Chapter 6
Developing Constraint-based Recommenders

Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach and Markus Zanker

6.1 Introduction

Traditional recommendation approaches (content-based filtering [48] and collaborative filtering[40]) are well-suited for the recommendation of quality&taste products such as books, movies, or news. However, especially in the context of products such as cars, computers, appartments, or financial services those approaches are not the best choice (see also Chapter 11). For example, apartments are not bought very frequently which makes it rather infeasible to collect numerous ratings for one specific item (exactly such ratings are required by collaborative recommendation algorithms). Furthermore, users of recommender applications would not be satisfied with recommendations based on years-old item preferences (exactly such preferences would be exploited in this context by content-based filtering algorithms).

Knowledge-based recommender technologies help to tackle these challenges by exploiting explicit user requirements and deep knowledge about the underlying product domain [11] for the calculation of recommendations. Those systems heavily concentrate on knowledge sources that are not exploited by collaborative filtering and content-based filtering approaches. Compared to collaborative filtering and content-based filtering, knowledge-based recommenders have no cold-start problems since requirements are directly elicited within a recommendation session. However, no advantage without disadvantage, knowledge-based recommenders suffer from the so-called knowledge acquisition bottleneck in the sense that knowledge

Alexander Felfernig (contact author)
Graz University of Technology e-mail: alexander.felfernig@ist.tugraz.at

Gerhard Friedrich
University Klagenfurt e-mail: gerhard.friedrich@uni-klu.ac.at

Dietmar Jannach
TU Dortmund e-mail: dietmar.jannach@tu-dortmund.de

Markus Zanker
University Klagenfurt e-mail: markus.zanker@uni-klu.ac.at

F. Ricci et al. (eds.), *Recommender Systems Handbook*,
DOI 10.1007/978-0-387-85820-3_6,

engineers must work hard to convert the knowledge possessed by domain experts into formal, executable representations.

There are two basic specifics of knowledge-based recommenders: case-based [3, 4, 36] and constraint-based recommenders [11, 13].[1] In terms of used knowledge both are similar: user requirements are collected, repairs for inconsistent requirements are automatically proposed in situations where no solutions could be found [12, 13, 43], and recommendation results are explained. The major difference lies in the way solutions are calculated [11]. Case-based recommenders determine recommendations on the basis of similarity metrics whereas constraint-based recommenders predominantly exploit predefined *recommender knowledge bases* that contain explicit rules about how to relate customer requirements with item features. In this chapter we will focus on an overview of constraint-based recommendation technologies. For a detailed review of case-based recommender technologies the reader is referred to [3, 4, 36].

A *recommender knowledge base* of a constraint-based recommender system (see [16]) typically is defined by two sets of variables (V_C, V_{PROD}) and three different sets of constraints (C_R, C_F, C_{PROD}). Those variables and constraints are the major ingredients of a constraint satisfaction problem [54]. A solution for a constraint satisfaction problem consists of concrete instantiations of the variables such that all the specified constraints are fulfilled (see Section 6.4).

Customer Properties V_C describe possible requirements of customers, i.e., requirements are instantiations of customer properties. In the domain of financial services *willingness to take risks* is an example for a customer property and *willingness to take risks = low* represents a concrete customer requirement.

Product Properties V_{PROD} describe the properties of a given product assortment. Examples for product properties are *recommended investment period, product type, product name, or expected return on investment*.

Constraints C_R are systematically restricting the possible instantiations of customer properties, for example, *short investment periods are incompatible with high risk investments*.

Filter Conditions C_F define the relationship between potential customer requirements and the given product assortment. An example for a filter condition is the following: *customers without experiences in the financial services domain should not receive recommendations which include high-risk products*.

Products Finally, allowed instantiations of product properties are represented by C_{PROD}. C_{PROD} represents one constraint in disjunctive normal form that defines elementary restrictions on the possible instantiations of variables in V_{PROD}.

A simplified recommender knowledge base for the domain of financial services is the following (see Example 6.1).

[1] Utility-based recommenders are often as well categorized as knowledge-based, see for example [4]. For a detailed discussion on utility-based approaches we refer the interested reader to [4, 13].

Example 6.1. Recommender knowledge base (V_C, V_{PROD}, C_R, C_F, C_{PROD})

V_C = {kl_c: [expert, average, beginner] ..... /* level of expertise */
wr_c: [low, medium, high] ..... /* willingness to take risks */
id_c: [shortterm, mediumterm, longterm] ..... /* duration of investment */
aw_c: [yes, no] ..... /* advisory wanted ? */
ds_c: [savings, bonds, stockfunds, singleshares] ..... /* direct product search */
sl_c: [savings, bonds] ..... /* type of low-risk investment */
av_c: [yes, no] ..... /* availability of funds */
sh_c: [stockfunds, singlshares] ..... /* type of high-risk investment */ }

V_{PROD} = {$name_p$: [text] ..... /* name of the product */
er_p: [1..40] ..... /* expected return rate */
ri_p: [low, medium, high] ..... /* risk level */
$mniv_p$: [1..14] ..... /* minimum investment period of product in years */
$inst_p$: [text] ..... /* financial institute */ }

C_R = {CR_1: $wr_c = high \rightarrow id_c \neq shortterm$,
CR_2: $kl_c = beginner \rightarrow wr_c \neq high$}

C_F = {CF_1: $id_c = shortterm \rightarrow mniv_p < 3$,
CF_2: $id_c = mediumterm \rightarrow mniv_p \geq 3 \wedge mniv_p < 6$,
CF_3: $id_c = longterm \rightarrow mniv_p \geq 6$,
CF_4: $wr_c = low \rightarrow ri_p = low$,
CF_5: $wr_c = medium \rightarrow ri_p = low \vee ri_p = medium$,
CF_6: $wr_c = high \rightarrow ri_p = low \vee ri_p = medium \vee ri_p = high$,
CF_7: $kl_c = beginner \rightarrow ri_p \neq high$,
CF_8: $sl_c = savings \rightarrow name_p = savings$,
CF_9: $sl_c = bonds \rightarrow name_p = bonds$ }

C_{PROD} = {$CPROD_1$: $name_p = savings \wedge er_p = 3 \wedge ri_p = low \wedge mniv_p = 1 \wedge inst_p = A$;
$CPROD_2$: $name_p = bonds \wedge er_p = 5 \wedge ri_p = medium \wedge mniv_p = 5 \wedge inst_p = B$;
$CPROD_3$: $name_p = equity \wedge er_p = 9 \wedge ri_p = high \wedge mniv_p = 10 \wedge inst_p = B$}

On the basis of such a recommender knowledge base and a given set of customer requirements we are able to calculate recommendations. The task of identifying a set of products fitting a customer's wishes and needs is denoted as *recommendation task* (see Definition 6.1).

Definition 6.1. A *recommendation task* can be defined as a constraint satisfaction problem (V_C, V_{PROD}, $C_C \cup C_F \cup C_R \cup C_{PROD}$) where V_C is a set of variables representing possible customer requirements and V_{PROD} is a set of variables describing product properties. C_{PROD} is a set of constraints describing product instances, C_R is a set of constraints describing possible combinations of customer requirements, and C_F (also called filter conditions) is a set of constraints describing the relationship

between customer requirements and product properties. Finally, C_C is a set of unary constraints representing concrete customer requirements.

Example 6.2. Based on the recommender knowledge base of Example 6.1, the definition of a concrete recommendation task could be completed with the following set of requirements C_C=$\{wr_c = low, kl_c = beginner, id_c = shortterm, sl_c = savings\}$.

Based on the definition of a recommendation task, we can introduce the notion of a solution (*consistent recommendation*) for a recommendation task.

Definition 6.2. An assignment of the variables in V_C and V_{PROD} is denoted as *consistent recommendation* for a recommendation task (V_C, V_{PROD}, $C_C \cup C_F \cup C_R \cup C_{PROP}$) iff it does not violate any of the constraints in $C_C \cup C_F \cup C_R \cup C_{PROD}$.

Example 6.3. A consistent recommendation with regard to the recommender knowledge base of Example 6.1 and the customer requirements defined in Example 6.2 is $kl_c = beginner, wr_c = low, id_c = shortterm, sl_c = savings, name_p = savings, er_p = 3, ri_p = low, mniv_p = 1, inst_p = A$.

In addition to the recommender knowledge base we have to define the intended behavior of the recommender user interface. In order to support intuitive dialogs, a recommender interface must be adaptive (see Section 3). There exist different alternatives to describe the intended behavior of recommender user interfaces. For example, dialogs can be modeled explicitly in the form of finite state models [20] or can be structured even more flexibly in a form where users themselves are enabled to select interesting properties they would like to specify [37].

In this chapter we will focus on the first alternative: recommendation dialogs are modeled explicitly in the form in finite state models [20]. Transitions between the states are represented as acceptance criteria on the user input. For example, an expert ($kl_c = expert$) who is not interested in a recommendation session regarding financial services ($aw_c = no$) is automatically forwarded to q_4 (search interface that supports the specificiation of technical product features). Figure 6.1 depicts a finite state model of the intended behavior of a financial services recommender application.

The remainder of this chapter is organized as follows. In Section 6.2 we give an overview of knowledge acquisition concepts for the development of recommender knowledge bases and recommender process definitions. In Section 6.3 we introduce major techniques for guiding and actively supporting the user in a recommendation dialog. A short overview of approaches to solve recommendation tasks is given in Section 6.4. In Section 6.5 we discuss successful applications of constraint-based recommender technologies. In Section 6.6 we present future research issues in constraint-based recommendation. With Section 6.7 we conclude the chapter.

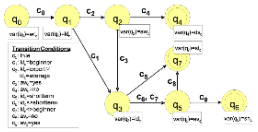

Fig. 6.1: Recommender user interface description: a simple example recommendation process for financial services. The process starts in state q_0, and, depending on the user's knowledge level, is forwarded to either state q_2 or state q_3. In the final state (one of the states q_4, q_6, q_7) the recommended items are presented. Each state q_i has an assigned customer property var(q_i) that represents a question to be asked in this state.

6.2 Development of Recommender Knowledge Bases

The major precondition for successfully applying constraint-based technologies in commercial settings are technologies that actively support knowledge engineers and domain experts in the development and maintenance of recommender applications and thus help to limit knowledge acquisition bottlenecks as much as possible. Due to very limited programming skills of domain experts, there typically is a discrepancy between knowledge engineers and domain experts in terms of knowledge base development and maintenance know-how [13]. Thus domain experts are solely responsible for knowledge provision but not for the formalization into a corresponding executable representation (recommender knowledge base).

The major goal of the commercially available *CWAdvisor* environment presented in [13] is to reduce the above mentioned knowledge acquisition bottleneck: it supports autonomous knowledge base development and maintenance processes for domain experts. In the following sections we will present parts of the *CWAdvisor* environment for demonstration purposes. The *CWAdvisor* knowledge acquisition environment (CWAdvisor Designer) takes into account major *design principles* that are crucial for effective knowledge acquisition and maintenance [8, 13].

- First, *rapid prototyping* processes support the principle of *concreteness* where the user can immediately inspect the effects of introduced changes to explanation texts, properties of products, images, recommender process definitions, and recommendation rules. This functionality is implemented in the form of templates that enable a direct translation of graphically defined model properties into a corresponding executable recommender application.
- Second, changes to all the mentioned information units can be performed on a graphical level. This functionality is very important to make knowledge aquisition environments more accessible to domain experts without a well-grounded technical education. Domain experts are protected from programming details - an approach that follows the principle of a strict *separation of application logic and implementation details*.

- Third, an integrated testing and debugging environment supports the principle of *immediate feedback* in the sense that erroneous definitions in the recommender knowledge base and the recommendation process are automatically detected and reported (end-user debugging support). Thus, knowledge bases are maintained in a structured way and not deployed in a productive environment until all test cases specified for the knowledge base are fulfilled. As a direct consquence, domain experts develop a higher trust level since erroneous recommendations become the exception of the rule.

Figure 6.2 provides examples for major modeling concepts supported by the *CWAdvisor* recommender development environment [13]. This environment can be used for the design of a recommender knowledge base (see Example 6.2), i.e., customer properties (V_C), product properties (V_{PROD}), constraints (C_R), filter conditions (C_F), and the product assortment (C_{Prod}) can be specified on a graphical level. Figure 6.2 (upper part) depicts an interface for the design of filter conditions (C_F) whereas the lower part represents an interface for the context-oriented specification of compatibility constraints. Figure 6.3 shows the *CWAdvisor* Process Designer user interface. This component enables the graphical design of recommendation processes. Given such a process definition, the recommender application can be automatically generated (see, e.g., Figure 6.4).

Sometimes recommendation processes are faulty, for example, the transition conditions between the states are defined in a way that does not allow the successful completion of a recommendation session. If we would change the transition condition $c_1 : kl_c = beginner$ in Figure 6.1 to $c_1' : kl_c = expert$, users who have nearly no knowledge about the financial services domain would not be forwarded to any of the following states (q_2 or q_3). For more complex process definitions, the manual identification and repair of such faults is tedious and error-prone. In [20] an approach is presented which helps to automatically detect and repair such faulty statements. It is based on the concepts of model-based diagnosis [20] that help to locate minimal sets of faulty transition conditions.

In addition to a graphical process definition, *CWAdvisor* Designer supports the automated generation of test cases (input sequences including recommended products) [16]. On the one hand, such test cases can be exploited for the purpose of regression testing, for example, before the recommender application is deployed in the production environment. On the other hand, test cases can be used for debugging faulty recommender knowledge bases (if some of the test cases are not fulfilled) and faulty process definitions (e.g., when the recommender process gets stuck).

The basic principle of recommender knowledge base debugging [10, 12, 13, 16] will now be shown on the basis of Example 6.4.[2] Readers interested in the automated debugging of faulty recommender process definitions are referred to [20]. A typical approach to identify faults in a recommender knowledge base is to test the knowledge base with a set of examples (test cases) $e_i \in E$. For simplicity, let us assume that $e_1 : wr_c = high \wedge rr_c \geq 9\%$ is the only example provided by domain experts up to now. Testing $e_1 \cup C_R$ results in the empty solution set due to the fact

[2] For simplicity, we omit the specification of V_{PROD}, C_F, and C_{PROD}.

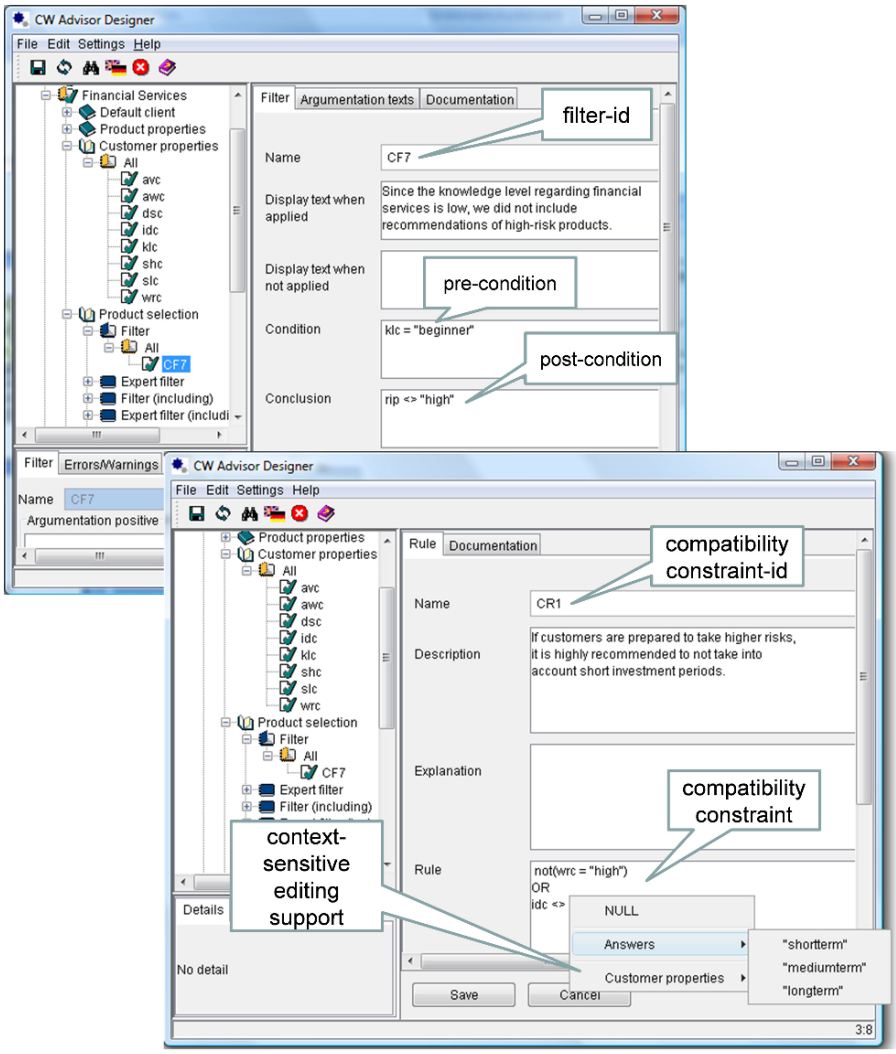

Fig. 6.2: CWAdvisor Designer Environment. Filter constraints (conditions) as well as compatibility constraints can be defined in a context-sensitive editing environment.

that e_1 is inconsistent with C_R. A more detailed look at the example shows that the constraints CR_2, CR_3 are inconsistent with e_1. CR_2, CR_3 is denoted as *conflict set* [33, 45] that can be resolved (under the minimality assumption) by simply deleting one of its elements. For example, if we delete CR_3 from C_R, the consistency of $e_1 \cup C_R$ is restored. The calculation of conflict sets can be realized using the conflict detection algorithm proposed by [33], the automated resolution of conflicts is shown in detail in [20].

Example 6.4. Faulty Recommender knowledge base (V_C, V_{PROD}, C_R, C_F, C_{PROD})

V_C = {rr_c: [1-3%, 4-6%, 7-9%, 9%] ..... /* return rate */
wr_c: [low, medium, high] ..... /* willingness to take risks */
id_c: [shortterm, mediumterm, longterm] ..... /* duration of investment */ }

C_R = {CR_1: $wr_c = medium \rightarrow id_c \neq shortterm$
CR_2: $wr_c = high \rightarrow id_c = long$
CR_3: $id_c = long \rightarrow rr_c = 4-6\% \vee rr_c = 7-9\%$
CR_4: $rr_c \geq 9\% \rightarrow wr_c = high$
CR_5: $rr_c = 7-9\% \rightarrow wr_c \neq low$ }

V_{PROD} = {} C_F = {} C_{PROD} = {}

Experiences from commercial projects in domains such as financial services [18], electronic equipments [13], or e-tourism [74] clearly point out the importance of the above mentioned principles regarding the design of knowledge acquisition and maintenance environments. Within the scope of user studies [10] significant time savings in development and maintenance processes have been detected due to the availability of a graphical development, test, and automated debugging environment. Experiences from the financial services domain [18] show that initially knowledge bases have to be developed within the scope of a cooperation between domain experts and technical experts (knowledge engineers). Thereafter, most development and maintenance requests are directly processed by the domain experts (e.g., updates in product tables, adaptations of constraints, or recommender process definitions).

6.3 User Guidance in Recommendation Processes

As constraint-based recommender systems operate on the basis of explicit statements about the current customer's needs and wishes, the knowledge about these user requirements has to be made available to the system before recommendations can be made. The general options for such a *requirements elicitation process* in increasing order of implementation complexity include the following.

1. Session-independent customer profiles: users enter their preferences and interests in their user profile by, for example, specifying their general areas of inter-

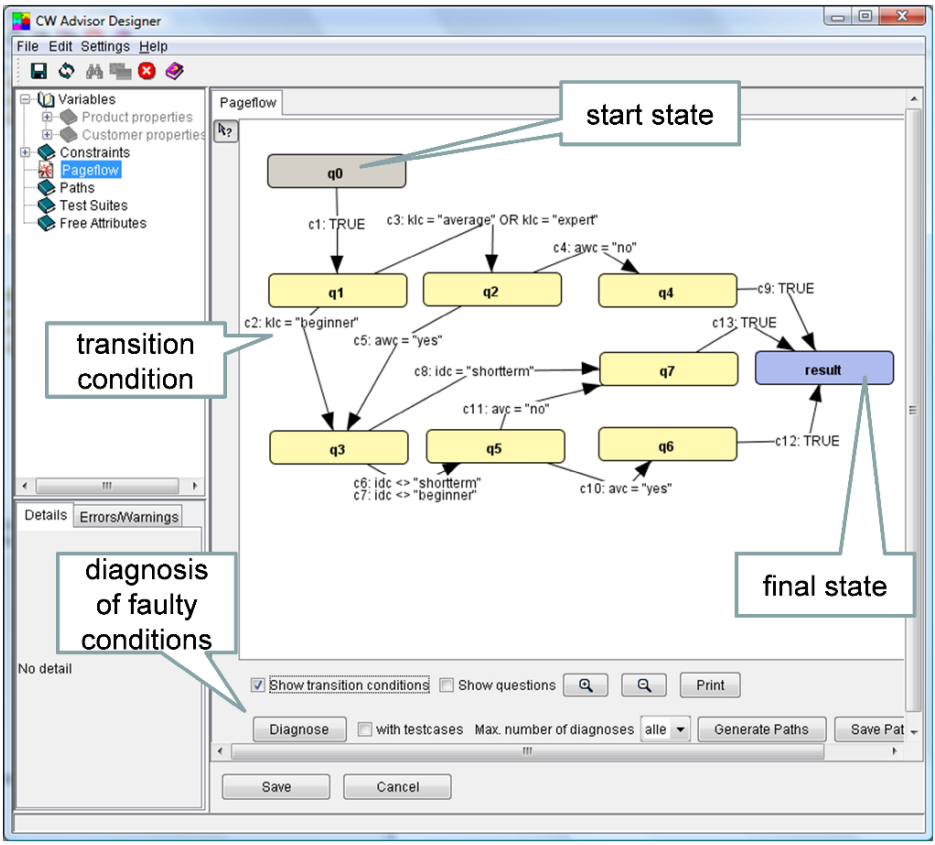

Fig. 6.3: CWAdvisor Designer Environment. Recommendation processes are specified on a graphical level and can be automatically translated into a corresponding executable representation. Faulty transition conditions can be identified automatically on the basis of model-based diagnosis [20].

est (see also Chapter 22). This is a common approach in web portals or social networking platforms.

2. Static fill-out forms per session: customers fill out a static web-based form every time they use the recommender system. Such interfaces are easy to implement and web users are well-acquainted with such interfaces, which are often used on web shops search for items.
3. Conversational recommendation dialogs: the recommender system incrementally acquires the user's preferences in an interactive dialog, based on, for example, "critiquing" [8] (see also Chapter 13), "wizard-like" and form-based preference elicitation dialogs [13], natural-language interaction [59] or a combination of these techniques.

In the context of constraint-based recommendation, particularly this last type of preference elicitation plays an important role and will be in the focus of this chapter, because recommendation in complex domains such as financial services [18] or electronic consumer goods [25] often induces a significant cognitive load on the end user interacting with the system. Thus, adequate user interfaces are required to make sure that the system is usable for a broad community of online users.

Of course, the static information available in some user-specified customer profile can also be an input source for a constraint-based recommender. The integration of such general profile information (including particularly demographic information) into the recommendation process is straightforward. In many cases, however, this information is rather unspecific and broad so that the utility of these information pieces is limited for an in-detail knowledge-based recommendation process.

Static fill-out forms for some applications work well for the above-mentioned reasons. However, in knowledge-intensive domains, for which constraint-based recommenders are often built, this approach might be too simplistic, particularly because the online user community can be heterogeneous with respect to their technical background, so that it is inappropriate to ask all users the same set of questions or at the same level of technical detail [25].

Finally, we will also not focus on natural language interaction in this chapter as only few examples such as [59] exist, that use a (complementing) natural language recommender system user interface. Despite the advances in the field of Natural Language Processing and although human-like virtual advisors can be found as an add-on to different web sites, they are barely used for recommending items to users today, for which there are different reasons. First, such dialogs are often user-driven, i.e., the user is expected to actively ask questions. In complex domains, however, in particular novice users are not capable of formulating such questions about, for example, the right medium-term investment strategy. In addition, the knowledge-acquisition effort for such systems is relatively high, as the system should also be capable of conducting casual conversation. Finally, end users often attribute more intelligence to such human-like avatars than is warranted which carries the risk of leaving them disappointed after interacting with the system.

Critiquing Critiquing is a popular interaction style for knowledge-based recommender systems, which was first proposed in [6] in the context of *Case-Based Reasoning* (CBR) approaches to conversational recommendation. The idea is to present individual items (instances), for example, digital cameras or financial products, to the user who can then interactively give feedback in terms of critiques on individual features. A user might, for instance, ask for a financial product with a "shorter investment period" or a "lower risk". This recommend-review-revise cycle is repeated until the desired item is found. Note that although this method was developed for CBR recommendation approaches[3], it can also be applied to constraint-based recommendation, as the critiques can be directly translated into additional constraints that reflect the user's directional preferences on some feature.

[3] The general idea of exploring a database by criticizing successive examples is in fact much older and was already proposed in the early 1980s in an information retrieval context [55].

When compared with detailed search forms that can be found on many online shops, the critiquing interaction style has the advantage that it supports the user in interactively exploring the item space. Moreover, the approach, which is often also called *tweaking*, is relatively easy to understand also for novice users. Developing a critiquing application, however, requires some domain knowledge, for example, about the set of features the user can give feedback, suitable increment values for number-valued attributes or logical orderings of attributes with enumeration domains. In addition, when mappings from customer needs to product features are needed, additional engineering effort is required.

The basic critiquing scheme was later on extended to also support *compound critiques* [44, 52], where users can give feedback on several features in a single interaction cycle. In the domain of financial services, a user could therefore ask for a product that has lower risk and a longer investment horizon in one step, thus decreasing the number of required interaction cycles. While some sort of pre-designed compound critiques were already possible in the initial proposal from [6], it is argued in [44] that the set of possible critiques should be dynamically determined depending on the remaining items in the current user's item space and in particular on the level of variation among these remaining items. The results of experimental evaluations show that such compound critiques can help to significantly reduce the number of required interaction cycles, thus making the whole interaction process more efficient. In addition, the experiments indicate that compound critiques (if limited to a size that is still understandable to the user) can also help the user understand the logic of the recommendations generated by the system.

Recent developments in critiquing include the use of elaborate visual interfaces [62], the application of the approach in mobile recommender systems [74], or the evaluation of critiquing styles regarding decision accuracy and cognitive effort [20].

Personalized preference elicitation dialogs Another form of acquiring the user's wishes and needs for a constraint-based recommender system is to rely on explicitly modeled and adaptive preference elicitation dialogs. Such dialog models can for instance be expressed using a *dialog grammar* [2] or by using finite-state automaton as done in the *CWAdvisor* system [20, 13].

In the later system, the end user is guided by a "virtual advisor" through a series of questions about the particular needs and requirements before a recommendation is displayed, see Figure 6.4 for an example dialog. In contrast to static fill-out forms, the set of questions is personalized, i.e., depending on the current situation and previous user answers, a different set of questions (probably also using a different technical or non-technical language [29]) will be asked by the system.

In the *CWAdvisor* system, the required user interface adaptation is based on manually-engineered personalization rules and on an explicit dialog model in the form of a finite-state automaton as shown in Figure 6.1. Thus, a method is chosen that represents a compromise between fill-out forms to which web users are well-acquainted and fully free conversation as provided by approaches based on Natural Language Processing.

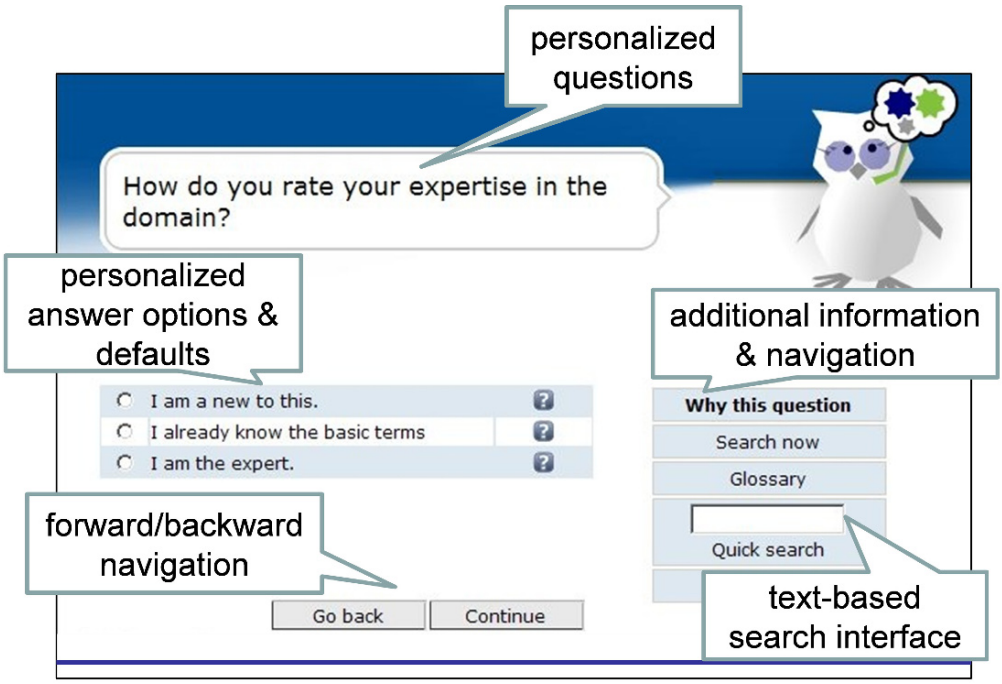

Fig. 6.4: Interactive and personalized preference elicitation example. Customers specify their preferences by answering questions.

Technically, the vertices of the finite-state automaton in Figure 6.1 are annotated with logical expressions over the constraint variables that are used to capture the user requirements. The process of developing the dialog and personalization model is supported in the *CWAdvisor* system by an end-user oriented graphical process modeling editor. At run time, the interaction-handling component of the framework collects the user inputs and evaluates the transition conditions in order to decide how to continue the dialog, see [13] for more details.

Beside the personalization of the dialog, different other forms of adaptation on the level of content, interaction and presentation are implemented in the system [30] in order to support the design of preference elicitation and explanation dialogs that support the end user in the best possible way.

While highly-dynamic and adaptive web applications can be valuable in terms of ease-of-use and user experience, the technical realization and in particular the maintenance of such flexible user interfaces for a constraint-based recommender can be challenging. The main problem in this context are the strong interrelationships between the "model", the "view" and the control logic of such applications: consider, for instance, the situation, where the dialog model should be extended with a new question (variable), a new answer option (new variable domain), or whole dialog page (new dialog automaton state). In each case, the web pages that represent

the "view" of the recommender application, have to be adapted accordingly. Therefore, toolkits for developing personalized preference elicitation processes, have to provide mechanisms to at least partially automate the process of updating the user interface, see [30] for details of the template-based approach in *CWAdvisor*.

Dealing with unfulfillable or too loose user requirements The issue of the development of the user interface is not the only challenging problem in the context of personalized preference elicitation in constraint-based recommenders. In the following, we will sketch further aspects that have to be dealt with in practical applications of this technology (see also Chapter 15 and Chapter 16).

In constraint-based recommenders, the situation can easily arise that no item in the catalog fulfills all the constraints of the user. During an interactive recommendation session, a message such as "no matching product found" is however highly undesirable. The question therefore arises, how to deal with such a situation that can also occur in CBR-based recommenders that in many cases at least initially rely on some query mechanism to retrieve an initial set of cases from the product catalog (case base). One possible approach proposed in the context of CBR-based recommenders is based on *query relaxation* [39, 43, 47, 23, 27]. In the context of CBR recommenders, the set of recommendable items are conceptually stored in a database table; the case retrieval process consists of sending a conjunctive query Q (of user requirements) to this case base. Query relaxation then refers to finding a (maximal) subquery Q' of the original query Q that returns at least one item.

The general idea of query relaxation techniques can also be applied in constraint-based recommendation. Consider Example 6.5 (adapted from [27]), where the catalog of four items C_{PROD} is shown in tabular form in Figure 6.5.

$name_p$	sl_p (type of low risk inv.)	ri_p (associated risk)	$mniv_p$ (min. investment period)	er_p (expected return)	$inst_p$ (financial institute)
p1	stockfunds	medium	4	5 %	ABank
p2	singleshares	high	3	5 %	ABank
p3	stockfunds	medium	2	4 %	BInvest
p4	singleshares	high	4	5 %	CMutual

Fig. 6.5: Example item catalog (financial services).

Example 6.5. Query Relaxation
For sake of clarity and simplicity of the example, let us assume that the customer can directly specify the desired properties of the investment product on an "expert screen" of the advisory application. The set of corresponding customer properties V_c thus contains sl_c (investment type), ri_c (risk class), $minimum\_return_c$ (minimum value for expected return) and $investment\_duration_c$ (desired investment duration). The filter constraints (conditions) in this example simple map customer requirements from C_c to item features, i.e., $C_F = \{CF_1 : sl_c = sl_p, CF_2 : ri_c = ri_p, CF_3 : investment\_duration_c >= mniv_p, CF_4 : er_p >= minimum\_return_c\}$

Let the concrete customer requirements C_C be as follows: $\{sl_c = singleshares, ri_c = medium, investment\_duration_c = 3, minimum\_return_c = 5\}$.

As can be easily seen, no item in the catalog (see Figure 6.5) fulfills all relevant constraints in the given task, i.e., no consistent recommendation can be found for the recommendation task. When following a "constraint relaxation" approach, the goal now consists of finding a maximal subset of the constraints of C_F, for which a recommendation can be found. The maximization criterion is typically chosen because the constraints directly relate to customer requirements, i.e., the more constraints can be retained, the better the retrieved items will match these requirements.

While this problem of finding consistency-establishing subsets of C_F seems to be not too complex at a first glance, in practical settings, computational effectiveness becomes an issue. Given a constraint base consisting of n constraints, the number of possible subsets is 2^n. Since real-world recommender systems have to serve many users in parallel and typically the acceptable response time is about one second, naive subset probing is not appropriate.

Different techniques have therefore been proposed to solve this problem more efficiently. In [39], for instance, an incremental mixed-initiative to recovery from failing queries in a CBR recommender was suggested. In [47], a relaxation method based on manually-defined feature hierarchies was proposed, which despite its incomplete nature has shown to be an effective help in a travel recommender system. Finally, in [27] and [26] a set of complete algorithms for the query relaxation problem in constraint-based recommenders was developed. The algorithms not only support the computation of minimum relaxations in linear time (at the cost of a preprocessing step and slightly increased memory requirements) but also the computation of relaxations that lead to "at least n" remaining items. In addition, also a conflict-directed algorithm for interactive and incremental query relaxation was proposed which makes use of recent conflict-detection technology [33].

The main idea of the linear-time constraint relaxation technique can be sketched as follows. Instead of testing combinations of constraints, the relevant constraints are evaluated individually, resulting in a data structure that assigns to every constraint the list of catalog items that fulfill the constraint, see Figure 6.6.

ID	Product p1	Product p2	Product p3	Product p4
CF_1	0	1	0	1
CF_2	1	0	1	0
CF_3	0	1	1	0
CF_4	1	1	0	1

Fig. 6.6: Evaluating the subqueries individually. For example, product p1 is filtered out by the filter condition CF_1 under the assumption that $sl_c = singleshares$.

The table should be interpreted as follows. Constraint CF_1 on the type of investment (single shares) in line 1 of the table would filter out products $p1$ and $p3$.

Given this table, it can be easily determined, which constraints of a given set C_F have to be relaxed to have a certain product in the result set, i.e., consistent with the constraints and the user requirements. For example, in order to have $p1$ in the result set, the constraints CF_1 and CF_3 of C_F have to be relaxed. Let us call this a "product-specific relaxation" for $p1$. The main idea of the method from [27] is that the overall "best" relaxation for given products C_{PROD}, filter conditions C_F and a given set of concrete requirements C_C has to be among the product-specific relaxations. Thus, it is sufficient to scan the set of product-specific relaxations, i.e., no further constraint solving step is required in this phase.

In the example, the relaxation of constraint CF_2 is optimal, when the number of relaxed constraints determines the best choice as only one customer requirement has to be given up. All other relaxations require at least two constraints to be ignored, which can be simply determined by counting the number of zeros in each column. Note that the number of involved constraints is only one possible optimization criterion. Other optimization criteria that take additional "costs of compromise" per constraint into account can also be implemented based on this technique as long as the cost function's value is monotonically increasing with the size of the relaxation.

Technically, the computation of product-specific relaxations can be done very efficiently based on bit-set operations [27]. In addition, the computation can partially also be done in advance in the start-up phase of the recommender.

Suggesting alternatives for unfulfillable requirements In some application domains, the automated or interactive relaxation of individual constraints alone may be not suffice as a means to help the user out of a situation, in which his or her requirements cannot be fulfilled. Consider, for instance, a situation where the recommender in an interactive relaxation scenario proposes a set of alternatives of constraints to be relaxed. Let us assume that the user accepts one of the proposals, i.e., agrees to relax the constraints related to two variables of V_C, for example, A and B. If, however, the values of A and B are particularly important to him (or mandatory), he will later on put different constraints on these variables. These new values can, however, again cause an inconsistency with the other requirements of the user. This might finally lead to an undesirable situation, in which the user ends up in trying out different values but gets no clear advise, which values to take to receive a consistent recommendation.

Overall, it would be thus desirable, if the system could immediately come up with suggestions for new values for A and B, for which it is guaranteed that some items remain in the result set when the user's other requirements are also further taken into account.

Let us first consider the basic CBR-style case retrieval problem setting as used in [39, 43, 47], in which constraints are directly placed on item features. The constraints in this example shall be $\{sl_p = singleshares,\ ri_p = medium,\ minv_p < 3,\ er_p >= 5\ \}$. Again, no item fulfills these requirements.

In such a setting, the detailed information about the catalog items can be used to compute a set of suggestions for alternative constraints ("repairs") on individual features. Based on this information, the system could – instead of only proposing the

user to relax the constraints on the investment type and on the investment duration – inform the user that "if the single shares requirement is abandoned and the minimum investment duration is set to 4" one or more items will be found. Thus, the user will be prevented from (unsuccessfully) trying a minimum investment duration of 3.

In this example, the calculation of such alternative values can be accomplished by the system by choosing one relaxation alternative (investment duration and investment type) and searching the catalog for items that fulfill the remaining constraints. The values for the investment duration and the investment type (e.g., of product 1 in Figure 6.5) can be directly taken as suggestions for the end user [19] [14].

While this approach seems intuitive and simple, in practical applications the following problems have to be dealt with.

- The number of possible repairs. In realistic scenarios, the number of possible repair alternatives is typically very large as for every possible relaxation – there might be already many of them – various solutions exist. In practice, however, end users cannot be confronted with more than a few overall alternatives. Thus, the problem exists to select and prioritize the repair alternatives.
- The size/length of the repair proposals. Repair suggestions that contain alternative values for more than three features are not easy to understand for end users.
- Computational complexity for non-trivial constraints. When only simple constraints on product features are allowed, the information from the item catalog can help to determine possible repairs as described above. In constraint-based systems such as *CWAdvisor*, however, the definition of constraints that relate often qualitative user needs to (technical) product features is possible. Consequently, also the repair suggestions must relate to user requirements, which means that the search space of possible repair alternatives is determined by the domains of the user-related variables. In addition, determining whether or not a specific combination of user requirements (i.e., a repair alternative) leads to a non-empty result set, requires a probably costly catalog query.

In order to address these issues at least to some extent, the *CWAdvisor* system uses a combination of query relaxation and different search heuristics and additional domain-specific knowledge for the calculation of repair suggestions in a financial services application [17].

The method implemented in the system interleaves the search for relaxations with a bounded search for repair alternatives. The possible relaxations are determined in increasing order of their cardinality. For each relaxation, repair alternatives are determined by varying the values of the variables that are involved in the relaxed constraints. The selection of alternative values can for instance be guided by a "nearness" heuristic that is based on an externally or implicitly defined order of the values. Thus, when varying for instance a user requirement of "at least 5 % expected return", the neighboring value of "4 %" is evaluated, assuming that such an alternative will be more acceptable for the end user than an even stronger relaxation. In order to avoid too many similar repair suggestions, the algorithm can be parameterized with several threshold values that, for example, determine the number

of repairs for a relaxation, the maximum size of a relaxation and so forth. Overall, anecdotal evidence in the financial service domain indicates that such a repair feature, even if it is based on heuristics, is well-appreciated by end users as a means for shortening the required dialog length.

Query tightening Beside having no item in the result set, having *too many* items in the result set is also not desirable in an interactive recommender. In many real-world applications the user is informed that "too many items have been found" and that more precise search constraints have to be specified. Often, only the first few results are displayed (as to, e.g., avoid long page loading times). Such a selection may however not be optimal for the current user, since the selection is often simply based on the alphabetic order of the catalog entries.

In order to better support the user also in this situation, in [48] an *Interactive Query Management* approach for CBR recommenders is proposed, that also includes techniques for "query tightening". The proposed tightening algorithms takes as an input a query Q and its large result set and selects – on the basis of information-theoretic considerations and the entropy measure – three features that are presented to the user as proposals to refine the query.

Overall, an evaluation of *Interactive Query Management* within a travel recommender system that implemented both query relaxation and query tightening [50], revealed that the relaxation feature was well-appreciated by end users. With respect to the tightening functionality, the evaluation indicated that query tightening was not that important to end users who were well capable of refining their queries by themselves. Thus, in [41] a different feature selection method was proposed, that also take a probabilistic model of feature popularity into account. An evaluation showed that in certain situations the method of [41] is preferable since it is better accepted by end users as a means to further refine their queries.

6.4 Calculating Recommendations

Following our characterization of a recommendation task (see Definition 1), we will now discuss corresponding problem solving approaches. Typical approaches to solve a recommendation task are *constraint satisfaction algorithms* [54] and *conjunctive database queries* [46] .

Constraint Satisfaction Solutions for *constraint satisfaction problems* are calculated on the basis of search algorithms that use different combinations of *backtracking* and *constraint propagation* - the basic principle of both concepts will be explained in the following.

Backtracking. In each step, backtracking chooses a variable and assigns all the possible values to this variable. It checks the consistency of the assignment with the already existing assignments and defined set of constraints. If all the possible values of the current variable are inconsistent with the existing assignments and the constraints, the constraint solver backtracks which means that the previously

instantiated variable is selected again. In the case that a consistent assignment has been identified, a recursive activation of the backtracking algorithm is performed and the next variable is selected [54].

Constraint Propagation. The major disadvantage of pure backtracking-based search is "trashing" where parts of the search space are revisited although no solution exists in these parts. In order to make constraint solving more efficient, constraint propagation techniques have been introduced. These techniques try to modify an existing constraint satisfaction problem such that the search space can be reduced significantly. The methods try to create a state of *local consistency* that guarantees consistent instantiations among groups of variables. The mentioned modification steps turn an existing constraint satisfaction problem into an equivalent one. A well known type of local consistency is *arc consistency* [54] which states that for two variables X and Y there must not exist a value in the domain of Y which does not have a corresponding consistent value in X. Thus, arc consistency is a directed concept which means that if X is arc consistent with Y, the reverse must not necessarily be the case.

When using a constraint solver, constraints are typically represented in the form of expressions of the corresponding programming language. Many of the existing constraint solvers are implemented on the basis of Java (see, for example, *jacop.osolpro.com*).

Conjunctive Database Queries Solutions to *conjunctive queries* are calculated on the basis of database queries that try to retrieve items which fulfill all of the defined customer requirements. For details on the database technologies and the execution of queries on database tables see, for example, [46].

Ranking Items Given a recommendation task, both constraint solvers and database engines try to identify a set of items that fulfill the given customer requirements. Typically, we have to deal with situations where more than one item is part of a recommendation result. In such situations the items (products) in the result set have to be ranked. In both cases (constraint solvers and database engines), we can apply the concepts of multi-attribute utility theory (MAUT) [56] that helps to determine a ranking for each of the items in the result set. Examples for the application of MAUT can be found in [13].

An alternative to the application of *MAUT in combination with conjunctive queries* are *probabilistic databases* [35] which allow a direct specification of ranking criteria within a query. Example 6.6 shows such a query which selects all products that fulfill the criteria in the WHERE clause and orders the result conform to a similarity metric (defined in the ORDER BY clause). Finally, instead of combining the mentioned *standard constraint solvers with MAUT*, we can represent a recommendation task in the form of soft constraints where the importance (preference) for each combination of variable values is determined on the basis of a corresponding utility operation (for details see, for example, [1]).

Example 6.6. Queries in probabilistic databases

```
Result = SELECT *     /* calculate a solution */
FROM Products     /* select items from "Products" */
```

WHERE x_1=a_1 and x_2=a_2 /* "must" criteria */
ORDER BY score(abs(x_3-a_3), ..., abs(x_m-a_m)) /* similarity-based utility function */
STOP AFTER N; /* at most N items in the solution (result set) */

6.5 Experiences from Projects and Case Studies

The *CWAdvisor* system has been commercialized in 2002 and since then more than 35 different applications have been instantiated and fielded. They have been applied in commercial domains ranging from financial services [17] to electronic consumer goods or tourism applications [32] as well as to application domains that are considered rather untypical for recommender systems such as providing counseling services on business plans [28] or supporting software engineers in selecting appropriate software estimation methods [43].

Based on this installation base different forms of *empirical research* have been conducted that try to assess the impact and business value of knowledge based recommender systems as well as to identify opportunities for advancing their state-of-the-art. In the following we will differentiate them based on their study design into *user studies*, *evaluations on historical data* and *case studies of productive systems*.

Experimental user studies simulate real user interactions and research the acceptance or rejection of different hypotheses. [15] conducted a study to evaluate the impact of specific functionalities of conversational knowledge-based recommenders like explanations, proposed repair actions or product comparisons. The study assigned users randomly to different versions of the recommender system that varied the functionalities and applied pre- and post-interaction surveys to identify users' level of knowledge in the domain, their trust in the system or the perceived competence of the recommender. Quite interestingly, the study showed that study participants appreciate these specific functionalities as they increase their perceived level of knowledge in the domain and their trust in the system's recommendations.

The COHAVE project initiated a line of research that investigated how psychological theories might be applied to explain users' behavior in online choice situations. For instance, asymmetric dominance effects arise if proposed itemsets contain decoy products that are dominated by other products due to their relative similarity but a lower overall utility. Several user studies in domains such as electronic consumer goods, tourism and financial services showed, that knowing about these effects a recommender can increase the conversion rate of some specific items as well as a users confidence in the buying decision.

Algorithm evaluations on historical datasets are off-line experimentations [25]. A dataset that contains past user transactions is split into a training and testing set. Consequently, the training set is exploited to learn a model or tune algorithm's parameters in order to enable the recommender to predict the historic outcomes of the user sessions contained in the testing set. Such an evaluation scenario enables comparative research on algorithm performance. While collaborative and content-

based recommendation paradigms have been extensively evaluated in the literature, comparing knowledge-based recommendation algorithms with other recommendation paradigms received only few attention in the past. One reason is that they are hard to compare, because they require different types of algorithm input: collaborative filtering typically exploits user ratings while constraint-based recommender systems require explicit user requirements, catalog data and domain knowledge. Consequently, datasets that contain all these types of input data - like the Entree dataset provided by Burke [14] - would allow such comparisons, however they are very rare. One of the few is described in [61]. The dataset stems from a retailer offering premium cigars and includes implicit ratings signifying users' purchase actions, users' requirements input to a conversational recommender and a product catalog with detailed item descriptions. Therefore, offline experiments compared knowledge-based algorithm variants that exploited user requirements with content-based and collaborative algorithms working on ratings. One of the interesting results were that knowledge-based recommenders did not perform worse in terms of serendipity measured by the catalog coverage metric than collaborative filtering. This is especially true if a constraint-based recommender is cascaded with a utility-based item ranking scheme like the *CWAdvisor* system. However, collaborative filtering does better in terms of accuracy, if there are 10 and more ratings known from users. Nevertheless, an evaluation of a knowledge-based recommender always measures the quality of the encoded knowledge base *and* the inferencing itself.

Another study was instrumented in [60] that focuses on explicit user requirements as the sole input for personalization mechanisms. It compares different hybridization variants between knowledge-based and collaborative algorithms, where collaborative filtering interprets explicit requirements as a form of rating. Result sets of knowledge-based recommenders turn out to be very precise, if users formulated some specific requirements. However, when only few constraints apply and result sets are large the ranking function is not always able to identify the best matching items. In contrast, collaborative filtering learns the relationships between requirements and actually purchased items. Therefore, the study shows that a cascading strategy where the knowledge-based recommender removes candidates based on hard criteria and a collaborative algorithm does the ranking does best.

Consequently, in [57] a meta-level hybridization approach between knowledge-based and collaborative filtering was proposed and validated. There collaborative filtering learns constraints that map users' requirements onto catalog properties of purchased items and feeds them as input into a knowledge-based recommender that acts as the principal component. Offline experiments on historical data provided initial evidence that such an approach is able to outperform the knowledge base elicited from the domain experts with respect to algorithm's accuracy. Based on these first promising results further research on automatically extracting constraints from historic transaction data will take place.

Case studies on productive systems are the most realistic form of evaluation because users act under real-world conditions and possess an intrinsic motivation to use the system. In [13] experiences from two commercial projects in the domains

of financial services and electronic consumer goods are reported. In the latter domain a conversational recommender for digital cameras has been fielded that was utilized by more than 200.000 online shoppers at a large Austrian price comparison platform. Replies to an online questionnaire supported the hypothesis that advisor applications help users to better orientate themselves when being confronted with large sets of choices. A significantly higher share of users successfully completed their product search when using the conversational recommender compared to those that did not use it. Installations of knowledge-based recommenders in the financial services domain follow a different business model as they support sales agents while interacting with their prospective clients. Empirical surveys among sales representatives figured out that time savings when interacting with clients are a big advantage which in turn allows sales staff to identify sales opportunities [13, 17].

In [58] a case study researches how the application of a knowledge-based conversational sales recommender on a Webshop for Cuban cigars affects online shoppers behavior. Therefore the sales records in the period before and after introducing the recommender were analyzed. One interesting finding of this study is that the list of top ranked items in the two periods differs considerably. In fact items that were infrequently sold in the period before but very often recommended by the system experienced a very high demand. Thus the relative increase of items was positively correlated with how often the recommender proposed these items. The advice given by recommendation applications is followed by users and leads to online conversions. This confirms the results of user studies like [15] that were initially discussed. Finally, another evaluation of a knowledge-based recommender in the tourism domain was conducted to compare conversion rates, i.e., the share of users that turned into bookers, between users and non-users of the interactive sales guide [59]. This study strongly empirically confirms that the probability of users issuing a booking request is more than twice as high for those having interacted with the interactive travel advisor than for the others.

Thus, based on these results we are able to summarize that constraint-based recommendation has been successfully deployed in several commercial application domains and is well accepted by their users.

6.6 Future Research Issues

On the one hand constraint-based recommender systems have proven their utility in many fielded applications on the other hand we can identify several challenges for improvements. Such improvements will lead to enhancing the *quality* for users, the *broadness* of the application fields, and the *development* of recommender software.

Automated product data extraction A constraint-based recommender is only as good as its knowledge base. Consequently, the knowledge base has to be correct, complete, and up-to-date in order to guarantee high quality recommendations. This implies significant maintenance tasks, especially in those domains where data and

recommendation knowledge changes frequently, for example, electronic consumer products. Currently, maintenance is done by human experts, for example, collecting product data or updating rule-bases. However, in many domains at least product data is accessible for machines on the web. By exploiting the internet as a resource for data and knowledge almost all necessary pieces for many recommender applications could be collected. The major research focuses in this context are the automated extraction of product data from different information sources and the automated detection and adaptation of outdated product data. This includes identifying relevant information sources (for instance, Web pages), extracting product data, and resolving contradictions in those data. A related recent challenge is extracting product information directly from digital multimedia products such as books, CDs, DVDs, and TV programs.

However, the fundamental problem for machines is the presentation of data in the web. Data in the Web is usually presented with the goal that humans can easily access and comprehend information. Unfortunately, the opposite is true for computers which are currently not particulary capable in interpreting visual information. Therefore, a fundamental research question is how we can enable machines such that they can "read" the web similarly as humans do. In fact, this task goes far beyond recommender systems and is a central endeavor of the Semantic Web and on a more general level of Artificial Intelligence. Although it seems that currently this task is far too ambitious to be solved in the near future, we can exploit the particularities of recommendation domains. For example, when dealing with the extraction of product data from the web, we can search for product descriptions in tabular form, extract the data of these product descriptions, and instantiate a product database [31]. Of course the success of such methods depends on the domains. For example in the domain of electronic consumer products like digital cameras the description of cameras follows a common structure (e.g., data-sheets of different brands are very similar) whereas in other domains like holiday resorts product descriptions are mostly expressed by natural language text. It has to be mentioned that instead of an automatic translation of human readable content in machine processable data there is the alternative to provide such machine processable data in addition or instead of human readable content. Indeed strong market forces like internet search engine vendors might offer improved search services if machine processable information is provided. For example, product vendors supply their data in specific formats and benefit by an improved ranking in search results. However, in this scenario search machine vendors dictate which descriptions of which products are available for recommendations purposes which leads to a strong dependency on single authorities. Therefore, the aim to enable computers to read the web as humans do remains an important point on the research agenda.

Community-based knowledge acquisition The cornerstone of constraint-based recommendation is efficient knowledge acquisition and maintenance. This problem has been addressed in the past in different dimensions, the main focus lying on knowledge representation and conceptualization issues as well as on process models for capturing and formalizing a domain expert's knowledge. Historically, one

main assumption of these approaches was that there shall exist one single point of knowledge formalization and in consequence one (user-oriented) conceptualization and a central knowledge acquisition tool. In most cases in real world, however, the domain knowledge is in the heads of different stakeholders, typical examples being cross-department or cross-organization business rules or new types of applications, in which large user communities are sharing knowledge in an open-innovation , web-based environment. Only recently, with the emergence and spread of Web 2.0 and Semantic Web technologies, the opportunities and also the problems of collaborative knowledge acquisition have again become a topic of interest. With regard to the types of knowledge to be acquired, the main focus of these recent developments, however, is on acquiring "structural" knowledge, i.e., on terms, concepts, and relationships among them. New developments aim at going a step further and target at the collaborative acquisition and refinement of domain-constraints and business rules as they represent the most crucial, frequently updated, and thus costly part in many knowledge-based applications. The main questions to be answered comprise the following: How can we automatically detect and resolve conflicts if knowledge acquisition is distributed between different knowledge contributors? How can we assist the knowledge contributors to acquire knowledge by asking them the "right" questions, i.e., minimizing the interaction needed? How can we generate "good" proposals for changing the knowledge base from different, possibly only partially-defined knowledge chunks, i.e., find plausible (in the eyes of the contributors) changes of the knowledge base?

Usually the term *knowledge acquisition* refers to methods supporting the user to formulate rules, constraints, or other logical descriptions depending on the employed language. This task is complicated in recommender systems since in most cases the output includes a preference relation over the recommended items. Consequently, knowledge acquisition has to support also the formulation, debugging, and testing of such preference descriptions [21].

A further factor which complicates the search for a satisfying knowledge base is the demand for high quality explanations . Explanations in constraint-based recommender systems are generated by exploiting the content of the knowledge base. In fact, different knowledge bases can provide the equivalent input/output behavior with respect to recommendations but show significant differences in their explanatory quality. Consequently, a further important goal of knowledge acquisition is supporting the formulation of comprehensible knowledge bases which serve the user to gain confidence in the recommendations.

Knowledge bases for recommender systems have to be considered as dynamic. Unfortunately this dynamics are not only caused by changing product catalogs but also by shifts of customer preferences. For example, the pixel resolution of digital photos considered to be sufficient for printing an A4 picture changes over time because of higher demands for quality. Consequently, automatic detection of such shifts and supporting a subsequent adaptation of the knowledge base are of great interest.

Validation Successfully developing and maintaining recommender knowledge bases requires intelligent testing environments that can guarantee the recommendations' correctness. Particularly in application areas where a certain recommendation quality must be assured (e.g., financial products) a company employing recommender systems has to be sure about the quality of the recommendation process and its outcome. So, future research must focus on developing mechanisms to automatically configure optimal test suites that both maximize the probability of identifying faulty elements in the recommender knowledge base and minimize the number of test cases needed to achieve this goal. Minimizing the number of test cases is important because domain experts must validate them manually. This validation output fits nicely with supporting knowledge acquisition since any feedback from a knowledge engineer can be exploited for learning recommendation knowledge bases. In particular an interesting research question is to which extend arguments of a user in favor or against a recommendation can be exploited to improve knowledge bases. In [51] an algorithm is described which learns constraints based on arguments why an example (e.g., a product) should be recommended or not.

Recommendation of configurable products and services With the production of the Model T about 100 years ago, Henry Ford revolutionized manufacturing by employing mass production (the efficient production of many identical products). Nowadays, mass production is an outmoded business model, and companies must provide goods and services that fit customers' individual needs. In this context, mass customization – the production of highly variant products and services under mass production pricing conditions – has become the new paradigm. A phenomenon accompanying mass customization is mass confusion, which occurs when items are too numerous and complex for users to survey. Developing recommender technologies that apply to configurable products and services can help tackle mass confusion. For example, recommender technology could be adapted to help the uninformed customer to discover her wishes, needs, and product requirements in a domain of almost unlimited product variants. However, recommendation of configurable products pushes the limits of current recommender technologies. Current techniques assume that items to be recommended can be extensionally represented. But configuration domains frequently offer such a high product variance that the set of all possible configurations can only be intensionally characterized by configuration descriptions. For example, configurable systems may comprise thousand of components and connections. In these domains searching for the most preferred configurations satisfying the customer requirements is a challenging task.

Intelligibility and explanation To be convincing, recommendations must be explained to customers. When they can challenge a recommendation and see why a system recommended a specific product customers will start to trust that system. In general, explanations are provided for outputs of recommender systems and serve a wide spectrum of tasks, for example, increase transparency and trust, persuade a customer, or improve customer satisfaction just to name some. These explanations depend on the state of the recommendation process and the user profile, for example, her aims, desires, and prior knowledge. The vision of future recommender

systems is that pro-actively information is provided to the user such that explanation goals are optimized, i.e., if the recommender recognizes that a customer does not understand the differences between alternative products then explanations of these differences are offered. Conversely, customers with a rich background of a product domain and a clear understanding what they want can be offered a quick jump to a recommendation with a detailed technical justification. Consequently, the research challenge is to create an artificial recommender agent that acts flexibly to the needs of customers. Explanations are a cornerstone in such a general endeavor.

Theories of consumer buying behavior A truly intelligent recommender agent adapts to the user. This implies that the recommender has a model of the user which allows predictions about her reaction depending on the information provided. In particular, if we have a model about the influencing factors of consumer buying behavior then it is possible to reason about the best next actions a recommender agent can take. Therefore, research in recommender technology can greatly benefit from insights of cognitive and decision psychology. One can argue that such "intelligent" behavior of recommender agents is questionable from an ethical point of view. However, every information provided to a customer influences her buying behavior. Therefore, it is important to understand the consequences of communications with the customer thus allowing a more planned design of recommender systems.

Context awareness and ambient intelligence Recommender systems may not only be regarded as simple software tools accessible via a PC but rather as intelligent agents recommending actions in various situations. For example, in future cars artificial assistants will provide advice for various driving tasks, such as, overtaking, turning, or parking. In order to give recommendations in such environments the recommender has to be aware of the situation and the goals of a user. Other typical scenarios are recommendations for tourists during their journeys. In such situations, recommendations depend not only on customer preferences but also on the context, which can include attributes such as time of day, season, weather conditions, and ticket availability. Note, that the mentioned scenarios requires so called ambient intelligence. Not the traditional computer is the only interface to the customer but speech and gesture play an important role for the communication between user and recommender.

Semantic Web The W3C states "The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries." In particular Semantic Web technologies offer methods to relate data in the web. This can be exploited to implement a decentralized web of entities who trust each other or relations between customers and products they rated. Based on such relations between customers or products many improvements are feasible. We already mentioned that the extraction of product data and knowledge acquisition can benefit from the machine readable content descriptions. However, we can go a step further and use the information in the Semantic Web to improve the quality of recommendations [22, 63]. In particular, an agent can consider only those ratings of trustworthy agents in order to avoid intentional misguidance. Further-

more, the Semantic Web allows to integrate data of various sources in the reasoning process. On the one hand this enhances knowledge-based recommendation since knowledge is contributed and maintained by a community on a decentralized computing infrastructure and therefore knowledge-acquisition efforts are shared. However, on the other hand many research questions for this scenario arise: How can the quality of recommendations be guaranteed or at least assessed? How can we assess the trustworthiness and quality of knowledge sources? How can we make sure that for the description of products and services there is a common agreement on the concepts and values used? How can we deal with differences in the meaning of concepts and values? How can we assess not only the correctness of recommendations but also their completeness?

6.7 Summary

In this chapter we provided an overview of major constraint-based recommendation technologies. These technologies are especially applicable to large and potentially complex product assortments where collaborative filtering and content-based filtering technologies have their drawbacks. The usefulness of constraint-based recommendation technologies has been shown in different commercial applications - those applications have been analyzed in this chapter. Finally, to trigger further research in the field, we provide an extensive overview of important future research directions.

References

1. Bistarelli, S., Montanary, U., Rossi, F.: Semiring-based Constraint Satisfaction and Optimization. Journal of the ACM **44**, 201–236 (1997)
2. Bridge, D.: Towards Conversational Recommender Systems: a Dialogue Grammar Approach. In: D.W. Aha (ed.) Proceedings of the EWCBR-02 Workshop on Mixed Initiative CBR, pp. 9–22 (2002)
3. Bridge, D., Goeker, M., McGinty, L., Smyth, B.: Case-based recommender systems. Knowledge Engineering Review **20**(3), 315–320 (2005)
4. Burke, R.: Knowledge-Based Recommender Systems. Encyclopedia of Library and Information Science **69**(32), 180–200 (2000)
5. Burke, R.: Hybrid Recommender Systems: Survey and Experiments. User Modeling and User-Adapted Interaction **12(4)**, 331–370 (2002)
6. Burke, R., Hammond, K., Young, B.: Knowledge-based navigation of complex information spaces. In: Proceedings of the 13th National Conference on Artificial Intelligence, AAAI'96, pp. 462–468. AAAI Press (1996)
7. Burke, R., Hammond, K., Young, B.: The FindMe Approach to Assisted Browsing. IEEE Intelligent Systems **12**(4), 32–40 (1997)
8. Burnett, M.: HCI research regarding end-user requirement specification: a tutorial. Knowledge-based Systems **16**, 341–349 (2003)
9. Chen, L., Pu, P.: Evaluating Critiquing-based Recommender Agents. In: Proceedings of the 21st National Conference on Artificial Intelligence and the Eighteenth Innovative Ap-

plications of Artificial Intelligence Conference, AAAI/IAAI'06, pp. 157–162. AAAI Press, Boston, Massachusetts, USA (2006)
10. Felfernig, A.: Reducing Development and Maintenance Efforts for Web-based Recommender Applications. Web Engineering and Technology **3**(3), 329–351 (2007)
11. Felfernig, A., Burke, R.: Constraint-based recommender systems: technologies and research issues. In: Proceedings of the 10th International Conference on Electronic Commerce, ICEC'08, pp. 1–10. ACM, New York, NY, USA (2008)
12. Felfernig, A., Friedrich, G., Jannach, D., Stumptner, M.: Consistency-based diagnosis of configuration knowledge bases. Artificial Intelligence **152**(2), 213–234 (2004)
13. Felfernig, A., Friedrich, G., Jannach, D., Zanker, M.: An integrated environment for the development of knowledge-based recommender applications. International Journal of Electronic Commerce **11**(2), 11–34 (2007)
14. Felfernig, A., Friedrich, G., Schubert, M., Mandl, M., Mairitsch, M., Teppan, E.: Plausible Repairs for Inconsistent Requirements. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI'09, pp. 791–796. Pasadena, CA, USA (2009)
15. Felfernig, A., Gula, B.: An Empirical Study on Consumer Behavior in the Interaction with Knowledge-based Recommender Applications. In: Proceedings of the 8th IEEE International Conference on E-Commerce Technology (CEC 2006) / Third IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (EEE 2006), p. 37 (2006)
16. Felfernig, A., Isak, K., Kruggel, T.: Testing Knowledge-based Recommender Systems. OEGAI Journal **4**, 12–18 (2007)
17. Felfernig, A., Isak, K., Szabo, K., Zachar, P.: The VITA Financial Services Sales Support Environment. In: Proceedings of the 22nd AAAI Conference on Artificial Intelligence and the 19th Conference on Innovative Applications of Artificial Intelligence, AAAI/IAAI'07, pp. 1692–1699. Vancouver, Canada (2007)
18. Felfernig, A., Kiener, A.: Knowledge-based Interactive Selling of Financial Services using FSAdvisor. In: Proceedings of the 20th National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference, AAAI/IAAI'05, pp. 1475–1482. AAAI Press, Pittsburgh, PA (2005)
19. Felfernig, A., Mairitsch, M., Mandl, M., Schubert, M., Teppan, E.: Utility-based Repair of Inconsistent Requirements. In: Proceedings of the 22nd International Conference on Industrial, Engineering and Other Applications of Applied Intelligence Systems, IEAAIE 2009, Springer Lecture Notes on Artificial Intelligence, pp. 162–171. Springer, Taiwan (2009)
20. Felfernig, A., Shchekotykhin, K.: Debugging user interface descriptions of knowledge-based recommender applications. In: Proceedings of the 11th International Conference on Intelligent User Interfaces, IUI 2006, pp. 234–241. ACM Press, New York, NY, USA (2006)
21. Felfernig, A., Teppan, E., Friedrich, G., Isak, K.: Intelligent debugging and repair of utility constraint sets in knowledge-based recommender applications. In: Proceedings of the ACM International Conference on Intelligent User Interfaces, IUI 2008, pp. 217–226 (2008)
22. Gil, Y., Motta, E., Benjamins, V., Musen, M. (eds.): The Semantic Web - ISWC 2005, 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005, *Lecture Notes in Computer Science*, vol. 3729. Springer (2005)
23. Godfrey, P.: Minimization in Cooperative Response to Failing Database Queries. International Journal of Cooperative Information Systems **6**(2), 95–149 (1997)
24. Herlocker, J., Konstan, J., Terveen, L., Riedl, J.: Evaluating Collaborative Filtering Recommender Systems. ACM Transactions on Information Systems **22(1)**, 5–53 (2004)
25. Jannach, D.: Advisor Suite - A knowledge-based sales advisory system. In: R.L. de Mantaras, L. Saitta (eds.) Proceedings of European Conference on Artificial Intelligence, ECAI 2004, pp. 720–724. IOS Press, Valencia, Spain (2004)
26. Jannach, D.: Techniques for Fast Query Relaxation in Content-based Recommender Systems. In: C. Freksa, M. Kohlhase, K. Schill (eds.) Proceedings of the 29th German Conference on AI, KI 2006, pp. 49–63. Springer LNAI 4314, Bremen, Germany (2006)
27. Jannach, D.: Fast computation of query relaxations for knowledge-based recommenders. AI Communications **22**(4), 235–248 (2009)

28. Jannach, D., Bundgaard-Joergensen, U.: SAT: A Web-Based Interactive Advisor For Investor-Ready Business Plans. In: Proceedings of International Conference on e-Business, pp. 99–106 (2007)
29. Jannach, D., Kreutler, G.: Personalized User Preference Elicitation for e-Services. In: Proceedings of the IEEE International Conference on e-Technology, e-Commerce, and e-Services, EEE 2005, pp. 604–611. IEEE Computer Society, Hong Kong (2005)
30. Jannach, D., Kreutler, G.: Rapid Development Of Knowledge-Based Conversational Recommender Applications With Advisor Suite. Journal of Web Engineering **6**, 165–192 (2007)
31. Jannach, D., Shchekotykhin, K., Friedrich, G.: Automated Ontology Instantiation from Tabular Web Sources - The AllRight System. Journal of Web Semantics **7**(3), 136–153 (2009)
32. Jannach, D., Zanker, M., Fuchs, M.: Constraint-based recommendation in tourism: A multi-perspective case study. Information Technology and Tourism **11**(2), 139–156 (2009)
33. Junker, U.: QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. In: Proceedings of National Conference on Artificial Intelligence, AAAI'04, pp. 167–172. AAAI Press, San Jose (2004)
34. Konstan, J., Miller, N., Maltz, D., Herlocker, J., Gordon, R., Riedl, J.: GroupLens: applying collaborative filtering to Usenet news. Communications of the ACM **40**(3), 77–87 (1997)
35. Lakshmanan, L., Leone, N., Ross, R., Subrahmanian, V.: ProbView: A Flexible Probabilistic Database System. ACM Transactions on Database Systems **22**(3), 419–469 (1997)
36. Lorenzi, F., Ricci, F., Tostes, R., Brasil, R.: Case-based recommender systems: A unifying view. In: Intelligent Techniques in Web Personalisation, no. 3169 in Lecture Notes in Computer Science, pp. 89–113. Springer (2005)
37. Mahmood, T., Ricci, F.: Learning and adaptivity in interactive recommender systems. In: Proceedings of the 9th International Conference on Electronic Commerce, ICEC'07, pp. 75–84. ACM Press, New York, NY, USA (2007)
38. Maimon, O., Rokach, L. Data Mining by Attribute Decomposition with semiconductors manufacturing case study, in Data Mining for Design and Manufacturing: Methods and Applications, D. Braha (ed.), Kluwer Academic Publishers, pp. 311–336 (2001)
39. McSherry, D.: Incremental Relaxation of Unsuccessful Queries. In: P. Funk, P.G. Calero (eds.) Proceedings of the European Conference on Case-based Reasoning, ECCBR 2004, no. 3155 in Lecture Notes in Artificial Intelligence, pp. 331–345. Springer (2004)
40. McSherry, D.: Retrieval Failure and Recovery in Recommender Systems. Artificial Intelligence Review **24**(3-4), 319–338 (2005)
41. Mirzadeh, N., Ricci, F., Bansal, M.: Feature Selection Methods for Conversational Recommender Systems. In: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service on e-Technology, e-Commerce and e-Service, EEE 2005, pp. 772–777. IEEE Computer Society, Washington, DC, USA (2005)
42. Pazzani, M.: A Framework for Collaborative, Content-Based and Demographic Filtering. Artificial Intelligence Review **13**(5-6), 393–408 (1999)
43. Peischl, B., Nica, M., Zanker, M., Schmid, W.: Recommending effort estimation methods for software project management. In: Proceedings of the International Conference on Web Intelligence and Intelligent Agent Technology - WPRRS Workshop, vol. 3, pp. 77–80. Milano, Italy (2009)
44. Reilly, J., McCarthy, K., McGinty, L., Smyth, B.: Dynamic Critiquing. In: Proceedings of the 7th European Conference on Case-based Reasoning, ECCBR 2004, pp. 763–777. Madrid, Spain (2004)
45. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence **32**(1), 57–95 (1987)
46. R.Elmasri, Navathe, S.: Fundamentals of Database Systems. Addison Wesley (2006)
47. Ricci, F., Mirzadeh, N., Bansal, M.: Supporting User Query Relaxation in a Recommender System. In: Proceedings of the 5th International Conference in E-Commerce and Web-Technologies, EC-Web 2004, pp. 31–40. Zaragoza, Spain (2004)

48. Ricci, F., Mirzadeh, N., Venturini, A.: Intelligent query management in a mediator architecture. In: Proceedings of the 1st International IEEE Symposium on Intelligent Systems, vol. 1, pp. 221–226. Varna, Bulgaria (2002)
49. Ricci, F., Nguyen, Q.: Acquiring and Revising Preferences in a Critique-Based Mobile Recommender System. IEEE Intelligent Systems **22**(3), 22–29 (2007)
50. Ricci, F., Venturini, A., Cavada, D., Mirzadeh, N., Blaas, D., Nones, M.: Product Recommendation with Interactive Query Management and Twofold Similarity. In: Proceedings of the 5th International Conference on Case-Based Reasoning, pp. 479–493. Trondheim, Norway (2003)
51. Shchekotykhin, K., Friedrich, G.: Argumentation based constraint acquisition. In: Proceedings of the IEEE International Conference on Data Mining (2009)
52. Smyth, B., McGinty, L., Reilly, J., McCarthy, K.: Compound Critiques for Conversational Recommender Systems. In: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, WI'04, pp. 145–151. Maebashi, Japan (2004)
53. Thompson, C., Goeker, M., Langley, P.: A Personalized System for Conversational Recommendations. Journal of Artificial Intelligence Research **21**, 393–428 (2004)
54. Tsang, E.: Foundations of Constraint Satisfaction. Academic Press, London and San Diego (1993)
55. Williams, M., Tou, F.: RABBIT: An interface for database access. In: Proceedings of the ACM '82 Conference, ACM'82, pp. 83–87. ACM, New York, NY, USA (1982)
56. Winterfeldt, D., Edwards, W.: Decision Analysis and Behavioral Research. Cambridge University Press (1986)
57. Zanker, M.: A Collaborative Constraint-Based Meta-Level Recommender. In: Proceedings of the 2nd ACM International Conference on Recommender Systems, RecSys 2008, pp. 139–146. ACM Press, Lausanne, Switzerland (2008)
58. Zanker, M., Bricman, M., Gordea, S., Jannach, D., Jessenitschnig, M.: Persuasive online-selling in quality & taste domains. In: Proceedings of the 7th International Conference on Electronic Commerce and Web Technologies, EC-Web 2006, pp. 51–60. Springer, Krakow, Poland (2006)
59. Zanker, M., Fuchs, M., Höpken, W., Tuta, M., Müller, N.: Evaluating Recommender Systems in Tourism - A Case Study from Austria. In: Proceedings of the International Conference on Information and Communication Technologies in Tourism, ENTER 2008, pp. 24–34 (2008)
60. Zanker, M., Jessenitschnig, M.: Case-studies on exploiting explicit customer requirements in recommender systems. User Modeling and User-Adapted Interaction: The Journal of Personalization Research, A. Tuzhilin and B. Mobasher (Eds.): Special issue on Data Mining for Personalization **19**(1-2), 133–166 (2009)
61. Zanker, M., Jessenitschnig, M., Jannach, D., Gordea, S.: Comparing recommendation strategies in a commercial context. IEEE Intelligent Systems **22**(May/Jun), 69–73 (2007)
62. Zhang, J., Jones, N., Pu, P.: A visual interface for critiquing-based recommender systems. In: Proceedings of the 9th ACM Conference on Electronic Commerce, EC'08, pp. 230–239. ACM, New York, NY, USA (2008)
63. Ziegler, C.: Semantic Web Recommender Systems. In: Proceedings of the EDBT Workshop, EDBT'04, pp. 78–89 (2004)