see Figure 31), query rank (the histogram surface right of the particle representing the object), output rating (scale attached to the container, see Figure 31), and output rank (the surface of all histogram parts right of the particle representing the object).

Histograms support the information integration process described above by providing the missing links in the computational chain. Individual histograms visualize how query ratings and query ranks of the respective query are associated, establishing the left “link” in Figure 32. The ensemble of all histograms again has histogram qualities, so that the sum of all histograms shows how output ratings relate to output ranks (see also the jelly interface in Figure 37 that visually merges all query histograms into a single huge histogram representing the profile). Figure 33 explains how histograms associate ratings and ranks and compares histograms to bar charts.

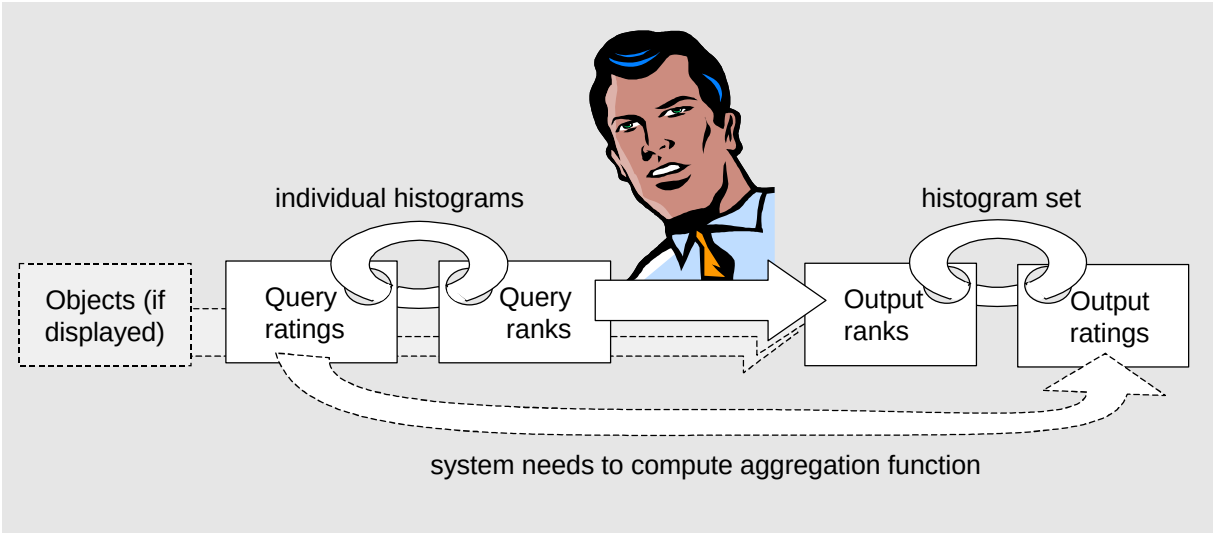


Figure 32: Histograms allow users to work on ranking information (center) and the systems to obtain the required rating information (bottom).

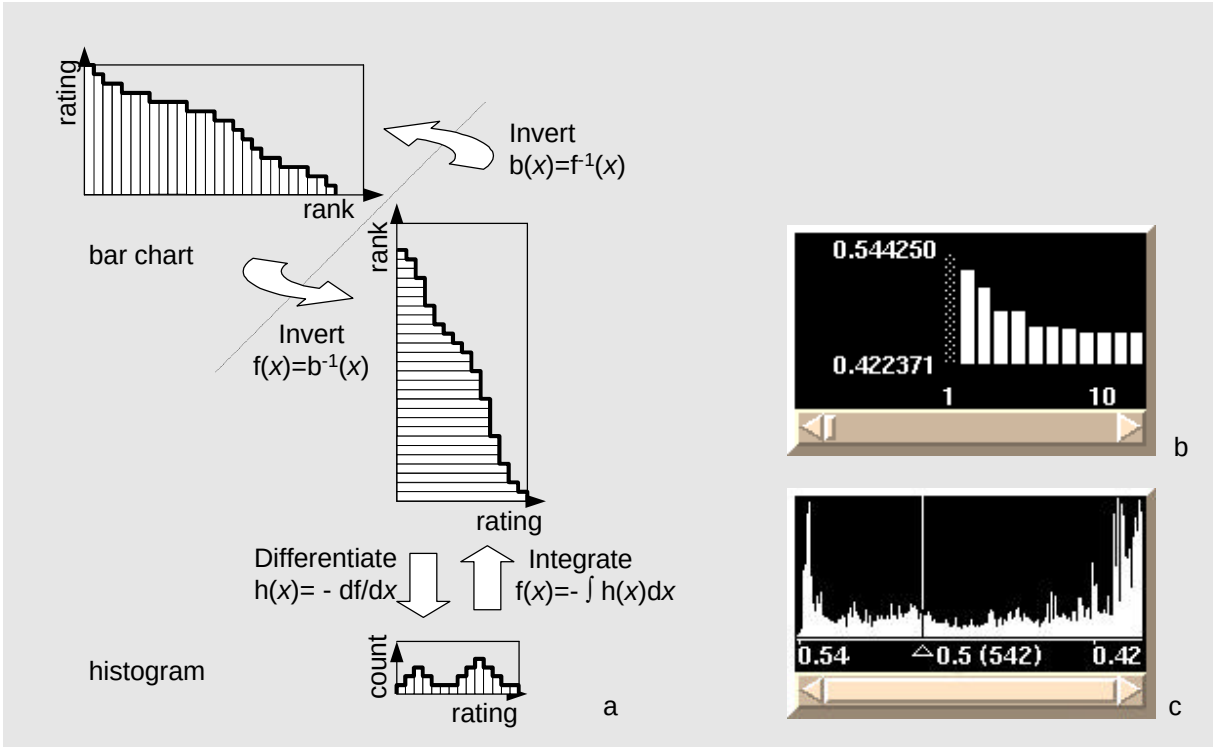


Figure 33: Histograms provide the same information contained in bar charts, and can be converted into each other (a). Histograms are more compact. While bar charts represent each object as a bar, histograms use only a single “grain” to represent each object. By varying the number of bars, histograms can be miniaturized to any size. While a bar chart (Xinquery [CCH92, INQUERY92]) can only show the ratings of a subset of a large ranking (b), a histogram can give an overview over *all* objects (c).

4.2.6 Interaction techniques for handling large histograms

The histograms of queries matching large amounts of objects may clutter the interface. If these queries are used to determine the scale of histograms, queries matching fewer objects may be displayed so small that their histograms are difficult to pick and their triangle handles overlap. The problem can be solved, because the user's mental "bandwidth" is limited. Because of this limitation, generally only a small portion of a large histogram will be delivered to the user; the majority will be cropped at the cut-off line. Allowing parts of histograms to stick out to the left solves the clutter problem, because the major portion of large histograms will then not be visible. The second histogram of the interface shown in Figure 36a uses this technique. (This interface is mirrored, so the histogram sticks out to the right.)

Another reason for leaving space left of the cut-off line (to the right, if the interface is mirrored as is the case in Figure 36a) is to support query reuse. If a sufficient amount of space outside the area selected for output is available, users can store *inactive* queries in this area, i.e. queries that do not currently contribute to the filtering output. Such queries can be useful for representing subjects that are repeatedly of interest, but not all the time. Whenever users want to use such a query they may execute it in an ad hoc fashion (bookmark mode, see also Section 3.4.1) or they may manually activate it by dragging it into the area selected for output at the other side of the cut-off line. The *martial arts query* at the bottom of Figure 36a shows an example.

Dragging histograms out of the visible range requires space outside the visible range. Some rating scales (e.g. some utility-oriented scales) are infinite in both directions and therefore automatically provide such space. For probabilistic scales this is not the case. If a probabilistic output scale is used, space for low-rated objects can be created by using a logarithmic scale (e.g. by converting ratings using $\ln(r)$). The visible range then displays the interval between some minimum range and one. This creates the required space to store the vast masses of objects of low relevance matched by unspecific queries.

If histograms are allowed to stick out of the visible container, a method has to be provided that allows dragging histograms out. The handles used so far cannot do that for obvious reasons. A solution involving a second set of handles as used by the Photoshop *levels dialog* (Figure 23c) is unsatisfactory, because the relation between the additional handles and the histogram is visually unclear. We solved the problem by providing our interfaces with an additional "handle" that allows horizontally dragging the entire histogram. We used the entire histogram surface as a handle (Figure 34a). To facilitate the handling especially of small histograms even further, most interfaces we built make the entire container region usable for histogram dragging.

For interfaces providing two degrees of freedom per histogram, we experimented with an enhancement of this dragging technique. In this *2D drag interaction model*, dragging the histogram upwards makes it tall and thin, dragging it downwards makes it flat and wide (Figure 34b). While the rigid histograms shown in Figure 34 do not invite users to drag histograms vertically, the jelly design shown in Figure 37 provides a much better affordance (see Section 4.2.7). If possible, handles are attached to the top of each histogram; pulling at the top of these histograms will result in the same behavior that a piece of jelly would show—the jelly stretches vertically while becoming thinner. Again, histograms can be grabbed at any position;

the purpose of the handle is only to invite users to discover this functionality. This 2D drag interaction model provides the same degrees of freedom that interfaces with two handles provide, but requires no handles but the histogram surface itself. While handles can disappear if the histogram sticks out of the visible region, the 2D drag interaction technique always remains applicable. The main advantage of this interaction technique is that it is highly efficient. Since both adjustment interactions can be carried out simultaneously, histograms can be adjusted with a single mouse drag interaction. See Section 4.3 for a user study evaluating this interaction technique.

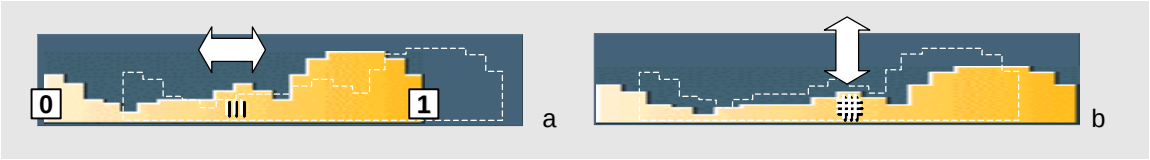


Figure 34: Dragging the entire histogram horizontally (a) and vertically (b).

4.2.7 Example applications

To test our user interface concept, we built a prototypical Web page filtering application, several interface prototypes, and the TV Scout user interface (see also Chapter 5).

The Web page filtering system *Histograms* is a simple prototypical QuerySet Architecture system. It is used as follows. Users create a number of queries using search engine syntax and insert them into their user profile. The profile is initialized according to Zipf's law (see Section 3.3.2), so that more specific queries receive positive rating offsets. When users execute their profiles, the system executes each query from the user's profile on a Web search engine (Excite) and merges them as specified in the QSA profile. The resulting ranked list is formatted like a search engine result list and displayed in a Web browser or fed into an offline reader. The result list can be restricted to unread pages by filtering it with the user's Web browser history file, so that the system provides users only with unread Web pages.

Histograms supports only manual profile updating. To update their profiles, users invoke the system's histogram-based profile editor. To prepare the histogram display, the system test-executes the queries on the search engine, extracts the ratings from the returned pages, and aggregates these ratings to form the histograms. Then the histogram-based profile editor is displayed. Figure 35 shows a screenshot from an example session. The textual information at the bottom right informs the user that the currently displayed user profile *Team members and hobbies* matched 494 web pages, out of which 200 will be delivered to the user. The flying window at the top right shows that the profile consists of the three text searches *Georg Martin Wenzel*, *Juggling Torches*, and *Patrick Baudisch*.

Each of the three queries is represented by a histogram. Histograms are displayed in an overlapping fashion. Clicking a histogram brings it to the front and allows it to be manipulated. The dark histogram that is currently in front represents the query *Georg Martin Wenzel*. This query has the smallest histogram, which indicates that this query retrieved fewer pages than the other two queries; only 37 web pages, as confirmed by the textual query

the other two queries; only 37 web pages, as confirmed by the textual query information at the bottom right of the interface. The histogram's shape gives an impression of the rating distribution of this query. It shows that there is one web page with a particularly high rating (the single bar at the outer right), that the majority of pages have ratings in the upper quarter (the peak next to it), and that there are a few pages with lower ratings (the remaining bars of equal height).

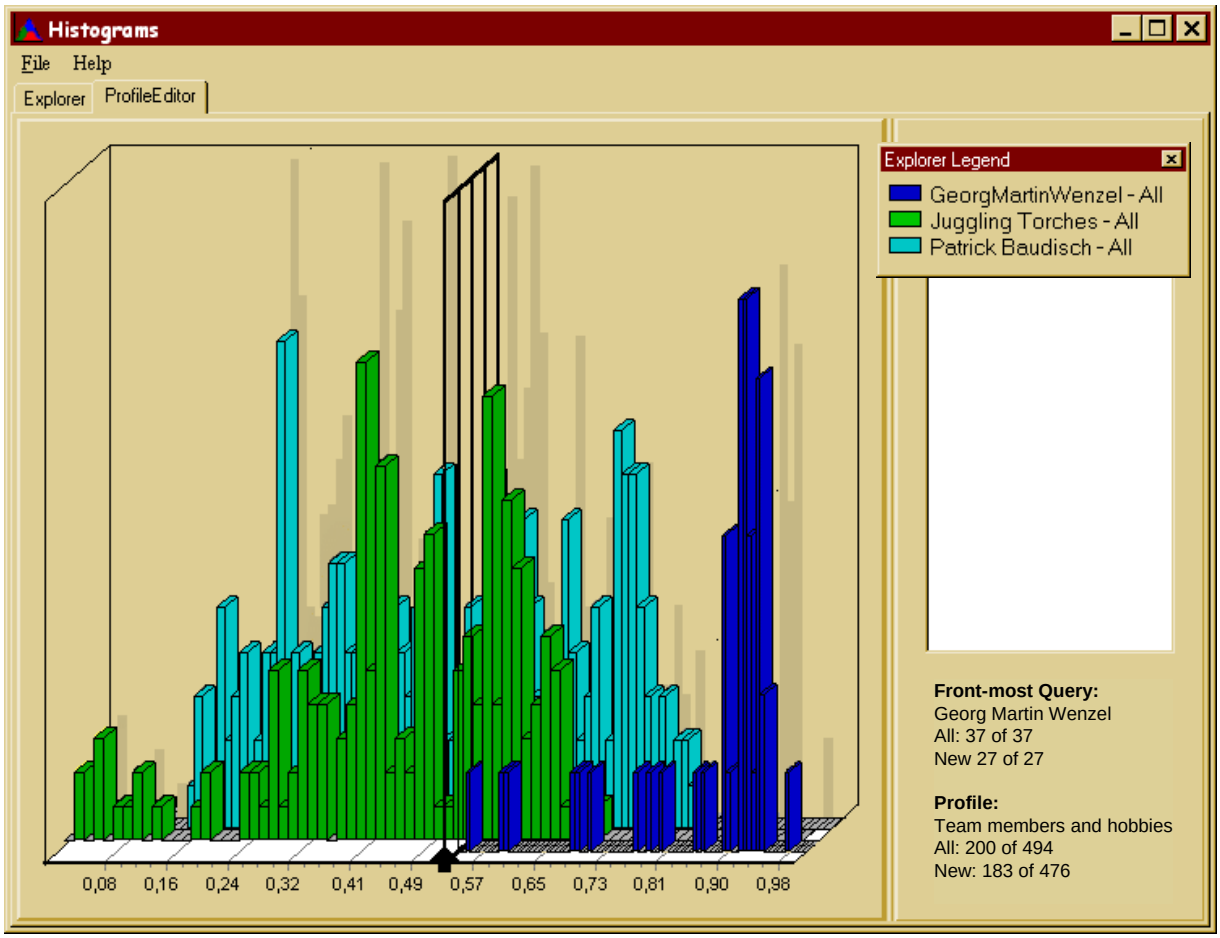


Figure 35: Screenshot of the histogram-based Web page filtering system *Histograms* (Screenshot post processed to improve readability on black and white printouts).

The positions of the individual histograms define which output ratings will be assigned to the individual web pages. The output ratings scale at the bottom of the histogram container labels container areas with output ratings between zero and one. Since the *Georg Martin Wenzel* histogram is completely in the right half of the container, its web pages are all assigned output ratings above 0.5. Since the cut-off plane, i.e. the black wire fence in the middle of the histogram area, is located left of this histogram, all these pages will be delivered to the user. To reduce cluttering, query ratings are not explicitly labeled, so that users cannot tell the exact query ratings assigned to the individual web pages. However, histograms use bars of zero

height to fill up their entire query rating range (the “riffles” at the bottom of each histogram), which gives a rough visual impression of each histogram’s query rating scale.

Users can update their profiles using the 2D drag interaction style described earlier, i.e. they can grab histograms, drag them horizontally and deform them using vertical mouse movements. To give users an overview over their current information intake volume, a “shadow” on the back of the container displays the sum of all query histograms, i.e. if all histograms where stacked, this would be their shadow.

For the TV Scout system (see Chapter 5), we designed the profile editor shown in Figure 36. To avoid cluttering, as was caused by the overlapping histograms in the *Histograms* application, this interface provides each histogram with its own container. To still allow visually comparing multiple queries, containers are relatively flat so that multiple histograms can be viewed simultaneously atop of each other. TV Scout queries can deliver between one and several thousand objects, i.e. TV programs, per week. Since also small queries should be clearly visible and easy to grab and manipulate, the size ratio of 10,000:1 exceeds the available screen surface. Therefore, this interface scales histogram surfaces using a geometric function if necessary. Smallest and largest queries are assigned constant histogram surfaces; histogram surfaces in between are interpolated. In the interface shown in Figure 36, the minimum histogram size is 10x10 pixels; the maximum size is two thirds of the container size. The price for keeping histograms easy to pick and to manipulate is that this distortion biases the correspondence between surfaces and amounts of objects, which makes it more difficult to visually compare the amounts of objects delivered by two queries of different sizes.

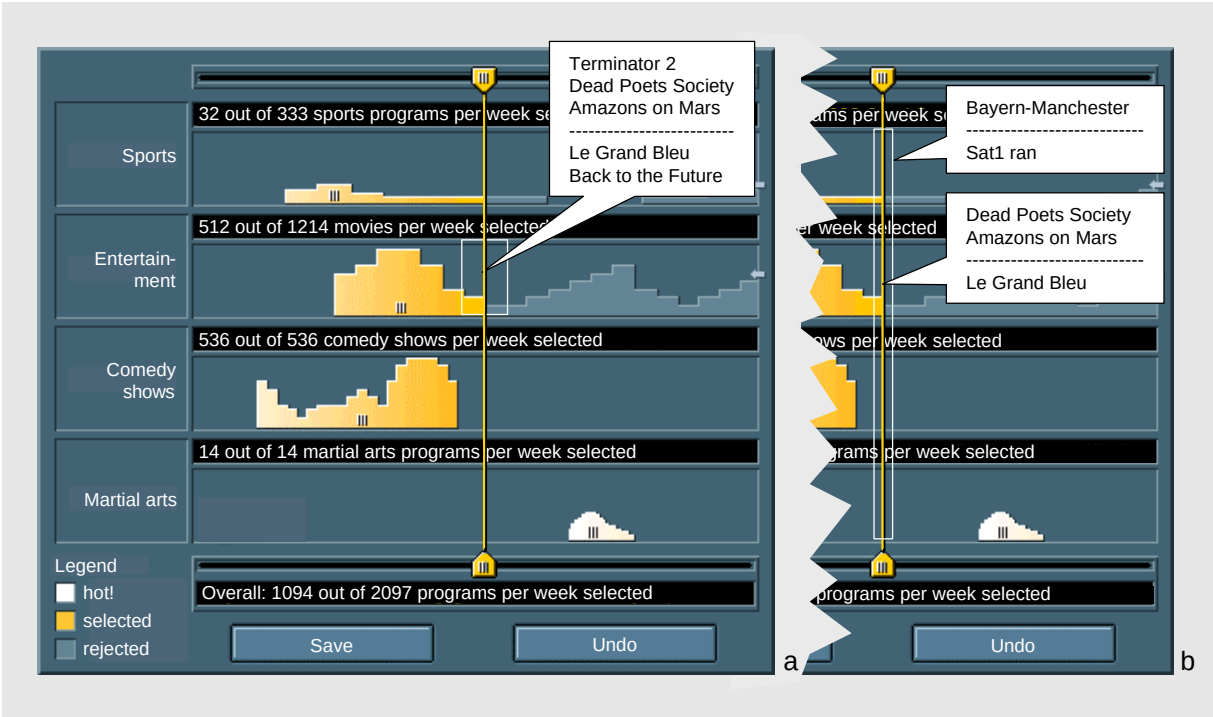


Figure 36: A histogram-based profile editor applied to the TV domain. This prototype of the TV Scout profile editor illustrates histograms with lists of example objects.

Compared to the interfaces presented so far, this interface is mirrored horizontally, so that those histogram parts representing the most relevant objects are now on the left. Some users found this orientation more appropriate, because with this orientation the output order of objects coincides with western reading habits from left to right. This interface uses color to visualize output ratings, i.e. light background color for the cropped part, yellow for selected parts, and a gradient towards white for highly relevant parts. As before, histograms are manipulated using the 2D drag interaction and the cut-off line can be dragged horizontally.

The shown prototype provides users with object examples. Clicking a histogram with the right mouse button brings up a “magnifier” frame around the selected part of the histogram and an attached text bubble that lists example objects having the respective query rating (Figure 36a). Dragging the mouse with depressed right mouse button horizontally updates the bubble’s content continuously. Dragging the mouse vertically extends the magnifier across all histograms (Figure 36b), so that individual queries can be compared based on the example objects. Double clicking the right mouse button locks the magnifier to the cut-off line, so that the effect of histogram drags can be viewed in real-time. For bandwidth reasons, the profile editor applied in the TV Scout system does not support these example bubbles. See [Bau97, Bau98a] for experiences with earlier versions of this interface.

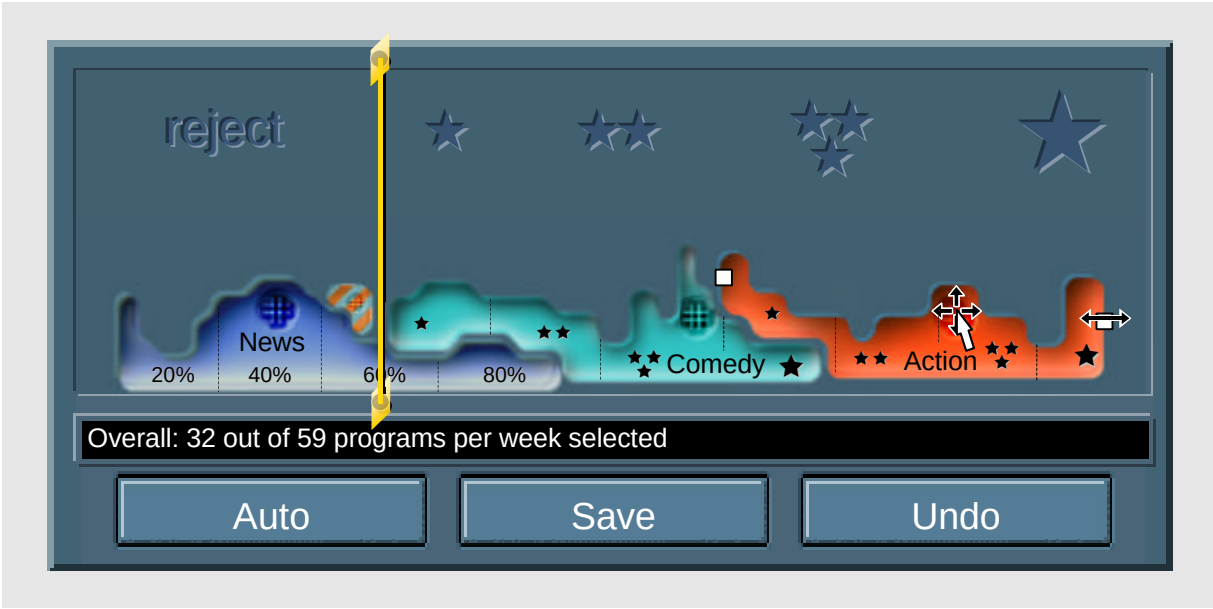


Figure 37: Profile editor using a histogram stack, jelly look. Jelly histograms can be manipulated by dragging them horizontally (shown as a cross cursor) or by grabbing one of the handles attached to a histogram’s edges (white squares) and dragging them, which moves only this edge so that the histogram is deformed (shown as a double arrow cursor).

The latest development in this interface series is a profile editor that uses a *stack of histograms* (Figure 37). In this version, the histograms of all queries are displayed in the same container. Compared to the interface styles used so far, histogram stacks have three advantages. First, they are highly compact without occluding anything. Second, the proximity of the individual queries makes it easier to relate two query histograms to each other. Third, the *output*

ranking is immediately represented in the interface, namely by the histogram that is defined by the sum of all query histograms. The price for these benefits is that the individual query histograms become ragged and are more difficult to recognize as histograms.

To ascertain that histograms remain one piece and are not split into several parts when pulled across steep edges of histograms under them, histograms behave like they were made of jelly. Jelly blobs sink towards the ground as far as possible, but never split into two parts. If necessary, holes in the stack will occur to preserve histogram continuity. The interface shown in Figure 37 uses a jelly-like look for the histograms to emphasize this behavior (see also the discussion in Section 4.3.5).

As mentioned above, the sum histogram in histogram-stack-based interfaces provides a preview of the filtering result. To ascertain the correctness of the amount information shown by the sum histogram, objects matching multiple queries should be represented exactly once. With their separation into multiple containers, the histogram-based interfaces presented earlier required creating an entire new container for each such case or picking *one* histogram to put the objects matching multiple queries into it. All criteria for picking a histogram that we have looked at so far were unintuitive or changed during manipulation. By using only a single container with histograms, stacked histograms provide a natural solution to this problem. Objects matching multiple queries are displayed as additional histograms in the common container. Figure 37 shows an example of such a histogram. The small histogram on top of the *news* histogram represents *action comedies*, i.e. all objects that match the query *action and* the query *comedy*. To reduce cluttering, it is not labeled but striped in the colors of its parent histograms *action* and *comedy*. All aggregation function models that we looked at base the ratings of objects matching multiple queries are typically on the ratings assigned to the individual queries; histograms representing multiple matches therefore automatically follow their "parent histograms" whenever one of them is dragged. In QSA systems that allow assigning independent ratings to objects matching multiple queries (see Section 3.3.3), these histograms can also be dragged individually. In the shown case, the user used this feature to get rid of *action comedies*. While *action* and *comedy* are both accepted entirely, the user obviously dislikes the combination of both and thus dragged the *action comedy* histogram beyond the cut-off line.

4.3 EXPERIMENTAL EVALUATION AND COMPARISON

To verify the validity of our interface concepts we conducted a controlled experiment on four different types of profile editors (two form-based and two histogram-based editors) plus a control group using an interface that only allowed users to select queries, but not to enter user-aggregated relevance feedback. The goal of the experiment was (1) to prove that the individual profile editors are usable, useful, and learnable and (2) to gather evidence about the strengths and weaknesses of the four interfaces as a basis for deciding, which of these interfaces or which combination should be applied in QSA systems, especially the TV Scout system (Chapter 5).

4.3.1 Subjects, apparatus, and interfaces

The subjects were 30 students from Darmstadt University of Technology who volunteered for the experiment. Subject ages ranged from 19 to 31 and 37% were female. All subjects had at least some previous computer experience, but no experience with any of the interfaces used in the experiment. There was no significant influence of age, sex, and education on performance during the experiment.

Experiments were run on Toshiba Tecra 740 CDT notebook computers with a 13.3 inch (33.8 cm) TFT color display and an external two-button mouse. The operating system was Microsoft Windows 95. Screen resolution was adjusted to 1024 x 768 pixels. Form-based interfaces were programmed in html, histogram-based interfaces in Java, and were run on Netscape Navigator 4.05. All three form-based interfaces covered the same screen size of 14.5 cm x 17 cm; the two histogram-based interfaces measured 12 cm x 14.5 cm.

Four different versions of QuerySet Architecture profile editors plus a control group editor were included in the experiment. These editors allowed subjects to view the current state of their user profiles and to adjust the weighting function of the individual queries. All editors were applied to given default profiles. The interfaces did *not* include the functionality necessary to create such profiles (e.g. query formulation). Instead, we used predefined profiles to assure comparability between the individual test groups.

To give subjects easier mental access to the experiment, we set up the experiment with the background of being about TV interests. But since users dealt only with predefined interests and not with any TV programs or program descriptions, the actual application area would have no influence on the experiment[18]. This predefined profile contained the seven queries *table tennis*, *women's programming*, *basketball*, *action*, *movies*, *sports*, and *information*, described as delivering 1, 2, 5, 15, 30, 100, and 700 programs per week. Using this distribution, very specific interests ("table tennis", 1 program per week), as well as very general interests ("Information", 700 programs per week) were included. Altogether the profile would deliver 853 TV programs per week, which corresponds to the amount of programs these queries would return for about 30 TV channels available in Germany.

Figure 38 shows the individual interfaces used in the experiment. The form-based interfaces basically corresponded to the form-based interface presented in Section 4.1 (Figure 20 and Figure 21). The 2F interface allowed subjects to enter an amount and a rating parameter per query (Figure 38a). The 1F interface (Figure 38b) used the faucet interaction technique, allowing users to enter a pair of such parameters per query. The 0F interface (control group) allowed subjects only to select or deselect entire queries for inclusion in the profile[19]. Toggle switches were used for this purpose (Figure 38c).

---

[18] The experiment was designed to evaluate the appropriateness of the user interfaces, not the performance of the underlying query subsystems. We therefore chose not to display any actual objects, i.e. actual TV program descriptions, to avoid affecting subjects' judgments of the UI performance. Future experiments evaluating the *filtering performance* of QSA systems will, of course, have to present actual content to users.

[19] This interface style is different from the others in that it does not allow users to define *how* queries will be combined into a ranked output. This experiment, however, only looked at interfaces; it *did not* compare the retrieval quality obtainable with the respective interfaces and which output style, i.e., single ranked output or multiple ranked outputs, subjects prefer.

The histogram-based interface 2H (Figure 38d) corresponded to the interfaces shown in Figure 36 with the difference that it used color gradients from yellow to red to visualize relevance instead of gradients from yellow to white. It did not provide the example bubble feature to keep the subject's taste about TV programs from influencing the experiment. It used the 2D drag interaction (see Section 4.2.6) and rating scales were not labeled. The rating distributions visualized by the histogram-based interfaces corresponded to the size of the audience ratings that were taken from [GFK97]. The 1H interface was identical to the 2H interface, but did not support histograms deformation, so that histograms could only be dragged horizontally, but not deformed.

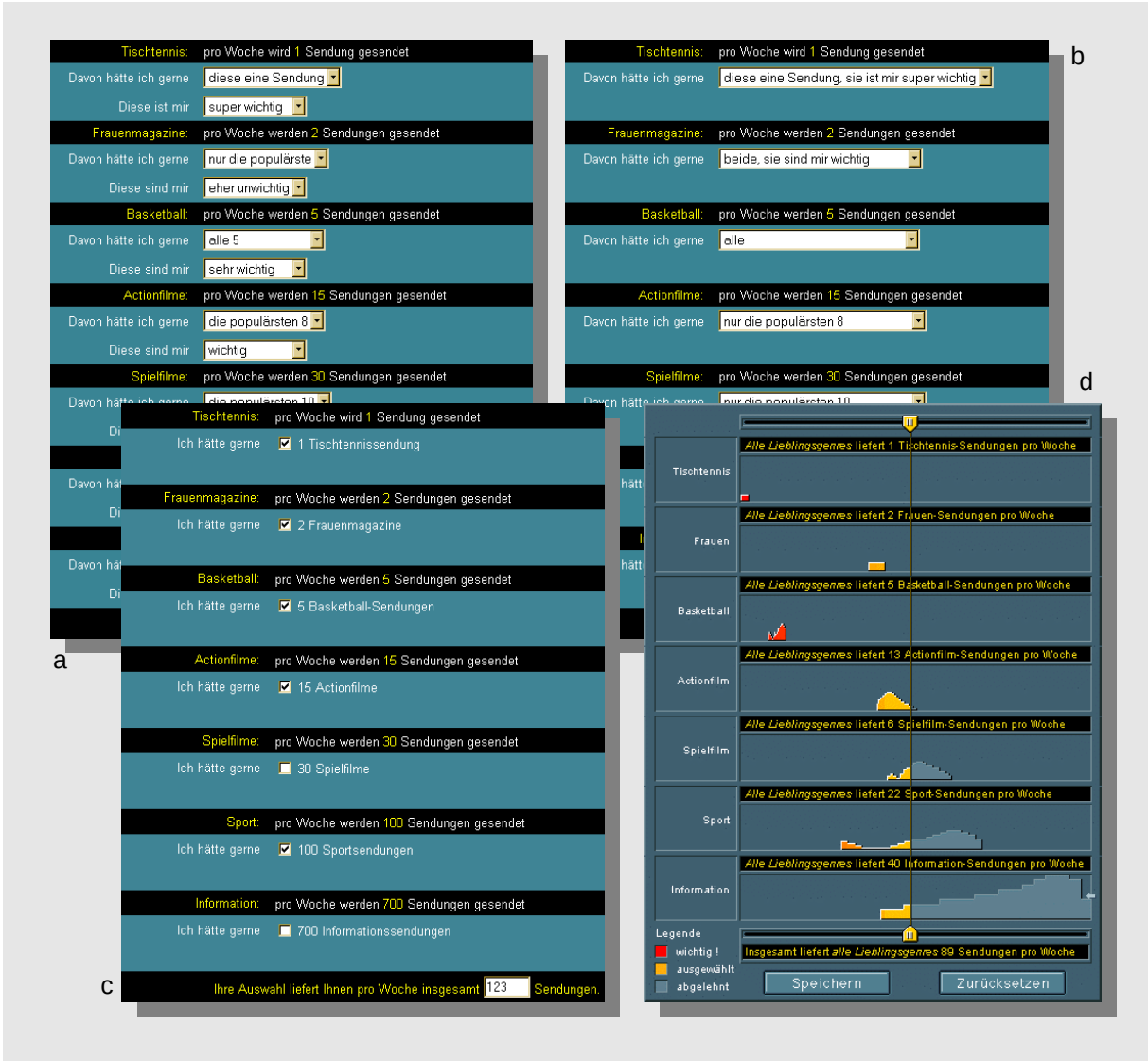


Figure 38: The 2F-interface (a), the 1F (b), the 0F interface of the control group (c), and the 1H/2H interfaces (d).

Histogram-based interfaces basically provided the same functionality as the corresponding form-based interfaces. The difference was that the histogram-based interfaces provided a finer granularity and visualized rating distributions. We did not include a “zero-dimensional” version of the histogram-based style, because its functionality would have been identical to the form-based 0F interface.

4.3.2 Hypotheses

The experiment was designed to examine the validity of the following six hypotheses.

Hypothesis 1 (Learnability form-based): Form-based interfaces are especially easy to learn. Form-based interfaces are immediately learnable for first-time users without any training.

Hypothesis 2 (Learnability histogram-based): Histogram-based interfaces are learnable in a situation where users already have pre-experience with form-based interfaces (see below).

Hypothesis 3 (Usability and utility): All four proposed profile editors (1F, 2F, 1H, and 2H) are sufficiently easy to operate, efficient, and useful.

Hypothesis 4 (Preference over control group interface): Although subjects cannot see the effect of more precisely defined profiles on retrieval quality during the experiment (precision of the ranked output), they still experience QSA editors (1F, 2F, 1H, and 2H) as more flexible and expressive and will therefore judge them as preferable to the control group interface (0F).

Hypothesis 5 (1F vs. 2F): The 1F interface style is more efficient, but the 2F interface style provides higher precision.

Hypothesis 6 (Histogram vs. form-based): Subjects prefer histogram-based editors to the form-based styles.

Hypotheses 1 and 3 to 6 are rather straightforward; only Hypothesis 2 requires additional explanation. The straightforward hypothesis would have been to evaluate the learnability of histogram-based interfaces without preconditions. We chose this hypothesis and the resulting experimental design to help answering the question which user interface(s) to deploy in the TV Scout system (see Chapter 5). Since form-based interfaces contain only standard interface elements, i.e. pull down menus, the learnability question is basically reduced to whether the parameters are chosen appropriately. Since the histogram-based interfaces require users to enter the same type of parameters *plus* to learn a new interaction style, it was clear that the form-based interface would be easier to learn. Therefore, form-based interfaces became the default interface for the TV Scout system. The remaining question was whether to provide histogram-based interfaces as an add-on. This involved the question of learnability, i.e. would subjects be able to transfer the task-related experience from the form-based interfaces as expressed in Hypothesis 2? Would users decide to switch to the more powerful, but also more complicated histogram-based editors or would they decide to stick with the form-based interfaces (Hypothesis 6)? We modeled this application scenario with an experimental design in which subjects first had to complete tasks using form-based interfaces, then switched to histogram-based interfaces. A direct comparison of all five interfaces (using five different test groups) still contains many interesting questions and may be the subject of future experiments.

To have a maximum impact on the TV Scout design, the experiment was designed to deal with first-time users. The TV Scout is an Internet-wide available system that first of all has to address itself to first-time users. To model the situation encountered by actual users of the TV Scout on the Web, we provided users with *no* training. The usability of the individual profile editors with respect to *experienced* users may be the subject of future experiments.

4.3.3 Procedures

Subjects were randomly assigned to one of the three groups referred to as 0D group, 1D group, and 2D group, so that there were ten subjects in each group. All subjects were given the same general instructions. Before the actual experiment, subjects filled in a questionnaire containing information about age, sex, education, and six different dimensions of computer pre-experience, including technical skills, such as mouse usage, as well as the knowledge about involved metaphors, such as histograms.

The experiment consisted of three parts, i.e. a set of tasks on form-based interfaces, the same tasks on histogram-based interfaces, and an informal comparison of all five interfaces. In all parts of the experiment, performance was measured as several facets of subjective satisfaction with the interaction task. The questionnaire we used for recording this information was loosely based on the QUestionnaire of User interface Satisfaction (QUIS) [QUIS, CDN88]. Most user ratings were given on a nine-item scale, such as “rigid (1)…flexible (9)”. See Appendix A for a copy of the Questionnaire used in the experiment (German). Throughout the whole experiment, subjects were encouraged to think aloud.

Part 1 (Form-based tasks): In the first part of the experiment, subjects had to enter a set of given interests into the form-based profile editor assigned to the subject’s test group. All subjects were given the same description of a fictitious interest profile that subjects were asked to consider as their own during the experiment. This interest profile defined the relation of the subject to the seven interests contained in the interest profile held by the profile editors. The textual description included vague information about the intensity of interests and vague information about their desired amounts of objects. For example “You actively play table tennis. Therefore, table tennis is very important to you and you do not want to miss a single program” or “You want to be up to date on current information, but you do not want to invest too much time in it. You find it sufficient to have the choice between a few information programs”.

When subjects had internalized the profile, they were presented the respective profile editor. Subjects in the 0D, 1D, and 2D groups were given 0F, 1F, and 2F interfaces, respectively. Subjects were told that this profile editor interface was the user interface to an information filtering system allowing them to receive personalized TV schedules. They were told to assume that TV programs in each interest category were sorted by an excellent, though not necessarily perfect system according to their personal preferences. Then subjects were asked to enter the described profiles. Subjects were allowed to enter settings in any order until satisfied with the result. We did not restrict the time for entering user profiles, but none of the subjects required more than two minutes for this task. All subjects were able to enter a user profile that corresponded to the given description.

Next, subjects had to perform two refinement tasks. For the first refinement, they were told that their interests had changed in a way that they would now like to receive many information programs, instead of the few ones they had selected before. When subjects were done with these adjustments, they were told that their available time for TV watching had decreased. Subjects were asked to adjust their profiles to reflect a reduction of their TV consumption down to roughly 30%[20]. After subjects had completed the profile adjustment task and the two refinement tasks, their subjective satisfaction was recorded using the adapted QUIS.

Part 2 (Histogram-based tasks): In part two of the experiment, subjects from the 1D or 2D groups were given the corresponding histogram-based interface and were instructed to repeat the same three tasks with this interface, i.e. entering the given profile as well as the two refinements. The only explanation given was that this interface was functionally similar to the form-based interface they had tried before and that histograms could be dragged with the mouse to enter settings. When subjects were done, again their subjective satisfaction was recorded using the adapted QUIS. We included three additional questions directly comparing histogram-based and form-based interfaces. To evaluate the subjects' understanding of the histogram-based style, we asked subjects to explain individual features of the histogram-based interface and assessed their responses.

Part 3 (Comparison and ranking): In the third and final part of the experiment, subjects were presented with all remaining interfaces, so that they could compare all five interfaces 0F, 1F, 2F, 1H, and 2H. In this phase, subjects were allowed to freely experiment with all these interfaces. Filling in another questionnaire comparing and ranking all five interface styles concluded the experiment. Each session took about 45 minutes.

4.3.4 Results

We will present the results in the order of the individual hypotheses.

Hypothesis 1: "Form-based interfaces are especially easy to learn". The questionnaire filled in at the end of part 1 of the experiment clearly confirmed this hypothesis. Subjects learned to use the form-based interfaces rapidly and without any help or training. Subjects had no difficulty understanding the used URF parameters (ratings and amounts), which confirms the choice of these URF parameters for building profile editors. Subjects confirmed the learnability of form-based interfaces; the 1F and 2F interfaces received learnability ratings of 8.2 (standard deviation 1.32) and 8.3 (standard deviation 1.89) (8.9(0.33) after removal of one outlier), see Table 1, top row.

[20] The original histogram-based interfaces allow dragging the cut-off line to carry out this type of amount adjustment task especially efficiently. To make all interface styles comparable, this function was disabled during the experiment.

		Results: mean (standard deviation)				
		0F	**1F**	**2F**	**1H**	**2H**
	Difficult (1)… easy (9) to learn (*Learnability*)	9.0 (0.00)	8.2 (1.32)	8.3 (1.89) 8.9(0.33#)	7.0 (0.25)	4.5 (2.46)
	Difficult to operate (1)… simple (9) (*Usability*)	8.7 (0.48)	7.6 (1.43)	7.6 (1.65)	6.4 (1.65)	4.7 (2.79)
Efficiency	Entering the given profile was inefficient (1)… efficient (9)	5.8 (2.6)	7.1 (0.88)	6.5 (1.72)	7.0 (1.41)	6.4 (2.5)
Efficiency	Entering interest changes is inefficient (1)… efficient (9)	5.8 (1.75) 6.2(1.20#)	6.6 (1.43)	7.0 (1.7)	7.7 (1.57	7.4 (1.78)
Utility	Entering the given profile was insufficient (1) exhaustive (9)	3.3 (2.3)	6.4 (1.43)	7.7 (1.06)	7.5 (1.08)	8.4 (0.7)
Utility	Entering subj.'s *own* interests insufficient (1) exhaustive (9)	2.8 (1.5)	5.7 (2.06) 6.2(1.30#)	7.3 (1.7)	6.8 (2.3) 7.4(1.13#)	8.0 (1.41)
Utility	Interests can be represented imprecisely (1)…precisely (9)	1.4 (0.52)	5.1 (2.33)	6.0 (1.89)	7.3 (1.34)	6.3 (2.00)
Utility	Controlling amount of objects impossible (1)… well pos. (9)	3.5 (2.3) 2.9(1.36#)	6.1 (1.73)	6.9 (1.85)	7.7 (0.95)	8.7 (0.67)
Utility	Rigid (1)… flexible (9)	1.3 (0.7)	4.6 (1.84)	4.2 (2.66)	6.8 (1.32)	8.1 (0.88)
Utility	Functionality too little (1)… appropriate (5)…too much (9)	1.6 (0.84)	3.5 (1.27)	4.0 (1.05)	4.2 (1.03)	5.6 (1.43)
Overall	Horrible (1) … wonderful (9)	3.3 (1.95)	5.6 (1.35)	5.1 (1.66)	7.0 (1.89) 7.6(0.73#)	5.3 (2.63)
Overall	Frustrating (1)… satisfying (9)	2.7 (1.16)	5.8 (1.69) 6.2 (1.1#)	5.0 (2.05)	7.0 (1.89) 7.6(0.73#)	5.4 (2.46)

Table 6: Results of the questionnaire filled in by subjects after part 1 and part 2 of the experiment. Ratings of interface styles 1F and 1H provided by subjects of group 1D, those for 2F and 2H by group 2D, and those for 0F by group 0D. Table cells contain mean (standard deviation) (# after removal of one outlier).

Hypothesis 2 "Histogram-based interfaces are learnable in a situation where users already have pre-experience with form-based interfaces". For the 1H interface, i.e. the interface that allows histograms to be shifted, but not to be deformed, this expectation was fulfilled. All subjects were able to carry out the assigned tasks correctly and felt sure about the correctness of their solution. Obviously, all subjects had gotten the transfer from the form-based interfaces to the histogram-based interfaces.

To check which aspects of the interaction were understood, we asked subjects to explain selected functions of this interface and assessed their answers (Table 2). All subjects were able to explain that dragging histograms horizontally selected amounts; almost all subjects could explain the function of cut-off line and the color gradient. Only about half of the subjects, however, understood the surface property of histograms, i.e. that histogram surfaces corre-

spond to the amount of represented objects. For the 1H interface, however, this did not lead to any problem and the interface received a mean of 7 in learnability with a std. deviation of 0.25 (Table 6, row 1).

	1H	**2H**
Horizontal histogram dragging (0-2)	2.0 (0.00)	2.0 (0.00)
Vertical histogram dragging (0-2)	–	1.6 (0.70)
Histogram surface = amount of objects (0-2)	1.1 (0.99)	1.3 (0.95)
Cut-off line controls overall amount (0-2)	1.8 (0.42)	2.0 (0.00)
Color gradient denotes ratings (0-2)	1.9 (0.32)	2.0 (0.00)

Table 7: Which properties of histogram-based editors did subjects understand; where did problems occur? Result of quiz. Experimenter rated subjects answers as 0 = not understood, 1 = partially understood, and 2 = fully understood.

The lack of understanding of the surface property, however, led to problems with the 2H interface, which allowed histograms to be deformed. Those subjects who did not understand the surface property found the height changes of histograms during the deformation confusing. Thinking aloud, several subjects wondered what the "height of these boxes" meant. Although all subjects were able to successfully complete the tasks and eight out of ten subjects were still able to explain the use of deformation (namely to assign different ratings to top- and bottom ranked objects, results shown in row 2 of Table 7), the non-understanding of the surface property (row 3 of Table 7) was significantly correlated with a lower value in the satisfaction rating. Consequently, 2H interfaces received a mean learnability rating of only 4.5 (standard deviation 2.46, Table 6, row 1).

Hypothesis 3: "All four proposed profile editors (1F, 2F, 1H, and 2H) are sufficiently easy to operate, efficient, and useful". The usability of the four interface styles was judged positively with mean ratings in the range 6.4 to 7.6 for usability and efficiency (Table 6, rows 2, 3, and 4), for both entering a user profile from scratch and for making updates. The only interface that received below-average usability ratings was the 2H interface, with a mean of 4.7 in usability (standard deviation 2.79). The result of the 2H interface may partially be explained by the poor learnability of this interface, but should mainly be caused by the 2D drag interaction technique, which some subjects mentioned to be confusing.

Positive results were achieved for the individual utility dimensions (Table 6, rows 5 to 10). Figure 39 illustrates these results. According to the subjects' ratings, all four interfaces allowed subjects to enter the given user profiles rather exhaustively (means between 6.4 and 8.4, Table 6, row 5). Maybe even more important, subjects judged all editors to be able to represent *their own interests* rather exhaustingly (means between 5.7 and 8.0, Table 6, row 6). All four interfaces allowed subjects to control the amounts of objects their profiles would deliver (means between 6.1 and 8.7, Table 6, row 8) and all four interfaces allowed representing interests rather precisely (means between 5.1 and 7.3, Table 6, row 6). Maximum precision was obtained with the 1H interface (mean 7.3, standard deviation 1.34); the coarse amount

steps of the pull-down menus limited the achievable precision of the form-based interfaces (1F: mean 5.1, 2F: mean 6.0). Some subjects mentioned that the simultaneous manipulation of histogram position and shape using horizontal and vertical mouse movements made it difficult to obtain accurate results, which caused that the 2H interface was judged less precise than the 1H (mean 6.3). The form-based styles were judged rather rigid (means 4.6 and 4.2, Table 6, row 9), while the histogram-based interfaces, especially the 2H, were judged rather flexible (means 6.8 and 8.1).

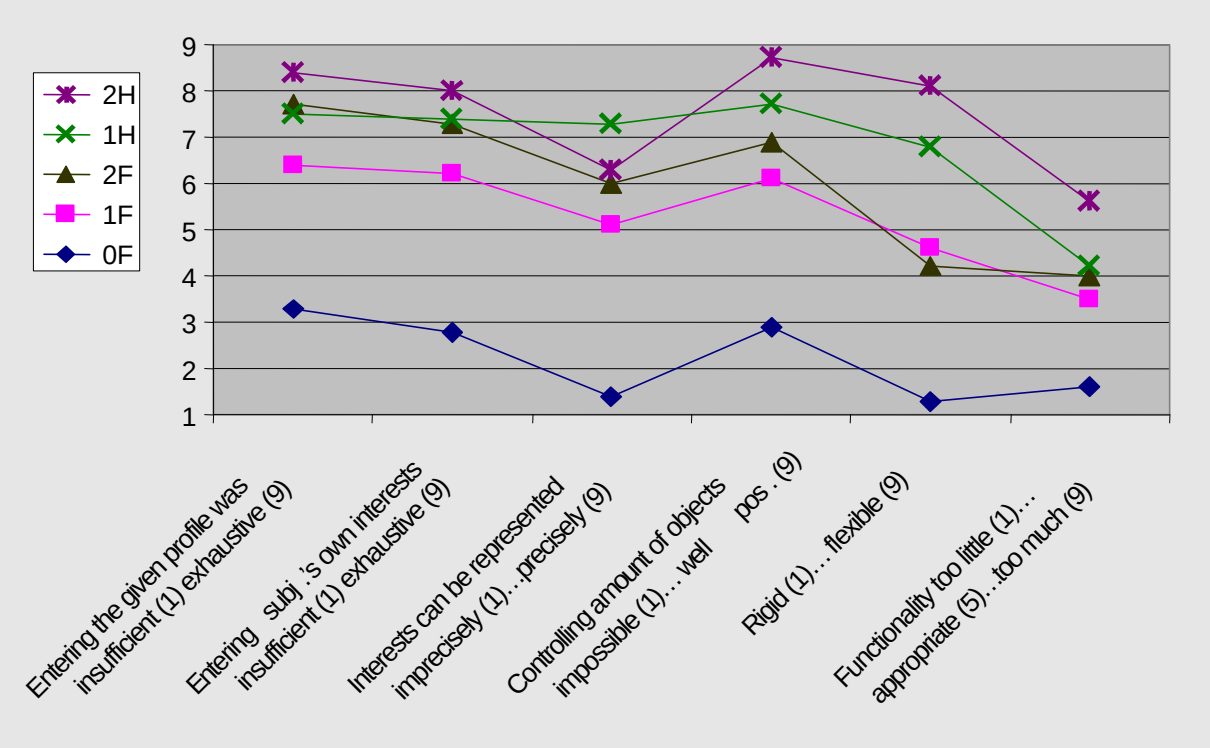


Figure 39: Means of utility and functionality related ratings (Table 6, rows 5–10).

Subjects judged the functionality of all four interfaces as close to appropriate (Table 6, row 10). The functionality of the 1F (means 3.5) and 2F interfaces (means 4.0) was judged as slightly too little, 1H (means 4.2) closer to fully appropriate (corresponding to a rating of 5). Only the 2H interface overshot the goal; with its means of 5.6 it already offers more functionality than would have been appropriate. These results were confirmed in part three of the experiment where all subjects rated the functionality of all five interfaces (Figure 40).

a

	0F	1F	2F	1H	2H
Func-tionality	1,1 (0,43)	2,8 (0,75)	4,2 (0,95)	4.1 (0.72)	5.4 (0.74)

b

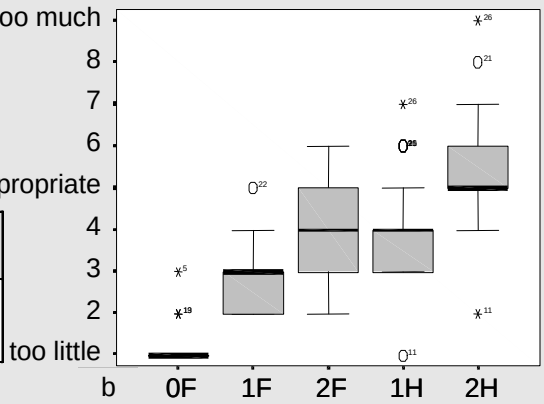


Figure 40: Rating of functionality during the direct comparison of all five interfaces. All thirty subjects rated each interface, also those they had not used during the tasks. Table cells contain mean (standard deviation). Ratings: 1=too little, 5=appropriate, 9=too much. All pair-wise differences besides 1H/2F are significant at 0.01 (paired/independent T-test).

Hypothesis 4: "Subjects experience QSA editors (1F, 2F, 1H, and 2H) as more flexible and expressive and therefore judge them as preferable to the control group interface (0F)." Table 8, columns 1 and 2, shows a comparison of the control group interface 0F with the 1F and 2F interfaces (individual results in Table 6). Since the 0F required only clicking toggle switches, the 0F was judged even slightly easier to learn (Table 8, row 1). With respect to usability and efficiency there where no significant differences. All utility and functionality related ratings of the 0F were significantly lower than the ratings of all other interfaces[21] (Table 8, rows 5-10, and chart in Figure 39). The 0F was judged very rigid (mean 1.3, standard deviation 0.7) and as providing too little functionality (mean 1.6 standard deviation 0.84). Consequently, the 0F was judged significantly less "wonderful" and "satisfying" than all other interfaces (Table 8, rows 11 and 12). In the direct comparison, all subjects preferred any other interface to the 0F interface (Figure 41), so that hypothesis 4 was clearly confirmed.

[21] Since 1H and 2H were assessed in a different situation, a direct comparison with the 0F interface is of course only justified with the 1F and 2F interfaces.

		Comparison: difference, *significance					
		1F-0F	***2F-0F***	***1H-1F*** [22]	***2H-2F*** [22]	***2F-1F***	***2H-1H*** [23]
	Difficult (1)… easy (9) to learn (*Learnability*)	*-0.8**	*-0.7* *–0.1*[#]	*-1.2**	*-3.8** *-4.4**[#]	*+0.1* *+0.7**[#]	-2.5*
	Difficult to operate (1)… simple (9) (*Usability*)	*-1.1*	*-1.1*	-1.2	*-2.9**	*0.0*	-1.7
Efficiency	Entering the given profile was inefficient (1)… efficient (9)	*+1.3*	*+0.7*	*-0.1*	*-0.1*	*-0.6*	*-0.6*
Efficiency	Entering interest changes is inefficient (1)… efficient (9)	*+0.8* *+0.4*[#]	+1.2 +0.8[#]	*+1.1**	+0.4	*+0.4*	*-0.3*
Utility	Entering the given profile was insufficient (1) exhaustive (9)	+3.1*	+4.4*	*+1.1**	+1.7[*]	+1.3*	*+0.9**
Utility	Entering subj.'s *own* interests insufficient (1) exhaustive (9)	+2.9* +3.4**[#]	+4.5*	+1.1 +1.2**[#]	*+0.7*	+1.6 +1.1[#]	*+1.2* *+0.6*[#]
Utility	Interests can be represented imprecisely (1)…precisely (9)	+3.7*	+4.6*	*+2.2**	*+0.3*	+0.9	*-1.0*
Utility	Controlling amount of objects impossible (1)… well pos. (9)	+2.4* +3.2**[#]	+2.4* +3.0**[#]	+1.6*	*+1.8**	+0.8	*+1.0**
Utility	Rigid (1)… flexible (9)	*+3.3**	*+2.9**	+2.2*	*+3.9**	-0.4	*+1.3**
Utility	Functionality too little (1)… appropriate (5)…too much (9)	*+1.9**	*+2.4**	*+0.7**	*+1.4**	+0.5	*+1.4**
Overall	Horrible (1) … wonderful (9)	+2.3*	*+1.8**	*+1.4** *+2.0** [#]	*+0.2*	*-0.5*	*-1.7* *-2.3**[#]
Overall	Frustrating (1)… satisfying (9)	+3.1* +3.5**[#]	*+2.3**	*+1.2** *+1.4***[#]	*+0.4*	*-0.8*	*-1.6* *-2.2**[#]

Table 8: Differences of the user ratings from Table 6. Positive values indicate that the second of the two respective interfaces received better ratings; negative values indicate that the first interface received better ratings. Asterisks indicate significance: * significant at 0.05, [#] after removal of one outlier. Applied tests: 1. White cells: variables were normally distributed (Shapiro-Wilk test). A t-test for independent samples was used if interfaces were tested by different subjects, e.g. 1H vs. 2H. A t-test for paired samples was used if the two interfaces were evaluated by the same subject, e.g. 1F vs. 1H. 2. Light Gray cells = variables were not normally distributed. Variables were compared using a Mann-Whitney-U-Test if interfaces were tested by different subjects, e.g. 1H vs. 2H, otherwise using a Wilcoxon test, e.g. 1F vs. 1H.

[22] Because of the sequential design of the experiment (1F followed by 1H or 2F followed by 2H), the evaluation of form-based and histogram-based interfaces differed in that users already had pre-experience with the form-based interfaces when starting to use the histogram-based interfaces (see Section 4.3.2).

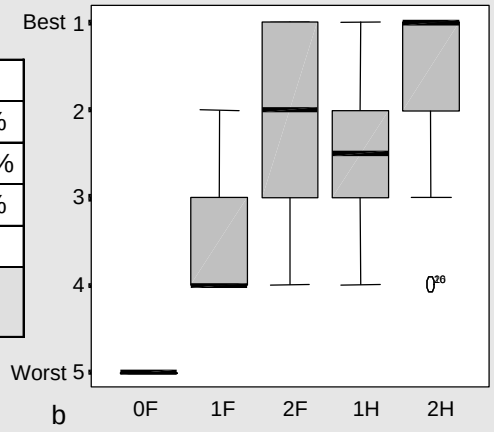
[23] Subjects testing the 1H and 2H interfaces differed in they pre-experience, namely with either the 1F of the 2F interface.

Hypothesis 5 "The 1F interface style is more efficient, but the 2F interface style provides higher precision". Although the 1F interface requires operating only half as many pull down menus as the 2F interface, the differences in the usage efficiency were not significant (Table 8, column 5, rows 3 and 4). Apparently, this result was not determined by the manual effort, but by the cognitive effort for perceiving the different menu options and for making a decision. The 2F interface received slightly higher means in most utility-related ratings (Figure 39), but the difference was only significant for the exhaustiveness of entering the given profile (Table 8, row 5, column 5). This plus in exhaustiveness may have been the reason why most subjects (93%, Figure 41a) preferred the 2F interface in the final comparison.

Hypothesis 6 "Subjects prefer histogram-based editors to the form-based styles." Subjects rated their respective form-based interface and their histogram-based interface according to the two dimensions horrible/wonderful and frustrating/satisfying (Table 6, last 2 rows). In this comparison, the 1H interface received significantly better ratings than the 1F interface (Table 7, last 2 rows) in both dimensions. The differences between 2H and 2F interface, however, were not significant. Possible explanations are that the reduced learnability caused by the deformation option and the 2D drag interaction had lowered the subjects' satisfaction with this interface style.

0F better	–	0/0%	0/0%	0/0%	0/0%
1F better	30/100%	–	2/7%	7/23%	4/13%
2F better	30/100%	28/93%	–	17/57%	11/37%
1H better	30/100%	23/77%	13/43%	–	4/13%
2H better	30/100%	26/87%	19/63%	26/87%	–
	than 0F	than 1F	than 2F	than 1H	than 2H

a



b

Figure 41: Preference ranking of all five interfaces by all subjects in part 3 of the experiment. Number/percentage of subjects who preferred the interface named at the left to the interface named at the bottom (a). Ranks assigned to each of the five interfaces: 1 = preferred interface, 5 = least preferred interface (b). All pairwise differences but the 1H/2F pair are significant at 0.01 (Wilcoxon Signed-Rank test/Mann-Whitney-U-Test)

These results changed in the final comparison, where subjects were presented with all five interface styles. Before filling in the final questionnaire, subjects had additional opportunity to experiment with the interfaces, to explore their functionality, and to understand their design principles. When subjects were asked to rank the five interfaces at the end of the experiment, the results differed from what would have been expected from the earlier ratings (Figure 41b). This time, 63% of the subject preferred the 2H interface to the 2F interface (Figure 41a). There was no systematic relation between this preference and the interface that subjects had

used during the experiment. This time, most subjects (87%, Figure 41a) preferred the 2H interface even to the 1H interface. Apparently, the additional exploration phase had helped subjects to better understand the 2H interface and to learn to handle the 2D drag interaction.

Correlating subjects' pre-experience with their preference for form-based or histogram-based interfaces showed that the more experienced subjects were, the more likely they were to prefer the histogram-based styles (see Table 9). Computer pre-experience in general (which was highly correlated with intense usage of the Internet) showed a strong positive correlation with the preference of the histogram-based styles (gamma 0.615, Table 9, first two rows). Further positive correlations were found with experience with diagrams and a curious attitude towards new interfaces (Table 9, row 5 and 7). No significant correlation was found with experience with histograms.

	Preference for histogram over form-based
Computer usage: never (1), …, often (9)	.615*
Experience with the Internet: beginner (1), expert (9)	.634*
Experience with mouse/trackball/joystick: beginner (1), expert (9)	.542
Experience with pull-down menus: beginner (1), …, expert (9)	.056
Experience with diagrams (e.g. Excel): beginner (1), …, expert (9)	.471*
Experience with histograms: beginner (1),…, expert (9)	.386
Attitude towards new interfaces: uninterested (1), …, curious (9)	.554*

Table 9: Relation between pre-experience and preference of the histogram-based styles. Cross-tabulation symmetric measures. Gamma values range from -1 (perfect negative correlation) to 1 (perfect positive correlation). * significant at 0.05.

There are different possible explanations for the subjects' preference for the histogram-based styles. On the one hand, the preference may be explained by the fact that histogram-based editors were judged more flexible and more precise, and that their functionality was judged more appropriate (Table 8, column 3, see also Figure 39). On the other hand, subjects found the histogram-based styles not only more informative, but they also preferred the graphical representation and had more fun using their direct-manipulative interfaces (see Table 10).

	1F vs. 1H	**2F vs. 2H**
Which is more informative: form(1)… histogram(9)	6.6 (1.71)	6.5 (2.01)
Preferred graphical representation: form(1)… histogram(9)	7.8 (1.62)	7.4 (2.59)
Which one was more fun to operate: form(1)… histogram(9)	8.3 (0.82)	7.3 (2.16)

Table 10: Direct comparison form-based and histogram-based styles by the subjects. Table cells contain mean (standard deviation). Subjects provided these ratings at the end of part 2 of the experiment.

4.3.5 Discussion

The presented experiment has implications for QSA system design. While the form-based interfaces 1F and 2F were judged sufficiently useful and usable, the biggest strength of these interfaces is their learnability. The fact that first-time user subjects did not require any training before using these interfaces makes these user interfaces the interfaces of choice for new and inexperienced users of QSA systems. Since most subjects preferred the 2F interfaces to the 1F interface in the direct comparison, the 2F interface may be the better choice for building systems.

The 1H interface proved to fulfill the basic requirements. In the experimental situation, where subjects had already gathered experience with a form-based interface, the 1H interface showed to be sufficiently easy to learn, easy to operate, efficient, and useful. Subjects liked this interface style not only because of the provided functionality, but also because it was fun to use. In the preference ranking, however, the 1H interface failed to surpass the 2F interface, so it is doubtful whether users of a system providing both interfaces would switch from an introductory 2F interface to a 1H interface that is offered as an alternative interface for experienced users.

The interface that has the potential to surpass the 2F interface is the 2H interface, as the preference ranking showed, where the majority of all subjects preferred the 2H interface over all other interfaces (Figure 41a). The problem with the 2H interface is that although it produced satisfactory to good results with sufficiently computer-experienced users, it is likely to overwhelm inexperienced users.

The experiment showed that what makes the histogram-based interfaces difficult to understand is (1) the surface property and (2) the 2D drag interaction. We will briefly discuss them in the following.

(1) The fact that many users did not understand the surface property (i.e. that surfaces represent amounts of objects and histogram heights only change to conserve the histogram surface) may indicate that the user interface caused users to associate an inappropriate metaphor. A possible explanation is that those subjects who recognized some sort of diagram in the interfaces associated them with bar charts, such as election results, where *height* is the property of primary importance—not surface. Even if subjects had experiences with histograms, not many of them will have ever compared two histograms by their surface. A metaphor is only useful if subjects are familiar with the associated real-world object—it is at least doubtful whether this association worked in the case of histograms.

To overcome the problem, histograms should better be associated with something that is characterized by the same three properties that characterize the histograms in this interface style. These characteristics are deformability, surface-conservation, and largely shape-conservation. A real-world object that has these properties is *jelly*. Jelly can be deformed, but it cannot be compressed. When deformed, jelly conserves shape details as far as possible. For this reason, we suggest the jelly metaphor to the basis for the next generation of histogram-based interfaces. This interface style was already to preview at the example of the stacked histogram in Figure 37. To also avoid the term histogram, histograms may be described as heaps of objects (e.g. TV programs) that are each represented as a “spoonful” of jelly.

(2) The subjects' difficulties with the 2D drag interaction may have been caused by this interaction technique's way of aggregating its two parameters. The applied 2D drag interaction deforms histograms by expanding or contracting the histogram simultaneously to the left *and* to the right, leaving only the center stationary. In the experiment, subjects had to enter two parameters per histogram, i.e. the rating for the top-ranked objects and the amount of objects. The first parameter could always be entered by dragging the histogram horizontally, e.g. to align the left edge of the histogram with the respective rating. When entering the second parameter, e.g. the position of the histogram relative to the cut-off line, the histogram had to be deformed to not affect the already entered first parameter (see Section 4.2.4). The 2D drag interaction was not appropriate for this task, because the symmetric expansion/contraction affected also the first parameter, so that subjects had to iterate. For this type of adjustment task, the interaction model described in Section 4.2.6 that combines histogram-attached handles with the option to drag the histogram horizontally may provide better results (see also the handles at the jelly interface in Figure 37). An interface using this interaction technique behaves like the 1H interface evaluated in the experiment, unless the additional handles are used, which may also allow users to learn this interaction technique in a stepwise fashion. For experienced users who do not want to miss the efficiency of the 2D drag interaction, a version that leaves the left edge constant and expands only to the right may allow for better results than the version used in the experiment.

Several aspects of profile editors may be the subject of future experiments. The experimental design used in this experiment, i.e. using a sequential combination of form- and histogram-based interfaces, limited the possibilities of comparing form- and histogram-based styles and also of comparing 1H and 2H interfaces. Future experiments may evaluate all interfaces using individual groups. Furthermore, this experiment focused on amount information; future experiments may provide rating scales as described in Section 4.2.4 to allow subjects to take both ranking *and* rating information into account. Other focuses of future experiments may be the alternative designs of histogram-based interfaces discussed above and also the stacked histograms described in Section 4.2.7. Finally, the combination of user interfaces and retrieval quality is an interesting issue. In the presented experiment, interfaces were only compared with respect to their interface qualities, but not with respect to which retrieval quality the resulting profiles enabled. Future experiments may take obtainable retrieval quality into account.

4.4 EFFICIENT PAINTING-BASED EDITORS

In the previous sections, we presented and discussed two families of profile editors designed for maximum learnability and accuracy, respectively. In this section, we will present a user interface family designed for simultaneously updating multiple query weighting functions with maximum efficiency. As we will discuss in Section 4.5, these interfaces perform best in the situation of repetitive interest changes.

The interaction concepts presented in this section is ongoing work. We will present a selection of the interfaces we built so far, but we will also describe possible extensions and point out

future directions of research. First results of our research effort related to these interaction techniques can be found in [Bau98d, Bau99a].

4.4.1 Multiple select and painting

The interfaces presented so far require a user effort that is linear with the number of queries to be manipulated. The only possibility to improve efficiency beyond O(*n*) is to manipulate multiple interests simultaneously. One way of accomplishing that is by providing a number of defaults that set the entire profile or parts of the profile to a specific, previously stored state (see Section 4.4.5). The other way is to provide users with an interaction technique that allows them to directly manipulate multiple queries at a time.

An interaction technique for manipulating multiple objects that is common in other application areas is *multiple select*. In this approach, selectable objects, e.g. cells in spreadsheet programs or icons in desktop GUIs are distributed over a two-dimensional surface. Users interactively select a subset of these objects, e.g. using mouse drag operations or using multiple mouse clicks while keeping a qualifier key depressed. Then, users select a method to be applied, e.g. “clear selected cells” or “delete selected files”.

Multiple select can often reduce the effort for manipulating objects significantly. First, the user effort for invoking the manipulation method is always reduced to one, independent of the number of selected objects. Second, the user effort for making the selection is usually reduced as well. How big that gain is depends on the arrangement of the objects on the surface (see Section 4.4.3) and on the selection tool (see Section 4.4.2). The more objects can be selected with a single user interaction and the shorter the path the selection tool has to go, the faster the selection task can be carried out. If, for example, a rectangular rubber band selection tool is used and if the objects to be selected are arranges in the form of a square matrix, a drag interaction of length O(√*n*) (the diameter of the square) is sufficient for selecting all objects, which is substantially shorter then any path traversing the objects individually. Furthermore, the usage of a multiple select tool, such a rubber band, results in a continuous select operation that is not intertwined with method invocations for individual objects. The interaction is therefore generally more fluid and thus faster.

To apply multiple select to queries, we have to arrange queries in a two-dimensional plane, provide one or more selection tools, and one or more updating methods. Figure 42 shows such an interface that offers a rectangular rubber band selection tool and two updating methods that allow setting the output ratings of the top-ranked object of the selected queries to zero and to some maximum value.

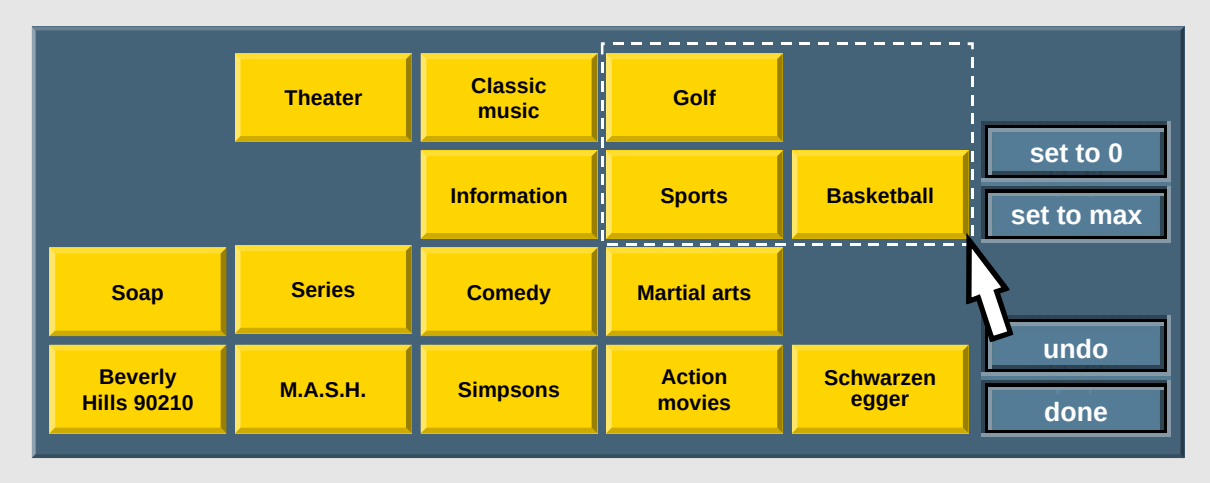


Figure 42: Applying multiple select to QSA user profiles

This selection mechanism is restricted to Boolean selections, i.e. queries are either selected or not. To produce gradual effects, more updating methods have to be provided, e.g. methods that allow the output ratings of the top-ranked object to be set to 25%, 50%, and 75%. Selecting methods, however, interrupts the selection interaction, which diminishes the benefit from multiple select. To generalize the basic idea of simultaneously manipulating queries, a more flexible mechanism is required. The goal is to allow entering interest changes with a single mouse drag interaction, i.e. without switching tools.

An interaction concept matching these requirements can be found in image processing; it is the *painting* interaction. To apply painting to QSA profiles, we consider each query in the user profile as a pixel of an image[24]. While a painting interaction in a paint program assigns colors to the pixels of an image, painting applied to a QSA profile assigns user-aggregated relevance feedback to queries. Multiple select and painting have the following properties.

- *Multiple select* uses the so-called *noun-verb* order (e.g. [SIK+82]), i.e., items are selected first, and then the method is selected. The main strength of the noun-verb order is that it allows restricting the list of available methods to those methods that are applicable to the items within the current selection. The method “empty trashcan”, for example, is only applicable to non-empty trashcans. This, however, is of no use if there is only a single object type, as is the case in paint programs (pixels) and QSA profiles (queries). The disadvantage of the noun-verb order is that during the selection process, the method that the user plans to apply to the selection is not yet known, and consequently no preview of the expected result can be given. Highlighting the selection does not have the quality of a realistic preview.
- Being based on the *verb-noun* order, *painting* is the dual approach to multiple select. Using painting, a method is chosen first, e.g. a colored airbrush, and then it is applied continually to all subsequently selected pixels. The immediate effect and visual feedback of

[24] This is different from the analogy introduced in Chapter 3 where pixels corresponded to objects. In the approach described in the current section, pixels correspond to entire queries.

painting allows users to immediately perceive the effect of their current action. This allows the effective use of a wide range of different painting tools, as underlined by the wide range of painting tools provided by today's paint programs.

The interface directly corresponding to the multiple select example from Figure 42 is one that provides users with a painting tool and a color palette to pick color from. Users would now first select a color (= a URF value) and then continuously apply it to queries. This, however, would not bring any benefit compared to the multiple select interface, because the process of picking colors would still interrupt the painting process. To finally obtain an interaction technique that allows users to continuously work on the rating task, the right painting tools are required. As we will discuss in the following sections, painting provides several types of *incremental* painting techniques (e.g. airbrushes) that allow obtaining gradual effects while almost entirely avoiding the necessity to interrupt the user's task for switching colors. All these tools provide immediate visual feedback and therefore allow users to monitor the gradual effects they produce. Since there is no natural way of visualizing "degree of selectedness", there is no corresponding way of achieving the same effect with multiple select[25].

In the following, we will refer to interfaces supporting painting interactions as *painting-based interfaces* or *paintable interfaces*. In the following, we will examine the individual design possibilities. Interface designers have to decide (1) how to visualize queries, (2) which interaction methods to provide, and (3) how to layout queries in two-dimensional space.

4.4.2 Painting tools

Painting interactions update all pixels that are swept by the painting tool during the painting interaction. Figure 43 gives an overview over possible functionalities of painting and drawing tools, as provided by a commercial image processing package [Adobe96]. Painting tools are characterized by the following properties.

1. *Effect of multiple passes*. Brushes affect each pixel only once per paint interaction. To affect the same pixel again, they require stopping and restarting the paint interaction, e.g. by releasing and repressing the mouse button. Airbrushes continuously affect pixels. The longer an active airbrush is held over a pixel, the more intense the effect becomes. The slower an airbrush is moved, the more intense its trace becomes.
2. *Color*
3. *Tool tips*, i.e. brush tips, airbrush tips, etc. depending on which tool is used allow determining the width and the shape of the trace left during painting. They come in different shapes and with different edge softness.
4. *Opacity* determines the intensity with which painted-over pixels are affected. Opacity is important for creating gradual effects (see below).

[25] Some existing painting programs support gradual selections. If these are to be manipulated interactively, however, the degree of selection is mapped to color and *painting* tools are used to modify the selection, so that the problem of manipulating selections is solved by mapping it to a painting problem (e.g. Photoshop mask mode [Adobe96]).

5. *Painting modes* allow defining how pixel color and brush color are combined to define the new pixel color. Painting programs provide a large variety of choice, such as “multiply mode”, a mode that always darkens painted pixels, or “screen” (negative multiply), which always brightens them.
6. If an appropriate input device is available (e.g. a pressure sensitive stylus), tool tip size (2) or opacity (3) may be varied during painting.

See [Bau98d] for a discussion of especially efficient painting tools for paintable interfaces on Boolean values.

Alternatively to painting (i.e. pixel-oriented interaction), *drawing* (i.e. graphical object-oriented) techniques may be used as the basic interaction technique of paintable interfaces. Drawing allows affecting pixels by covering them with a geometric shape. For paintable interfaces, typically only filled shapes are of interest. Unlike painting, the set of affected pixels is not determined by the entire path of the input device, but only by selected positions. A rectangle, as for example used by the multiple select interface shown in Figure 42, is defined by starting and end point. Shapes may be filled using various fill effects, e.g. a circular gradient that assigns a specific value to the center of a shape while fading to transparent towards the periphery. Similar to painting tools, drawing tools may use opacity and different modes defining the combination of covered pixel and drawn shape. Drawing tools allow efficiently covering large surfaces if complemented by an appropriate layout (see Section 4.4.3 and the TV channel dialog shown in Figure 47a).

Drawing is different from multiple select, despite the fact that the same tools used for multiple select may also be used for drawing. Drawing differs from multiple select mainly in that it results in an object-oriented representation of what has been drawn (the *scene graph*), so that this representation can be manipulated at an object-oriented level, e.g. by manually resizing a drawn shape using handles attached to that shape. Paintable interfaces may combine drawing and painting tools in the same interface, e.g. by considering painted surfaces as objects. For a discussion on the integration of painting and drawing see also Section 3.1.

The physical user effort for using paintable interfaces is mainly determined by the sum of the efforts for moving the mouse to the individual start positions, the lengths of the painting interactions, and the effort for switching tools. Painting tools should be selected such that they minimize these three efforts. Painting tool selection is therefore usually the result of an optimization process.

Paintable interfaces may allow users to manipulate multiple URF samples per query, e.g. the two URF samples *rating of top-ranked objects* and *rating of bottom-ranked objects/amount of objects* (see, for example, the form-based interface shown in Figure 20). Pixels also have multiple properties (e.g. hue, saturation, value, and transparency [Fol90]) and painting programs typically allow addressing them using different painting modes or tools. Similarly, paintable interfaces may use different painting modes or tools to address individual URF samples or combinations of URF samples, e.g. as the faucet interaction does (see Section 4.1).

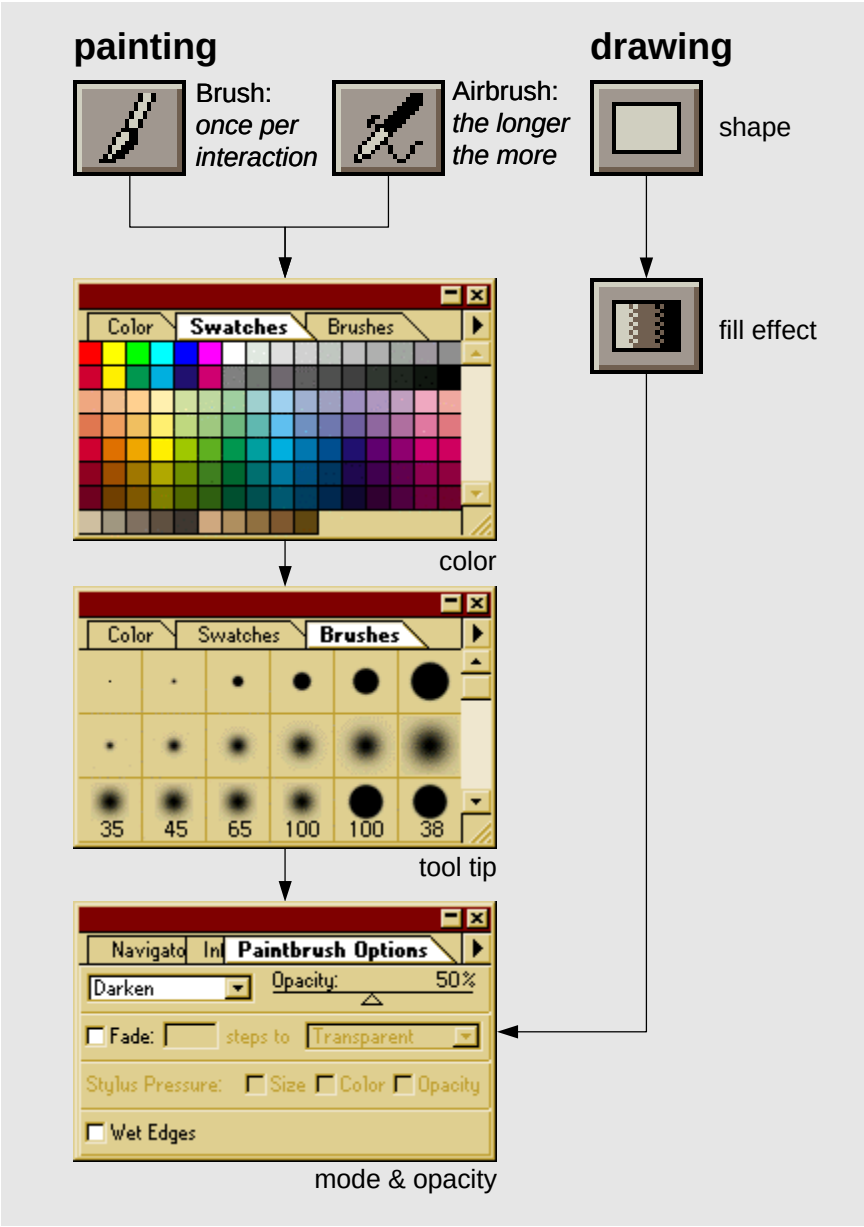


Figure 43: Parameters of painting and drawing tools useful for paintable interfaces. All flying window screenshots and all icons except the rectangle taken from Adobe Photoshop [Adobe96].

For each degree of freedom per query, i.e. for each URF sample to be manipulated independently, at least two painting tools configurations are required, i.e. one that increments its value (painting white) and one that decrements it (painting black). The value of opacity determines precision and speed of painting; higher values allow faster painting, but limit the obtainable precision of URF samples. There are various ways of offering these two painting tools, e.g. via a tool chest, by mapping them to left and right mouse button (See Section 4.4.5), by mapping them to a mouse button and a qualifier key, by selecting them based on the color of the pixel at which painting starts, etc. Additional tools may be offered for obtaining better paint-

ing efficiency, e.g. tools with different tool tip sizes or different opacities. However, interface designers have to weigh whether the benefit from additional painting tools justifies the additional learning and handling effort. Unless the number of objects to be manipulated with a paintable interface is very large, providing a larger number of tools and options generally will not be justified. To avoid option palettes, such as the ones shown in Figure 43 and the usage interruptions they cause, a selection of pre-initialized tool configurations, i.e. tools with fully defined settings will often be preferable.

4.4.3 Layout

The goal of layout is to maximize usage speed by optimizing two factors.

First, layout should support users in locating individual interface objects, here queries, easily and rapidly. To minimize recognition time, layout should group objects of subjective similarity, so that recognizing one object already gives a clue about what the neighboring objects might be. Lin [Lin93] showed that layouts generated by neural network clustering could be used equally effective by users locating documents as human-produced maps.

Second, the scope of paint operations should be maximized, so that only a minimum number of paint operations is required to make the desired adjustments. Therefore, layout should spatially group objects that are frequently manipulated together. Common techniques to accomplish that are multidimensional scaling [HB97, KB97] and disjoint cluster analysis, e.g. using bottom-up clustering [LN93]. The requirements for paintable layout are very similar to the layout requirements of spatial menus (e.g. [Nor91, p. 261-280]), insofar as in both cases multiple objects that are laid out in 2D space are to be selected. Consequently, many concepts can be transferred. In spatial menus, layout based on the *frequency of co-occurrence* generally offers good results [MDM86, MMN88].

With respect to the layout of paintable interfaces, this means that those objects should be laid out adjacently that are *manipulated* the same way (we will call this *layout based on frequency of co-manipulation*). Since this includes that the same painting tools be used, painting tools have to be taken into account when computing layout. If painting tools increment and decrement the ratings of top-ranked objects, for example, then queries whose ratings are often incremented or decremented by the same amount should be laid out adjacently. The data required for building such layouts may be extracted from logs of the user's past sessions. An initialization may be based on usage data from other users having similar user profiles or on content-based similarity between queries, such as the overlap in objects retrieved by two queries.

Layout for paintable interfaces depends on shape and size of the tool tip. Unlike the layout of spatial menus, paintable interfaces require similar objects not only to be in relative proximity (to reduce hand movements as analyzed by [GHA+90], see also [Fit54]) but also to be directly adjacent and optimized for the respective painting tool. If, for example, a rectangle tool is used, a modified tree map layout [TJ92, Shn92] can support efficient usage (see Section 4.4.5).

If the same layout is used for a longer period of time, then the perception factor becomes increasingly less important, because users may find the objects based on past experience. The

more often the same interest changes occur, the more a user may profit from past experience to save *mental* effort (see [Bau98d]); users may have learned to associate a desired profile state, such as a specific mood, with the painting *gesture* that produces the desired settings. To make this possible, the layout should be kept relatively constant over time, even when queries are added or removed. For related work on gesture-like usage of marking menus/pie menus see for example [Nor91].

4.4.4 Visualization

QSA user profiles may be considered to contain graph structures. The nodes of these graphs are the queries; the node properties to be updated are the queries' weighting functions. The edges between these nodes are the relations between queries, e.g. the interest interaction. Paintable interfaces may therefore be considered tools for annotating graphs.

User interface designers creating paintable interfaces can pick which node information to visualize (e.g. the URF samples *rating of top-ranked objects*, *rating of bottom-ranked objects*, *percentage* or *amount of selected objects*, *rating distribution*) and how to visualize it (e.g. color, gauge, color gradient, button surface, histogram, etc.). The same freedom applies to edges, where interface designers may choose which edge properties to visualize (e.g. *frequency of co-manipulation*, *overlap in matching objects*, *frequency of co-occurrence in other users' profiles*, *inter-interest relation*) and how to visualize them (e.g. proximity, (annotated) connecting lines, color). For the reasons described in Section 4.4.3, proximity will usually be used to represent frequency of co-manipulation.

A lot of research has been done on the 2D visualization of graphs (for a survey see [HMM00]), that we will not repeat here. Figure 44 shows a small selection of design possibilities.

a. This layout uses proximity to visualize relatedness. One way of generating such layouts is by mining usage data (e.g. "how many users simultaneously retain queries X and Y in their profiles") and converting this data into two-dimensional layouts, e.g. using multidimensional scaling. The brightness of nodes represents the rating currently assigned to the top-ranked objects of that query and can be updated using painting. As an additional feature, the shown layout also offers access to selected sets of objects matching two queries. For this purpose, thick lines connecting nodes are used to represent objects matching both queries. An edge between martial arts and action movies, for example, represents action movies involving martial arts. The meaning of edge color corresponds to the meaning of node color; the especially bright edge between martial arts and action movies, for example, expresses the users preference for this combination of movies. Edge color can also be manipulated using the same painting interactions that are used to update node color.
b. This interface uses a regular layout of button-like nodes. Again node brightness is used to represent the rating currently assigned to the top-ranked objects of that query. The button design is designed to encourage users to click nodes, which is one way of discovering painting operations. Furthermore, the button-like design provides a strong visual feedback while making efficient use of the available screen space. This is especially useful when using these interfaces on devices with small screens, such as handheld devices. The regular

layout is especially efficient in combination with a rectangle drawing tool (see also Section 4.4.6).

c. This design extends the one from Figure 44b in that is provides users with additional information about ratings and amounts. First, it uses variable button sizes to represent the amount of objects matched by the respective query. The query information, for example, obviously delivers more objects than the query theater. Button sizes are scaled non-linearly if necessary to ensure label readability. Second, the buttons of queries that are partially below cut-off value bear gauge-like displays informing the user about how many of this query's objects will be dropped. The query series, for example, is cropped and only the top 70% will be delivered to the user. Third, color gradients on the buttons are used to represent the output ratings of top-ranked and bottom-ranked objects. Top-ranked action movies, for example, are assigned very high ratings, while bottom-ranked action movies received only mediocre ratings. The nested rectangular layout of this interface (modified tree map layout) will be described in Section 4.4.5.

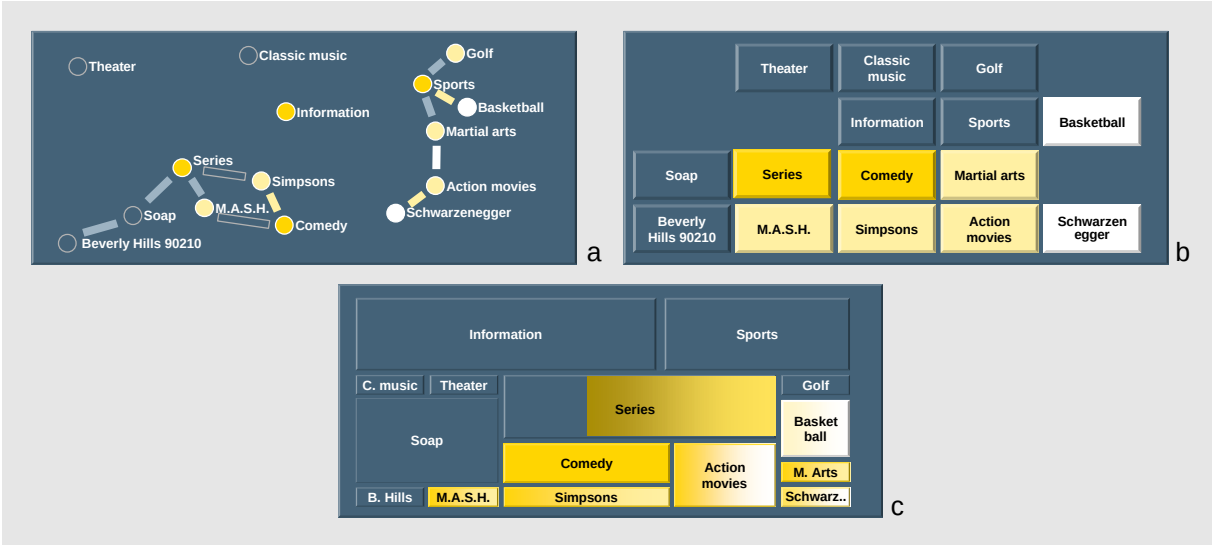


Figure 44: A small selection of possible visualizations for paintable interfaces. A graph-like layout (a), a regular layout a button-like representation of queries (b), and an interface style visualizing amount and rating information (c).

4.4.5 Handling repetitive interest changes with paintable interfaces

Compared to the other two user interface classes presented in this chapter, paintable interfaces have the potential to reach higher physical efficiency, because users can manipulate multiple queries with a single mouse drag interaction. Therefore, paintable profile editors are especially interesting for handling interest changes where many queries are affected simultaneously. The user interface shown in Figure 45 is designed for that purpose.

This interface uses the visualization presented in Figure 44c that is designed for the use by experienced users; interfaces for less experienced users may omit part of the visualization

features. The interface provides users with an airbrush tool. Painting with the left mouse button increases the rating of top-ranked objects while simultaneously increasing the rating of bottom-ranked objects/the amount of selected objects (faucet interaction, see Section 4.1); the right mouse button decreases them. Since all painting functionality is mapped to mouse buttons, color and tool palettes have been avoided. To support both efficiency and precision, the airbrush uses a dynamic tool tip. If the airbrush is moved rapidly (a typical type of interaction for making a first "sketch"), the airbrush uses a larger tool tip and higher opacity, thus emitting more color on a wider area. The slower the mouse movement becomes, the smaller opacity and tool tip become, allowing users to fine-tune their profiles.

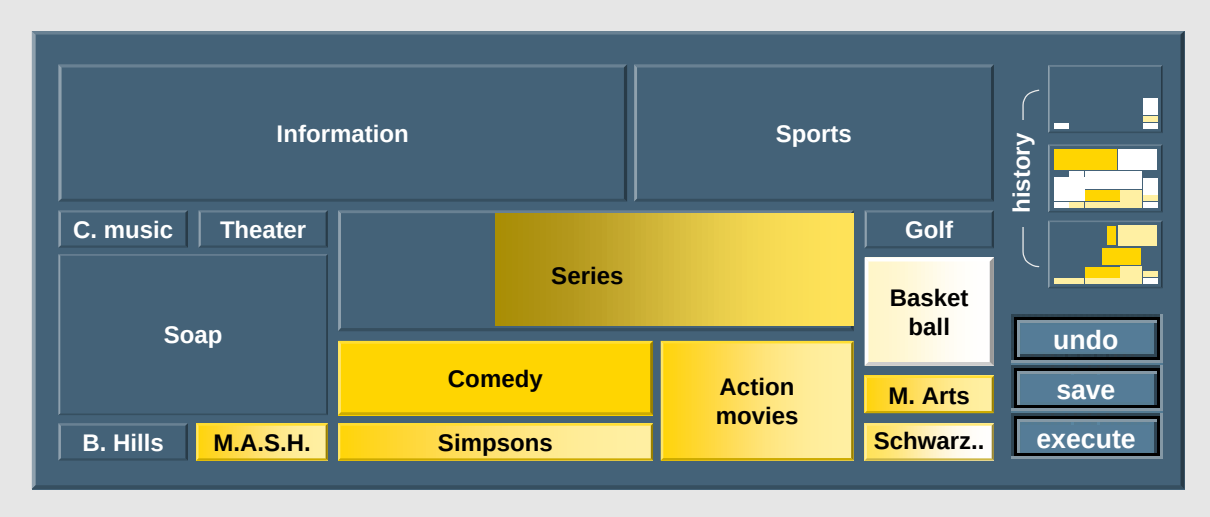


Figure 45: A paintable interface providing special support for repetitive interest changes

To allow users to memorize the painting "*gestures*" leading to the desired adjustments, the layout is kept as constant as possible over time, even when queries are added or removed. This is accomplished by using a tree map layout. This layout is computed in two steps. 1. The query set is converted into a tree by successively clustering queries according to frequencies of co-manipulation. 2. The resulting tree structure is converted into a 2D layout using a modified tree map algorithm. The parent node is initialized to a rectangle filling the entire space. The map is then recursively decomposed by replacing each parent rectangle alternatingly by a horizontally or vertically sequence of child rectangles [TJ92, Shn92, Lin93][26].

While insertions into tiled layouts (e.g. Figure 44b) make significant layout rearrangements necessary, insertions into tree map layouts keep local neighborhoods practically untouched as shown in Figure 46. When a new query is inserted, it pushes other buttons away to obtain the required space. All buttons in the interface are compressed equally so that amount ratios remain correct. Because of the similarity between layout before and after the insertion, positions

[26] Another benefit of tree map layouts not exploited by this interface is that any tree node (not only leaves) and any set of adjacent nodes form a rectangle in the resulting layout. This allows for especially efficient manipulation if a rectangle drawing tool is provided.

of button and button groups can easily be identified in the resulting map. The recognition of the modified layout may be improved further by using a smooth animation showing how a new interest is inserted with a surface of zero and successively grows until reaching its actual size.

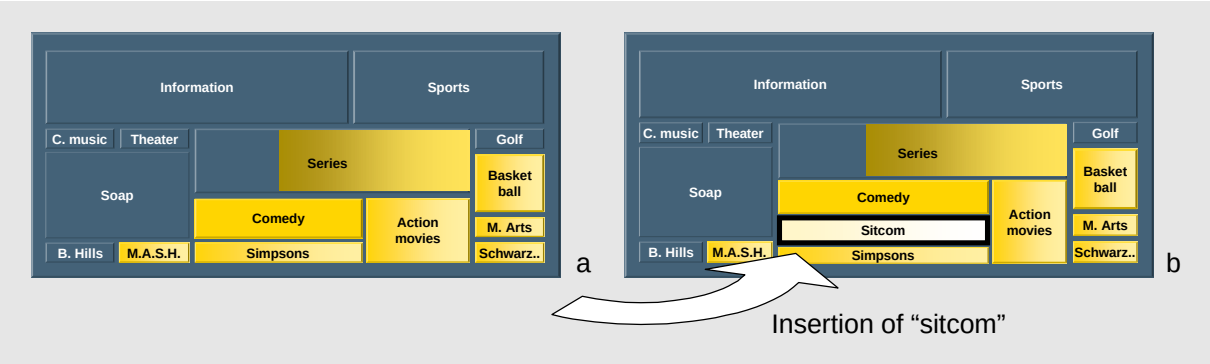


Figure 46: Insertion of a new interest into a tree map layout

The history module (Figure 45, top right corner) provides even faster access in the case that complete profile states occur repeatedly. It allows recalling entire settings from past sessions. The graphical nature of paintable interfaces and especially the button design supports the recognition of profile states even in the shown thumbnail size. Whenever a session is completed, the current setting is stored in the history. The history entry to be replaced is selected based on the criteria low recency of use, low frequency of use, and similarity with other profile states in the history.

The shown interface can be used to define a temporary offset for a one-shot execution or to make permanent changes to the user’s profile. The distinction between temporary and permanent change is made by providing distinct “save” and “execute” button. Unless the save button is pressed, all changes are of temporary nature, so that the previous settings will be restored after executing the profile.

4.4.6 Other application areas

Because of their universal nature, paintable interfaces can be applied to a wide range of applications that require users to enter larger numbers of parameters of the same data type. In this section, we will briefly present some of the paintable interfaces that we built for different applications. See [Bau98d, Bau99a] for more details on these interfaces and their application.

Paintable interfaces allow rating large numbers of objects. The paintable interfaces shown in Figure 47 allow users to specify which TV channels they are able to receive. In Germany, the set of available TV channels depends not only on the support, i.e. channel vs. satellite vs. antenna, but also varies widely across different regions. For this reason, the selection task cannot be handled effectively by aggregating channels or by providing defaults. If the interface provided extra buttons for every useful default configuration, the number of these buttons could easily exceed the number of TV channels in the interface. Using a paintable interface

solves the problem. Because of the regular layout of Figure 47a (a mixture between layout by subjective similarity and correlation data extracted from existing user profiles), a rectangle drawing tool proved to be highly efficient for this interface (see also Section 4.4.7). This interface is part of the TV Scout system (see Chapter 5). The layout of the interface shown in Figure 47b is grouped according to their geographical locations (semantic space layout, e.g. [Nor91, p. 269]). The black line shows one possible path to activate the highlighted switches with a single mouse interaction using a pen tool.

Both interfaces use a special interaction technique optimized for Boolean settings. Instead of using two mouse buttons for determining whether to set or reset buttons, we used a technique found in the early black-and-white paint program MacPaint [Ake95]. During painting, the first button is toggled; subsequent buttons are set to the new state of the first button. In this mode, a rectangle tool, for example, paints rectangles of set buttons if painting starts on an unset toggle; otherwise it paints rectangles of unset buttons. Although using a single mouse button only, over-painting fragmentized regions (e.g. to set or reset a whole map) never requires more than a click and a mouse drag (unlike for example painting modes that *invert* painted regions). As proven by user tests [Bau98d], users easily understand this mode for two reasons: First, this painting mode is consistent with the behavior of individual toggle switches, because painting a single button is equivalent to clicking it. Second, since at least the first button is inverted, this mode always provides users with visual feedback, which simplifies trial-and-error learning. Another advantage is that all interactions can be triggered with a single mouse button. This allows paintable interfaces using this painting mode to be run on single button mouse systems or on touchscreen-based systems such as palm-top computers.

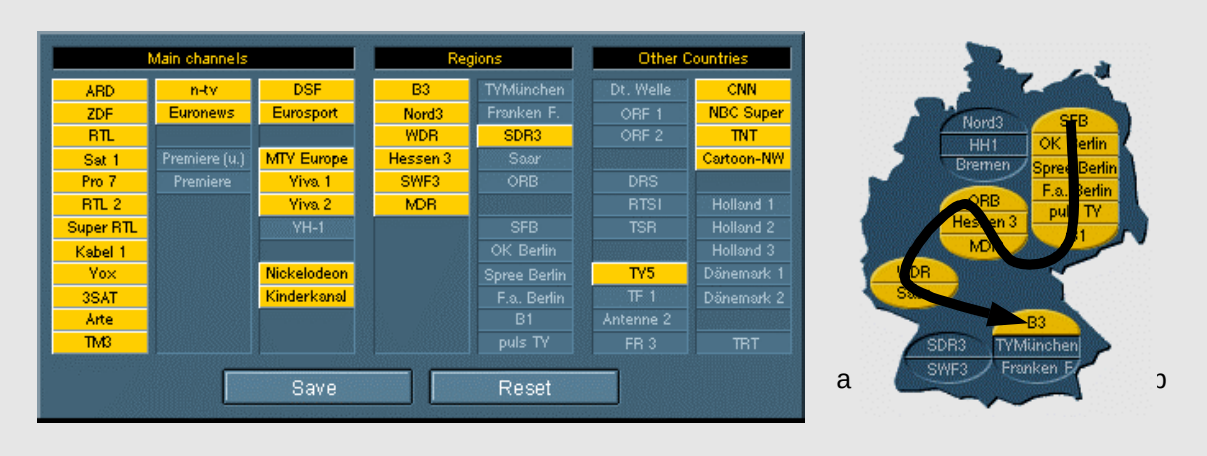


Figure 47: A paintable TV channel selection dialog.

We applied similar interaction techniques to the manual configuration of consumer interest profiles [BL98, LB98, BLF98, BL00]. Figure 48 shows an interface that allows users to manually adjust their consumer interest profiles, as used by advertisers. The relatively large number of product classes makes the compact appearance and the efficient manipulation of a paintable interface preferable over more complex interface strategies. The interface restricted to Boolean input (Figure 48a) uses a rectangle drawing tool, the gradual version (Figure 48b)

an airbrush. Profiles with deeper hierarchies can be handled using tree map layouts (see Section 4.4.5).

Figure 48: Paintable interfaces allowing users the selection (a) or the rating (b) of preferred advertising categories.

Paintable interfaces replace interval sliders. When a continuous variable like time is segmented, it can be represented as an array of Boolean variables. These variables in turn can then be manipulated using a paintable interface. Figure 49 shows a timer interface based on this strategy. Programming the shown state (e.g. controlling house lighting during absence) is possible with only three rectangle drawing operations. At the shown moment, the time intervals for the weekend are enlarged by adding the hours starting at 9 o'clock. When the mouse button is released old and new intervals unite automatically. Although the basic interface components are buttons, on a higher level they behave like interval sliders. Intervals of set switches are labeled as single intervals to reduce cluttering and to underline this high level behavior. Unlike classical interval sliders, paintable interfaces can do without any handles. Enlarging an interval only requires painting the addition. In a similar way, intervals can be shortened or even split. Furthermore, paintable interval sliders are especially easy to read, because a large portion of the screen surface is used for visual feedback. Being part of the TV Scout system, this interface allows users to select preferred viewing times (see Chapter 5). See [SHA97, PSB95, Shn98, Nor88, GBM+89, Hum89] for related work on entering times and dates.

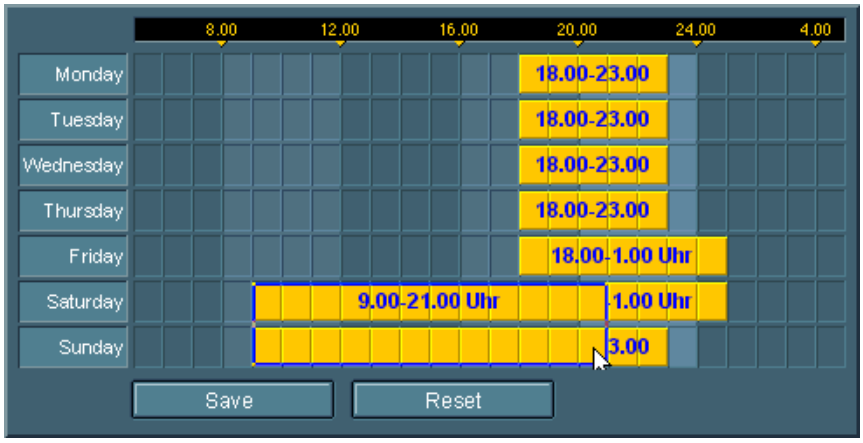


Figure 49: A paintable timer interface. It allows users to input intervals for a whole week.

N-dimensional painting. The timer interface is extremely efficient because of the strong internal structure on hours and days that implies a high subjective similarity, as well as a high frequency of co-manipulation between neighboring buttons. Unlike unstructured, high-dimensional data, such as interests represented in an IF system that have to be mapped to 2D using multidimensional scaling (see 4.4.3), the data objects of the timer interface are highly structured.

When designing paintable interfaces for highly structured data of a dimensionality higher than two, it may be desirable to conserve the internal structure instead of mapping it to 2D. Figure 50b and Figure 50c shows paintable interfaces for such situations. The shown interfaces allow users to express their preferences with respect to computer monitors, e.g. to retrieve corresponding classified ads from a database. Figure 50a shows an interface arranging monitors according to two dimensions. Figure 50b and Figure 50c are generated by successively introducing two more dimensions. To keep the original 2D painting interaction applicable, we used so-called *explosion displays* that solve the problem of occlusion. Layers of buttons are spread; the resulting gaps provide access to buttons that would otherwise be occluded. Using this technique, buttons at any depth can still be accessed directly with a 2D input device, such as a mouse. To provide maximum efficiency, n-dimensional drawing tools are used. Figure 50b sketches the usage of a rectangle drawing tool extended to *n*D. This tool allows drawing filled (hyper-) cubes by dragging the mouse from one vertex to the diametrically opposed one. An overview over n-D visualizations may be found in [Fol90]; a tool for supporting users in creating 3D layouts in presented in [Bau96a].

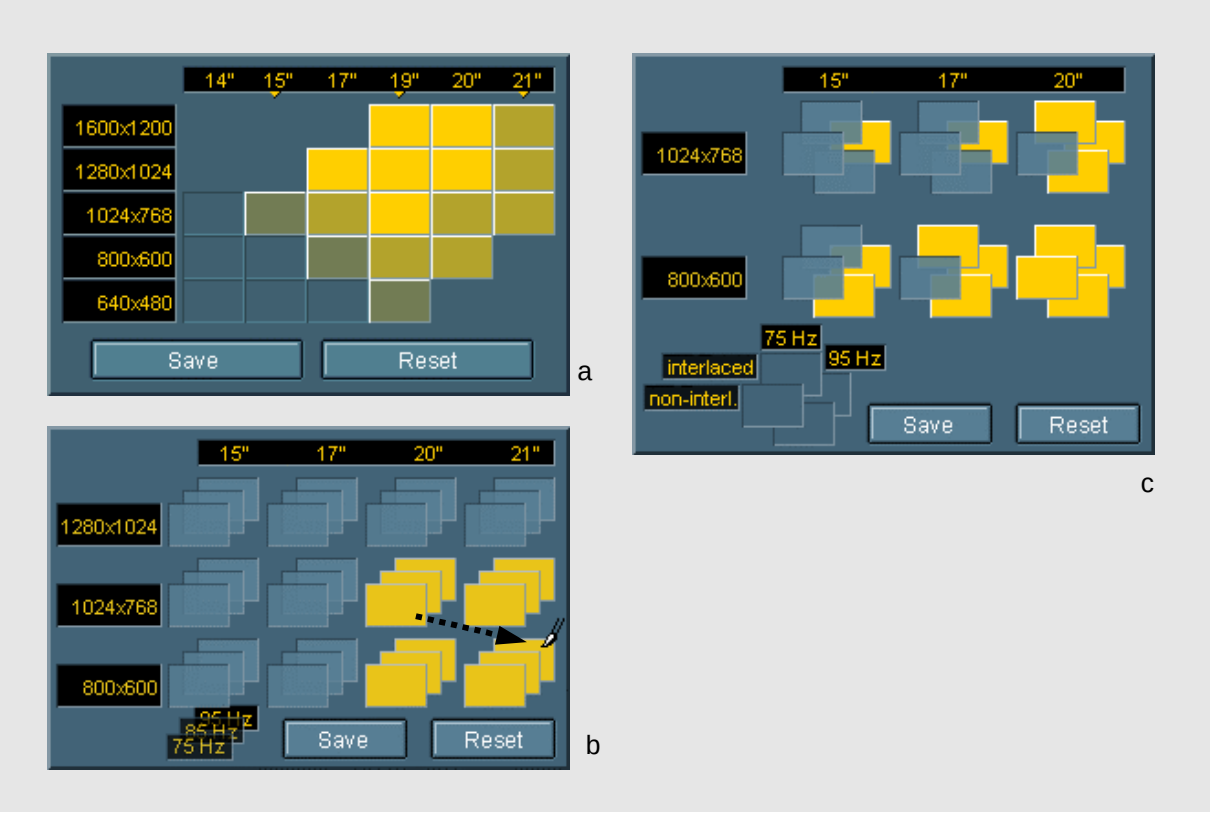


Figure 50: Paintable interfaces allowing users to enter their preferences about computer monitors with respect to two (a), three (b), or four dimensions (c).

4.4.7 Experimental evaluation

A systematic experimental evaluation of the painting-based profile editors for QSA filtering systems should a ripe field for future research that we have not yet tackled. A first impression of the efficiency gain to expect can be found in an earlier experimental study that compared paintable interfaces for entering Boolean settings with interfaces without painting functionality, i.e. interfaces only providing toggle switches [Bau98d, Bau99a]. In this experiment, 64 students who volunteered for the experiment had to enter a given set of TV channels using the user interface shown in Figure 47a. There were four groups of subjects. Users in the groups 1 and 2 used a version of the interface providing a rectangle drawing method, while the interface given to users in groups 3 and 4 supported only toggling individual switches. Groups 1 and 3 had to enter profiles that consisted of buttons grouped in three larger chunks (“good layout” condition), while the groups 2 and 4 had to enter profiles consisting of seven smaller button groups (“poor layout” condition).

The experiment confirmed that users provided with a painting method perform better than users provided with a click-only interface. In the timed selection task the speed-up of painting showed clearly. First-time users required significantly less time to complete the given task (“good layout” condition: 6.4 sec instead of 10.2 sec, significant at $p<0.001$, “poor layout”

condition: 14.3 sec instead of 16 sec, not significant). When repeating the experiment with trained expert users, results showed even clearer. In the “good layout” condition painting users required an average of 1.97 seconds for task completion, which was more than twice as fast as the 4.1 seconds of the click-only group (significant at $p < 0.001$). In the “poor layout“ condition paint users were, at 4.86 seconds, only slightly faster than the click-only group at 5.5 seconds (significant at $p < 0.01$). In all subject groups, layout quality (i.e. “good layout” vs. “poor layout”) had a significant ($p < 0.01$) interaction with task completion time, which emphasizes the importance of layout.

Beside the main hypothesis that paintable interfaces result in efficiency gains, the experiment showed that paintable interfaces are easy to learn and operate. Without having received any training, subjects of all individual groups learned to operate the interface rapidly and were able to operate them correctly. When asked about their preference for any of the four interface styles, 88% of all subjects chose an interface providing a painting method. This ratio was independent of the interface type used during the experiment. The preference for interfaces with a painting method was especially high in the “good layout” groups.

Although this experiment dealt with Boolean settings, not with gradual painting tools as used by painting-based editors for QSA filtering systems, it had implications on the efficiency of paintable interfaces in general. What remains to be proven is in how far these results apply to gradual painting tools, such as airbrushes, when applied in paintable QSA profile editors.

4.5 DISCUSSION

The decomposition of user profiles into queries allows users to manipulate their profiles at the level of the aggregation function by providing user-aggregated relevance feedback. In this chapter, we presented three different user interface strategies that allow users to enter user-aggregated relevance feedback and that are optimized for three different objectives, i.e. learnability (form-based interfaces), accuracy (histogram-based interfaces), and efficiency (paintable interfaces).

Table 11 shows a classification of these three user interface classes according to two dimensions. Table rows distinguish interfaces according to the number of URF samples per query they support, which determines the obtainable profile accuracy. Only the histogram-based interfaces provides the additional ranking and rating information that allows users to control higher degrees of freedom; the other interfaces are restricted to entering a limited number of URF samples per query. Table columns distinguish query-wise and URF sample-wise processing. While form-based and histogram-based interfaces process user profiles query by query, paintable interfaces take the opposite way by processing user profiles URF sample-wise (e.g. using different painting tools to address the individual URF samples). Therefore, painting may be considered the orthogonal approach to the other two approaches.

	Query-wise processing, preferable if few queries Manipulate several properties at once, but only one query at a time	*Property-wise processing, preferable if many queries* Manipulate several queries at once, but only one property at a time
Preferable for few URF samples (simplicity): Adjust rating of top- and bottom ranked object/amount of objects	form-based interfaces	paintable interfaces
Preferable for many URF samples (accuracy): Adjust also mid-ranked samples to obtain perfect merged output ranking	histogram-based interfaces	

Table 11: Classification of profile editing tools

Table 11 has two sweet spots. Query-wise processing (histogram-based interfaces) is more efficient if many URF samples per query are to be updated; URF sample-wise processing (paintable interfaces) is more efficient if one URF sample of many queries is to be updated. If, for example, a new query is inserted into the profile (and no other query requires updating), histogram-based interfaces are more efficient than paintable interfaces. If a profile-wide interest change occurs that can be represented by updating a single URF sample (e.g. using the faucet interaction), paintable interfaces are more efficient. With respect to functionality and efficiency, form-based interfaces are subsumed by the other two interface styles. The justification for form-based interfaces is that they are easier to learn. Consisting of standard pull-down menus only, they are especially appropriate for first-time users and casual users. Interfaces for manipulating both multiple queries and multiple properties (bottom right cell of Table 11) have not been proposed yet, but may turn out to become the subject of future research.

Together, these three interfaces form an interface toolkit that allows system designers to build customized interfaces to QSA applications. Since the three interface styles can substitute for each other within limits, it may sometimes be sufficient to provide only one or two of them. In a complete QSA profile editor interface, i.e. one providing all three interface styles, histogram-based interfaces allow users to initialize newly inserted queries or to reinitialize them after substantial interest changes, and to fine-tune their profiles. Paintable interfaces allow users to efficiently handle repetitive interest variations, such as moods, especially if they affect many queries simultaneously. Form-based interfaces give new users an easy start and can be replaced by the other styles as users gain more experience.

CHAPTER 5 A *QUERYSET ARCHITECTURE*-BASED TV PROGRAM RECOMMENDATION SYSTEM

There is nothing worse than white canvas
[Pablo Picasso about the cold-start problem]

As a proof of concept, we designed and implemented the *TV Scout*, a TV program recommendation system based on the QuerySet Architecture [Bau96, Bau97, Bau98b, Due97, Bru99]. The goal of the TV Scout system is to support users in planning their personal TV consumption. Before we focus on the system and its filtering capabilities, we will briefly motivate why TV recommendation is an appropriate application area for demonstrating our filtering concept. In 1992, Belkin and Croft wrote "In particular, applications such as the recreational use of television programming pose special problems and opportunities for research in filtering" [BC92, p.37]. So what is interesting about the application area TV?

5.1 FILTERING TV PROGRAMS

TV viewers face an information overload situation. During the past twenty years, a number of technical improvements, such as cable, satellite, and digital TV technology have caused a substantial increase of the number of available TV channels. In Germany, cable TV currently distributes 500 programs per day on over thirty channels, satellite TV makes over a hundred European channels available, and digital TV, still in its infancy, continues this trend. Since the amount of content that is of interest for a given viewer has not increased proportionally, *planning* ones TV consumption has become a challenge.

As the number of TV channels increases, two trends in TV program guides try to help viewers to find what they are interested in. First, the number of TV program guides has increased, so TV viewers can choose the one that best supports their interests. Second, TV program guides provide extra columns for selected categories and genres to provide faster access for some interests. Since both these strategies of "customization" are limited by the minimum required number of sold copies and the maximum number of pages readers can handle, it is doubtful for how much longer viewers will find the "retrieval quality" of printed TV program guides satisfactory. Ehrmantraut [EHWS96] states that TV guides for digital TV (500 channels)

would be 350 pages long—without extra category indexes. Channel surfing, as an alternative program selection strategy, would take over an hour to scan all channels.

Attempting to meet the changed requirements, several web-based program guides have emerged during the past years, e.g. the Gist (http://www.thegist.com) and the Compass (http://www.compass.de). Furthermore, several digital TV receivers, so-called *set-top boxes*, provide built-in *electronic program guides* (EPG, see [Wit95, WG95, CGG+96]), e.g. the D-Box (http://www.df1.de), Prevue (http://www.prevue.com), Starsight (www.starsight.com), or VideoGuide (http://www.vgi.com) [Fru97]. All these systems allow at least paging through the TV program by date, time, and channel; some allow searching by genre or support text search. The profiling capabilities of these systems, however, are still very limited. Some web-based systems allow users to save and recall simple content-based queries (e.g. http://www.tvmovie.de). Only recently the first set-top box-based system was presented in Germany that automatically builds user profiles based on the user's relevance feedback (The *Tivo* system [Scr99]). None of the currently available systems uses advanced filtering techniques, such as collaborative filtering.

5.1.1 Related work in TV program recommendation and related IF application areas

There have been few research projects about TV recommendation, e.g. [Ehr95, EHWS96, DtHh98]. They mainly focus on technical aspects, such as the integration of IF functionality into set-top boxes and on the technical possibilities for monitoring user behavior. There are also some US patents related to the application of IF techniques to the problem of TV program selection, e.g. [Str93, Str96, Herz94].

Much more research has been done in related research areas, such as the recommendation of movies, CDs, news group articles, web pages, or scientific reports (see Section 1.1.2 and Chapter 2). To estimate how far techniques from these related application areas are applicable to the TV domain, we will briefly analyze the main characteristics of the TV domain and compare them to the related areas. The TV application area is mainly characterized by the following three criteria that are relevant to IF system design.

1. Most of the content on TV is non-textual. This restricts the applicability of text-based filtering techniques. While textual information objects, such as news articles, can directly be handled with text-based filtering methods (e.g. full text indexing), TV programs require surrogates, such as descriptions or close captions to be assigned, before text-based techniques can be applied (see Section 5.4.1).

2. TV is a broadcast medium. Unlike publishing media, where objects are available at least for a certain amount of time after being published, TV programs are only available *while* broadcast. This has a triple impact on the application of collaborative filtering techniques.

- Collaborative filtering systems work by recording the *reactions of users to data objects* ([GNOT92], see also Section 1.1.2.2.1), but unless a program has been broadcast before (which is the case for many entertainment programs, but not for live broadcasts, such as news programs), none of the users know the program. In this situation, users can only share their *expectations* that may for example be reactions to TV program descriptions or

previews these users have seen, but they cannot share judgments based on experience with the actual programs.

- The limited duration of life of TV program descriptions causes users' annotations to lose part of their utility when the corresponding program is broadcast. Outdated annotations may still be used to identify neighborhoods of users with similar tastes, but they cannot be used as recommendations for other users (see Section 5.4.2.3), unless the program is broadcast repeatedly.
- For TV program *descriptions*, the property of being *new* is not associated with any additional value. While news articles are preferably retrieved while new, there is no specific incentive to retrieve TV program descriptions immediately when issued. On the one hand, earlier planning may allow viewers to adapt their personal schedule to not miss highly relevant programs. On the other hand, to take interest changes and moods into account users may delay planning their TV consumption until the last minute[27]. The lack of incentives to look at newly issued TV program descriptions has a significant impact on the so-called *first-rater problem* (see Section 5.4.2.2).

3. The medium TV delivers a broad range of content. Unlike more specific media such as movies or news articles, TV provides contents ranging from information to entertainment and that satisfy user interests ranging from task-specific interests to pastime (see, for example, [KC90]). These interests have different durations of life and are subject to different types of interest changes (Figure 51). This diversity of interests causes a corresponding diversity of planning behaviors. Since TV programs are broadcast at a fixed point in time, users planning their TV consumption are required to estimate what their interests will be when the respective program will be broadcast (or in the time after the program is broadcast, if a VCR is used). This extrapolation is more reliable for more stable interests, so that these interests can be planned for a longer period ahead. More dynamic interests, e.g. those that are subject to moods, are better not planned very far ahead or may even be handled on ad hoc basis (see also Figure 57 in Section 5.4.2.4). Consequently, TV program recommendation systems have to support long-term interests and short-term interests, long-term planning and ad hoc querying.

[27] Problems caused by last-minute interest shifts and moods can be avoided by using a VCR or their modern counterparts, such as *Tivo* [Scr99]. Such systems allow users to record what they *might* be interested in and to decide later whether and if so when to watch it. The work presented in this chapter, however, will not investigate this TV consumption strategy, but will focus on *immediate* TV watching.

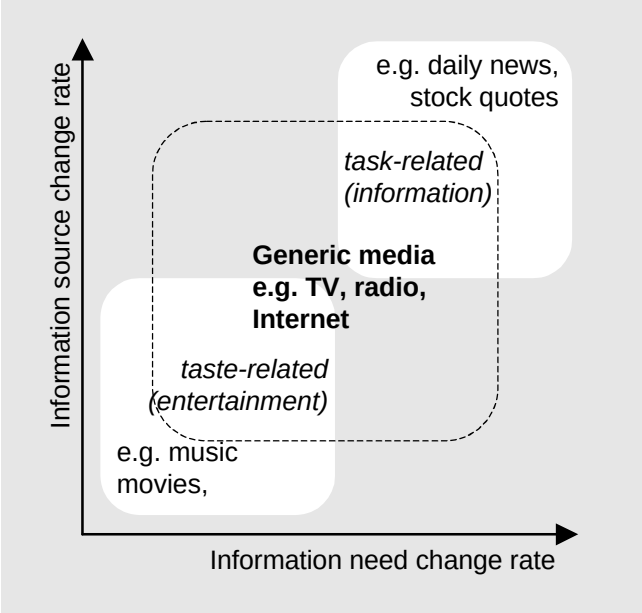


Figure 51: Properties of generic media, such as TV

Table 12 summarizes the above discussion and compares the TV domain to related application areas. None of the other application areas corresponds to the TV domain in all properties, so that existing IF concept for these application areas cannot be expected to be directly applicable to TV. However, there are several similarities. Web pages and Usenet news, for example, have a similar breadth of content; online auctions have the same limited duration of life and time-value curve; and movies and CDs also have to deal with surrogates instead of the actual objects. We will take these similarities into account by incorporating the respective filtering techniques as subsystems into our TV recommendation system (see the *query classes* of the TV Scout in Section 5.4).

	Information or entertainment content	Value of an object over time	Mostly textual or non-textual content
TV	both	⇗ (then ⇩)	non-text
Well-maintained Web pages	both	⇨	Text
Usenet news, bulletin boards	both	⇘	text
Online auctions	information	⇗ (then ⇩)	non-text
Movies or CDs	entertainment	⇨	non-text
Tech reports, memos	information	⇘	Text

Table 12: Correspondence between TV and related filtering application areas. Shaded cells correspond to TV properties.

Especially its breadth of interests and interest changes makes the TV domain an appropriate application area for the QuerySet Architecture. The combination of automatic and manual update mechanisms provided by this architecture allows handling the whole bandwidth of interest changes. As we will see in Sections 5.4 and 5.5, the capability of plugging in multiple content-based and collaborative filtering components as query-executing subsystems allows QSA systems to handle a wide range of interests that users have with respect to TV.

5.2 OVERVIEW OVER THE TV SCOUT SYSTEM

The TV Scout project was conducted in cooperation with the TV program guide publisher TV TODAY, who provides the TV program description content, consisting of a database of program descriptions and an additional movie database, and a genre classification.

5.2.1 Implementation

The TV Scout is implemented as a web-based system, not as a set-top box. While a set-top box implementation would have provided extended means of monitoring user behavior (see also Section 5.3), a web-based implementation provides better support for system development and system updates, which was judged especially important. Another reason was that set-top boxes were not in wide use in Germany when the project started, so that it was questionable whether a set-top box-based system would be able to reach a sufficient number of users for running collaborative filtering techniques. However, there are no architectural problems in porting the resulting system to a set-top box.

The TV Scout is built as a client-server system. To support collaborative filtering functionality (see Section 5.4.2.2 and 5.4.3), user profiles are stored on the server side. Users access their accounts using a self-selected login name and password. The computation described in the following sections is mainly performed by program scripts (Perl) that are compiled into an Apache web server and by Java and JavaScript programs that are downloaded to the client's web browser. Query execution subsystems run either on a Sybase database management system [SyBase89, SQL91] or on a FreeWAIS retrieval system [GP97].

5.2.2 The user interface

Figure 52 shows the TV Scout user interface. The TV Scout user interface consists of profile editing tools (top), menu and search components (middle), and TV listing components (bottom). This illustration serves only as a first overview; details will be shown in the following figures.

The web browser-based interface (Figure 52 center right) consists of the menu frame on the left and the content frame on the right. The menu frame provides users with access to all retrieval and filtering functions and is permanently visible. It consists of the exact match menu (top, see also Figure 53), the query class menu (center, see also Figure 54), and the retention tool menu (bottom). The content frame is used to display various types of TV listings and all profile editing tools. When users launch the TV Scout, a starting page is loaded that highlights some current recommendations.

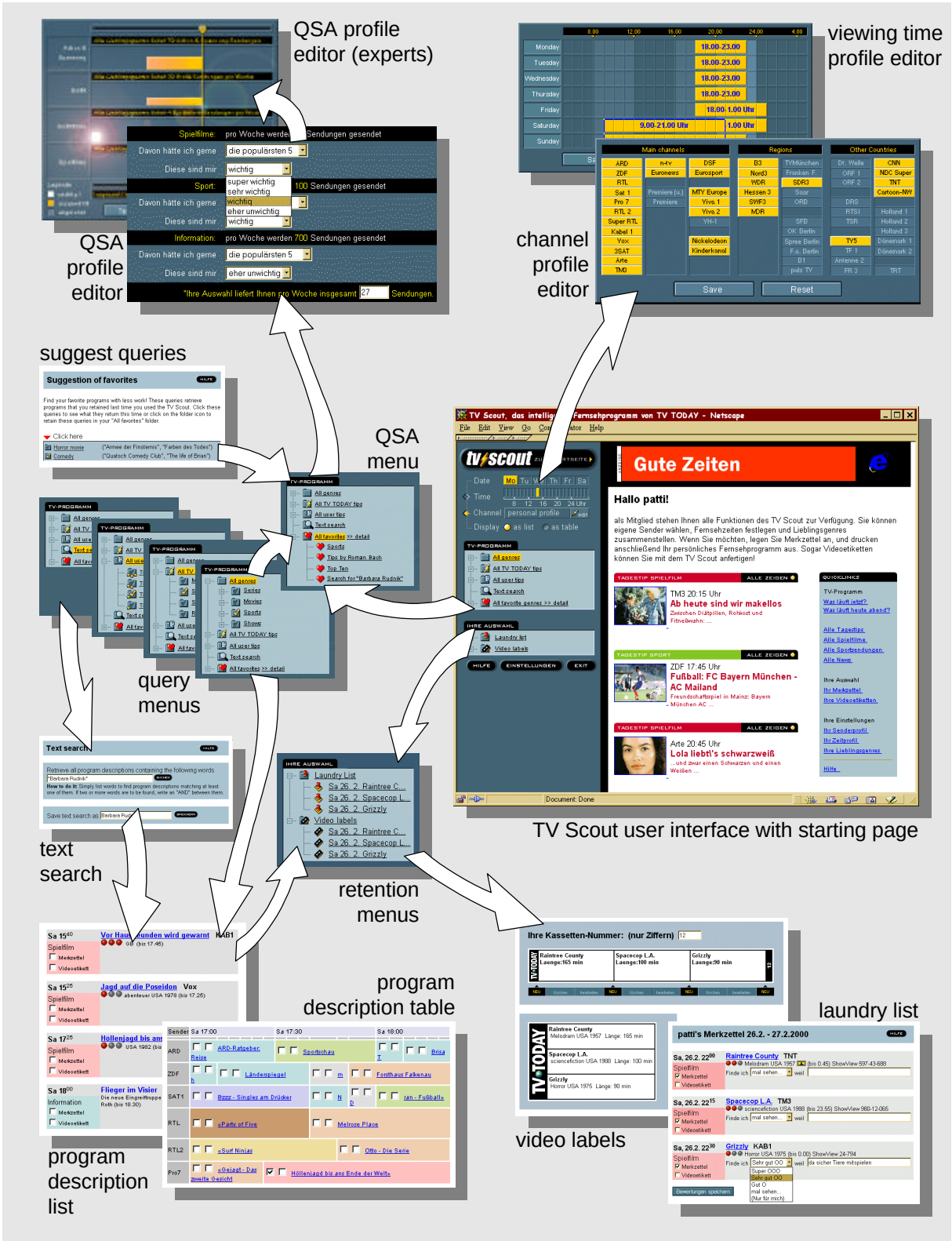


Figure 52: Overview: the TV Scout user interface (screenshots partially translated from German)

Query menu: Users can retrieve TV program descriptions by executing a query from the query menu (see Figure 54 for several close-ups of this menu). The query menu holds five submenus that will be presented in Section 5.4. Most menus contain a hierarchy of submenus that can be browsed in a file system explorer-like fashion; queries in submenus match subsets of the objects matched by the parent query. Only the text search window requires users to enter parameters; all other queries are parameter-less queries that are executed by simply picking them.

Exact match settings and profile: All queries are executed as a conjunction with the current exact match settings. The three parameters date, time, and channel can be entered using individual controls (Figure 53a). Time and date intervals can be entered using mouse drag interactions; channels can be selected from two predefined sets. Their default setting is current day, current hour, and all channels.

Settings for time and channel can be stored permanently in the users exact match profile. Figure 53 shows how users can switch to profile mode (Figure 53a) and how the respective editors are invoked (Figure 53b). The viewing time profile allows users to enter arbitrary time intervals for each day of the week. The channel profile allows users to select an arbitrary subset of all available channels. Both editors are implemented as paintable interfaces in Java and have been presented in Section 4.4.6, Figure 47a and Figure 49.

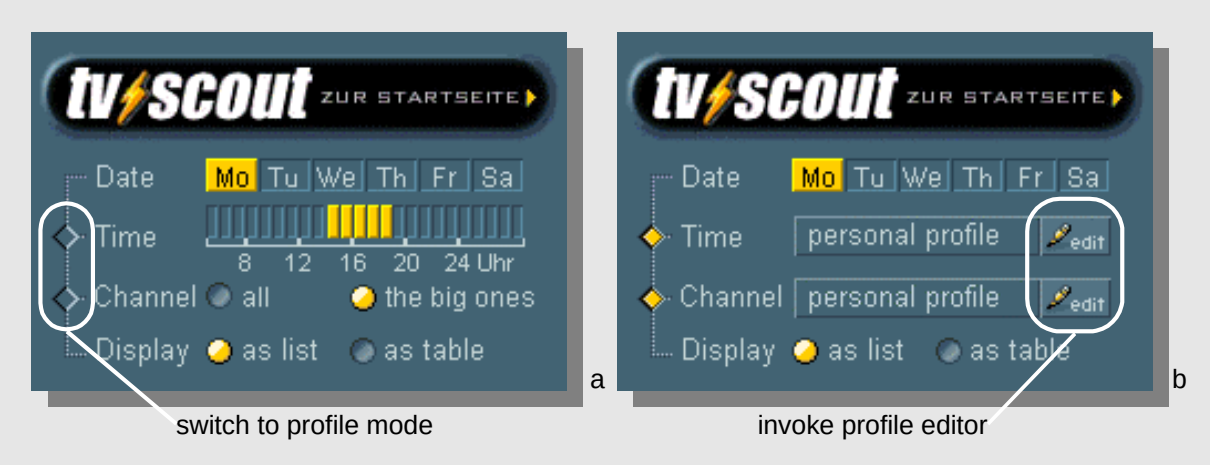


Figure 53: The exact match menu, switching to the exact match profiles (a), and invoking the respective profile editors (b).

Displaying and retaining program descriptions: When a query is executed, the resulting set of TV program descriptions is displayed in the content area. Users can pick the display styles *ranked list* or *table* (Figure 52 bottom left) using the corresponding buttons shown at the bottom of the exact match menu (Figure 53). In program lists and tables, colored fields visualize program category (hue) and rating (saturation). Depending on visualization style and available space, the title of the programs and an extract of the program description are shown. The full program description including screenshots and detailed ratings can be obtained by clicking the program title.

Toggle switches accompanying program descriptions allow users to retain the program they plan to watch in the so-called *retention tools*. The retention tool *laundry list* can be used to print a list of programs; *video labels* are designed to retain and print programs to be recorded on video tape (Figure 52 bottom right). The retention menu at the bottom of the menu frame allows users to display the content of their retention tools for review or printing. To provide immediate visual feedback of retention operations, both menus open automatically and show the updated content whenever programs are inserted or removed.

Creating the QSA profile: Users can retain queries in their QSA profiles by clicking the toggle switch that accompanies each query menu entry (a folder symbol with a check mark, see Figure 54). Since text searches require additional parameters, retaining them requires using the "Save text search as [name]" control in the search dialog. The QSA menu that is labeled "All favorites" contains all retained queries, independent of the query submenu they originated from. The QSA menu provides the same visual feedback for insertion and deletion as the retention menus. Since the QSA profile can be queried like any other query submenu, its menu is displayed in a container together with the query submenus.

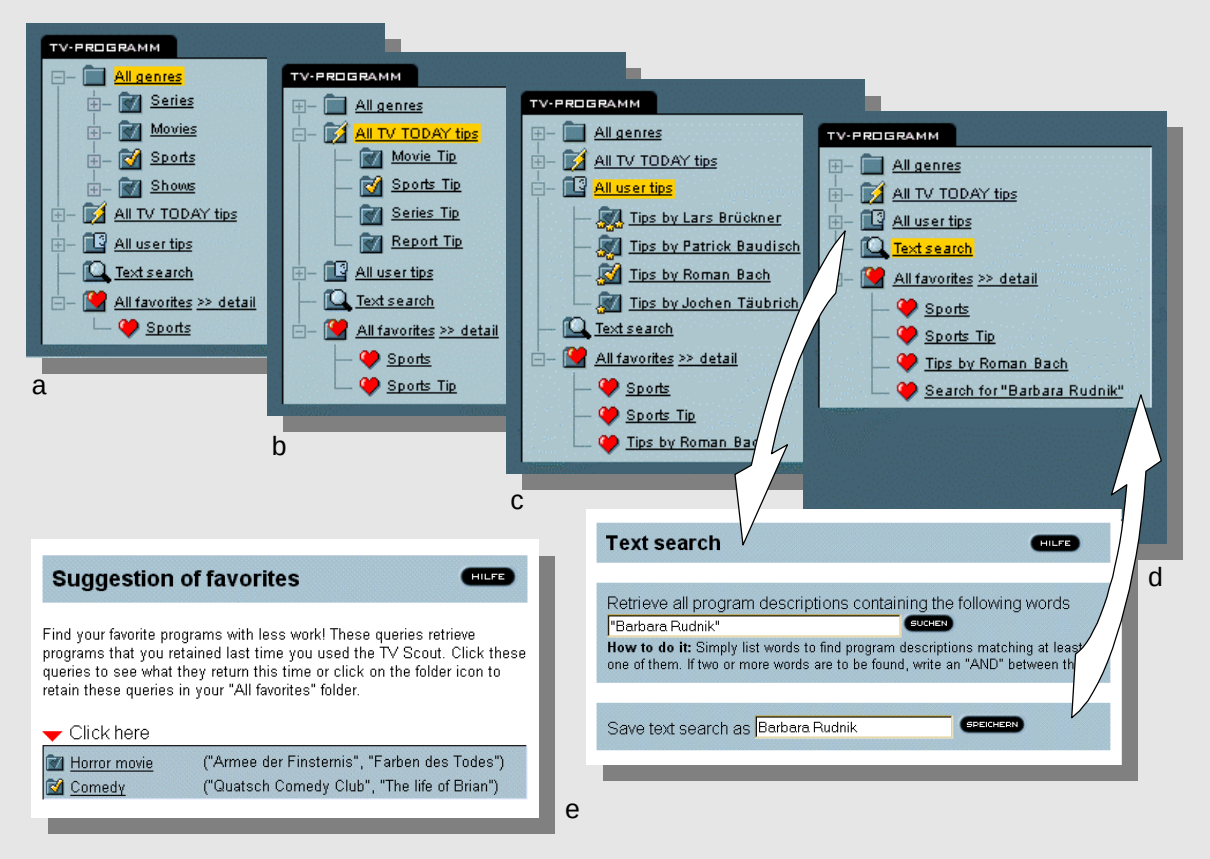


Figure 54: By clicking a hook-shaped toggle switch (a-c) or clicking a *save* button (d), queries can be retained in the user's QSA profile labeled "All favorites". Queries can also be suggested for retention (e).

When the TV Scout has collected a sufficient amount of information about the user (see Section 5.3.2) it starts suggesting queries as shown in Figure 54e. The TV Scout uses a pop-up dialog to show suggested queries that are computed offline. A future version suggesting queries in real-time may offer suggested queries as another permanently visible query submenu.

Using the QSA profile: There are two ways of using the QSA profile. First, users can execute retained queries in a *bookmark*-like manner by clicking the individual entries of the QSA menu. Second, users can execute the entire QSA profile by clicking the top menu entry "All favorites". This invokes the profile execution described in Chapter 3 (see also Section 5.5, Figure 60). Queries from the QSA menu and the QSA profile are both executed in the context of the current exact match settings, *not* the settings that were used when the query was retained.

Editing the QSA profile: Clicking the ">>details" link in the QSA menu invokes the form-based profile editor that was presented in Section 4.1, Figure 20. It allows users to manually adjust their profiles by entering ratings of top-ranked objects and desired amounts for the individual queries. Via a link located on this page, experienced users may also access a histogram-based profile editor (see Section 4.2.7, Figure 36). For performance reasons, the currently used histogram-based editor uses rectangles instead of actual histograms. The TV Scout does currently not offer a paintable profile editor. When the TV Scout was released, browsers required more time for launching the Java Virtual Machine than what users would have saved by using this more efficient interface.

Beside the presented interface components, the TV Scout offers a collection of help pages and several preference dialogs that allow users to customize the system.

In the remainder of this chapter, we will present and discuss the underlying filtering mechanism implemented by the TV Scout. We will begin by looking at how the TV Scout gathers relevance feedback that allows it to learn about users' interests and interest changes. Then we will present the individual query classes that implement the individual submenus of the query menu.

5.3 GATHERING IMPLICIT RELEVANCE FEEDBACK

The profile learning mechanism presented in Section 3.3 is based on the availability of relevance feedback. Relevance feedback may be obtained from explicit feedback, i.e. by asking users to rate objects, or from implicit ratings, i.e. by estimating, based on observed user behavior, how the user would rate objects [OK98]. In the TV Scout, we decided to use implicit feedback, because the incremental guidance of users from ad hoc querying to filtering (see Section 3.4.1) renders the usage of explicit feedback problematic, as we will discuss in the following.

To motivate users, their effort to provide explicit ratings has to be rewarded with some sort of benefit. The more immediate this benefit is the better. The benefit that the TV Scout can give to users providing ratings (there is more benefit for the community, see Section 5.4) is that it will suggest queries and automatically refine the users' QSA profiles. Both techniques support users in obtaining improved retrieval quality.

But how much of an incentive is retrieval quality for casual or first-time users? The 5000 TV program descriptions that the TV Scout system holds per week are relatively few compared to, for example, millions of web pages. As long as the system is used in a one shot fashion, the retrieval quality provided by the predefined queries will often be perceived as sufficient. Optimizing queries pays off if the optimized queries are used repeatedly, e.g. by storing them in a user profile, so that instead of the 5000 programs of a week the 250,000 programs of a year can be processed. Therefore, building an elaborate user profile is mainly an investment in the future. For frequent users, profiling functionality may be a sufficient incentive for providing explicit relevance feedback. For casual users, however, the promised benefit may be too far in the future.

Implicit feedback is not hampered with these problems, because it does not require users to spend additional effort. If implicit feedback is used, the system can gather relevance feedback also from casual and first-time users, even if they do not have any ambitions to optimize their retrieval quality or to build a user profile. This has two advantages. First, assuming that there are a substantial number of casual users, more implicit feedback than explicit feedback can be gathered. This can improve the prediction quality of queries based on collaborative filtering (Section 5.4.2). Second, the system can automatically build a user profile based on gathered implicit feedback or propose components for building one. Accepting such a profile causes little effort to users, so that this approach makes it easier for users to take the step towards a more filtering-oriented usage of the system that will save a substantial amount of work in the long run.

The problem with implicit feedback is the low quality of the individual feedback units. Implicit feedback is restricted to what is observable. From observation, it is for example usually not possible to quantify how *useful* an object is for a user. Furthermore, the observed behavior is usually ambiguous (see Section 5.3.2), so that less information can be extracted from implicit feedback than what would be obtainable from explicit input. Implicit feedback has to compensate for this by gathering proportionally more input.

5.3.1 Which type of user behaviors to use as a source of implicit feedback?

Different types of user behavior are potential candidates for being used as sources of implicit feedback. To gather implicit feedback, at least one type of user behavior has to be monitored. Figure 55 shows the schema of implicit feedback methods by Nichols and Oard [Nic97, OK98] that classifies the sources of implicit feedback into the three categories examination, retention, and reference behavior.

Before users can retain or forward TV program descriptions, they will usually have to examine them. Examination behavior can therefore be expected to occur more frequently than the other behavior types, which makes it attractive as a source for implicit feedback. Different types of examination behavior, e.g. requests for a full program description, may be used as input. Also reading time is a potential source of implicit examination feedback [MS94].

The problem with examination behavior in a *web-based* TV recommendation system is the unclear correspondence between examination behavior and perceived relevance. Since these

systems cannot monitor users watching the actual *programs*[28], they are limited to monitoring users examining the programs' *descriptions*. This problem distinguishes the TV domain from many other application areas. In TV, at least some of the actual objects, e.g. live broadcasts TV programs, *cannot* be made available at filtering time (see Section 5.1.1). But what does it mean if users spend time examining a TV program *description*? Since TV programs can be broadcast repetitively or can belong to a series, a TV program can often be identified rapidly and users can decide whether or not to watch it without reading the description details. If users retrieve program descriptions and examine them for an instant only, the system cannot know whether this indicates that the user already knows (and maybe likes) the program or whether the user judged it to be of little interest. Because of these ambiguities, implicit feedback deduced from object examination is of limited use in the TV domain.

Category	**Observable Behavior**
Examination	Selection Duration Edit wear Repetition Purchase (object or subscription)
Retention	Save a reference or save object (with or without annotation) (with or without organization) Print Delete
Reference	Object→Object (forward, reply, post follow up) Object→Object (hypertext link, citation) Object→Object (cut, paste, quotation)

Figure 55: Observable behavior that may serve as implicit feedback [OK98].

The remaining choices are retention and reference behavior. Forwarding references not only requires the system to provide the respective functionality, but also requires users to have somebody to forward a reference to. Unless users already bring their pre-existing social networks into the system, building up a social network requires time. During this period, users would not produce any implicit reference feedback, which is basically the critique we had launched against explicit feedback.

[28] A set-top box-based system *can* monitor what users are actually watching [EHWS96, Cossmann 96]. This additional information source allows gathering a large amount of implicit feedback. However, the gathered information is of limited use. While it allows refining profiles based on a content analysis of what the user has watched, it does not support making predictions based on collaborative filtering. Implicit ratings are obtained too late to be useful to other users. At the limit, such feedback can help channel surfers to find out which already running program they should switch to.

Retention does not require a social network and is a very natural behavior in a system that helps users to *plan* their TV consumption. Because of the distinction between TV programs and program descriptions, users *have to* retain their intention to watch a program until the program is actually broadcast. This can be done either mentally or with the help of the computer, e.g. by saving or printing program descriptions. The help of the computer may not only be perceived as more convenient, it will also produce system-observable behavior.

Since current web browsers do not allow monitoring saving and printing activities, the TV Scout provides the two *retention tools* presented in Section 5.2.2. Program descriptions retained using the retention tools are stored at the server side, which allows the TV Scout to simultaneously access program retention information about all users, e.g. to support collaborative filtering techniques (see Section 5.4.2).

5.3.2 Deducing implicit ratings from retention tools

To assign implicit ratings to TV programs, the TV Scout not only monitors which program descriptions have been *retained*, but also which program descriptions have been *displayed* and which have been *avoided*. This allows the system to limit the negative implicit rating *not retained* to those programs that the user *decided* not to retain.

Implicit ratings are assigned as follows. The user's ratings about all program descriptions are initialized to the value *not inspected*. When program descriptions are displayed to the user, the ratings of all displayed program descriptions with *not inspected* rating are overwritten with the negative implicit rating *not retained*. Whenever the user retains a program description, this program description's rating is overwritten with the positive implicit rating *retained*.

By using queries representing their interests, users can limit the set of program descriptions that are displayed to what is at least potentially interesting and can thereby *avoid* uninteresting programs. The usage of queries[29] makes a statement about the relevance of programs—not necessarily about the retrieved programs, but about those programs that are never retrieved. To constantly exclude a program from retrieval by using only queries not matching the program is considered deliberate avoidance. These programs are assigned the negative implicit rating *avoided*, which the TV Scout handles as an equivalent to *not retained*. To assign this rating, the algorithm described above is enhanced by inserting an additional step. Before *not retained* ratings are written to all programs matching the query, *all programs matching the current exact match settings* that are currently rated *not inspected* are overwritten with *avoided*.

Using this algorithm, only programs with exact match attributes not matching any of the user's queries keep their *not inspected* ratings. The system has no evidence for that the user is *able* to receive these other programs. The user may have no time to watch these programs, may not be able to receive the respective channel at the respective date and time, or may not

[29] This refers to queries searching for programs matching a given interest, e.g. genre or text search queries. This does *not* include the exact match menu. The fact that a program is in a data/time/channel segment that the user cannot access does not imply that the user actively avoids a given program.

have used the system to plan for this time interval. There is no evidence for that the user intentionally avoided these programs.

The assignment of *avoided* ratings assumes that the system can deduce which programs users are able to watch based on the user's exact match settings. This assumption is only justified if users use the exact match menu in the maximally restrictive way, i.e. they only retrieve programs that they would also be willing to retain, i.e. from channels they can receive and at times they have time to watch TV. If the user has not yet set up a channel profile and uses one of the predefined channel sets in a query, this assumption is not justified. In this situation, the TV Scout constructs a temporary channel profile consisting only of those channels that the user has already retained programs from. If users issue queries including all 24 hours of a day, the system accepts this as a fact—users are then assumed to plan to use a video recorder.

5.3.3 The meaning of implicit retention ratings

The notion of relevance reflected by output ratings is strongly related to the relevance notion of the relevance feedback they are constructed from. In the case of the TV Scout, the monitored behavior considered as relevance feedback is *retention*. Consequently, all predicted output ratings that are solely based on implicit ratings (e.g. predicted popularity and automated collaborative filtering, see Section 5.4.2) are estimates of the *probability of retention* of a program. We will briefly compare this relevance notion to other relevance notions.

There is a difference between *retention* and *planning to watch*. To retain programs, users have the choice between the usage of the retention tools and other means, e.g. mental memorization. Programs that can easily be retained mentally, such as favorite series, may therefore be under-represented in implicit ratings extracted from retention behavior.

Furthermore, there is a difference between *planning to watch* and *watching*. On the one hand, TV consumption usually is a combination of planned TV consumption and opportunistic TV consumption, such as channel surfing, so users also watch programs they had not planned to watch. On the other hand, people may not always watch what they have planned to watch, because their time schedule or their interests have changed in the meantime. Program retention delivers an estimate for what users plan to watch. Neither does it attempt to predict opportunistic TV consumption nor time schedule or interest changes until broadcast time.

Finally, *to watch* a program is different from *to have liked the program in the past*. While past experiences may provide more accurate data for correlating users (therefore used by many Automated collaborative filtering systems, such as [Sha94]), past experiences may not necessarily be the best predictor for future watching behavior. Users may have undergone interest changes in the meantime, especially *while* watching the program. Therefore, users may choose not to watch a program again, although they had liked it the first time. For predicting user behavior, it is more important how much users expect to like a program *next time*. Program retention expresses such expectations.

As stated above, the implicit retention ratings of the TV Scout allow computing output ratings related to the *probability of retention*. It cannot predict what users will channel surf into, what programs users have liked in the past, or what users do not have to plan because they already

know when its on. Retention-based ratings support users only in doing what their task in the TV Scout is—to find the programs that they will want to retain.

5.4 THE QUERY CLASSES

The TV Scout supports users in finding desired programs by providing a wide range of different queries. Queries are grouped in four so-called *query classes;* a query class is the set of queries that is executed on the same subsystem. In its current version, the TV Scout offers the four query classes *text search*, *genres*, *user tips*, and *editor's tips* [Bau98b]. The queries provided by these query classes can be executed ad hoc, to immediately find matching programs, or they can be inserted into the user's QSA profile to automate the handling of repetitive and long-term interests. In this section, we will describe the subsystems underlying the individual query classes.

5.4.1 Text search

The TV Scout text search is implemented using the Wide Area Information Server *FreeWAIS* [GP97]. To initiate indexing, the fields *title*, *description* (including optional text from the movie database), *date*, *time*, and *channel* of all TV program descriptions are converted to plain text and uploaded into FreeWAIS. FreeWAIS performs full text indexing on *title* and *description*. The fields *date*, *time*, and *channel* are indexed to support search in an exact match fashion.

At runtime, the search string that users type into the search dialog (Figure 54, bottom right) is combined with the current settings of the exact match menu (Figure 53) and sent to FreeWAIS via TCP-IP. FreeWAIS returns the matching program descriptions ranked according to TF-IDF weighting. FreeWAIS supports search for keywords and combinations of keywords with optional Boolean syntax.

5.4.2 Genres combined with predicted popularity and automated collaborative filtering

Genres are a historically grown classification of TV programs. They refer to a mixture of several program properties, including program content, e.g. *sports* or *western*, involved emotions, e.g. *comedy* or *romance*, and organizational structure of the programs, e.g. *series*. See the Service Information, part of the Digital Video Broadcast standard [DVB96], for a classification of TV program genres.

Genres are in widespread use and TV viewers usually know at least some genres from their past experience with TV and TV program guides. This pre-experience simplifies the genre user interface; it is usually sufficient to label genres with their name to make users recognize them. Unlike text search, genre queries do not require any parameters, so genre queries can be executed by simply picking them from a predefined set; typing can be avoided.

The genre user interface is built on the subset relations between genres (e.g. *action comedy* $\subseteq$ *action movies* $\subseteq$ *movies*). Since genres can be subsets of multiple parent genres (e.g. *action comedy* $\subseteq$ *action movies* but also *action comedy* $\subseteq$ *comedy*), genres form a directed acyclic graph. The TV Scout displays this graph using the file explorer-like interface shown in Figure 54a. Genres can be expanded into subgenres by clicking the +-symbol that accompanies each non-leaf genre. Since this interface can only display tree structures, genres with multiple parent genres can be found in multiple submenus. The file explorer-style interface supports users in interactively searching for more specific genres that express their interests more precisely.

The TV Scout distinguished between three levels of genres. The twelve top-level genres (*sports*, *movies*, *series*, etc.) are indexed manually by the company providing the raw program data for TV TODAY. Mid-level genres (e.g. *action movies* or *horror comedy*) are indexed manually by TV TODAY editors. Low-level genres are indexed automatically using regular expressions over program description and high- and mid-level genres, such as a search for the term *basketball* in the top-level genre *sports*. The enhancement of the manually indexed genre structure with low-level genres is possible because of the absence of space limitations that printed TV program guides have to face (see Section 5.1.1). Providing more detail than manually indexed genres, but still the convenient handling of parameterless queries, low-level genres close the gap between manually indexed genres and text search.

Top- and mid-level genres are also used by other products of TV TODAY, mainly their printed TV program guide. This genre information is therefore already stored in program descriptions. Low-level indexing is done automatically using a set of continuous queries, so that the entire genre indexing process of the TV Scout is maintenance-free.

High- and mid-level genres are indexed as exact-match categories; either a program is part of a genre or it is not. Although some action movies, for example, are more action-oriented than other action movies, this fuzzy categorization is not considered by the manual indexing process. Although such Boolean ratings can be handled by QSA systems, query classes are preferred to provide gradual ratings to allow users to optimize their search strategies. Inserted into QSA profiles, gradual ratings better mix with other queries in the profile. For imposing gradual ratings on genres, the TV Scout uses an approach based on collaborative filtering, as we describe in the following.

5.4.2.1 Ranking programs within genres according to predicted popularity

The size of the audience is a widely accepted measure of the success of TV programs. Statistics about German TV programs [GFK97] show that there are significant differences in the size of the audience of TV programs. Figure 56a shows an example of a top sports event (right bar) that has substantially more viewers than any other sports program at that day.

If the success of a program is to be compared between groups of different sizes, the size of the audience is biased toward the larger group, because the larger group has more potential viewers. The impact of a program on individual users is therefore better measured by the quotient of size of the audiences and sample size, i.e. which *portion* of the group members has watched the program. This quotient of the number of viewers and the number of people in the monitored group is called *rating* or *household rating* (http://www.nielsenmedia.com/

whatratingsmean). To avoid overloading the already heavily used term rating, we will use the term *popularity* instead.

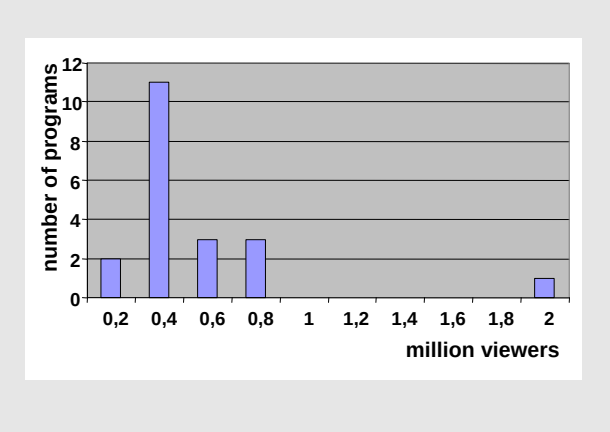


Figure 56: Example of a size of the audience distribution over German sports programs from December 3rd 1998 [GFK97].

The popularity of a program represents the probability that a randomly selected viewer has seen the given program. For not further qualified users, popularity would therefore be a useful criterion for ranking programs, e.g. within genres. Unfortunately, the popularity of a program is not known before the program is broadcast (although TV stations may come up with estimates for the size of the audience of their programs when computing ad rates).

The TV Scout uses a related measure that can be computed in advance. We define the *predicted popularity* of a program as the number of users who decided to *retain* this program divided by the sample size [Bau98c, Fru97]. Predicted popularity tells users which percentage of their fellow users have decided to retain this program. Since retaining a program is related to the plan to watch that program, predicted popularity also gives some indication about how many users plan to watch that program (see Section 5.3.2).

Because program descriptions are checked out successively by the users of the system, the sample size in the denominator of predicted popularity is variable. It is therefore not possible to simply use all users as the sample. Instead, we have to restrict the sample to the subset of users who have already have checked out the respective program description or that have *avoided* it (see Section 5.3.2). We therefore compute the predicted popularity p of a program *b* as

$$p(b) = \frac{r(b)}{n(b) + r(b) + a(b)} \qquad \text{(Equation 5)}$$

with r(*b*), n(*b*), and a(*b*) denoting how often *b* was retained, not retained, and avoided respectively (see Section 5.3.2).

This definition restricts the sample to those users who have implicitly told the system that they are *able* to watch the program using their exact match menu settings. This restriction is appropriate, because the users who receive predicted popularity values for a program have ex-

pressed with *their* exact match settings that they are also able to receive these programs, otherwise they would never see the program description and its predicted popularity. This way, users are given popularity predictions based on the subgroup of users who are also able to receive this program. This means that p(*b*) is computed as the conditional probability that users will retain the program under the assumption that they are able to receive it. This approach allows users to compare programs that are broadcast under *different* date, time, and channel situations. This approach allows a program that is broadcast on a local station and also on a nationwide station to receive the same predicted popularity for both broadcastings, although the nation-wide broadcast will typically be retained more often. For a user known to be able to receive both programs this is the desired effect.

Predicted popularity becomes more accurate the bigger the sample is. Therefore, the predicted popularity of a program becomes increasingly accurate as retention ratings accumulate towards broadcast day. Achievable accuracy depends largely on the number of users and the usage of the retention tools.

5.4.2.2 Embedding an automated collaborative filtering subsystem

The next step in improving predicted popularity is to base the prediction only on that sample of users that are known to have interests/tastes similar to the user seeking predictions, the user's so-called *neighborhood*. This restriction turns predictions into personalized predictions; predicted popularity becomes automated collaborative filtering (see Sections 1.1.2.2.2 and 2.2.4).

The TV Scout maintains two taste-related entities that can be compared to determine the similarity between users. These entities are programs and queries. While the user's preference with respect to programs can be observed by monitoring retention in the retention tools, the user's preference with respect to queries can be observed by monitoring the retention of queries in the user's QSA profile[30]. Neighborhoods can be determined based on interest similarities with respect to programs (retention tools), with respect to queries (QSA profile), or with respect to both.

Measuring the similarity of two users' interests can be done using existing Automated collaborative filtering algorithms. ACF can be handled by an independent subsystem and since ACF systems are now available off the shelf, e.g. from *Net Perceptions* (http://www.netperceptions.com), we will restrict ourselves to demonstrating how to *embed* an ACF subsystems into a QSA system, such as the TV Scout. See [Sha94, SM95, MRK97] for more details on the basic algorithms required for writing an ACF system from scratch.

To compute similarity based on retained programs, the implicit ratings of each user, i.e. *retained* and *not retained/avoided* are stored in the rows of the common rating table (see Section 1.1.2.2.2, Table 1). Since *not retained* and *avoided* correspond to the same meaning, they should be handled as equivalent; a program rated *not retained* by one user and *avoided* by the

[30] Unlike the examination of program descriptions, the "examination" of queries, i.e. the execution of queries, may also be taken as a source of implicit feedback. The critique launched against the usage of implicit examination feedback with TV programs descriptions does not apply to queries.

other should be counted as a match. As usual in ACF, a user's neighborhood is now determined as those users having the closest matching table rows. The comparison is usually restricted to those programs that both users have assigned implicit ratings, i.e. *retained* or *not retained/avoided*. Since retention has a higher expressiveness than non-retention, similarities in *retained* ratings should be given a higher weight than similarities in *not retained*/*avoided* ratings.

Measuring the similarity of two users' interests based on retained queries[31] requires some additional computation. Simply considering queries as entities and using these entities as a basis for comparison [Fru97] results in a poor similarity measure, because this approach does not take into account that two non-identical queries can still be similar to a certain degree. A QSA profile consisting of *action movies* and *comedies*, for example, should be considered similar to a QSA profile consisting of the query *action comedies*, although the two profiles have no queries in common. However, especially for complex user-defined text searches, hardly any matches with other users' profiles will be found.

User profiles are compared more accurately by "expanding" them, i.e. by applying them to a test set of programs and comparing the predicted ratings. This approach works for queries from arbitrary query classes, including manually written text searches, and also the additional information contained in interest intensities and normalization functions is taken into account. Using this approach, the similarity measure on QSA profiles is reduced to a similarity measure on sets of program ratings. It can be handled by feeding the predicted ratings into an arbitrary Automated collaborative filtering system.

Since retained queries usually match more programs than users would actually select for retention, computing neighborhoods based on expanded QSA profiles can produce more matches than neighborhood computation based on retained programs. This may be preferable in situations where there are only few users using the system, so that the collaborative filtering table is very sparse (This approach of coping with sparcity is similar to the Filterbots concept [GSK+99, SKB+98]). Furthermore, QSA profiles can be adapted rapidly to interest changes, which allows users to rapidly find their new neighborhoods after an interest change. The price for these two advantages is that the found matches will usually be less accurate, because QSA profiles only circumscribe what users are interested in. Even two users having exactly the same queries, e.g. something as unspecific as *movies*, may not even have a single program retention in common.

Best results may be obtained by combining both approaches, i.e. by taking program retention *and* QSA profile into account. One way of accomplishing that is to initialize the user's row in the ACF table with the ratings predicted by the QSA profile and to overwrite ratings where available with ratings from the retention tools. To reflect the higher level of confidence, program retention ratings should be given a higher weight in the computation of neighborhoods (see also [CGM+99]).

[31] For users not yet having a QSA profile, a temporary profile consisting of all queries the user has executed so far may be used.

5.4.2.3 Critical mass and cold start problem

In ACF, a user's ratings have two functions. First, they provide the basis for determining the user's neighborhood. Second, they contribute to the predictions of all users that have this user in their neighborhoods. Both these functions require a sufficient number of ratings to work properly.

If a user has provided only few ratings, the system has only little evidence about which other users are the user's "soul mates". As a consequence, the user's neighborhood is blurred with non-soul mates, while some soul mates are accidentally excluded. At the limit, when a user's profile is empty, there is *no* information for determining the neighborhood (the so-called *cold-start problem*). In this situation, the prediction mechanism falls back to predicted popularity.

To achieve a satisfactory rating accuracy, a sufficient number of ratings have to be aggregated for each predicted rating. Two properties of the TV Scout system result in an increased demand for ratings. First, since implicit retention ratings are Boolean ratings, the number of ratings to be aggregated is higher then it would be for real-valued ratings. Second, since the TV Scout retrieves ACF-rated program descriptions via genres, *all* programs are expected to be rated (full so-called *coverage*), not just a few highly recommended programs, as done for example by movie recommender systems.

Since different TV programs may receive different number of implicit ratings, neighborhood sizes may be determined on a per-program basis. For each program, ratings from the smallest neighborhood that provides *n* ratings for that program may be aggregated; since the higher confidence is provided by *retained* ratings, the smallest neighborhood that provides *n retained* ratings may be used. Ratings from closer soul mates are given more weight when aggregated [SM95]. Using program-specific neighborhood sizes, the predicted rating for a program in a core interest area of the neighborhood can be highly personalized, while ratings for programs outside the focus of the closest neighborhood will be computed by also taking other users' ratings into account. See [HKBR99] for a comparison of different aggregation strategies, including fixed neighborhood sizes.

There are situations where not even all users together provide enough implicit ratings to predict a rating for a given program. When the system is launched for the first time, for example, no ratings are available at all, so that no collaborative predictions can be made (*critical mass problem*). This problem is increased further by the fact that TV is an application area where objects have a limited duration of life. While outdated ratings can still be used to base neighborhood computations upon, they do not provide any rating material to be aggregated for making predictions[32]. TV recommender systems have to constantly gather ratings for new programs to stay above the critical mass.

---

[32] In application areas where objects have an unlimited duration of life, e.g. movies, ratings provided by past generations can be eternally reused to make recommendations. This allowed for example to jumpstart the MovieLens system by initializing its ACF database with the "dead" rating data of the abandoned EachMovie database [KRBH98]. This approach does not work for application areas where objects have a limited duration of life, such as TV or newsgroup postings [MRK97, KMM+97], unless higher-level concepts, such as series or the repeated broadcasting of programs are taken into account.

The potential unavailability of collaborative predictions is the reason why the TV Scout combines collaborative filtering techniques with genres. Since genres are purely content-based, they do not suffer from the critical mass problem and allow users to narrow down the set of potentially interesting programs when collaborative filtering is unavailable. As implicit ratings are successively collected, predicted popularity and ACF are incrementally activated [Bau98c] and predictions incrementally become more accurate and more personalized. This transition is smooth and users will not feel any break in the system's behavior. With the increasing quality of collaborative filtering predictions, even unspecific genres, at the limit the top-level genre *all genres*, can be used to find personally interesting programs. This incremental exploitation of ratings allows the collaborative filtering subsystem to be built incrementally. The current version of the TV Scout implements only predicted popularity. An ACF system may be added when the number of users and their retention ratings will suggest that the introduction of ACF will result in a significant improvement in prediction quality.

5.4.2.4 First rater problem and coverage

A special case of lacking ratings is the so-called *first-rater problem*. Program descriptions that are newly inserted into the system's database have not yet received any ratings, so that no collaborative ratings can be predicted for these programs. In the case of the TV Scout, 700 new program descriptions are added every day. Compared to application areas such as movies, TV has a rather high information source change rate, so that TV recommendation systems face a significant first-rater problem, similar to recommender systems for news articles.

As mentioned in the previous section, providing ratings has two positive effects. The first effect, i.e. to provide a basis for neighborhood determination, leads to better predictions for the user. This is a benefit *for the user*. The second effect, i.e. to contribute to the predictions for other users, results in benefit *for the community*. How big the benefit for the community is depends on when a rating is provided. Providing the last rating before the program description dates out does not allow this rating to be used for making recommendations to any other user; the benefit for the community is zero. The earlier a rating is provided the more predictions can take this rating into account and the bigger the benefit for the community.

Unfortunately, work and benefit associated with predicted popularity and ACF are distributed very unjust. The longer a user waits, the more other users have already rated the programs, and the better collaborative predictions become. Users who prefer planning ahead for a longer period involuntarily become early raters and have to cope with a large percentage of objects having undefined ratings. Early raters can only rely on content-based queries, i.e. genres and text search. While early raters in news-oriented application areas profit from the newness of the read news articles, there is no such effect in TV program descriptions (see Section 5.1.1).

Fortunately, there is a group of users who naturally qualify as early raters. Figure 57a shows that people have different preferences with respect to how far they plan ahead. In a survey with 101 subjects carried out in August 1997, the 73 subjects who used *printed* TV program guides planned their TV consumption as shown in Figure 57a. While most subjects stated to plan not further than the other day, there is a group of subjects planning for the entire week (12%). Figure 57b shows a sample of the retention distribution in the TV Scout [Gil99]. Note that, since retention is accumulative, the right diagram is different to read. The two diagrams

show similar breadth in the subjects/users planning behavior; users of the TV Scout system even plan a bit further ahead of the subjects of our survey, (35% plan further ahead than the next day, opposed to 12% in the survey).

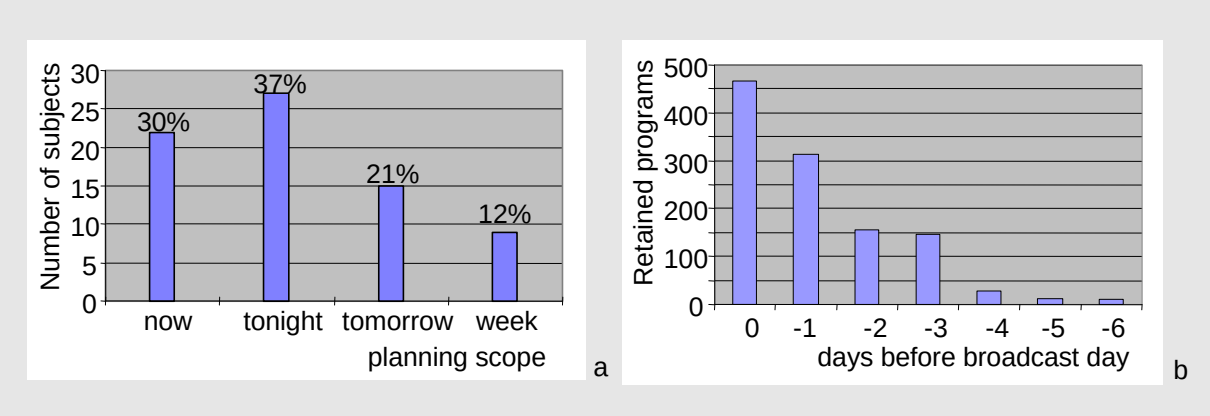


Figure 57: Planning scope of subjects in survey printed about the usage of TV program guides (a) and number of programs retained by TV Scout users during February 1999 broken down by days before broadcast day (b)

Unfortunately, these users may not necessarily be happy with the role they have been assigned by the system. Users that plan ahead for a longer period receive worse filtering quality than other users and are therefore disadvantaged. The risk is that these users could be less satisfied with the system's prediction quality and therefore change their planning behavior or even stop using the system (see for example [Gru87, Gru89] for a detailed discussion on work and benefit in collaborative systems.) Collaborative filtering systems, however, need users and especially those willing to contribute ratings to the community.

What can be done to improve the situation of early raters? Two classes of solutions for the first-rater problem have been proposed. One class deals with content-based enhancements. A popular approach falling into this class is to use agents implementing content-based predicates that play the role or early raters; these agents are called *virtual users* [Mae94], *Filterbots* [KRBH98, SKB+98, GSK+99], or *Archetypes* [Gre98]. The TV Scout uses a similar, though slightly less flexible approach. The TV Scout initializes predicted popularity values for newly inserted programs with aggregates of the ratings of past programs who's descriptions have content-based similarities with the new program description [Fru97]. A program is considered similar if it matches the same content-based queries, i.e. text searches and genre. Similarity categories are *same title* (which usually means that the same program is repeated), *same series*, *same genre*, and *similar description text*.

The other class of solutions uses various economic approaches providing compensation for early ratings [Gl98, AZ97, RV97, ARZ99]. The website Epinions.com, for example, pays people for writing reviews that other read and like; Amazon.com gives status to people who have written many book reviews. The TV Scout provides a special economic incentive model for early raters, the *opinion leader* concept. This concept identifies a relatively small subgroup of early raters that are given the task to provide the very first ratings and that are explicitly

rewarded in exchange. Other users of the system, also those planning relatively far ahead, profit from opinion leaders by already obtaining some basic prediction quality of their collaborative queries.

5.4.3 Opinion leaders serve as early raters

In many recommender systems, e.g. [HRF95, RIS+94, and SM95], all users have the same role; all users contribute the same types of work in return for the same type of benefit. If we assume that the effort that users are willing to spend on a recommendation system differs between individual users, this means to waste part of the potential. On the one hand, users that are not willing to spend the required effort will quit using the system, i.e. they will stop contributing anything at all. On the other hand, highly motivated users will only contribute as much as everybody else, although they would be willing to contribute more. To maximize the productivity of the system, i.e. to achieve optimum prediction quality, a system should challenge each individual user by offering multiple effort levels and associating them with corresponding incentives.

In the related work, there are countless examples of users spending an over-proportionally large effort for the benefit of the community. When analyzing newsgroup postings, Terveen found that a relatively small minority of people expends the effort of judging information and volunteering their opinions to others [THA+97]. People running moderated newsgroups, so-called *moderators*, spend a significant amount of work on filtering the postings to "their" newsgroup. In free software, such as the Linux operating system, individual developers spend a huge effort without receiving financial reward [Ray99]. Individual "editors" fight information overload in email conferences [Pal84]. In the Tapestry system, "eager" users provide annotations for the rest of the community [GNOT92].

Many of these individuals play the role that editors or critics play in traditional print media. Unlike "real" editors, they are usually self-proclaimed and receive no monetary reward for their work. What makes people moderate newsgroups, program free software, or post long listings of URL recommendations? The answer is that communities have something to give that single-user systems cannot give—*social reward*. In exchange for their work, the often self-proclaimed users who contribute the lion's share of work have outstanding positions in the communities built on their work; they have more influence and a higher social status [Ray99].

The *opinion leader* concept [Bau98c] is an algorithm for formally managing such editors. What distinguishes the opinion leader concept from the examples found in related work is that the opinion leader concept provides a formal selection mechanism. To maximize the community's profit, it seeks those users that are most appropriate for performing the task of providing the very first ratings and encourages *them* to take this role.

5.4.3.1 User interfaces for users and opinion leaders

Opinion leaders (OLs) are a group of users that make recommendations to the entire community of users. We will begin by looking at how opinion leaders are presented to the rest of the

user community. These recommendations are offered in the *user tips* menu (Figure 58, also shown in Figure 54c at Page 122) that contains one entry for each individual opinion leader. Entries are labeled with the respective opinion leader's name or, for privacy reasons, pseudonym.

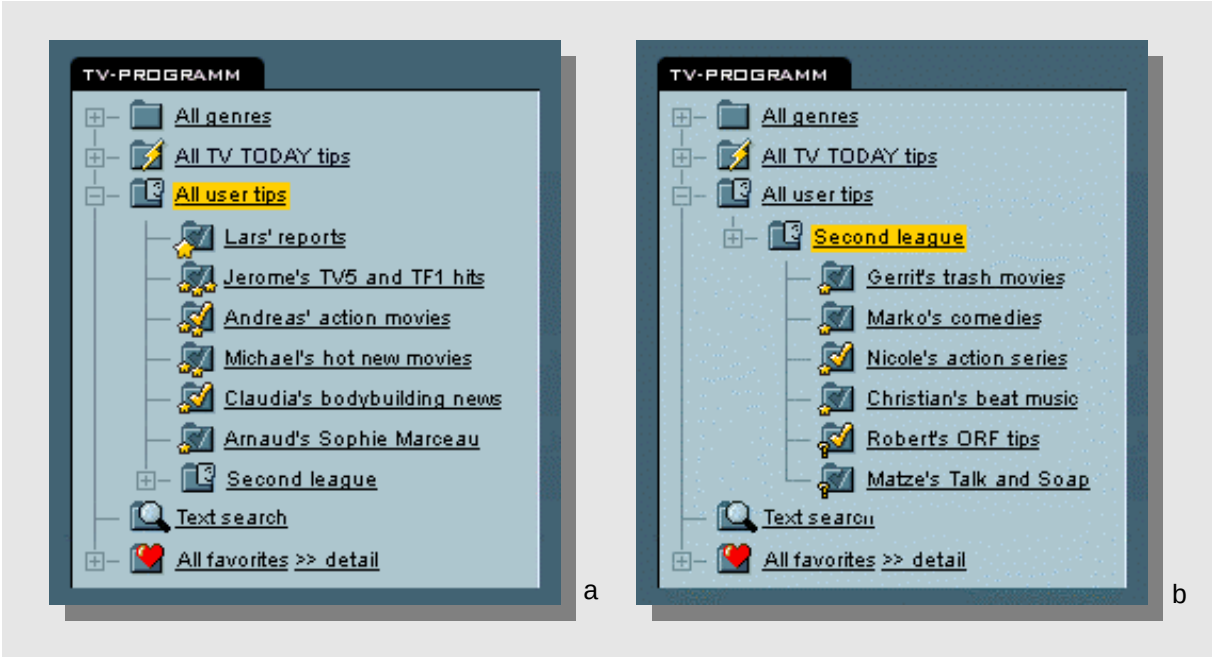


Figure 58: Ordered display of opinion leaders in the *user tips* menu and graphical visualization of their success.

Users operate the *user tips* menu similar to the genre menu. Users can pick entries from the *user tips* menu to obtain program descriptions within the current date/time/channel settings and users can include their favorite OLs in their QSA profiles. The difference compared to genres is that the selection and ranking of program descriptions obtained by clicking an entry of the *user tips* menu reflects the respective OL's *taste*, not a content-based criterion like a genre. The type of filtering implemented by the *user tips* menu is also referred to a *pull active collaborative filtering* (see Section 1.1.2.2.2).

Opinion leaders are provided with specialized user interfaces for entering recommendations. Since judging a program recommendable will often coincide with planning to watch that program (see Section 5.3.3 for a discussion about the differences), the TV Scout saves OLs a double effort by overloading the OL functionality over the retention tools. OLs select programs they want to recommend by selecting them for retention into a retention tool, the *laundry list*.

While the rest of the TV Scout user interface is kept unchanged, the laundry list is enhanced with the simple annotation capability "*I judge this program {excellent, very good, good, wait and see, just for me} because [textbox]*" (Figure 52 on Page 120, bottom right). This annotation serves four purposes. 1. "Just for me" excludes the respective program from public visibility and thereby allows OLs to distinguish between retention for their own use and public

recommendation. 2. “Wait and see” allows OLs to explain that they have not previously seen this program and base their recommendation only on the program’s description. This is the default setting. 3. Ratings impose a ranking on recommended programs and thereby support a weakly ranked output. 4. The textbox gives OLs a limited means of self-expression. OL ratings and annotations are incorporated into the corresponding program descriptions and are publicly visible in the system.

To allow opinion leaders to provide the very first ratings, the TV Scout provides OLs with TV program descriptions *before* they become accessible to all other users. While normal users have access only to a one-week program, opinion leaders are already given access to the program descriptions of the following week.

5.4.3.2 Selection of candidates

As mentioned above, the main idea of the opinion leader concept is to make those users become opinion leaders that will bring the biggest benefit to the community. OL candidates are evaluated according to the following criteria.

1. *Represent the community’s taste:* One primary function of OLs is to provide the very first ratings to startup collaborative predictions. Since there are too few OLs to allow the following users to receive personalized ACF predictions, the OLs should be a representative sample of the community’s taste, so that predicted popularity based on that limited sample has an optimum retrieval quality. To contribute to predicted popularity, OLs *together* should cover the most popular programs. New OL candidates should therefore complement existing OLs towards predicted popularity. When aggregating OL ratings, the system may assign weights proportional to the number of users each OL represents.
2. *Serve as useful queries:* The other primary function of OLs is to implement the entries of the *user tips* menu. Here every OL counts individually. Like any other query, the *user tips* queries are only useful if they bring a benefit to the system’s users. Candidates have to have a maximum overlap with a large number of other users, to serve as useful queries for them. Since the more accurate algorithms are computational expensive, the current TV Scout version uses the following approximation algorithm [Koe98]. The utility of each OL candidate is computed as the quotient of the correspondence between the candidate and all other users in the system (based on implicit feedback, aggregation using a modified Dice coefficient, e.g. [Wil88, p. 578-579]) and the candidate’s correspondence with all other queries in the system (candidates must not be redundant with any other query in the system).
3. *Reliability:* Candidates are expected to be accustomed to performing the rating behavior that they will have to perform as OLs, i.e. they should use the retention tools on a regular basis and plan ahead as far as possible. These aspects can be assessed based on the user’s past usage of exact match menu and retention tools. It is important that OLs do not quit using the system, because if they do, all users having retained this OL in their QSA profiles lose their effort. Therefore, only users that have used the system for a certain while can become candidates. Last not least, candidates have to be willing to

do it. Since OLs lose part of their privacy, the system has to ask candidates for their agreement before making them OLs.

5.4.3.3 Creation and refinement of opinion leader boards

When the opinion leader query class is launched, the set of all OLs that we will call the *opinion leader board* is initialized with an arbitrary number of the best available candidates. From now on, the board is continuously optimized. Unlike in the initialization phase, OLs can be assessed based on their actual success. Success can be measured as the number of programs that users decide to retain based on a recommendation from an OL[33], as the number of users retaining the OL in their QSA profiles ("subscribers" [MGT+87]), or a combination of both. Based on this assessment, the performance of the opinion leader board is optimized using the following natural selection mechanism that is very similar to the management of sports leagues.

1. Opinion leaders are rewarded according to their performance. Social reward is given by publicly displaying the OLs' performance, e.g. by displaying the OLs entries in the *user tips* menu as shown in Figure 58. Note that displaying the performance of the OLs does not benefit users seeking recommendation, because this performance is not personalized; displaying them is important because it is a reward *for the OL*. If there are more OLs than the user interface is able to display in a single list, OLs may be displayed as a nested tree structure; tree levels then correspond to individual *leagues* (Figure 58b). If available, also monetary rewards may be given (e.g. [AZ97]).
2. OLs whose performance has dropped below a given threshold lose their OL status and are removed from the board[34].
3. New candidates are added to the board if their expected performance lies above the threshold. To allow new OLs to be found by their potential followers, they are given a certain period of time before they are evaluated for the first time. During that period, their unevaluated performance is indicated by a question mark icon (Figure 58b, last two OLs). To shorten this period, the system may introduce new OLs by actively suggesting them to users for whom the respective OL may be useful.

The number of OLs is limited by what can be effectively handled by the user interface. While the *user tips* menu may be deeply nested to hold more OLs, this may not be the best solution for the following two reasons. First, trying out OLs is a process of trial and error that users may not be willing to spend too much time on. Second, the social reward for each individual OL is reduced the more OLs there are. Since the overall amount of monetary and also social reward is typically limited (not everybody can be "outstanding"—in a user interface), more

[33] To not overestimate OLs that are queried because of their reputation or position in the user interface, we used a weighted difference between how often recommended programs from that OL were retained (benefit to the community) and the number of program recommendations that all individual users have received from this OL (since (unsuccessful) recommendations cause users reading effort, the latter is considered costs to the community).

[34] This selection scheme can, of course, also be applied to any other query class in the system. Queries that perform poorly, e.g. genres that are never used may be removed to simplify the user interface.

OLs generally mean less reward per person, especially for those at the lower end of the board. For OLs, there may not be much incentive in finding their entries buried deeply in the menu where hardly any user will find it. The size of the opinion leader board will therefore usually be limited.

5.4.3.4 Personalized opinion leader boards

The system described above is the current implementation status of the TV Scout. As a next step, the opinion leader concept may be generalized by allowing potentially every user to become an OL *for some local community of followers*. The formal requirements for becoming such a local OL are similar to those of global OLs. Candidates have to be early raters, must have used the system for a certain amount of time and must have provided implicit ratings during that time, and have to be willing to share their implicit ratings with other users. The main difference to a global OL board is that expected prediction quality is computed with respect to the respective user instead of the entire community. Beside that, local OLs may use the same user interfaces as global OLs, i.e. the annotation interface, and can be given access to the second week of program descriptions.

Unlike, for example, genres or text search, the possibilities for labeling OLs are of very limited expressiveness (see for example Figure 58). Manually searching them is therefore highly inefficient; it basically means to try them out. While this effort may be feasible for the limited set of the global OL board, sifting through larger numbers of users is not appropriate. Therefore, the system always has to make a pre-selection of possible candidates.

This perception has an impact on the user interface for personalized OLs. It would be possible to replace the global OLs incrementally with the local OLs, while the system receives more relevance feedback. Doing that would mean that the content of the *user tips* menu would become variable, so users would not find an OL in the *user tips* menu that was still there the session before. This, however, would not match the philosophy of the TV Scout interface, as we have applied it so far. Query submenus are stable to allow users to search manually. This allows users to search for things the system cannot suggest, e.g. due to the lack of relevance feedback indicating this interest.

Since the local OLs that are of potential interest for the user are selected entirely by the system and not by the user, local OL are better handled as part of the query suggestions mechanism—completely analogue to the suggestion of genres, text searches, and user tips. Since the large set of OLs will allow making suggestions very often, replacing the interrupting query suggestion that was shown in Figure 54 with a permanently visible *suggestion* submenu (Figure 59) might be more convenient for the user. The example shown in Figure 59 shows how menu labels can be used to indicate why the respective local OL was suggested. If the OL has a self-selected label, this label may be shown (“Claudia’s bodybuilding news”). Otherwise, without violating the OLs anonymity, the OL may be referred to by naming an overlapping program (“User liking Magnum P.I.”)

By personalizing the *user tips* menu, social reward can now be given to a much larger group of local OLs. To provide a reward for local OLs (and an incentive to provide early ratings), OLs have to be informed about their success. Figure 59b shows one possibility using a modi-

fied *laundry list* icon that shows a number of stars. Furthermore, the systems may give local OLs reports about their current performance, such as the number of their followers and the number of "recommended" programs that their followers decided to retain.

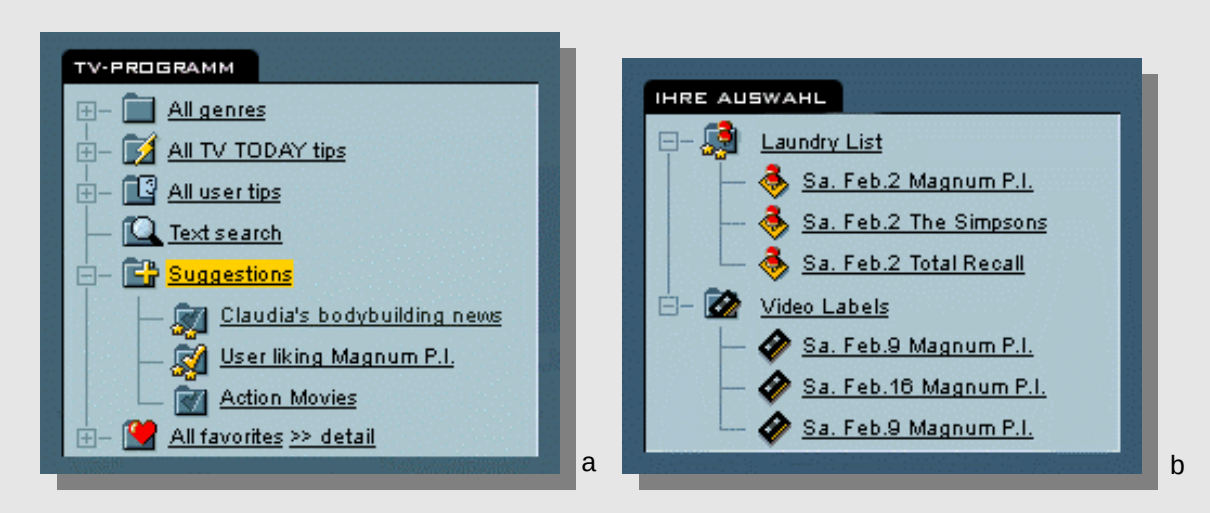


Figure 59: Suggesting an anonymous user (a). The laundry list of the local OL with stars (b).

The generalized opinion leader concept basically means to give users access to their neighborhoods (as used in ACF), such that they can manually select who they want to be in their "neighborhoods" and who not. Although there are essential differences in the selection of the suggested group of users and the neighborhood used in ACF (see Section 3.2.3) the generalized OL basically is a modified version of ACF where *automation* has been replaced by *suggestion*, resulting in a form of *pull active* collaborative filtering. Another distinguishing feature is that local OLs are manually selected by their followers and that local OLs are noticed as individuals. It would also be possible to tell users of an ACF system in how many users' neighborhood they are and how many predictions they have influenced in the past, but since nobody but the user him or herself would take notice of that fact, it would not mean any social reward.

5.4.3.5 Comparison with objectives

The opinion leader query class tries to achieve two objectives. The first objective is to reward users for providing early ratings. Early ratings are required to ascertain the prediction quality of predicted popularity and ACF for all other users. In the currently implemented version, a small selection of users is employed to provide the very first ratings. In the personalized version, the distinction between OLs and normal users is relaxed and all early raters can receive social reward for their contributions. This reward is intended to reimburse early users for the reduced prediction quality they receive.

It is clear that this type of reward is not an incentive for everybody. Role specialization in other existing systems shows that some people are tempted by social reward while other peo-

ple are not attracted by this type of incentive or may even perceive public visibility as unwelcome. The fact that users are different is fortunate though, because the system not only needs leaders but also followers[35]. The *system* cannot give social reward to producers—only followers can do that. The producer-consumer or leader-follower relation is a trading relation. Leaders provide an effort and trade it in for whatever it is that followers give them.

The second objective of the OL concept is to provide users with improved filtering performance by providing them with an additional query class, the *user tips*. Being based on the taste of users, user tips provide a type of recommendation that content-based queries cannot provide, because this recommendation may be based on properties that are not part of a program's description.

In the TV Scout, the opinion leader board displayed as *user tips* is complemented by a second board of professional editors. This query class is called *TV TODAY tips* (Figure 54b) and contains six queries labeled *movies*, *sports*, etc. that provide one recommendation per day provided by the editors of TV TODAY. These recommendations are imported from the printed TV program guide published by TV TODAY. Although it would be possible to integrate the *editor's tips* into the *user tips* menu, they were given their own top-level menu. The reason is that users may already know the TV TODAY editors and their classification scheme from the corresponding print media. While users have to spend some effort in finding their preferred opinion leader, editor's tips are a good means for new users to rapidly find recommendations by category.

5.5 SUMMARY AND DISCUSSION

Figure 60 summarizes the filtering functionality of the TV Scout filtering system. Block arrows show how TV program descriptions are filtered. When users execute their QSA profiles, each of the contained queries is executed on its respective subsystem, e.g. the FreeWAIS retrieval system. All subsystems receive their data from the program description database. Optionally, program descriptions may be enhanced with information from the movie database. The QSA queries rank program descriptions as described in Chapter 3 and remove program descriptions with output ratings below the cut-off value. Before program descriptions are delivered to the user, they are filtered according to the exact match settings. Alternatively to executing their QSA profiles, users may choose to execute a query as an *ad hoc query*, which bypasses the profile and directly executes the query.

Dashed lines show how relevance feedback is gathered. Users may select programs they plan to watch from the retrieved listings and retain them using the retention tools. The content of the retention tools is considered positive implicit feedback (Section 5.3) and is used to automatically improve the user's QSA profile as described in Chapter 3. The same input is also

[35] Of course users can simultaneously be producers and consumers. For example, seven groups of TV Scout users that all plan their TV consumption once a week, but on different weekdays, receive recommendations from each other. All these users are pure early raters with respect to the program descriptions inserted at "their" day, pure consumers with respect to the program descriptions broadcast at that day, and mixtures of both for all other program descriptions.

fed back into the system to serve as input to those query classes that are based on various types of collaborative filtering (Section 5.4).

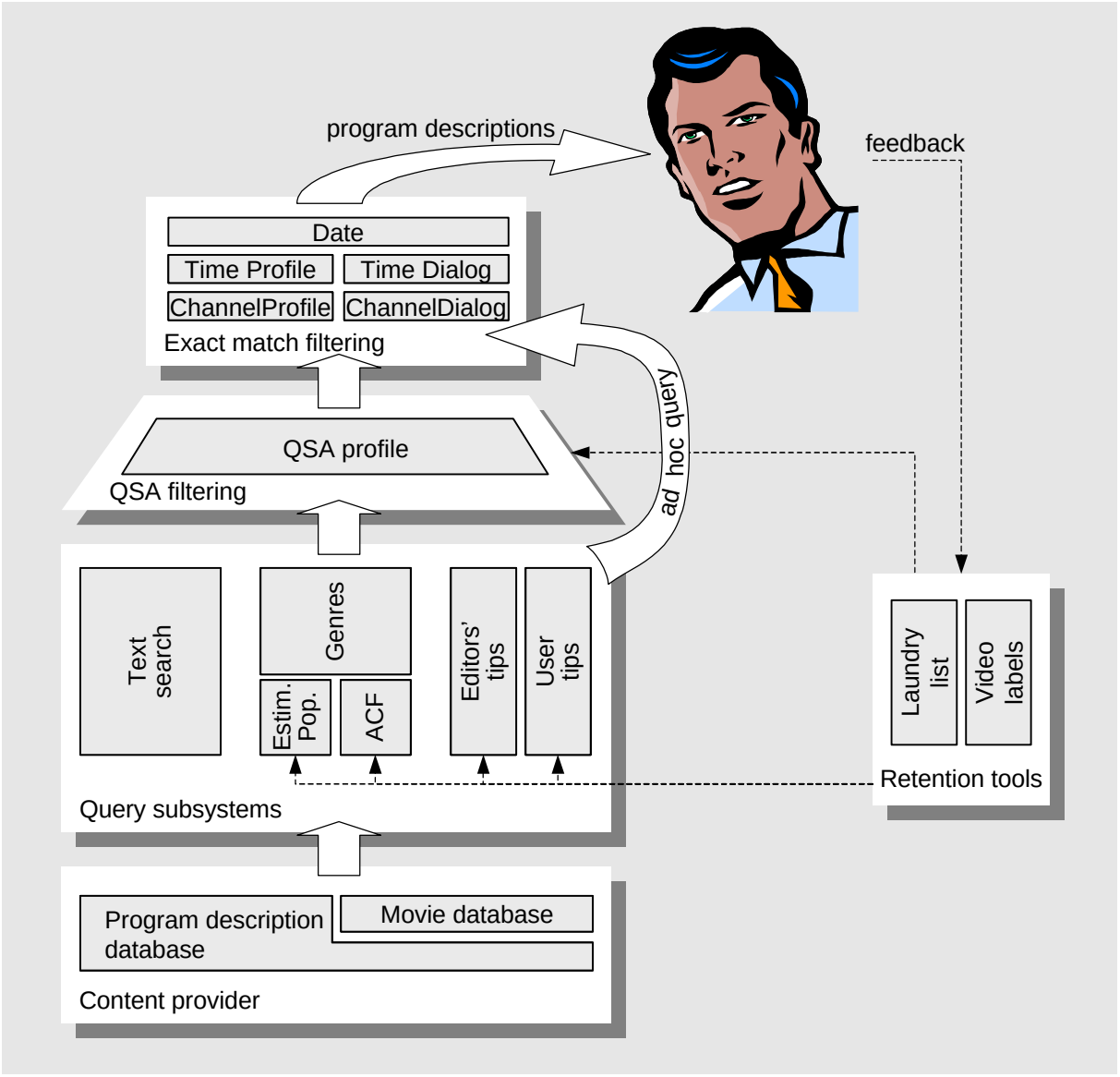


Figure 60: Overview: flow of program descriptions and ratings (block arrows) and generation of user feedback (dashed arrows).

5.5.1 Discussion of query classes

The TV Scout offers the four query classes *text search*, *genres*[36], *user tips*, and *editor's tips* to the user (see Section 5.4). Table 13 summarizes the technical aspects. The individual query classes use different content-based and collaborative filtering techniques and different indexing models. Since each query class uses its individual method for computing ratings, no ratings of any pair of query classes are compatible. Since a user's QSA profile can simultaneously hold the user's *favorite text searches*, *favorite genres*, *favorite opinion leaders*, and *favorite editor's tips*, query ratings have to be pre-converted as described in Section 3.3.4.

		Filtering technique	Ratings computed as	Range of query ratings
Text search		content-based, full text indexing	term frequency, TF-IDF	[0,1]
Genres		content-based, manual indexing	(see the two cells below)	(see the two cells below)
	Rank by popularity	autom. collaborative, non-personalized	how often retained	[0,1]
	Rank by ACF	autom. collaborative, personalized	how often retained in neighborhood	[0,1]
Opinion leaders	User tips	pull active collaborative filtering	opinion leader rating	{excellent, very good, good, wait and see}
	Editor's tips	pull active collaborative filtering	TV TODAY editor rating	{0,1,2,3} red bullets

Table 13: Summary of the individual query classes

Table 14 compares strengths and weaknesses of the individual query classes. It compares the appropriateness of the individual query classes for different tasks and for different situations. Depending on task and situation, different query classes provide the best support for finding the right programs (highlighted cells).

The content-based query classes (*text search* and *genres*) provide efficient means for finding programs if users already know what they are looking for and if they are able to describe it (dashed area in the top left corner). *Text search* is the first choice if what is sought can be identified by text contained in the title (find known program) or description text (find information on topic). Text search is the most efficient solution for finding, for example, the user's favorite shows or series. The potential problem that text search requires users to learn query syntax, is not too relevant here; since the TV Scout is a web-based system, many of its users already have pre-experience with the simple "bag of keywords" syntax used by FreeWAIS, because it is also used by many web search engines. The manual classification of programs

[36] Invisible to the user, genres are complemented with *predicted popularity* or *automated collaborative filtering* to impose a ranking on the retrieved programs.

into *genres* provides a fast access especially if specific types of entertainment, such as *action movies*, are sought. Being combined with ACF/predicted popularity, the ranking within genres helps users to rapidly find the most promising programs.

Collaborative filtering query classes (*predicted popularity*, *ACF*, *user tips*, and *editor's tips*, dashed area in the bottom right corner) provide better support for unspecified interests, such as pastime. Unlike content-based techniques, collaborative filtering queries help users finding information that users may have difficulties to describe abstractly.

		Text search	Genres			Opinion leaders	
				Popularity	ACF	User tips	Editor tips
Task	Find known program, e.g. *The Matrix*	++	o	--	--	-[1]	-[1]
	Find information on topic, e.g. *Clinton*	++	+	--	--	--	--
	Find specific entertainment, e.g. *Action*	o[2]	++	--	--	O[3]	-
	Find any program user will like	-	o	o[5]	++[7]	+[4]	o[5]
	Find any liked program broadcast now, for pastime (provides high coverage)	--[6]	-[6]	o[5]	++[7]	--[6]	--[6]
Situation	Appropriate for inexperienced users (Ease of manually finding right query)	+[8]	++	[9]	[9]	O[10]	++
	Works when retention tool is empty (Prediction quality during cold-start)	++	++	++	--[11]	++	++
	Works when system has few users (Prediction q. while not critical mass)	++	++	-	--	O[12]	++
	Works for early raters (long-time planners/opinion leaders)	++	++	-	--	+/-[13]	+[13]
	Efficiently identifiable as outdated after interest change (specificity & naming)	++	++	o[14]	--	O[15]	++

Table 14: Comparison of the individual query classes with respect to different tasks and situations. Possible ratings best to worst: ++, +, o, -, --. Annotations: [1] OLs may simplify search by labeling themselves accordingly, e.g. with a genre-derived name; Editor's tips have some genre structure. [2] Depends on whether genres are also text indexed. [3] OL may be known to specialize on that. [4] If the favorite OLs have been found already. [5] Not personalizable, depends on how "main stream" the user's taste is. [6] Limited coverage, can rate only some objects in given time interval. [7] Depends very much on cold-start and critical mass. [8] Users have to learn syntax. [9] Predicted popularity and ACF have no own user interface and are accessed via genres. [10] Labeling of OLs is of limited expressiveness; user may have to try them out. [11] Falls back to popularity. [12] Few OL candidates, therefore also candidates with low utility have to be accepted. [13] OLs are not supposed to copy recommendations from other OLs or editors. [14] Is only subject to interest changes in the rare case that user's taste becomes more or less "main stream". [15] Interests represented by an OL are not obvious.

If a user has provided a sufficient number of ratings, if these ratings are up to date, and if other users have provided a sufficient number of ratings, ACF is first choice. ACF combines several desirable properties by taking a large number of users into account while simultaneously being personalized. While selecting *all genres* allows users to receive predictions only based on ACF, the combination with genres allows users to restrict the ACF predictions to content-based categories. This allows "guiding" ACF in periods when the user's interests change.

If the user has not provided a sufficient number of implicit ratings, ACF takes *all* users into account and falls back into *predicted popularity*. The same effect occurs if the user's neighborhood has not provided enough ratings, e.g. when predictions for a given set of programs are requested. By taking *all* users into account, predicted popularity provides a better coverage than the more or less specific queries from other query classes that will often result in no matches if applied to a small date/time/channel interval (zero hit queries)[37].

Opinion leaders provide users with full control over the sources of their collaborative recommendations. Unlike ACF, where the system determines who is a good predictor for the user, opinion leaders allow users to judge who they want to receive their recommendations from. This allows users to update their "neighborhood" when interest changes occur. It also allows individual OLs to be queried individually as ad hoc queries. The price for the additional flexibility is that users have to invest more work in finding out which OLs match their tastes best.

Its dynamic allocation and refinement of OLs allows the *user tips* menu to follow trends and represents what the community is currently interested in, unlike the *editor's tips*. The main advantage of *editor's tips*, on the other hand, is that they come with a predefined category structure, such as a *movie tip* and a *sports tip*. This makes them easier to identify than user tips, especially for new users. Furthermore, users may already know these editors' tips from the printed TV program guide and may have a higher confidence in these recommendations, if they have agreed with them in the past.

5.5.2 TV Scout usage data

The TV Scout system has been publicly available at http://www.tvscout.tvtoday.de since October 16, 1998. At April 20, 2000, we conducted a data analysis based on the usage data gathered so far.

We had two objectives in this analysis. The first objective was to look at the design of the query classes, i.e. whether the query functionality provided by the TV Scout query classes would be appropriate. Would all query classes be used by a substantial number of users? Do the query classes complement each other, i.e. would users use them for different purposes (e.g. do text searches cover different interests than genres)? Or would we find the provided query classes to be redundant, so that the system could be tuned by removing one or more of them?

The second objective was to check in how far the conception of QSA systems to simultaneously serve as IR and IF systems would work (see Section 3.4.1). Would all three stages that

[37] Another interesting aspect of predicted popularity is that it may help users to share TV experiences with their real-world communities as a basis for social interaction; a property of TV partially lost with the increasing number of channels.

QSA system users can be in (retrieval, bookmarking, and filtering) be adopted by a substantial number of users or would users settle earlier, e.g. for retrieval functionality? How many users would make the step from retrieval to bookmarking; how many would make the step to creating and using personal QSA profiles?

The data analysis was carried out based on the 18 months worth of usage data that the system had gathered until April 20, 2000. All usage data was extracted from the web server log files and the system's database and reflects the system's usage by all 10676 users that had used the system until that point in time.

Being only based on log file data, our study was subject to a number of limitations. To answer any questions involving user satisfaction (e.g. which query classes users *perceived* as most useful), future experiments should complement the data presented here with data obtained through experiments, e.g. the direct assessment of the functionality by subjects via questionnaires, thinking aloud, etc. Another important aspect of the TV Scout and QSA systems that was not tackled by this study is the appropriateness of the QuerySet Architecture for handling interest changes. While the presented data analysis looks only at the usage frequency of the individual features of the system, a study on filtering and interest changes requires taking the *prediction quality* provided by QSA systems into account. Since changing user interests may be difficult to simulate in a lab setting, such a study would best be carried out as a long-term field study.

5.5.2.1 Logins

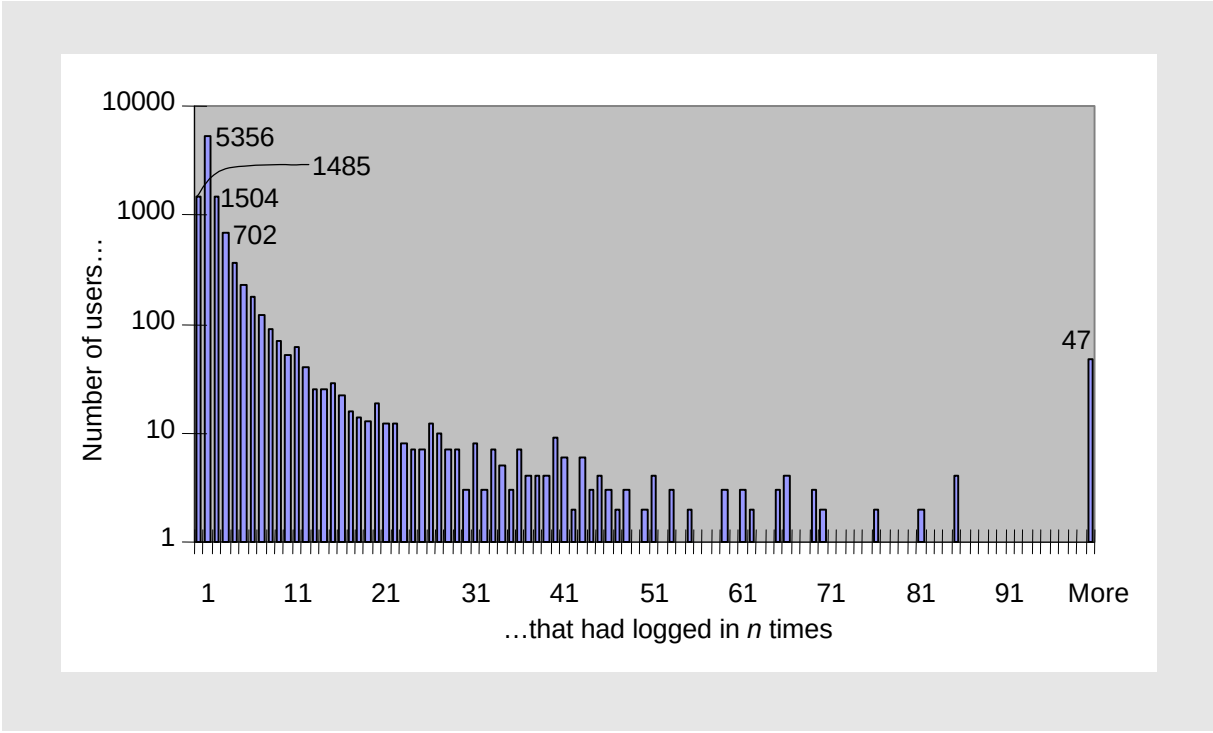


Figure 61: Login frequency of users

The TV Scout requires users to register before their first usage. Registration requires users to give themselves a username and password and then provides users with immediate access to the TV Scout system. At July 28th 2000, the system had 10676 subscribed users, of which 9190 users had logged in at least twice. Figure 61 shows how the number of logins was distributed across users. Forty-seven users logged in more than one hundred times; the most active user logged in almost daily (580 times).

5.5.2.2 Ad hoc queries

Together, users executed 48,956 queries. 53% of all queries (25,736 queries) were specific queries, i.e. different from "all genres", which is the default when the TV Scout comes up.

Genre queries: Genres were available from the first online day of the TV Scout. Figure 62 shows how the individual genre queries executed until April 2000 were distributed across the individual genres. Out of the 195 predefined genres offered by the system, 120 were used at least once. Beside the default *All genres* (executed 23220 times), the genres *Movies* (6495), *Series* (2805), *Erotic* (1656), *Sports* (843), and *Information* (721) were especially popular.

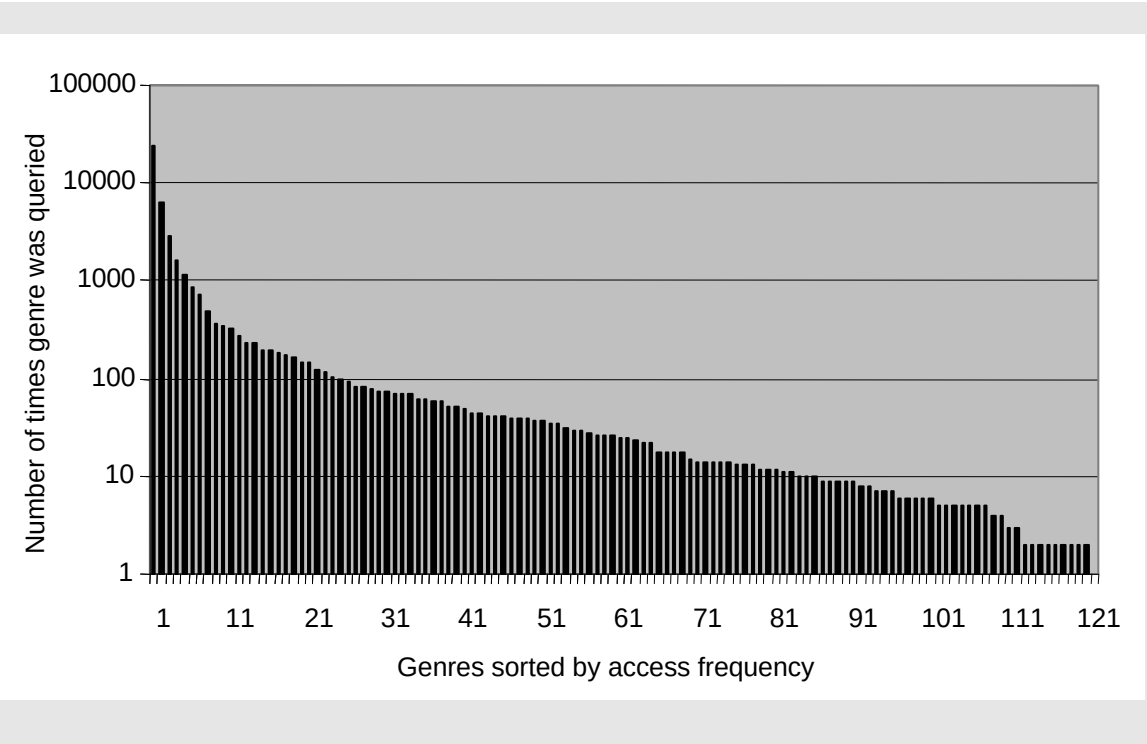


Figure 62: Distribution of queries over the individual genres.

User tips & editor's tips: When we conducted our study on TV Scout usage, the Opinion leader board *user tips* had been online for only a short period of time so that no sufficient amount of runtime data was available. The collaborative query *User top 10*, that we had made available before, gives a first impression of the potential of queries based on collaborative

properties. This query that returned the ten most popular programs of each day, was executed 1153 times, making it the 4th most popular query in the system.

At any time, the provided retention feedback given was sufficient to supply the *User top 10* query. Since the number of non-empty laundry lists however never grew above eighty, the coverage was not sufficient for supplying more elaborated recommendation features, such as ACF. One likely explanation is that the TV Scout user interface never actively pointed users to the retention tools. Either users discovered this feature on their own (by trying out the toggle switches next to the program descriptions in the program listings and tables, shown at the bottom of Figure 52) or this feature remained unused. Future versions of the TV Scout may increase retention tool usage by actively pointing users to it; also a combination with explicit ratings may be considered.

Text searches: Text searches were made available about a year after the TV Scout was brought online. To make the resulting usage data comparable to the genre data, we compared them on per week-basis. In the week April 14-20, 2000, 132 text searches were executed, which was 13.7% of the amount of all specific genre queries during this period (960 queries). The differences in usage frequencies may reflect the fact that text searches required users to manually enter a query string while genres are parameter less and can therefore be handled more efficiently.

As we had expected, text searches were used to complement genre queries, i.e. to search for all those things not covered by genres, i.e. series names (e.g. the German comedy shows "Bully Parade" and "TV Total"), search on a subject ("stars, planets, astronomy, earth"), "cooking food wine beer spice recipe"), concrete programs (e.g. the finals of the European Soccer Championship "em finale"), and actors (e.g. "Costner").

5.5.2.3 Bookmarks and QSA profiles

Bookmarks: 1770 users had bookmarked at least one query. Together, these users had bookmarked 4383 queries, mostly genres. The bar chart in Figure 63 shows how the bookmarked queries were distributed across users. The two most active users had bookmarked over 60 queries each.

The histogram in Figure 64 shows how bookmarking was distributed across queries. Beside the most frequently executed genre queries (*Movies* (bookmarked 736 times), *Information* (364 times), *Erotic* (350 times), *Series* (289 times), *Comedy* (297 times), *Culture* (267 times), *Entertainment* (266 times), *Action&Thrill* (266 times), and *Science* (256 times)), also *TV TODAY Movie tips* (369 times) were among the top-ranked bookmarks. The query *User top 10* was bookmarked 66 times. 395 text searches were bookmarked.

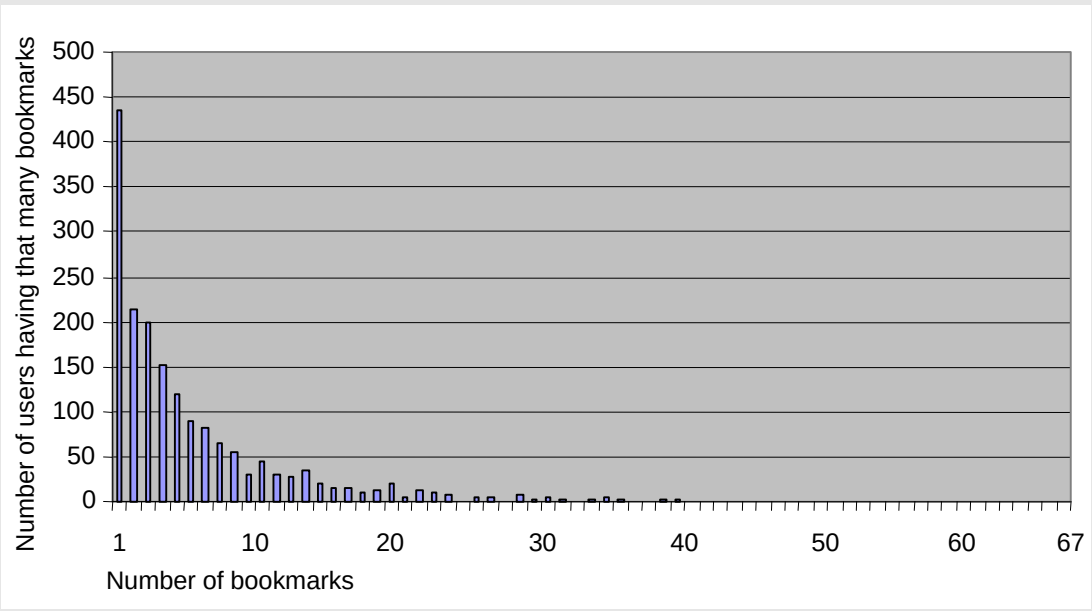


Figure 63: Number of users (x-axis) having how many queries in their QSA profiles (y-axis).

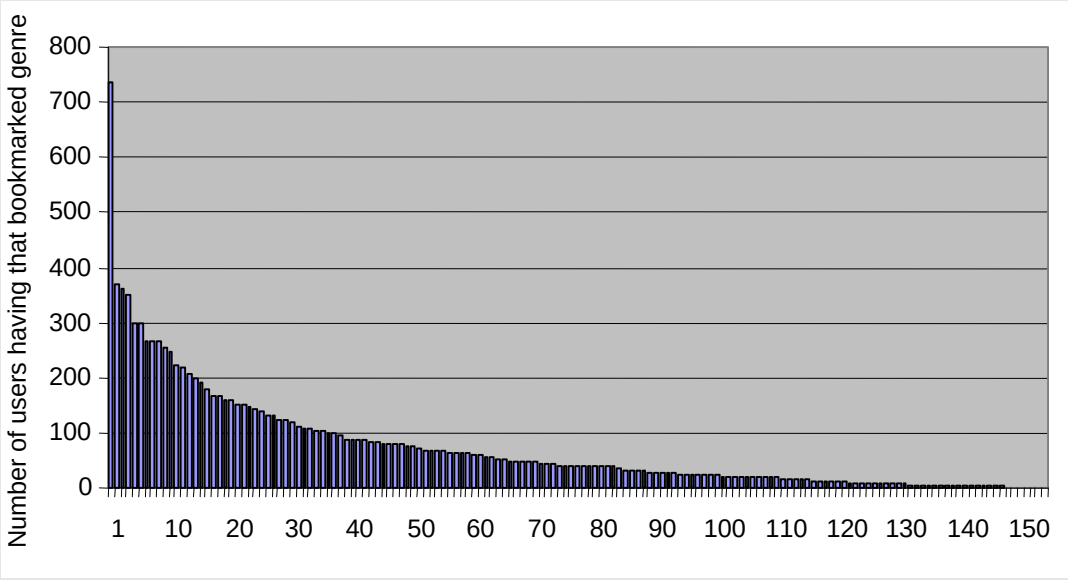


Figure 64: Number of times (y-axis) the individual queries (x-axis) were bookmarked by users.

Profile mode: Out of the 1770 users who had bookmarked at least one query, 270 users (about 15%) at least once executed their QSA profile as a whole, i.e. such that the QSA pro-

file query would return a combination of the results of all bookmarked queries. These users executed their QSA profiles together 5851 times, which is an average of 21 times per user. QSA profile users used the form-based profile editor to fine-tune their profiles 1213 times altogether; an average of 4.5 times per user. The advanced histogram-based editor was only accessible through the form-based editor and was therefore used substantially less often (280 times, an average of one time per user).

5.5.2.4 Discussion

Concluding, we will briefly discuss the results presented above and compare them with the two objectives of the study.

The first objective was to compare the individual query classes. The query class used most frequently so far were genres, which is most likely explained (1) by the pre-experience with genres that many users have from printed TV program guides and (2) by the convenient parameter-less usage of genre queries. Keyword search confirmed its usefulness by proving that it covers interests that genres cannot cover (e.g. series names, search on a topic, concrete programs, and actors). The opinion leader board had been active for too shortly to provide concrete quantitative results, but another collaborative query, collaborative query *User top 10* reached rank four among the most popular queries. Consequently, our hypothesis that the query classes provided so far complement each other was supported by the found results.

Our second objective was to investigate whether a substantial percentage of users would actually complete three usage stages and reach the filtering stage. The usage data presented above provides some evidence for the claim that the equation "*QSA = IR + IF*" worked in the case of the TV Scout. While all users used the ad hoc querying functionality provided by the TV Scout, 1770 users had bookmarked one or more queries. Out of these, 15% proceeded to the filtering stage, i.e. they executed their QSA profile at least once as a whole. The heavy use of the QSA profile (QSA profiles were executed 5851 times, i.e. users who had reached the filtering stage executed the QSA profile more often than any other query) shows that QSA profiles were highly appreciated by their users.

So what is about all the users who did not make use of bookmarks and QSA profiles? As discussed in Section 3.4.1, the fact that many users did not reach the filtering stage does not mean that these users did not reach the "goal" of QSA systems. Different users do have different strategies for planning their TV consumption. If we look for example at the different periods of time that people plan ahead when planning their TV consumption (Figure 57 in Section 5.4.2.4), we see that only a small percentage (12%) of all subjects in our survey on TV consumption plan ahead for longer than a day. The information filtering functionality provided by the TV Scout aims at supporting this relatively small subgroup of users—the larger number of users who did not make use of filtering functionality may not be interested in long-term planning—but may still have found the retrieval functionality of the TV Scout to be the appropriate support for *their* planning strategy.

CHAPTER 6 CONCLUSIONS AND FUTURE WORK

Live and don't learn – that's us.
[Bill Watterson: Calvin & Hobbes,
scientific progress goes "Boink"]

We conclude this work by presenting a short summary of the achievements of this thesis, discussing potential application areas, and describing open issues and directions of future research.

6.1 ACHIEVEMENTS OF THE DISSERTATION

The main achievement of this dissertation is the development of an information filtering architecture that not only tracks user interests based on relevance feedback, but also allows users to manipulate their profiles manually using an interaction that is derived from relevance feedback (user-aggregated relevance feedback, see Section 3.3.6). Because of these two distinct access levels to user profiles, *QuerySet Architecture* (QSA) systems can handle interest changes with different temporal behavior. Gradual interest changes, i.e. interest changes caused by a process, can be tracked based on relevance feedback. Abrupt interest changes, i.e. interest changes caused by events, as well as profile initialization, can be handled manually using user-aggregated relevance feedback. The high-level profile structure underlying the QuerySet Architecture is also beneficial for *reusing* profile states, which provides improved support for repetitive interest changes.

QSA systems facilitate the manual updating of QSA profiles by providing specialized user interfaces. *Histogram-based interfaces* are especially useful for profile updating operations involving a single query, e.g. the initialization of a query newly inserted into a QSA profile. *Paintable interfaces* are highly efficient if many queries are to be updated simultaneously, e.g. when reflecting mood changes in the user profile.

An important aspect of QSA systems is their support for incremental profile creation. Since QSA systems initially present themselves as retrieval systems to the user, queries can be formulated and tested individually before they are inserted into the QSA profile. New users therefore find themselves in an IR, *not* in an IF situation. When users have "bookmarked" at

least two queries, the filtering side of QSA systems introduced itself by compiling these bookmarked queries into a single ranking. This system design allows users to start using QSA systems in a non-planning, prototypical fashion that is only directed towards *immediate* benefit; long-term planning always remains optional. Furthermore, it allows users to go back and forth between retrieval and filtering mode, so that QSA systems can serve as both an IR and an IF system at any time.

By the example of the TV program recommendation system *TV Scout*, we demonstrated how both content-based and collaborative filtering techniques (e.g. the *opinion leader* concept that is based on active collaborative filtering) can be combined into a single QSA system and how QSA systems allow users to combine these filtering techniques in their user profile.

6.2 FURTHER RESEARCH TOPICS & OTHER APPLICATION AREAS

A number of issues are raised by this research, but could not be addressed in detail in this thesis. In the following, we discuss some of these issues and point out potential areas of future research based on this work. We see five principal directions for future research on QSA systems, i.e. (1) new query classes, (2) improved aggregation functions, (3) improved profile editor user interfaces, and (4) new application areas (5) empirical work.

(1) Although the presented framework of four query classes covers a wide range of interests (see Section 5.4), this framework is by no means comprehensive—not for the application area of TV programs and especially not for the QuerySet Architecture in general. One possible future research direction therefore is to enhance QSA systems by creating more query classes. Another approach would be to better exploit synergies between query classes by allowing users to formulate hybrid queries that combine expressions over multiple query classes in a single query, such as “User tips by *Lars* **and** *Action movie*”. See [Bau99b] for one possible concept supporting such queries.

A lot of work can be done on query classes that further develop the aspect of user communities. While users of the current TV Scout implementation communicate only with the system, future query classes may support users in directly communicating with each other. Related work shows that recommendations from known people can have qualities that recommendations from a “black box” do not have [ME95, GNOT92, HRF95, p. 196, KS98]. Local communities may be built based on common taste or on real-life relations. While local opinion leaders (Section 5.4.3.4) are a first step into this direction, communities may be provided with additional communication functionality, such as a means for forwarding recommendations to friends (*push* active collaborative filtering).

(2) The aggregation function forms the central component of the QuerySet Architecture. While the TV Scout uses the rather simple *weighted request and indexing retrieval model*, future QSA systems may explore different implementations of the aggregation function, e.g. the inference network-based solution sketched in Section 3.3.3.

(3) The user interfaces presented in this thesis explore new interaction techniques and thereby open a whole field of research. Future QSA user interface research may go into two directions.

On the one hand, new QSA profile editors may be developed, either by modifying the histogram or painting-based styles, or by exploring entirely different interaction concepts. On the other hand, the profile editing tools presented in Chapter 4 may be applied to different problem areas—within IF and outside of IF. Especially the paintable interfaces are highly generic, so that they apply to virtually all problems where users have to rate larger amounts of "things". We already sketched some applications of these interfaces in Section 4.4.6, but we continuously find more applications for them, ranging from user interfaces for palmtop computers to the programming of micro machine arrays. Additional user studies will help improving these designs.

(4) A lot of future work can be done in applying the QSA to other application areas. While our approach proved very successful in the TV application area, its generic structure makes the QuerySet Architecture applicable to a wide range of application areas. Since the QSA handles objects as encapsulated (object, rating) pairs, replacing the object type requires only adapting the query-execution subsystems. Replacing TV programs with news articles, for example, would allow reusing the entire TV Scout system as a news recommendation system, only the indexing mechanisms used by the content-based query classes and the exact match query menu would have to be adapted. Building QSA systems is especially simple for application areas that already come with retrieval functionality. The existing search functions can then be used to form the query-execution subsystems of the QSA. We have already shown this approach at the example of the Web search prototype *Histograms* (Section 4.2.7) that runs on top of a web search engine. This approach also allows upgrading any SDI system (e.g. the SIFT system) to a QSA system.

Many of the "classical" IF application areas, such as scientific publications or newsgroup postings are candidates for new QSA applications. Since the QSA is especially beneficial for "broad" application areas where users have many different interests, the recommendation of news and Web pages are may be the most promising candidates for becoming the next QSA applications. Another interesting application area for QSA systems is online auctions (e.g. http://www.ebay.com). If buyers are interested in multiple products, they may insert the corresponding queries in their QSA profile to monitor the respective offers to find the best products and the best deal. Queries in the profile could be used to buy many products over a long period of time (e.g. if the user is a collector) or may be removed from the profile when an appropriate object has been bought, so that these queries would serve as a kind of notifier. While online auctions allow reusing much of the work presented in this thesis, they also bring up new interesting questions. Unlike, say, TV programs, products offered in auctions usually cannot be sold to more than one buyer. Since this turns participating users into competitors, query classes based on collaborative filtering techniques may require modification. Can users "hunt" together and share the bounty?

Another interesting application area is portal sites that integrate multiple object types into a single web page. Such a system could provide users for example with the results of a search engine, several mailing lists and news groups, and a stock quote service by returning a single ranked output that merges objects of all these different data types.

(5) Finally, a substantial amount of empirical work can to be done on various aspects of the QuerySet Architecture. As already discussed in Section 5.5.2.4, an evaluation of filtering performance of QSA systems is an important next step. While the survey presented in Section 0

provided some evidence about how QSA systems combine IR and IF functionality, as well as about the selection of query classes, a comparison of the QuerySet Architecture with respect to its filtering performance is still outstanding. Since there is no reliable way of simulating interest changes in a lab study, a series of long-term studies would be required to quantify the benefit of QSA systems and to point out weaknesses and opportunities for improvement. Further experiments may help exploring the space of novel interaction techniques brought up by this thesis. User studies could help quantifying the benefits of the individual histogram- and painting-based interface styles and provide data that would allow further improving these interfaces.

APPENDIX A THE PROFILE EDITOR QUESTIONNAIRE

Einordnung in Testgruppe:
Ort:
Datum/Zeit:

1 Allgemeine Fragen

1.1 Geschlecht

- weiblich
- männlich

1.2 Alter

___ Jahre

1.3 Bildungsgrad

- Hauptschulabschluß
- Mittlere Reife
- Abitur
- Studium (oder gerade StudentIn)

1.4 Wie oft hast Du Umgang mit den folgenden Medien?

1.4.1 Fernsehen:

nie häufig
1 2 3 4 5 6 7 8 9

1.4.2 Computer:

nie häufig
1 2 3 4 5 6 7 8 9

1.5 Bisherige Erfahrung

Wie erfahren sind Sie im Umgang mit den folgenden Geräten, Systemen und Konzepten?
(Kreisen Sie bitte die entsprechenden Zahlen.) *NA= nicht anwendbar*

1.5.1 Maus/Trackball/Joystick:

Anfänger Experte
1 2 3 4 5 6 7 8 9 NA

1.5.2 Balkendiagramme (z. B. in Excel):

Anfänger Experte
1 2 3 4 5 6 7 8 9 NA

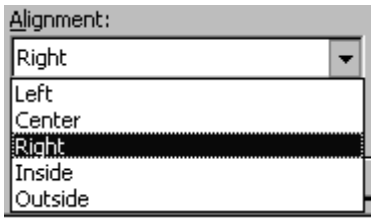
1.5.3 Internet:

Anfänger Experte
1 2 3 4 5 6 7 8 9 NA

1.5.4 Pulldown-Menüs:

Anfänger Experte
1 2 3 4 5 6 7 8 9 NA

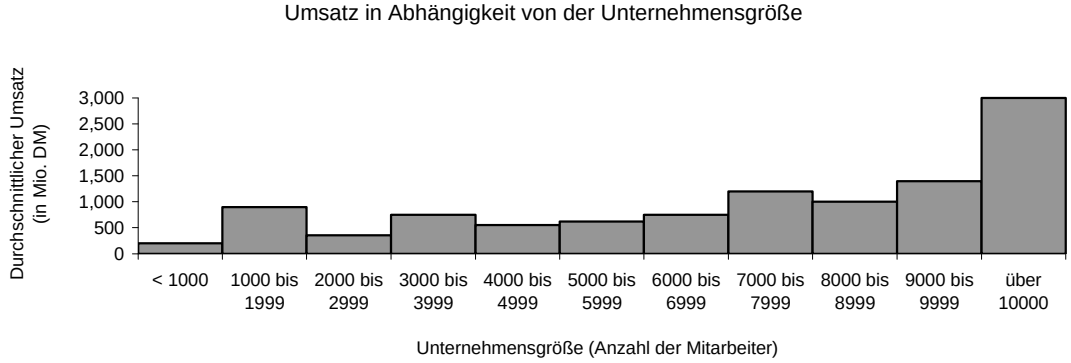
Beispiel:



1.5.5 Histogramme zur graphischen Manipulation:

Anfänger Experte
1 2 3 4 5 6 7 8 9 NA

Beispiel:[38]



[38] Das Histogramm wurde mit rein fiktiven Daten erstellt.

Sonstiges:

1.5.6 Verhalten Sie sich im Umgang mit neuen komplexen Eingabemöglichkeiten

zurückhaltend neugierig
1 2 3 4 5 6 7 8 9 NA

1.5.7 Die Zeit, die Sie von Woche zu Woche für die Auswertung des Fernsehprogramms aufwenden, ist

immer gleich sehr unterschiedlich
1 2 3 4 5 6 7 8 9 NA

1.5.8 Sie ändern Ihre Fernsehinteressen

nie häufig
1 2 3 4 5 6 7 8 9 NA

2 QUIS - Fragen nach Analyse der Formular-Schnittstelle

2.1 Formular: Allgemeine Benutzerreaktionen

Bitte kreisen Sie die Zahlen ein, die Ihre Eindrücke während der Benutzung am besten wiedergeben.

2.1.1		schrecklich	wundervoll
		1 2 3 4 5 6 7 8 9	NA
2.1.2		frustrierend	zufriedenstellend
		1 2 3 4 5 6 7 8 9	NA
2.1.3		schwierig	einfach
		1 2 3 4 5 6 7 8 9	NA
2.1.4		starr	flexibel
		1 2 3 4 5 6 7 8 9	NA

2.2 Formular: Erlernbarkeit und Nützlichkeit

2.2.1	Lernen, das System zu bedienen, war	schwierig	einfach
		1 2 3 4 5 6 7 8 9	NA
2.2.2	Die Erforschung der Systemeigenschaften durch Ausprobieren war	entmutigend	ermutigend
		1 2 3 4 5 6 7 8 9	NA
2.2.3	Ich konnte *das vorgegebene Profil* in diesem Formular abbilden	unzureichend	erschöpfend
		1 2 3 4 5 6 7 8 9	NA
2.2.4	Die Eingabe dieses Profils war	ineffizient	effizient
		1 2 3 4 5 6 7 8 9	NA
2.2.5	Ich könnte *meine eigenen Interessen* in diesem Formular abbilden	unzureichend	erschöpfend
		1 2 3 4 5 6 7 8 9	NA
2.2.6	Die angebotene Funktionalität ist	zu wenig	zuviel
		1 2 3 4 5 6 7 8 9	NA
2.2.7	Interessenänderungen einzugeben ist	ineffizient	effizient
		1 2 3 4 5 6 7 8 9	NA

2.2.8 Die Gesamtmenge von empfangenen Sendungsbeschreibungen zu kontrollieren, ist

unmöglich sehr gut möglich
1 2 3 4 5 6 7 8 9 NA

2.2.9 Die Anordnung der Information auf dem Bildschirm war

unübersichtlich übersichtlich
1 2 3 4 5 6 7 8 9 NA

2.2.10 Die von der Benutzerschnittstelle erlaubten Eingaben ermöglichen die Repräsentation der Interessen

unpräzise präzise
1 2 3 4 5 6 7 8 9 NA

3 Vergleich Formular – Histogramm-Darstellung

3.1 Verständnis

3.1.1 Wozu dient das horizontale Verschieben eines Histogramms?

- verstanden
- teilweise verstanden
- nicht verstanden

3.1.2 Wozu dient das Verformen eines Histogramms durch vertikale Mausbewegungen?

- verstanden
- teilweise verstanden
- nicht verstanden

3.1.3 Welche Bedeutung hat die Gesamtfläche der Histogramme?

- verstanden
- teilweise verstanden
- nicht verstanden

3.1.4 Wozu dient die senkrechte Linie?

- verstanden
- teilweise verstanden
- nicht verstanden

3.1.5 Welche Funktion hat der Farbverlauf der Histogramme?

- verstanden
- teilweise verstanden
- nicht verstanden

3.2 Histogrammbedienung: Allgemeine Benutzerreaktionen

Bitte kreisen Sie die Zahlen ein, die Ihre Eindrücke während der Benutzung am besten wiedergeben.

3.2.1		schrecklich	wundervoll
		1 2 3 4 5 6 7 8 9	NA
3.2.2		frustrierend	zufriedenstellend
		1 2 3 4 5 6 7 8 9	NA
3.2.3		schwierig	einfach
		1 2 3 4 5 6 7 8 9	NA
3.2.4		starr	flexibel
		1 2 3 4 5 6 7 8 9	NA

3.3 Histogrammbedienung: Erlernbarkeit und Nützlichkeit

3.3.1	Lernen, das System zu bedienen, war	schwierig	einfach
		1 2 3 4 5 6 7 8 9	NA
3.3.2	Die Erforschung der Systemeigenschaften durch Ausprobieren war	entmutigend	ermutigend
		1 2 3 4 5 6 7 8 9	NA
3.3.3	Ich konnte *das vorgegebene Profil* mit dieser Benutzeroberfläche abbilden	unzureichend	erschöpfend
		1 2 3 4 5 6 7 8 9	NA
3.3.4	Die Eingabe dieses Profils war	ineffizient	effizient
		1 2 3 4 5 6 7 8 9	NA
3.3.5	Ich könnte *meine eigenen Interessen* mit dieser Benutzeroberfläche abbilden	unzureichend	erschöpfend
		1 2 3 4 5 6 7 8 9	NA
3.3.6	Die angebotene Funktionalität ist	zu wenig	zuviel
		1 2 3 4 5 6 7 8 9	NA
3.3.7	Interessensänderungen einzugeben ist	ineffizient	effizient
		1 2 3 4 5 6 7 8 9	NA

3.3.8 Die Gesamtmenge von empfangenen Sendungsbeschreibungen zu kontrollieren, ist

unmöglich sehr gut möglich
1 2 3 4 5 6 7 8 9 NA

3.3.9 Die Anordnung der Information auf dem Bildschirm war

unübersichtlich übersichtlich
1 2 3 4 5 6 7 8 9 NA

3.3.10 Die von der Benutzerschnittstelle erlaubten Eingaben ermöglichen die Repräsentation der Interessen

unpräzise präzise
1 2 3 4 5 6 7 8 9 NA

3.4 Subjektiver Vergleich

3.4.1 Die dargestellten Information über TV-Interessengruppen waren informativer beim

Formular Histogramm
1 2 3 4 5 6 7 8 9 NA

3.4.2 Welche optische Darstellung spricht Sie eher an?

Formular Histogramm
1 2 3 4 5 6 7 8 9 NA

3.4.3 Bei welcher Benutzerschnittstelle machte Ihnen die Bedienung mehr Spaß?

Formular Histogramm
1 2 3 4 5 6 7 8 9 NA

4 Vergleich aller fünf Schnittstellen

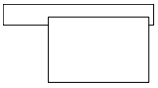
Beurteilen Sie die fünf Benutzerschnittstellen hinsichtlich ihrer Funktionalität und bringen Sie sie in eine Rangfolge.

Ihre Plazierung ▽

4.1 Funktionalität

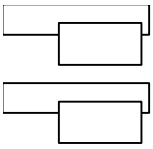
4.1.1  Funktionalität des Formulars, bei dem Interessen lediglich Fernsehsparten ausgewählt oder nicht ausgewählt werden konnten

zu wenig zuviel
1 2 3 4 5 6 7 8 9 NA

4.1.2 

Funktionalität des Formulars, bei dem nur entweder eine Angabe von Präferenzen oder von Mengen möglich war

zu wenig zuviel
1 2 3 4 5 6 7 8 9 NA

4.1.3 

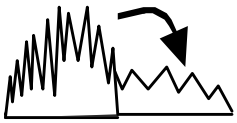
Funktionalität des Formulars, bei dem die Angabe der gewünschten Menge an Sendungen und der persönlichen Präferenzen getrennt möglich war

zu wenig zuviel
1 2 3 4 5 6 7 8 9 NA

4.1.4 

Funktionalität der Histogrammdarstellung, bei der die Höhe der Histogramme nicht verändert werden konnte

zu wenig zuviel
1 2 3 4 5 6 7 8 9 NA

4.1.5 

Funktionalität der Histogrammdarstellung, bei der die Höhe der Histogramme verändert werden konnte

zu wenig zuviel
1 2 3 4 5 6 7 8 9 NA

5 Vertrauen zum erstellten Profil

5.1 Stellen Sie sich vor, ein automatisches System stelle für Sie Ihr persönliches Fernsehprogramm zusammen. Ziehen Sie es vor, wenn dieses System Ihre Interessen durch Ihre *manuelle Eingabe* unter Ihrer direkten Kontrolle ermittelt oder ohne Ihr Zutun *automatisch* durch Beobachtung Ihrer Gewohnheiten im Hintergrund erstellt?

manuell automatisch

1 2 3 4 5 6 7 8 9 NA

5.2 Stellen Sie sich nun vor, ein solches System solle für Sie automatisch Aktien kaufen. Welche Methode der Profilerstellung bevorzugen Sie?

manuell automatisch

1 2 3 4 5 6 7 8 9 NA

BIBLIOGRAPHY

[AC99] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: building histograms without looking at data. In *Proceedings of the 1999 International Conference on Management of Data*, pages 181–192, 1999.

[ACM99] P. Resnick and H. Varian (Eds.). Special issue on Recommender Systems. *Communications of the ACM*, 40(3):56-89, March 1997.

[Adobe96] *Adobe Photoshop 4.0 User Manual*, Adobe Systems Incorporated, San Jose, CA, 1996. Online at http://www.adobe.com/prodindex/photoshop/main.html.

[Ake95] S. Aker. Most of MacPack: *Creative Activities with MacPaint and MacWrite*. Ashton Tate, February 1985.

[AKK97] J. Alspector, A. Kolcz, and N. Karunanithi. Feature-based and clique-based user models for movie selection: A comparative study. *User Modeling And User-Adapted Interaction*, 7(4):279-304, 1997.

[AKK98] J. Alspector, A. Kolcz., and N. Karunanithi. Comparing feature-based and clique-based user models for movie selection. In *Proceedings of the Third ACM Conference on Digital Libraries*, pages 11-18, Pittsburgh, PA, June 1998.

[All90] R. Allen. User Models: theory, method, and practice. *International Journal of Man-Machine Studies*, 32:511-543, 1990.

[ARZ99] C. Avery, P. Resnick, and R. Zeckhauser. The market for evaluations. *American Economic Review*, 89(3):564-584, 1999.

[AZ97] C. Avery and R. Zeckhauser. Recommender systems for evaluating computer messages. *Communications of the ACM*, 40(3):88-89, March 1997.

[Bac91] P.E. Baclace. Information intake filtering. In *Proceedings of Bellcore Workshop on High-Performance Information Filtering*, Morristown, NJ, November 1991.

[Bac92] P.E. Baclace. Competitive agents for information filtering. *Communications of the ACM*, 35(12):50, December 1992.

[Bal97] M. Balabanovic. An adaptive web page recommendation service. In *Proceedings of the 1st International Conference on Autonomous Agents*, pages 378-385, Marina del Rey, CA, February 1997.

[Bal98] M. Balabanovic. An interface for learning multi-topic user profiles from implicit feedback. In *Recommender Systems, Papers from the 1998 Workshop*, Technical Report WS-98-08, pages 6-10, Madison, WI. Menlo Park, CA: AAAI Press, 1998.

[Bar67] G.C. Barhydt. The effectiveness of non-user relevance assessments. *Journal of Documentation*, 23(2 and 3):146-149 and 251, 1967.

[Bau96] P. Baudisch. TV-Online: an adaptive TV-program guide on the World Wide Web. In *Proceedings of the ABIS '96 workshop*, pages D5.1-D5.4, Dortmund, October 1996.

[Bau96a] P. Baudisch. The Cage: efficient construction in 3d using a cubic adaptive grid. In *Proceedings of the 9th ACM Symposium on User Interface Software and Technology (UIST)*, pages 171-172, Seattle, WA, 1996.

[Bau97] P. Baudisch. Designing an evolving Internet TV program guide**.** Paper presented at *Human Computer Interaction Consortium Winter Workshop (HCIC '97)*, Snow Mountain Ranch, CO, 1997. Available online at http://www.darmstadt.gmd.de/~baudisch/Publications.

[Bau98a] P. Baudisch. The Profile Editor: designing a direct manipulative tool for assembling profiles. In *Proceedings of Fifth DELOS Workshop on Filtering and Collaborative Filtering*, pages 11-17, Budapest, November 1997. ERCIM Report ERCIM-98-W001. Le Chesnay Cedex, France, European Research Consortium for Informatics and Mathematics, 1998.

[Bau98b] P. Baudisch. Recommending TV programs. Paper presented at *Human Computer Interaction Consortium Winter Workshop (HCIC '98*), Snow Mountain Ranch, CO, 1998. Available online at http://www.darmstadt.gmd.de/~baudisch/Publications.

[Bau98c] P. Baudisch. Recommending TV Programs on the Web: how far can we get at zero user effort? In *Recommender Systems, Papers from the 1998* Workshop, Technical Report WS-98-08, pages 16-18, Madison, WI. Menlo Park, CA: AAAI Press, 1998.

[Bau98d] P. Baudisch. Don't click, paint! using toggle maps to manipulate sets of toggle switches. In *Proceedings of the 11th ACM Symposium on User Interface Software and Technology (UIST)*, pages 65-66, San Francisco, CA, November 1998.

[Bau99a] P. Baudisch. Using a painting metaphor to rate large numbers of objects. In *Ergonomics and User Interfaces, Proceeding of the HCI '99 Conference*, pages 266-270, Munich, Germany, August 1999. Mahweh: NJ: Erlbaum, 1999.

[Bau99b] P. Baudisch. Joining collaborative and content-based filtering. In *Online Proceedings of the CHI '99 Workshop Interacting with Recommender Systems*, 1999. Available online at http://www.darmstadt.gmd.de/rec99.

[BC87] N.J. Belkin and W.B. Croft. Retrieval techniques. In M.E. Williams (Ed.) *Annual Review of Information Science and Technology*, Chapter 4, pages 109-145, Elsevier, 1987.

[BC92] N.J. Belkin and W.B. Croft. Information filtering and information retrieval: two sides of the same coin? *Communications of the ACM*, 35(12):29-37, December 1992.

[BCB94] B.T. Bartell, G.W. Cottrell, and R.K. Belew. Automatic combination of multiple ranked retrieval systems. In *Proceeding of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 173-181, Dublin, Ireland, July 1994.

[BCCC93] N.J. Belkin, C. Cool, W.B. Croft, and J.P. Callan. The effect of multiple query representation on information retrieval performance. In *Proceeding of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 339-346, Pittsburgh, PA, June/July 1993.

[BCH92] D.A. Buell and D.H. Kraft. Threshold values and Boolean retrieval systems. *Information Processing and Management*, 17(3):127-136, 1981.

[Ber98] R.W. Berger. *Gamma.* Online at http://www.vtiscan.com/~rwb/gamma.html.

[BG89] H. Bandemer and S. Gottwald. *Einführung in Fuzzy-Methoden*. Berlin: Akademie Verlag, 1989/93.

[BGT87] G. Brajnik, G. Guida, and C. Tasso. User modeling in intelligent information retrieval. *Information Processing and Management*, 23:305-320, 1987.

[BHKR98] A. Borchers, J. Herlocker, J. Konstan, and J. Riedl. Ganging up on information overload. *Computer*, 31(4):106-108, April 1998.

[Bie98] M. Bienkovsky. Recommending Web resources to science educators. In *Recommender Systems, Papers from the 1998 Workshop*, Technical Report WS-98-08, pages 19-23, Madison, WI. Menlo Park, CA: AAAI Press, 1998.

[Bil98] D. Billsus, M.J. Pazzani. Learning collaborative information filtering. In *Recommender Systems, Papers from the 1998 Workshop*, Technical Report WS-98-08, pages 24-28, Madison, WI. Menlo Park, CA: AAAI Press, 1998.

[BL00] P. Baudisch and D. Leopold. Attention, indifference, dislike, action: Web advertising involving users. *Netnomics Journal*, 2:75-83, 2000.

[BL98] P. Baudisch and D. Leopold. User-configurable advertising profiles applied to web page banners**.** In *Proceedings of the First Berlin Internet Economics Workshop*, Berlin, October 1997.

[Bla90] D.C. Blair. *Language and Representation in Information Retrieval*. Amsterdam: Elsevier, 1990.

[BLF98] P. Baudisch, D. Leopold, and M. Frühwein. Benutzerangepaßte Auswahl von Bannerwerbung im World Wide Web**.** *Der GMD Spiegel,2*:46-48, 1998.

[BOB82] N. Belkin, R. Oddy, and H. Brooks. ASK for information retrieval. *Journal of Documentation*, 38:61-71 (Part 1); 38:145-164 (Part 2), 1982.

[Boo80] A. Bookstein. Fuzzy requests; An approach to weighted Boolean searches. *Journal of the American Society for Information Science*, 31(4):240-247, July 1980.

[Boo81] A. Bookstein. A comparison of two systems of weighted Boolean retrieval. *Journal of the American Society for Information Science*, 32(4):275-279, July 1981.

[Boy82] B. Boyce. Beyond topicality: A two-stage view of relevance and the retrieval process. *Information Processing & Management*, 18:105-109, 1982.

[Bru99] A. Brügelmann. *Implementierung eines kombinierten inhalts- und kollaborativ basierten Retrievalverfahrens (Implementation of a content-based and collaborative retrieval method).* Diploma Thesis, TU-Darmstadt, September 1999.

[BS97] M. Balabanovic and Y. Shoham, Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66-72, March 1997.

[BT99] P. Baudisch and L. Terveen. Interacting with recommender systems. *Conference companion on Human factors in computing systems*, page 164, Pittsburgh, PA, 1999.

[CCG95] N. Charoenkitkarn, M.H. Chignell, and G. Golovchinsky. Interactive exploration as a formal text retrieval method: how well can interactivity compensate for unsophisticated retrieval algorithms? In *Proceedings of the Third Text Retrieval Conference (TREC-3)*, pages 179-199, D.K. Harman (Ed.), National Institute of Standards and Technology (NIST) Special Publication 500-225, Gaithersburg, Maryland, 1995.

[CCH92] J.P. Callan, W.B. Croft, and S.M. Harding. The INQUERY retrieval system. In *Proceedings of the 3rd International Conference on Database and Expert System Applications*, pages 78-83, Valencia, Spain, September 1992. Berlin and New York: Springer, 1992.

[CDN88] J. Chin, V. Diehl, and K. Norman. Development of an instrument measuring user satisfaction of the human-computer interface, In *Proceedings of the 1988 ACM Conference on Human Factors in Computing Systems*, pages 213-218, 1988.

[CF98] L. Kovács (Ed.) *Proceedings of Fifth DELOS Workshop on Filtering and Collaborative Filtering*. Budapest, November 1997. ERCIM Report ERCIM-98-W001. Le Chesnay Cedex, France: European Research Consortium for Informatics and Mathematics, 1998.

[CGG+96] H. Cossmann, C. Griwodz, G. Grassel, M. Puhlhofer, M. Schreiber, R. Steinmetz, H. Wittig, L. Wolf. Interoperable ITV systems based on MHEG. In *Multimedia Computing and Networking, Proceedings of the SPIE - The International Society for Optical Engineering*, 2667:60-68, San Jose, CA, 1996. Bellingham, WA, SPIE, 1996.

[CGM+99] M. Claypool, A. Gokhale, T. Miranda, P. Murnikov, D. Netes, and M. Sartin. Combining Content-based and Collaborative Filters in an Online Newspaper. In *Online Proceedings of the ACM SIGIR '99 Workshop on Recommender Systems: Algorithms and Evaluation.* University of California, Berkeley, Aug. 1999. Available online at http://www.csee.umbc.edu/~ian/sigir99-rec.

[Coo73a] W.S. Cooper. On selecting a measure of retrieval effectiveness, Part 1: The "subjective" philosophy of evaluation. *Journal of the American Society for Information Science*, 24(2):87-100, 1973.

[Coo73b] W.S. Cooper. On selecting a measure of retrieval effectiveness, Part 2: Implementation of the philosophy. *Journal of the American Society for Information Science*, 24(6):413-424, 1973.

[Coo78] W.S. Cooper and M.E. Maron. Foundation of probabilistic and utility-theoretic indexing. *Journal of the Association for Computing Machinery*, 25(1):67-80, 1978.

[CR87] J.M. Carroll and M.B. Rosson. Paradox of the active user. In J.M. Carroll (Ed.) *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, pages 80-111. Cambridge MA: The MIT Press, 1987.

[CS94] S. Chen and S. Shen. Multimedia computing for digital libraries. *Newsletter of the Technical Committee on Multimedia Computing of IEEE Multimedia*, June 1994.

[CS98] L. Chen and K. Sycara. WebMate, a personal agent for browsing and searching. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 132-129, Minneapolis, MN, May 1998. ACM Press, New York, 1998.

[CSP69] Canada Institute for Scientific and Technical Information. CAN/SDI profile Design Manual, First Edition, Ottawa, 1969.

[DeJ80] K.A. De Jong. Adaptive system design: a genetic approach. *IEEE Transactions on Systems, Man and Cybernetics*, 10(9):566-574, 1980.

[Den82] P. Denning. Electronic junk. *Communications of the ACM*, 23(3):163-165, March 1982.

[Der77] B. Dervin. Useful theory for librarianship: Communication, not information. *Drexel Library Quarterly*, 13(3):16-32, 1977.

[DI98] J. Delgado and N. Ishii. Content + collaboration = recommendation. In *Recommender Systems, Papers from the 1998 Workshop*, Technical Report WS-98-08, pages 37-41, Madison, WI. Menlo Park, CA: AAAI Press, 1998.

[DtHh98] D. Das and H. ter Horst. Recommender systems for TV. In *Recommender Systems, Papers from the 1998 Workshop*, Technical Report WS-98-08, pages 35-36, Madison, WI. Menlo Park, CA: AAAI Press, 1998.

[Due97] J. Dugué. *Designing an Evolving Internet TV Program Guide*. Diploma Thesis, Institut de Recherches et d'Enseignement Supérieur aux Techniques de l'Electronique, 1997.

[Dum88] S. Dumais et al. Using latent semantic analysis to improve access to textual information. In *Proceedings of the 1988 ACM Conference on Human Factors in Computing Systems*, pages 281-285, 1988. New York, ACM, 1988.

[DVB96] M. Kuhn. *The New European Digital Video Broadcast (DVB) Standard*. Available online at ftp://ftp.informatik.uni-erlangen.de/local/cip/mskuhn /tv-crypt/dvb.txt.

[Ehr95] M. Ehrmantraut. *Verfahren zur Benutzerspezifischen Informationsrepräsentation in Interaktiven Fernsehsystemen.* Diploma Thesis. Univ. Kaiserslautern, IBM ENC Heidelberg, 1995.

[EHWS96] M. Ehrmantraut, T. Härder, H. Wittig, and R. Steinmetz. The Personal Electronic Program Guide—towards the pre-selection of individual TV Programs. In *Proceedings of the 5th International Conference on Information and Knowledge Management (CIKM'96)*, pages 243-250, Rockville, MD, 1996.

[FB90] N. Fuhr and C. Buckley. Probabilistic document indexing from relevance feedback data. In *Proceeding of the 13th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 45-62, Bruxelles University, September 1990.

[FB92] W.B. Frakes and R. Baeza-Yates. *Information Retrieval: Data Structures & Algorithms.* Englewood Cliffs, NJ: Prentice Hall, 1992.

[FC89] M.E. Frisse and S.B. Cousins. Information retrieval from hypertext: update on the Dynamic Medical Handbook Project. In *Hypertext '89 Proceedings*, *ACM SIGCHI Bulletin*, pages 199-212, November 1989.

[FC97] R. Fidel and M. Crandall. Users' perception of the performance of a filtering system. In *Proceeding of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. pages 198-205, 1997.

[FD92] P.W. Foltz and S.T. Dumais. Personalized information delivery: An analysis of information filtering methods. *Communications of the ACM*, 35(12):51-60, December 1992.

[Fel98] S. Feldman. The Internet "search-off". *Searcher Magazine*, 6(2), February 1998.Available online at http://www.infotoday.com/searcher/feb/story1.htm.

[Fit54] P.M. Fitts. The information capacity of the human motor system is controlled by the amplitude of movement, *Journal of Experimental Psychology*, 7:93-242, 1954.

[Fol90] J. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes. *Computer graphics: Principles and Practice.* Reading, Mass. Addison-Wesley, 1990.

[Fro94] T.J. Froehlich. Relevance reconsidered – towards an agenda for the 21st century: introduction to special topic issue on relevance research. *Journal of the American Society for Information Science*, 45(3):124-134, April 1994.

[Fru97] M. Frühwein. *Benutzerbasierte Relevanzbestimmung am Beispiel der WWW Fernsehzeitschrift TV-Online (User-based relevance measure for a Web-based TV program guide)*. Diploma Thesis, TU-Darmstadt, September 1997.

[FS91] G. Fischer and C. Stevens. Information access in complex, poorly structured information spaces. In *Proceedings of the 1991 ACM Conference on Human Factors in Computing Systems*, pages 63-70, New Orleans, LA, April 1991.

[FS94] E.A. Fox and J.A. Shaw. Combination of multiple searches. In *Proceedings of the Second Text REtrieval Conference (TREC-2)*, pages 243-252, National Institute of Standards and Technology (NIST) Special Publication 500-215, 1994.

[GBM+89] J. Gould, S. Boies, A. Meluson, M. Rasamny, and A. Vosburgh. Entry and selection methods for specifying dates. *Human Factors*, 31(2):199-214, 1989.

[GCG96] L. Gravano, C.C.K. Chang, H. Garcia-Molina, and A. Paepcke. STARTS: Stanford proposal for Internet meta-searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 207-218, Tucson, AZ., May 1997.

[GCT97] W. Greiff, W.B. Croft, and H. Turtle. Computationally tractable probabilistic modeling of Boolean operators. In *Proceedings of the 20th Annual International Conference on Research and Development in Information Retrieval SIGIR'97*, pages 119-128, Philadelphia, PA, 1997.

[GFK97] GfK Fernsehforschung. *TV-Rating Report*. 9th week 1997 (24 February 97–2 March 97). Baden Baden, Germany: Media Control, 1997.

[GG98] L. Gravano and H. García-Molina. Merging ranks from heterogeneous internet sources. In *Proceedings of the 23rd VLDB Conference*, pages 196-205, Athens, Greece, 1997.

[GGKS95] D.F. Geddis, M.R. Genesereth, A.M. Keller, and N.P. Sigh. Infomaster: a virtual information system. In *Proceedings of the CIKM 95 Workshop of Intelligent Information Agents*, Baltimore, Maryland, 1995.

[GHA+90] D.J. Gillian, K. Holden, S. Adam, M. Rudisill, and L. Magee. How does Fitts' law fit pointing and dragging? In *Proceedings of the 1990 ACM Conference on Human Factors in Computing Systems*, pages 227-234, Seattle, WA, April 1990. New York, ACM, 1990.

[Gil99] A. Gilbert. *TV Recommendation System: Interpreting and Structuring Humans' Behaviors Through Implicit Feedback*. Diploma Thesis, TU-Darmstadt, September 1999.

[GL91] M.L. Gordon and P. Lenk. A utility theoretic examination of the probability ranking principle in information retrieval. *Journal of the American Society for Information Science*, 42(10):703-714, 1991.

[Gl98] N. Glance. Putting recommender systems to work for organizations. In *Recommender Systems, Papers from the 1998 Workshop*, Technical Report WS-98-08, pages 51-52, Madison, WI. Menlo Park, CA: AAAI Press, 1998.

[GM99] S. Gabrielli and S. Mizzaro. Negotiating a multidimensional framework for relevance space. Tech-report UDMI/04/99, Department of Mathematics and Computer Science, University of Udine, 1999.

[GNOT92] D. Goldberg, D. Nichols, B.M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry, *Communications of the ACM*, 35(12)61-70, December 1992.

[Gnus] L. Magne-Ingebritson. *Gnus 5.4 Reference Manual*. Available online at http://www.gnus.org/manual.html.

[GP97] N. Gövert and U. Pfeiffer. *SFgate, The WWW Gateway for FreeWAIS-sf*. Edition 1.108, for SFgate 5.1. January 1997.

[Gre85] J.J. Grefenstette. *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, The Robotics Institute of Carnegie Mellon University, Pittsburgh, PA, 1985.

[Gre98] D.R. Greening. Collaborative filtering for Web marketing efforts. In *Recommender Systems, Papers from the 1998 Workshop*, Technical Report WS-98-08, pages 53-55, Madison, WI. Menlo Park, CA: AAAI Press, 1998.

[GroupLens] GroupLens homepage at the Computer Science Department of the University of Minnesota http://www.cs.umn.edu/Research/GroupLens.

[Gru87] J. Grudin. Social evaluation of the user interface: who does the work and who gets the BENEFIT? In *Proceedings of the IFIP INTERACT'87: Human-Computer Interaction*, pages 805-811, 1987.

[Gru89] J. Grudin. Why CSCW Applications Fail: Problems in the design and evaluation of organizational interfaces. *Office: Technology and People*, 4:245-264, 1989.

[GSK+99] N. Good, J.B. Schafer, J. Konstan, A. Borchers, B. Sarwar, J. Herlocker, and J. Riedl. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the 1999 Conference of the American Association of Artificial Intelligence (AAAI-99)*, pages 439-446, 1999.

[Har92] S.P. Harter. Psychological relevance and information science, *Journal of the American Society for Information Science*, 43(9):602-615, 1992.

[Har94] D.K. Harman. Overview of the third Text REtrieval Conference (TREC-3). In *Proceedings of the Third Text Retrieval Conference (TREC-3)*, pages 1-19, D.K. Harman (Ed.), National Institute of Standards and Technology (NIST) Special Publication 500-225, Gaithersburg, Maryland, 1995. Online at http://potomac.ncsl.nist.gov/TREC.

[HB97] T. Hofmann and J. Buhmann. Pair-wise data clustering by deterministic annealing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(1):1-14, January 1997.

[HC93] D. Haines and W.B. Croft. Relevance feedback and inference networks. In *Proceeding of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2-11, Pittsburgh, PA, June/July 1993. AC, New York, 1993.

[Herz94] F. Herz. System and Method for Scheduling Broadcast of and Access to Video Programs and Other Data Using Customer Profiles. U.S. Patent 5,758,257, May 26, 1998; U.S. Patent 6,020,883, February 1, 2000; U.S. Patent 6,088,722, July 11, 2000.

[HHWM92] W.C. Hill, J.D. Hollan, D. Wrobelwski, and T. McCandless. Edit wear and read wear. In *Proceedings of the 1992 ACM Conference on Human Factors in Computing Systems*, pages 3-9, Monterey, CA, May 1992.

[HKBR99] J. Herlocker, J. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. *Proceedings of the 22th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 230-237, Berkeley, CA, August 1999.

[HMM00] L. Herman, G. Melançon, and M.S. Marshall. Graph visualization and navigation in information visualization: a survey. In *IEEE Transactions On Visualization And Computer Graphics*, 6:24-43, 2000.

[Hoh94] H. Hohl. *Entwurf und Einsatz wissensbasierter Werkzeuge zur computergestützten Exploration von Informationsräumen.* Ph.D. thesis, Fakultät Informatik der Universität Stuttgart, 1994.

[Hou69] E.M. Houseman. Survey of current systems for selective dissemination of information. Technical Report SIG/SDI-1, American Society for Information Science, Special Interest Group on SDI, Washington, DC, June 1969.

[HRF95] W. Hill, L. Stead, M. Rosenstein, and G. Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the 1995 ACM Conference on Human Factors in Computing Systems*, pages 194-201, Denver, CO, May 1995.

[HRS94] W. Hill, M. Rosenstein, and L. Stead. Community and history-of-use navigation. In *Electronic Proceedings of the Second World Wide Web Conference '94.* National Center for Supercomputer Applications, Software Development Group, October 1994. Not available in print. Available online at http://community.bellcore.com/navigation/home-page.html.

[HT96] W. Hill and L. Terveen. Using frequency-of-mention in public conversation for social filtering. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW-96)*, pages 106-112, Boston, MA, November 1996.

[Hum89] B.D. Humm. A visual calendar for scheduling small group meetings. Technical report, Department of Computer Sciences, Univ. of North Carolina at Chapel Hill, 1989.

[Ing82] P. Ingwersen. Search procedures in the library analyzed from the cognitive point of view. *Journal of Documentation*, 38:165-191, 1982.

[Ing86] P. Ingwersen. Cognitive analysis and the role of the intermediary in information retrieval. In: R. Davies (Ed.). *Intelligent Information Systems*, p. 206-237. Chichester, West Sussex: Horwood, 1986.

[Ing92] P. Ingwersen. *Information Retrieval Interaction.* London: Taylor Graham Publishing, 1992.

[INQUERY92] *INQUERY 2.1 System Documentation*, Information Retrieval Laboratory, Dept. of Computer Science, University of Massachusetts, Amherst, MA, 1992-94.

[IP95] Y.E. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 233-244, San Jose, CA, May 1995.

[JF87] W.P. Jones and G.W. Furnas. Pictures of relevance: a geometric analysis of similarity measures. *Journal of the American Society for Information Science*, 38(6):420-442, 1987.

[JH92] A. Jennings and H. Higuchi. A personal news service based on a user model neural network. *IEICE Transactions on Information and Systems*, E75-D(2):198-209, March 1992.

[Kay95] J. Kay. The um toolkit for cooperative user modelling. *User Modelling and User-Adapted Interaction*, 4:149-196, 1995.

[KB97] H. Klock and J. Buhmann. Data visualization by multidimensional scaling: a deterministic annealing approach. *Pattern Recognition*, 33(4):651–669, April 2000.

[KBG74] E. Katz, J.G. Blumler, and M. Gurevitch. Utilization of mass communication by the individual. In J.G. Blumler and E. Katz (Eds.) *The Uses of Mass Communication*, pages 19-34. Beverly Hills, CA: Sage Publications, 1974.

[KC90] R. Kubey and M. Csikszentmihalyi. Television as escape: subjective experience before an evening of heavy viewing. *Commun. Rep.*, 3(2):92-100, 1990.

[KHL+94] J. Karlgren, K. Höök, A. Lantz, J. Palme, and D. Pargman. The glass box user model for filtering. Technical Report T94:09, Swedish Institute of Computer Science, July 1994.

[KMM+97] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Riedl. GroupLens: applying collaborative filtering to Usenet news. *Communications of the ACM*, 40(3):77–87, March 1997.

[Koc74] M. Kochen. *Principles of Information Retrieval.* Los Angeles: Melville Publishing Company, 1974.

[Koe98] M. Koehler. *Entwurf und Implementierung von Empfehlungskomponenten für WWW-Systeme (Design and Implementation of Recommender Components for Web-based Systems).* Diploma Thesis, TU-Darmstadt, 1998.

[Kor84] R.R. Korfhage. Query enhancement by user profiles. In *Proceedings of the Third Joint BCS and ACM Symposium*, pages 111-122, 1984.

[KR93] R. Keeney and H. Raiffa. *Decision with Multiple Objectives: Preferences and Value Tradeoffs.* Cambridge University Press, 1993. First published 1976 by John Wiley and Sons.

[KRBH98] J. Konstan, J. Riedl, A. Borchers, and J. Herlocker. Recommender systems: a GroupLens perspective. In *Recommender Systems, Papers from the 1998 Workshop*, Technical Report WS-98-08, pages 60-64, Madison, WI. Menlo Park, CA: AAAI Press, 1998.

[KS98] H. Kautz and B. Selman. Creating models of real-world communities with Referral Web. In *Recommender Systems, Papers from the 1998 Workshop*, Technical Report WS-98-08, pages 58-59, Madison, WI. Menlo Park, CA: AAAI Press, 1998.

[KSS96] H. Kautz, B. Selman, and A. Milevski. Agent-amplified communication. In *Proceedings of AAAI-96*, pages 3-9, Portland, Oregon, Cambridge, MA, MIT Press, 1996.

[KSS97a] H. Kautz, B. Selman, and M. Shah. Referral Web: combining social networks and collaborative filtering. *Communications of the ACM*, 40(3):63-65, March 1997.

[KSS97b] H. Kautz, B. Selman, and M. Shah. The hidden web. *AI Magazine*, 18(2):27-36, Summer 1997.

[Lan78] F.W. Lancaster. *Information Retrieval Systems: Characteristics, Testing and Evaluation*. 2nd Edition, New York: John Wiley and Sons, 1979.

[Lan95] K. Lang. News Weeder: Learning to filter Netnews. In *Proceedings of the 12th International Conference on Machine Learning*, Tahoe City, CA., 1995.

[LB98] D. Leopold and P. Baudisch. Konzepte benutzerangepaßter Bannerwerbung im World Wide Web am Beispiel der adaptiven Programmzeitschrift TV-Online (Concepts of User-Adapted Banner Advertising on the World Wide Web)**.** GMD-Report no. 9, GMD, Sankt Augustin, 1998.

[Lee95] J.H. Lee. Combining multiple evidence from different properties of weighing schemes. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 180-188, Seattle, WA, 1995.

[Lee97] J.H. Lee. Analysis of multiple evidence combination. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Devel-*

opment in Information Retrieval, pages 267-275, Philadelphia, PA, July 1997.

[Leg75] P. Leggate. Computer-based current awareness services. *Journal of Documentation*, 31:93-115, 1975.

[Lin93] X. Lin. *Self-Organizing Semantic Maps as Graphical Interfaces for Information Retrieval*. Unpublished Ph.D. thesis, University of Maryland, 1993.

[Liu82] J. Liu. *A Distance Approach Toward an Ideal Information Retrieval System.* M.S. thesis, Dallas, Texas: Southern Methodist University. 1982.

[LKH90] E. Lutz, H.V. Kleist-Retzow, and K. Hoerning. MAFIA—An active mail-filter agent for an intelligent document processing support. In *Proceedings of the IFIP WG84, Conference on Multi-User Interfaces and Applications*, pages 235-251, S. Gibbs and A.A. Verrijn-Stuart (Eds.), Crete, 1990.

[LMM94] Y. Lashkari, M. Metral, and P. Maes. Collaborative interface agents. *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 444-449, 1994.

[LMMP96] W. Lam, S. Mukhopadhyay, J. Mostafa, and M. Palakal. Detection of interest shifts for personalized information filtering. In *Proceeding of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 317-324, Zurich, Switzerland, August 1996.

[LN93] P.H. Lindsey and D.A. Norman. *Human Information Processing*. New York: Academic Press, 1977.

[LN98] F. Linton, A. Charron, and D. Joy. OWL: A recommender system for organization-wide learning. In *Recommender Systems, Papers from the 1998 Workshop*, Technical Report WS-98-08, pages 65-69, Madison, WI. Menlo Park, CA: AAAI Press, 1998.

[Loe92] S. Loeb. Architecting personalized delivery of multimedia information. *Communications of the ACM*, 35(12):39-48, December 1992.

[LT92] S. Loeb and D. Terry. Information filtering. *Communications of the ACM*, 35(12):26-28, December 1992.

[Luh58] H.P. Luhn. A business intelligence system. *IBM Journal of Research and Development*, 2(4):314-319, October 1958.

[Mae94] P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31–40, July 1994.

[Mal94] D.A. Maltz. *Distributing Information for Collaborative Filtering On Usenet Net News*. Master's thesis, MIT Department of EECS, Cambridge, Mass. May 1994.

[Mar95] G. Marchionini. *Information Seeking in Electronic Environments*. Cambridge, MA: Cambridge Univ. Press, 1995. (Cambridge series on human-computer-interaction; 9).

[MCBC89] C. Meadow, B. Cerny, C. Borgman, and D. Case. Online access to knowledge: System design. *Journal of the American Society for Information Science*, 40:86-98, 1989.

[MD89] D. McCarthy and U. Dayal. The architecture of an active database management system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 215-224, Portland, Oregon, May 1989.

[MDM86] J. McDonald, T. Dayton, and D.R. McDonald. Adapting menu layout to tasks. Memoranda in Computer and Cognitive Science MCCS-86-78, Computer Research Laboratory, New Mexico, State University, 1986.

[ME95] D. Maltz and K. Ehrlich. Pointing the way: active collaborative filtering. In *Proceedings of the 1995 ACM Conference on Human Factors in Computing Systems*, pages 202-209, Denver, CO, May 1995.

[MGT+87] T.W. Malone, K.R. Grant, F.A. Turbak, S.A. Brobst, and M.D. Cohen. Intelligent information sharing systems. *Communications of the ACM*, 30(5):390-402, May 1987.

[MGT86] T.W. Malone, K.R. Grant, and F.A. Turbak. The Information Lens: an intelligent system for information sharing in organizations. In *Proceedings of the 1986 ACM Conference on Human Factors in Computing Systems*, pages 1-8, Boston, April 1986. ACM, New York, 1986.

[Miz96] S. Mizzaro. Relevance: The whole history. *Journal of the American Society for Information Science*, 48(9):810-832, 1997.

[MMC+89] W.E. Mackay, T.W. Malone, K. Crowston, R. Rao, D. Rosenblatt, and S.K. Card. How do experienced Information Lens users use rules? In *Proceedings of the 1989 ACM Conference on Human Factors in Computing Systems*, pages 211-216, Austin, TX, April 1989. New York, ACM, 1989.

[MMN88] J. McDonald, M. Molander, and R. Noel. Color coding categories in menus. In *Proceedings of the 1988 ACM Conference on Human Factors in Computing Systems*, pages 101-106, 1988. New York, ACM, 1988.

[Moc96] K.J. Mock. *Intelligent Information Filtering via Hybrid Techniques: Hill Climbing, Case-based Reasoning, Index Patterns, and Genetic Algorithms.* Ph.D. thesis, University of California Davis, 1996. Available online at http://phobos.cs.ucdavis .edu:8001/~mock/infos/infos.html.

[MR98] R.J. Mooney and L. Roy. Book recommendation using text categorization. In *Recommender Systems, Papers from the 1998 Workshop*, Technical Report WS-98-08, pages 70-74, Madison, WI. Menlo Park, CA: AAAI Press, 1998.

[MRK+97] B. Miller, J. Riedl, J. Konstan, P. Resnick, D. Maltz, and L. Herlocker. *The GroupLens Protocol Specification*. Available online at http://www.cs.umn .edu/Research/GroupLens/old/protocol/protocol.html

[MRK97] B. Miller, J. Riedl, and J. Konstan. Experiences with GroupLens: making Usenet useful again. In *Proceedings of the 1997 USENIX Winter Technical Conference*, Anaheim, CA, Jan. 1997.

[MS94] M. Morita and Y. Shinoda. Information filtering based on user behavior analysis and best match text retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 272-281, Dublin, Ireland, July 1994. ACM, New York, 1994.

[Mya87] S.H. Myaeng. *The Roles of User Profiles in Information Retrieval.* Ph.D. thesis, Southern Methodist University, 1987.

[Mye80] T.H. Myer. Future message system design: Lessons from the Hermes experience. In *Proceedings of Distributed Computing COMPCON 80*, pages 76-84, Washington, D.C., NY:IEEE, September 1980.

[NH98] H. Nguyen and P. Haddawy. The Decision-Theoretic Video Advisor. In *Recommender Systems, Papers from the 1998 Workshop*, Technical Report WS-98-08, pages 77-80, Madison, WI. Menlo Park, CA: AAAI Press, 1998.

[Nic97] D.M. Nichols. Implicit ratings and filtering. In *Proceedings of Fifth DELOS Workshop on Filtering and Collaborative Filtering*, pages 31-36, Budapest, November 1997. ERCIM Report ERCIM-98-W001. Le Chesnay Cedex, France, European Research Consortium for Informatics and Mathematics, 1998.

[Nor88] D.A Norman. *The Psychology of Everyday Things*. New York: Basic Books, 1988.

[Nor91] K.L. Norman. *The Psychology of Menu Selection: Designing Cognitive Control at the Human/Computer Interface*. Norwood, NJ: Ablex, 1991.

[Oar94] D. Oard. *User Modeling for Information Filtering.* Paper available online at http://www.ee.umd.edu/medlab/filter/papers/umir.html.

[Oar97] D. Oard. The state of the art in text filtering. *User Modeling and User-Adapted Interaction*, 7(3):141-178, 1997.

[OK98] D.W. Oard and J. Kim. Implicit feedback for recommender systems. In *Recommender Systems, Papers from the 1998 Workshop*, Technical Report WS-98-08, pages 81-83, Madison, WI. Menlo Park, CA: AAAI Press, 1998.

[Ols98] T. Olsson. Decentralized social filtering based on trust. In *Recommender Systems, Papers from the 1998 Workshop*, Technical Report WS-98-08, pages 84-88, Madison, WI. Menlo Park, CA: AAAI Press, 1998.

[OM96] D.W. Oard and G. Marchionini. A conceptual framework for text filtering. Technical Report EE-TR-96-25, CAR-TR-830, CLIS-TR-96-02, CS-TR-3643, University of Maryland, College Park, May 1996.

[Pal84] J. Palme. You have 134 unread mail! Do you want to read them now? In *Proceedings IFIP WG 6.5 Working Conference on Computer-Based Document Services*, pages 175-184, Nottingham, England, May 1984.

[PB97] M. Pazzani and D. Billsus. Learning and revising user profiles: the identification of interesting Web Sites. *Machine Learning* 27:313-331, 1997.

[Pet96] P.P. Petrov. Shape preserving approximation by free knot splines. *East Journal on Approximations*, 2(1):41-48, 1996.

[Pol88] S. Pollock. A rule-based message filtering system. *ACM Transactions on Office Information Systems*, 6(3):232–54, July 1988.

[Poy98] C. Poynton. *The Gamma FAQ*. 1998. Paper is available online at http://www.inforamp.net/~poynton/GammaFAQ.html.

[PS79] K.H. Parker and D. Soergel. The importance of SDI for current awareness in fields with severe scatter of information. *J. Am. Soc. Inf. Sci.*, 30(3):125–135, 1979.

[PSB90] C. Plaisant, B. Shneiderman, J. Battaglia. Scheduling home control devices: a case study of the transition from the research project to a product. Technical report 90-10, University of Maryland, College Park, MD, 1990.

[PZM96] G. Pass, R. Zabih, and J. Miller. Comparing images using color coherence vectors. In *Proceedings of the Fourth ACM International Conference on Multimedia*, pages 65 – 73, 1996.

[QUIS] *QUestionnaire of User Interface Satisfaction (QUIS)*. Human Computer Interaction lab (HCIL), University of Maryland, College Park. Available online at http://www.lap.umd.edu/QUIS.

[Ray99] E.S. Raymond. *The Cathedral & the Bazaar, Musings on Linux and Open Source by an Accidental Revolutionary*. 1999.

[Rec96] H. Varian (Ed.). *Collaborative Filtering*. *Proceedings from the Berkeley Workshop on Collaborative Filtering*, University of California, Berkeley, CA, March 1996. Available online at http://www.sims.berkeley.edu/resources/collab.

[Rec98] H. Kautz (Ed.). *Recommender Systems, Papers from the 1998 Workshop.* Menlo Park, CA: AAAI Press, Technical Report WS-98-08, 1998.

[Rec99a] P. Baudisch and L. Terveen (Eds.). *Online Proceedings from the CHI '99 Workshop Interacting with Recommender Systems*. Pittsburgh, PA, May 1999. Available online at http://www.darmstadt.gmd.de/rec99.

[Rec99b] I. Soboroff, C. Nicholas, and M.J. Pazzani (Eds.). *Online Proceedings from the ACM SIG-IR'99 Workshop on Recommender Systems, Algorithms and Evaluation*. University of California, Berkeley, August 1999. Part of the 22nd International Conference on Research and Development in Information Retrieval. Available online at http://www.csee.umbc.edu/~ian/sigir99-rec.

[Ric79a] E.A. Rich. User modeling via stereotypes. *Cognitive Science*, 3:329-354, 1979.

[Ric79b] E.A. Rich. *Building and Exploiting User Models.* Ph.D. thesis, CMU, 1979.

[Ric83] E.A. Rich. Users are individuals: individualizing user models. *International Journal of Man-Machine Studies*, 18:199-214, 1983.

[RIS+94] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens: An open architecture for collaborative filtering of Netnews. In *Transcending*

Boundaries, Proceedings of the Computer Supported Cooperative Work Conference (CSCW '94), pages 175-186, Chapel Hill, NC, October 1994.

[RM96] P. Resnick and J. Miller. PICS—Internet access controls without censorship. *Communications of the ACM*, 39(10):87-93, October 1996.

[RMC82] S.E. Robertson, M.E. Maron, and V.S. Cooper. Probability of relevance: a unification of two competing models for document retrieval. *Information Technology (Research & Development)*, 1(1):1-21, 1982.

[Rob77] S.E. Robertson. The probability ranking principle in (IR) information retrieval. *Journal of Documentation*, 33(4):294–304, Dec 1977.

[Rob81] S.E. Robertson. The methodology of information retrieval experiment. In K. Sparck Jones (Ed.) *Information Retrieval Experiment.* Chapter 1, pages 9-31, Butterworths, 1981.

[RP97] J. Rucker and M.J. Polanco. Siteseer: personalized navigation for the Web. *Communications of the ACM*, 40(3):73-76, March 1997.

[RV97] P. Resnick and H. Varian. Recommender systems. *Communications of the ACM*, 40(3): 56-58, March 1997.

[Sal68] G. Salton. *Automatic Information Organization and Retrieval*. New York, NY: McGraw-Hill. 1968.

[Sal71] G. Salton. *The SMART Retrieval System—Experiment in Automatic Document Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1971.

[Sal83] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. New York: McGraw-Hill, 1983.

[Sar75] T. Saracevic. Relevance: A review of and a framework for the thinking on the notion in information science. *Journal of the American Society for Information Science*, 26(6):321-343, 1975.

[SB90] G. Salton and C. Buckley. Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*, 41(4):288-297, 1990.

[SBB96] M. Spenke, C. Beilken, and T. Berlage. FOCUS: the interactive table for product comparison and selection. In *Proceedings of the 9th ACM Symposium on User Interface Software and Technology (UIST)*, pages 41–50, Seattle, WA, 1996.

[SC96] J.R. Smith and S. Chang. VisualSEEK: a fully automated content-based image query system. In *Proceedings of the Fourth ACM International Conference on Multimedia*, pages 87–98, Boston, MA, November 1996.

[Sch94] L. Schamber. Relevance and information behavior. *Annual Review of Information Science and Technology*, 29:3-48, 1994.

[Scr99] J. Scriba. Freundliche Revolution. *Der SPIEGEL*, Dec 6, 1999. Available online at http://www.spiegel.de/spiegel/0,1518,55708,00.html.

[SEN90] L. Schamber, M.B. Eisenberg, M.S. Nilan. A re-examination of relevance: Toward a dynamic, situational definition. *Information Processing and Management*, 26(6):755-776, 1990.

[Sha94] U. Shardanand. *Social Information Filtering for Music Recommendation.* Master's thesis, Department of Electrical Engineering and Computer Science, MIT, September 1994.

[SHA97] S. Sen, T. Haynes, and N. Arora. Satisfying user preferences while negotiating meetings. *International Journal of Human-Computer Studies*, 47:407-427, 1997.

[Shn82] B. Shneiderman. The future of interactive systems and the emergence of direct manipulation. *Behav. Inf. Technol.*, 1(2):237-256, 1982.

[Shn92] B. Shneiderman. Tree visualization with Tree-maps: A 2D space-filling approach. *ACM Transactions on Graphics*, 11(1):92-99, 1992.

[Shn98] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Third edition. Reading MA: Addison-Wesley, 1998.

[SIK+82] D.C. Smith, C. Irby, R. Kimball, W. Verplank, and E. Harslem. Designing the Star user interface, *Byte* 7(4):242-282, 1982.

[SK92] I. Stadnyk and K. Kass. Modeling users' interests in information filters. *Communications of the ACM*, 35(12):49-50, December 1992.

[SKB+ 98] B. Sarwar, J. Konstan, A. Borchers, J. Herlocker, B. Miller, and J. Riedl. Using filtering agents to improve prediction quality in the GroupLens research collaborative filtering system. In *Proceedings of the 1998 Conference on Computer Supported Cooperative Work (CSCW)*, pages 345-354, November 1998.

[SL73] A. Schutz and T. Luckmann. *Structures of the Life World*. Evanston, Ill: Northwestern University Press, 1973.

[SM83] G. Salton, E. Fox, and H. Wu. Extended Boolean information retrieval. *Communications of the ACM*, 26(11):1022-1036, November 1983.

[SM93] B. Sheth and P. Maes. Evolving agents for personalized information filtering. In *Proceedings of the 9th IEEE Conference Artificial Intelligence for Applications*, pages 345-352, Orlando, FL, March 1993.

[SM95] U. Shardanand and P. Maes. Social information filtering: algorithms for automating "word of mouth". In *Proceedings of the 1995 ACM Conference on Human Factors in Computing Systems*, pages 210-217, 1995. ACM, New York, 1995.

[Soe94] D. Soergel. Indexing and retrieval performance: The logical evidence. *Journal of the American Society for Information Science*, 39(3):161-176, 1988.

[Ste67] W. Stephenson. *The Play Theory of Mass Communication*. Chicago: University of Chicago Press, 1967.

[Ste92a] C. Stevens. Automating the creation of information filters. *Communications of the ACM*, 35(12):48, December 1992.

[Ste92b] C. Stevens. *Knowledge-Based Assistance for Accessing Large, Poorly Structured Information Spaces*. Ph.D. thesis, University of Colorado, Department of Computer Science, Boulder, 1992.

[Ste99] R. Steinmetz. *Multimedia-Technologie, Grundlagen, Komponenten und Systeme*. 2nd Edition, Berlin: Springer, 1999.

[Str93] H.J. Strubbe. System and Method for Automatically Correlating User Preferences with a T.V. Program Information Database. U.S. Patent 5,233,924, June 29, 1993.

[Str96] H.J. Strubbe. System and Method for Finding a Movie of Interest in a Large Movie Database. U.S. Patent 5,483,278, January 09, 1996.

[SW00] R. Salvatierra, S.J. Wilson. *Director 8 and Lingo: Inside Macromedia Series.* 1st edition, Delmar Publishing, July 21, 2000.

[SW80] K. Sparck Jones and C.A. Webster. *Research on Relevance Weighting.* Cambridge: Cambridge University, Computer Lab. (BRLD, 5553), 1980.

[SW93] M.F. Schwartz and D.C.M. Wood. Discovering shared interests using graph analysis. *Communications of the ACM*, 36(8):78-89, August 1993.

[Swe88] J.A. Swets. Measuring the accuracy of diagnostic systems. *Science*, 240(4857):1285-1289, June 1988.

[SyBase89] *Transact-SQL User's Guide*. Sybase, Inc., Oct. 1989.

[Tay68] R. Taylor. Question negotiation and information seeking in libraries. *College and Research Libraries*, 29:178-194, 1968.

[TC90] H. Turtle and W.B. Croft. Inference networks and document retrieval. In *Proceedings of the 13th Conference on Research & Development in IR*, pages 1-24, NY: ACM, 1990.

[TC91] H.R. Turtle and W.B. Croft. Evaluation of an inference network-based retrieval model. *ACM Trans. Inf. Syst.*, 9(3):187-222, July 1991.

[TC92] H.R. Turtle and W.B. Croft. A comparison of text retrieval models. *The Computer Journal*, 35(3):279-290, 1992.

[Ter91] D.B. Terry. 7 steps to a better mail system. In P. Schicker and E. Schefferud (Eds.) *Message Handling Systems and Application Layer Communications Protocols*, pages 23-33. North Holland, 1991.

[Ter93] D.B. Terry. A tour through Tapestry. In *Proceeding of the ACM Conference on Organizational Computing Systems (COOCS)*, pages 21-30, November 1993.

[TGNO92] D. Terry, D. Goldberg, D. Nichols, and B. Oki. Continuous queries over append-only databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 321-330, San Diego, May 1992.

[THA+97] L. Terveen, W. Hill, B. Amento, D. McDonald, and J. Creter. Phoaks: a system for sharing recommendations. *Communications of the ACM*, 40(3):59-62, March 1997.

[TJ92] D. Turo and B. Johnson. Improving the visualization of hierarchies with Treemaps: design issues and experimentation. In *Proceedings of IEEE Visualization '92*, pages 124-131, Boston, MA, 1992. Los Alamitos, CA: IEEE Computer Science Press, 1992.

[TKI96] M. Tani, T. Kamiya, and S. Ichiyama. User interfaces for information strolling in a digital library. *NEC Research & Development*, 37(4):535-545, October 1996.

[TREC] *Proceedings of the Text REtrieval Conferences (TREC-1–TREC-8)*. 1992 to date. Available online at http://trec.nist.gov/pubs.html.

[TSD96] L. Tweedie, R. Spence, H. Dawkes, and H. Su. Externalising abstract mathematical models. In *Proceedings of the 1996 ACM Conference on Human Factors in Computing Systems*, pages 406-412, Vancouver, BC, Canada, April 1996.

[TSDS95] L. Tweedie, B. Spence, D. Williams, and R. Bhogal. The Attribute Explorer. In *Conference Companion on Human Factors in Computing Systems*, pages 435 – 436, 1994.

[TSDS95] L. Tweedie, B. Spence, H. Dawkes, and H. Su. The Influence Explorer. In *Conference Companion on Human Factors in Computing Systems*, pages 129-130, 1995.

[Twe97] L. Tweedie. Characterizing interactive externalizations. In *Proceedings of the 1997 ACM Conference on Human Factors in Computing Systems*, pages 375 – 382, 1997.

[vNe47] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. 2nd ed., Princeton, NJ: Princeton University Press, 1947.

[vRi77] C.J. van Rijsbergen. A theoretical basis for the use of co-occurrence data in information retrieval. *Journal of Documentation*, 33(2):106-119, 1977.

[vRi79] C.J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 1979.

[vRi80] C.J. van Rijsbergen, S.E. Robertson, and M.F. Porter. *New Models in Probabilistic Information Retrieval.* Cambridge: Cambridge University. Computer Lab. (BRLD, 5587), 1980.

[WBW96] J. Wood, K. Brodlie, and H. Wright. Visualization over the World Wide Web and its application to environmental data. In *Proceedings of the Conference on Visualization '96*, pages 81-86, San Francisco, CA, November 1996.

[WC79] W.G. Waller, and D.H. Kraft. A mathematical model for a weighted Boolean retrieval system. *Information Processing and Management*, 15(5):235-245, 1979.

[WG95] H. Wittig and C. Griwodz. Intelligent media agents in interactive television systems. In *Proceedings of the International Conference on Multimedia Computing and Systems '95*, pages 182-189, Boston, May 1995. Los Alamitos, CA: IEEE Computer Science Press, 1995.

[Wil88] P. Willett. Recent trends in hierarchical document clustering: A critical review. *Information Processing and Management*, 24(5):577-597, 1988.

[Wit95] H. Wittig. Agents and intelligent agencies for iTV. 7th Meeting of the Digital Audio Visual Council (DAVIC), Document number APP/03/10/13, London, March 1995.

[WS91] C. Williamson and B. Shneiderman. The Dynamic HomeFinder: Evaluating dynamic queries in a real-estate information exploration system. Technical Report CS-TR-2819, University of Maryland, College Park, MD, 1991.

[WS92] C. Williamson and B. Shneiderman. The Dynamic HomeFinder: evaluating dynamic queries in a real-estate information system, *Proceeding of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 338-346, Copenhagen, Denmark, June 1992. ACM, New York, 1992.

[YG94] T. Yan and H. Garcia-Molina. Distributed selective dissemination of information. In *Proceedings of the Third International Conference on Parallel and Distributed Information Systems*, pages 89-98, IEEE Computer Society, September 1994.

[YG95] T. Yan and H. Garcia-Molina. SIFT: A tool for wide-area information dissemination. In *Proceedings of the USENIX 1995 Winter Technical Conference*, pages 177-186, New Orleans, Louisiana. Berkeley, CA: USENIX Assoc., 1995.

[Zah73] L. Zadeh. The concept of a linguistic variable and its application to approximate reasoning. Memo. No. ERL-M411, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 15 October 1973.