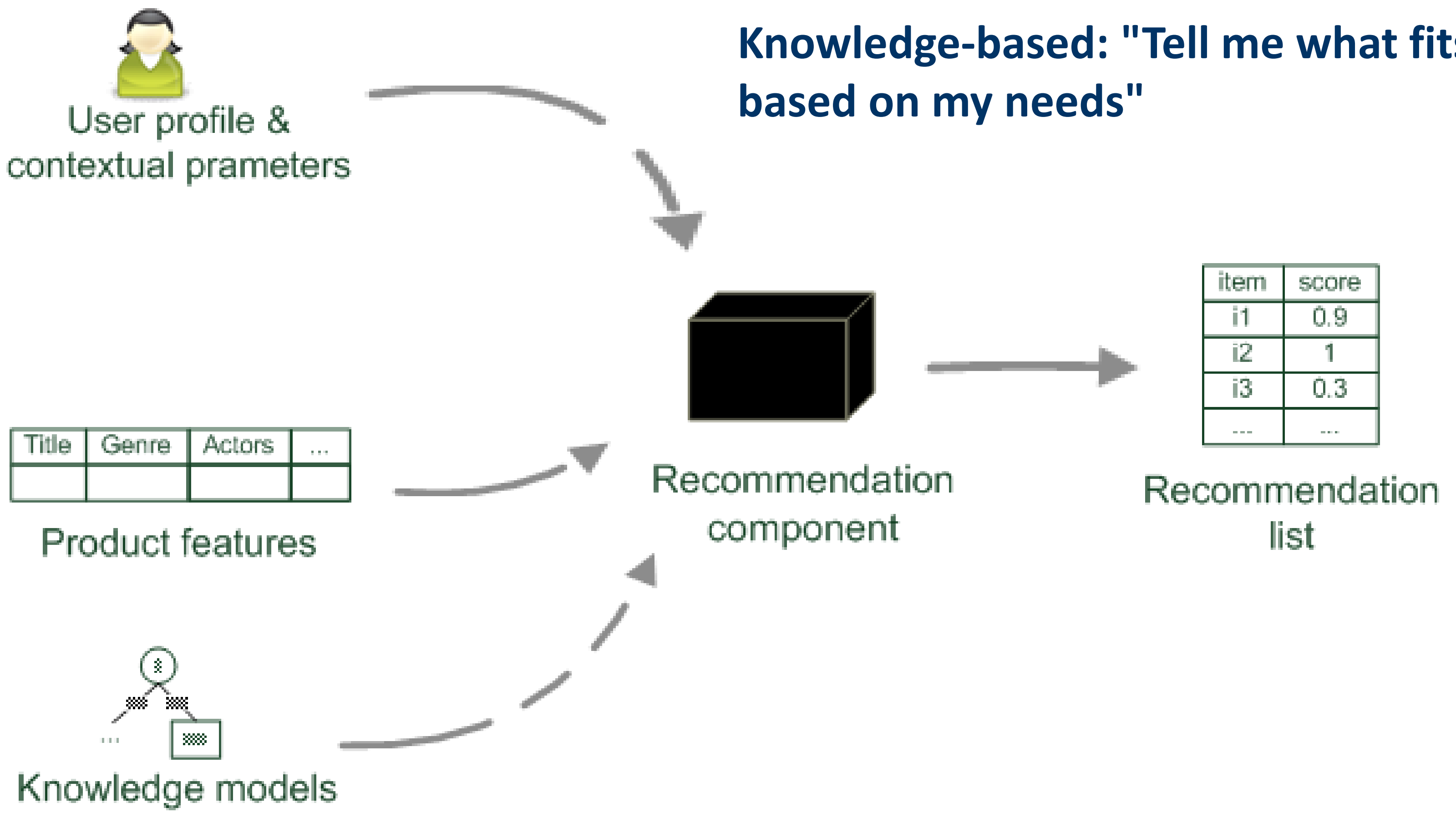
Knowledge-based recommendation

Basic I/O Relationship

Knowledge-based: "Tell me what fits based on my needs"



Why do we need knowledge-based recommendation?

- **Products with low number of available ratings**

- **Time span plays an important role**
  - five-year-old ratings for computers
  - user lifestyle or family situation changes
- **Customers want to define their requirements explicitly**
  - "the color of the car should be black"

Knowledge-based recommender systems

- **Constraint-based**
  - based on explicitly defined set of recommendation rules
  - fulfill recommendation rules
- **Case-based**
  - based on different types of similarity measures
  - retrieve items that are similar to specified requirements

- **Both approaches are similar in their conversational recommendation process**
  - users specify the requirements
  - systems try to identify solutions
  - if no solution can be found, users change requirements

Constraint-based recommender systems

- **Knowledge base**
  - usually mediates between user model and item properties
  - variables
    - user model features (requirements), Item features (catalogue)
  - set of constraints
    - logical implications (IF user requires A THEN proposed item should possess feature B)
    - hard and soft/weighted constraints
    - solution preferences
- **Derive a set of recommendable items**
  - fulfilling set of applicable constraints
  - applicability of constraints depends on current user model
  - explanations – transparent line of reasoning

Constraint-based recommendation tasks

- **Find a set of user requirements such that a subset of items fulfills all constraints**
  - ask user which requirements should be relaxed/modified such that some items exist that do not violate any constraint
- **Find a subset of items that satisfy the maximum set of weighted constraints**
  - similar to find a maximally succeeding subquery (XSS)
  - all proposed items have to fulfill the same set of constraints
  - compute relaxations based on predetermined weights
- **Rank items according to weights of satisfied soft constraints**
  - rank items based on the ratio of fulfilled constraints
  - does not require additional ranking scheme

Constraint-based recommendation problem

- **Select items from this catalog that match the user's requirements**

id	price(€)	mpix	opt-zoom	LCD-size	movies	sound	waterproof
P_1	148	8.0	4×	2.5	no	no	yes
P_2	182	8.0	5×	2.7	yes	yes	no
P_3	189	8.0	10×	2.5	yes	yes	no
P_4	196	10.0	12×	2.7	yes	no	yes
P_5	151	7.1	3×	3.0	yes	yes	no
P_6	199	9.0	3×	3.0	yes	yes	no
P_7	259	10.0	3×	3.0	yes	yes	no
P_8	278	9.1	10×	3.0	yes	yes	yes

- **User's requirements can, for example, be**
  - "the price should be lower than 300 €"
  - "the camera should be suited for sports photography"

Constraint satisfaction problem (CSP)

- **A knowledge-based RS with declarative knowledge representation**

$$CSP\,(X_I \cup X_U, D, SRS \cup KB \cup I)$$

- **Def.**
  - X_I, X_U: Variables describing product and user model with domain D
  - KB: Knowledge base with domain restrictions (e.g**. if** purpose=*on travel* **then** lower focal length < *28mm*)
  - SRS: Specific requirements of user (e.g. purpose = *on travel*)
  - I: Product catalog
- **Solution: Assignment tuple** θ $\forall x \in X_I (x = v) \in \theta \wedge v \in dom(x)$

$s.t. SRS \cup KB \cup I \cup \theta$ **is satisfiable**

Conjunctive query

- **Different from a constraint solver**
  - it is not to find valid instantiations for a CSP

- **Conjunctive query is executed in the item catalog**
  - a conjunctive database query
  - a set of selection criteria that are connected conjunctively

- **σ[criteria](P)**
  - *P:* product assortment
  - example: σ[mpix≥10, price<300](P) = {p4, p7}

Interacting with constraint-based recommenders

- **The user specifies his or her initial preferences**
  - all at once or
  - incrementally in a wizard-style
  - interactive dialog
- **The user is presented with a set of matching items**
  - with explanation as to why a certain item was recommended
- **The user might revise his or her requirements**
  - see alternative solutions
  - narrow down the number of matching items

Defaults

- **Support customers to choose a reasonable alternative**
  - unsure about which option to select
  - simply do not know technical details

- **Type of defaults**
  - static defaults
  - dependent defaults
  - derived defaults

- **Selecting the next question**
  - most users are not interested in specifying values for all properties
  - identify properties that may be interesting for the user

Unsatisfied requirements

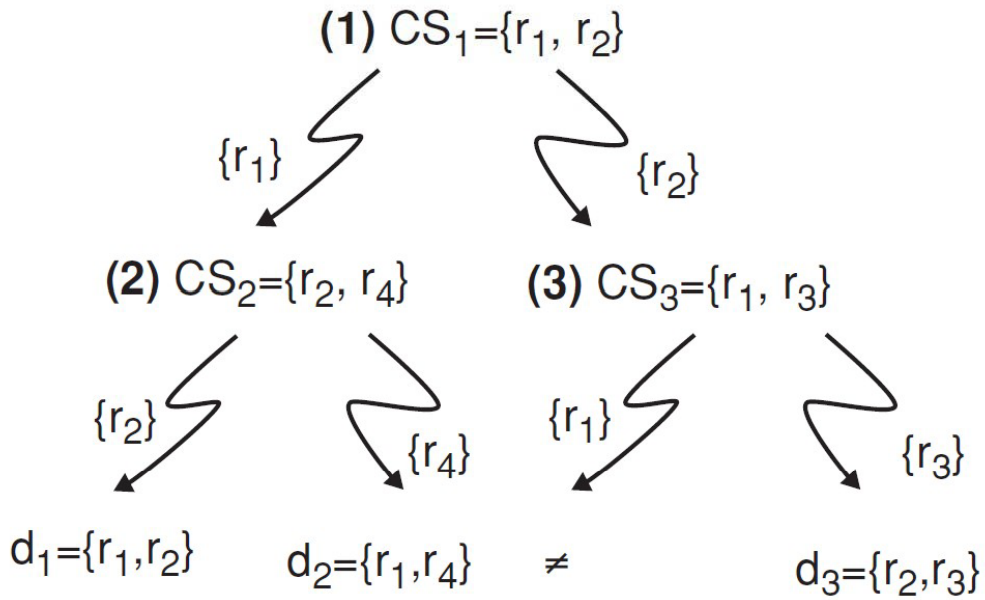
- **"no solution could be found"**

- **Constraint relaxation**
  - the goal is to identify relaxations to the original set of constraints
  - relax constraints of a recommendation problem until a corresponding solution has been found

- **Users could also be interested in repair proposals**
  - recommender can calculate a solution by adapting the proposed requirements

Deal with unsatisfied requirements

- **Calculate diagnoses for unsatisfied requirements**



- **The diagnoses derived from the conflict sets *{CS_1,CS_2,CS_3} are {d_1:{r_1, r_2}, d_2:{r_1, r_4},d_3:{r_2, r_3}}***

QuickXPlain

- **Calculate conflict sets**

Algorithm 4.1 QuickXPlain(*P, REQ*)

Input: trusted knowledge (items) *P; Set of requirements REQ*
Output: minimal conflict set *CS*
if $\sigma_{[REQ]}(P) = \emptyset$ *or* $REQ = \emptyset$ *then return* $\emptyset$
else return QX' (*P, ∅, ∅, REQ);*

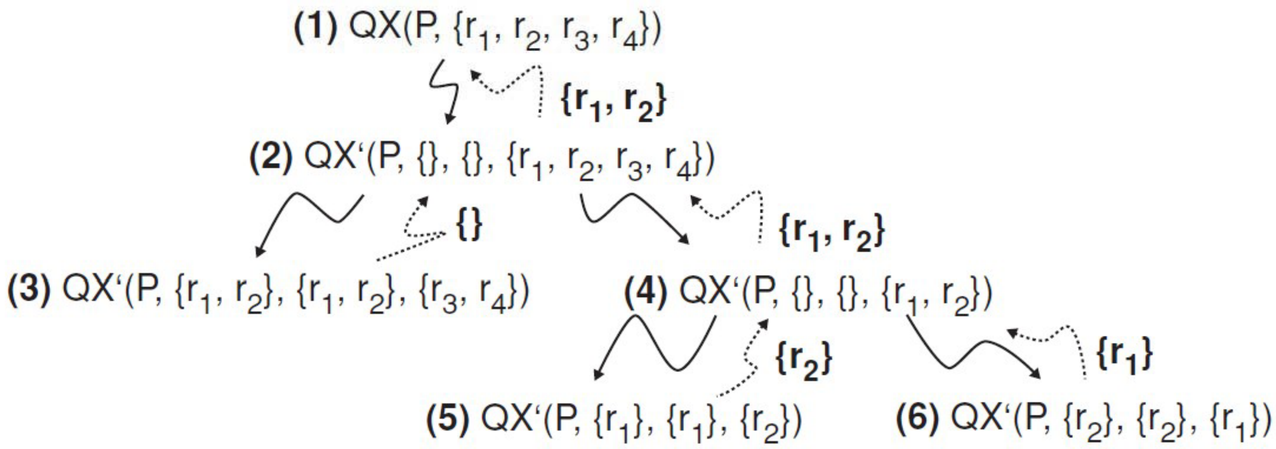
Function QX' (*P, B, Δ, REQ)*
if *= ∅ and* $\sigma_{[B]}(P) = \emptyset$ *then return ∅;*
if *REQ = {r} then return {r};*
let $\{r_1, . . . , r_n\} = REQ$;
let *k be n/2*;
$REQ_1 \leftarrow r_1, . . . , r_k$ *and* $REQ_2 \leftarrow r_{k+1}, . . . , rn$;
$\Delta_2 \leftarrow QX(P, B \cup REQ_1, REQ_1, REQ_2)$;
$\Delta_1 \leftarrow QX(P, B \cup \Delta 2, \Delta 2, REQ_1)$;

return $\Delta_1 \cup \Delta_2$;

Example of QuickXPlain

id	Price(€)	mpix	opt-zoom	LCD-size	movies	sound	waterproof
P_1	148	8.0	4×	2.5	no	no	yes
P_2	182	8.0	5×	2.7	yes	yes	no
P_3	189	8.0	10×	2.5	yes	yes	no
P_4	196	10.0	12×	2.7	yes	no	yes
P_5	151	7.1	3×	3.0	yes	yes	no
P_6	199	9.0	3×	3.0	yes	yes	no
P_7	259	10.0	3×	3.0	yes	yes	no
P_8	278	9.1	10×	3.0	yes	yes	yes

- **REQ = *{r1:price≤150, r2:opt-zoom=5x, r3:sound=yes, r4:waterproof=yes}***



Repairs for unsatisfied requirements

- **Identify possible adaptations**
- **Or query the product table *P with π[attributes(d)]σ[REQ−d](P)***
  - *π[attributes(d1)]σ[REQ−d1](P) = {price=278, opt-zoom=10×}*
  - *π[attributes(d2)]σ[REQ−d2](P) = {price=182, waterproof=no}*
  - *π[attributes(d3)]σ[REQ−d3](P) = {opt-zoom=4×, sound=no}*

repair	price(€)	opt-zoom	sound	waterproof
Rep_1	278	10×	√	√
Rep_2	182	√	√	no
Rep_3	√	4×	no	√

Ranking the items

- **Multi-attribute utility theory**
  – each item is evaluated according to a predefined set of dimensions that provide an aggregated view on the basic item properties
- ***E.g. quality and economy are dimensions in* the domain of digital cameras**

id	value	quality	economy
price	≤250	5	10
	>250	10	5
mpix	≤8	4	10
	>8	10	6
opt-zoom	≤9	6	9
	>9	10	6
LCD-size	≤2.7	6	10
	>2.7	9	5
movies	Yes	10	7
	no	3	10
sound	Yes	10	8
	no	7	10
waterproof	Yes	10	6
	no	8	10

Item utility for customers

- **Customer specific interest**

Customer	quality	economy
Cu_1	80%	20%
Cu_2	40%	60%

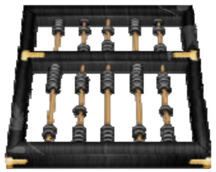
- ***Calculation of Utility***

quality	economy	cu_1	cu_2
P_1 Σ(5,4,6,6,3,7,10) = 41	Σ (10,10,9,10,10,10,6) = 65	45.8 [8]	55.4 [6]
P_2 Σ(5,4,6,6,10,10,8) = 49	Σ (10,10,9,10,7,8,10) = 64	52.0 [7]	58.0 [1]
P_3 Σ(5,4,10,6,10,10,8) = 53	Σ (10,10,6,10,7,8,10) = 61	54.6 [5]	57.8 [2]
P_4 Σ(5,10,10,6,10,7,10) = 58	Σ (10,6,6,10,7,10,6) = 55	57.4 [4]	56.2 [4]
P_5 Σ(5,4,6,10,10,10,8) = 53	Σ (10,10,9,6,7,8,10) = 60	54.4 [6]	57.2 [3]
P_6 Σ(5,10,6,9,10,10,8) = 58	Σ (10,6,9,5,7,8,10) = 55	57.4 [3]	56.2 [5]
P_7 Σ(10,10,6,9,10,10,8) = 63	Σ (5,6,9,5,7,8,10) = 50	60.4 [2]	55.2 [7]
P_8 Σ(10,10,10,9,10,10,10) = 69	Σ (5,6,6,5,7,8,6) = 43	63.8 [1]	53.4 [8]

Case-based recommender systems

- **Items are retrieved using similarity measures**
- **Distance similarity**

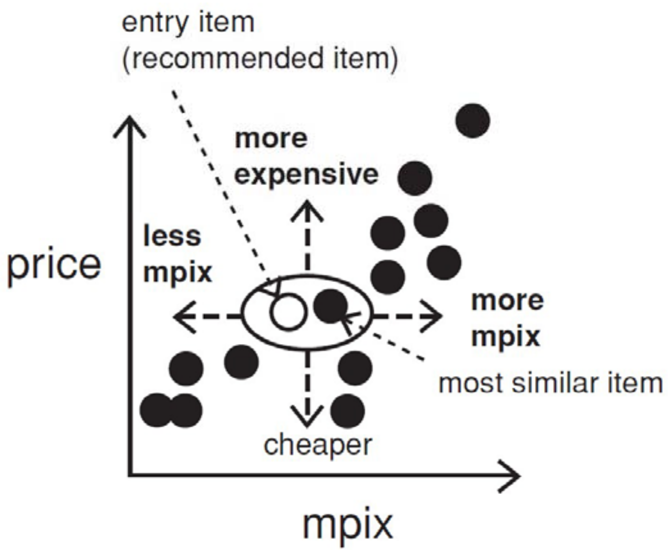
$$similarity(p, REQ) = \frac{\sum_{r \in REQ} w_r * sim(p, r)}{\sum_{r \in REQ} w_r}$$



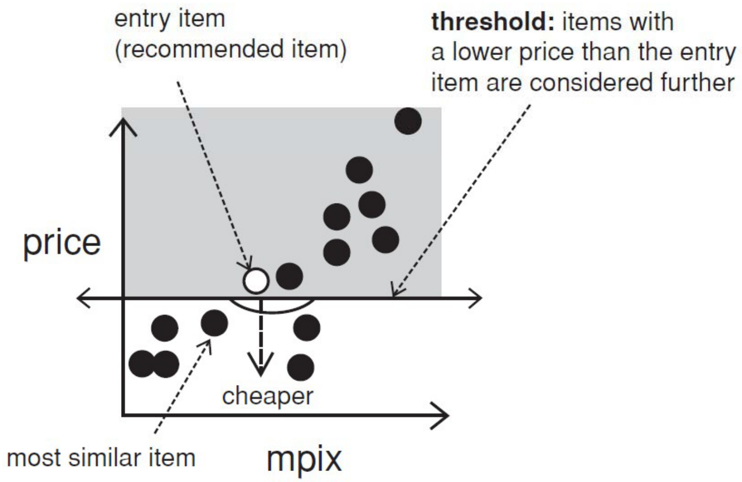
- **Def.**
  - sim (p, r) expresses for each item attribute value ϕr (p) its distance to the customer requirement r ∈ REQ.
  - w_r is the importance weight for requirement r
- **In real world, customer would like to**
  - maximize certain properties. i.e. resolution of a camera, "more is better"(MIB)
  - minimize certain properties. i.e. price of a camera, "less is better"(LIB)

Interacting with case-based recommenders

- Customers maybe not know what they are seeking
- Critiquing is an effective way to support such navigations
- Customers specify their change requests (*price or mpix*) that are not satisfied by the current item (*entry item)*

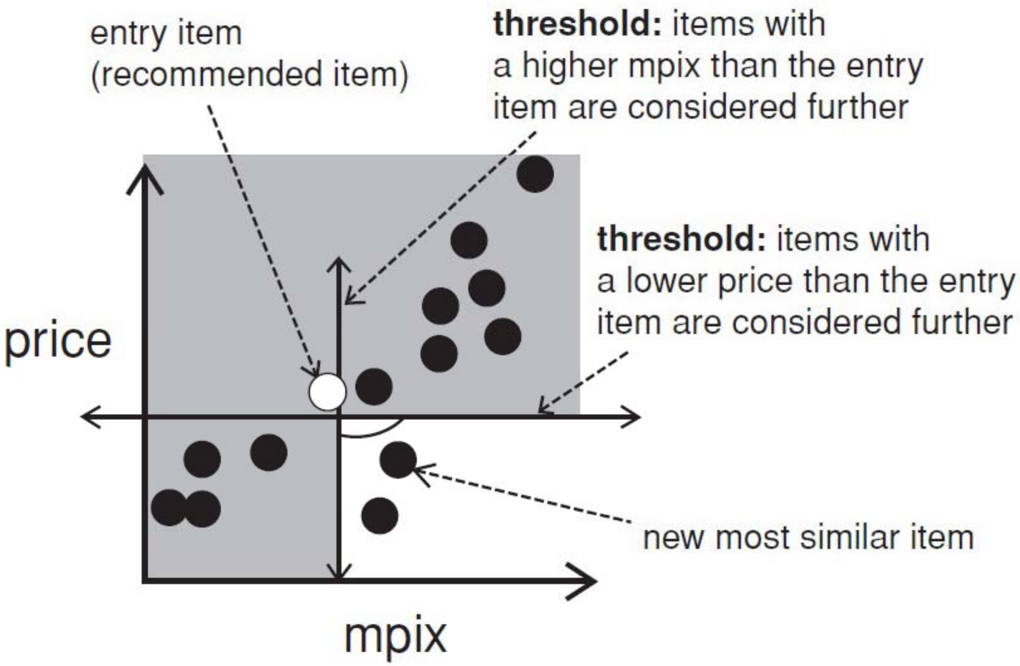


Critique on price



Compound critiques

- **Operate over multiple properties can improve the efficiency of recommendation dialogs**



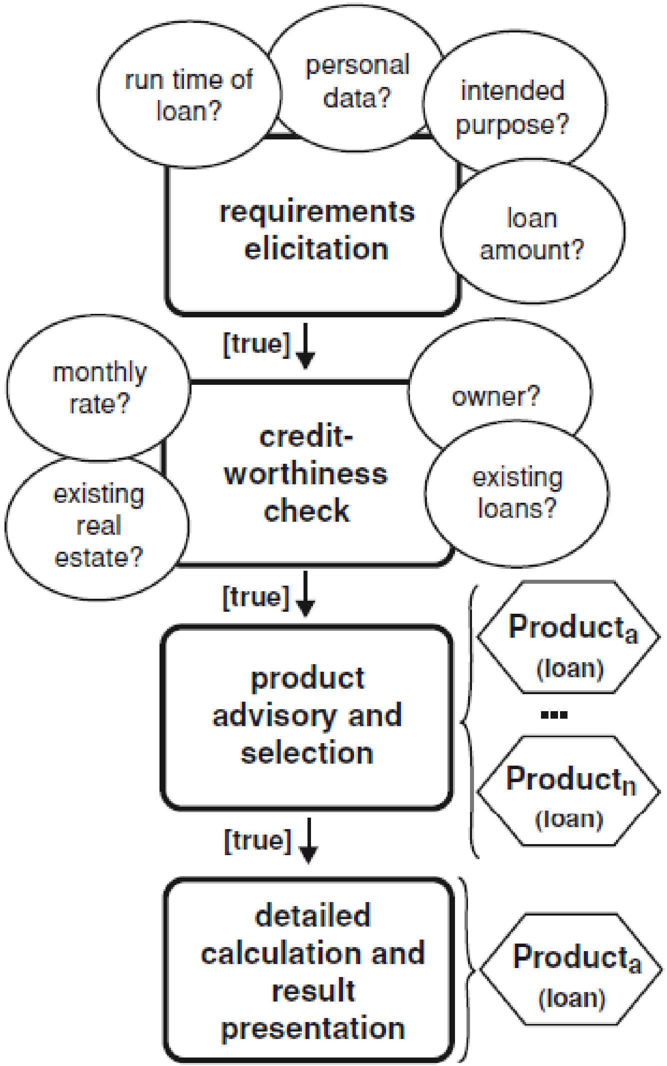
Dynamic critiques

- **Association rule mining**
- **Basic steps for dynamic critiques**
  - *q:* initial set of requirements
  - *CI:* all the available items
  - *K: maximum number of compound critiques*
  - σ_{min} *: minimum support* value for calculated association rules.

Algorithm 4.4 DynamicCritiquing(*q,CI*)
Input: Initial user query *q; Candidate items CI;*
number of compound critiques per cycle *k;*
minimum support for identified association rules σ_{min}

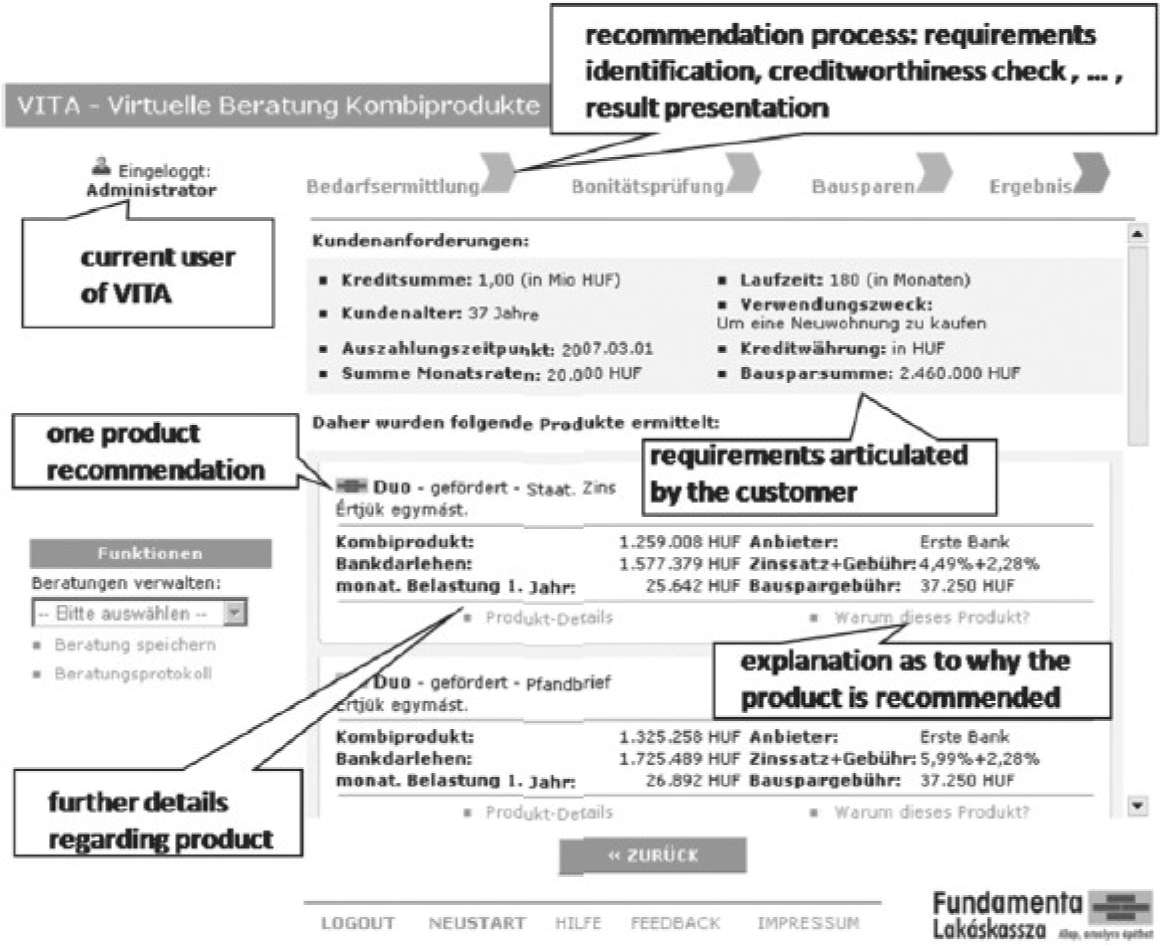
```
procedure DynamicCritiquing(q, CI, k, σmin)
repeat
r ←ItemRecommend(q, CI);
CC ←CompoundCritiques(r, CI, k, σmin);
q ←UserReview(r, CI, CC);
until empty(q)
end procedure

procedure ItemRecommend(q, CI)
CI ← {ci ∈CI: satisfies(ci, q)};
r ←mostsimilar(CI, q);
return r;
end procedure

procedure UserReview(r, CI, CC)
q ←critique(r, CC);
CI ←CI – r;
return q;
end procedure

procedure CompoundCritiques(r, CI, k, σmin)
CP ←CritiquePatterns(r, CI);
CC ←Apriori(CP, σmin);
SC ←SelectCritiques(CC, k);
return SC;
end procedure
```

Example: sales dialogue financial services



- **In the financial services domain**
  - sales representatives do not know which services should be recommended
  - improve the overall productivity of sales representatives
- **Resembles call-center scripting**
  - best-practice sales dialogues
  - states, transitions with predicates
- **Research results**
  - support for KA and validation
    - node properties (reachable, extensible, deterministic)

Example software: VITA sales support



Example: Critiquing

- **Similarity-based navigation in item space**
- **Compound critiques**
  - more efficient navigation than with unit critiques
  - mining of frequent patterns
- **Dynamic critiques**
  - only applicable compound critiques proposed
- **Incremental critiques**
  - considers history
- **Adaptive suggestions**
  - suggest items that allow to best refine user's preference model

Summary

- **Knowledge-based recommender systems**
  - constraint-based
  - case-based
- **Limitations**
  - cost of knowledge acquisition
    - from domain experts
    - from users
    - from web resources
  - accuracy of preference models
    - very fine granular preference models require many interaction cycles
    - collaborative filtering models preference implicitly
  - independence assumption can be challenged
    - preferences are not always independent from each other