Recommendation Technologies for Configurable Products

Andreas Falkner, Alexander Felfernig, and Albert Haag

■ *State-of-the-art recommender systems support users in the selection of items from a predefined assortment (for example, movies, books, and songs). In contrast to an explicit definition of each individual item, configurable products such as computers, financial service portfolios, and cars are represented in the form of a configuration knowledge base that describes the properties of allowed instances. Although the knowledge representation used is different compared to nonconfigurable products, the decision support requirements remain the same: users have to be supported in finding a solution that fits their wishes and needs. In this article we show how recommendation technologies can be applied for supporting the configuration of products. In addition to existing approaches we discuss relevant issues for future research.*

Configuration is a basic form of design activity where the target product is composed from a set of predefined parts in a way that is consistent with a given set of constraints (Stumptner 1997). Similar to knowledge-based recommendation (Burke 2000) configuration is a process where users specify (and often adapt) their requirements and the configuration system provides feedback. Requirements specifications range from feature value definitions to textual queries specified on an informal level. Feedback is provided, for example, in terms of further questions that need to be answered, solutions (configurations), explanations of solutions, and proposals for relaxations of the user requirements in situations where no solution can be found.

A major difference between configuration systems and recommender systems in general is the way in which product knowledge is represented. Configuration systems are operating on a configuration knowledge base (Stumptner 1997), which describes the properties of all allowed instances. In contrast to configuration systems, recommender systems are operating on the basis of an assortment of explicitly defined solution alternatives. The reason for using a configuration knowledge base is the large number of solution alternatives (possible configurations), which make an explicit representation infeasible. Although the used knowledge representations are different, the decision support goal is quite the same for both types of systems: users have to be proactively supported in finding a solution that fits their wishes and needs.

Configuration systems often achieve this goal only partially since the amount and complexity of options presented by the configurator outstrip the capability of a user to identify an appropriate solution (configuration). Users are unable to find the features they would like to specify, they are unsure about their preferences regarding complex technical product properties, and they do not know how best to adapt their requirements in the case of inconsistencies (if no solution can be identified). Even experienced sales persons show a tendency of recom-

$V=\{v_1:\text{type}, v_2:\text{pdc}, v_3:\text{fuel}, v_4:\text{skibag}, v_5:\text{4-wheel}, v_6:\text{color}\}$

$D=\{$ dom(type)={city, limo, combi, xdrive},
dom(pdc)={yes, no},
dom(fuel)={1.7, 2.6, 4.2},
dom(skibag)={yes, no},
dom(4-wheel)={yes, no},
dom(color)={red, blue, grey, black}}

$P_{KB}=\{$ c_1: 4-wheel=yes $\Rightarrow$ type = xdrive,
c_2: skibag=yes $\Rightarrow$ type $\neq$ city,
c_3: fuel=1.7 $\Rightarrow$ type = city,
c_4: fuel=2.6 $\Rightarrow$ type $\neq$ xdrive,
c_5: type=combi $\Rightarrow$ skibag = yes,
c_6: type=limo $\Rightarrow$ pdc = yes}

$C_R=\{$ c_7: type = city, c_8: fuel = 1.7, c_9: 4-wheel = no,
c_{10}: pdc = yes, c_{11}: color=black}

Figure 1. Example of a Configuration Task.

mending configurations they already know from previous sales dialogues and thus are potentially overlooking solutions that would better fit the wishes and needs of a customer (Tiihonen and Felfernig 2010). The goal of this article is to show how recommendation technologies can improve this situation.

In commercial environments (Haag and Riemann 2011, Fleischanderl et al. 1998) we can observe an increasing demand for functionalities that proactively support users throughout the configuration process. These environments already integrate basic functionalities such as the recommendation of feature values (Haag and Riemann 2011) or the recommendation of specific component connections (Fleischanderl et al. 1998). By the majority the recommendations are knowledge-based in the sense that they are determined on the basis of explicit rules or constraints (Felfernig and Burke 2008).

The idea of applying recommendation techniques to support configuration scenarios becomes increasingly popular (Coester et al. 2002, Ardissono et al. 2003, Tiihonen and Felfernig 2010). Examples are the recommendation of features and feature values (Coester et al. 2002, Tiihonen and Felfernig 2010), the recommendation of relaxations (Felfernig et al. 2009), and the reuse of cases for the determination of new configurations (Tseng, Chang, and Chang 2005). The reuse of cases is and has been intensively investigated by the CBR community (Mantaras et al. 2005).

The remainder of this article is organized as follows. First, we introduce basic concepts of knowledge-based configuration. In the following we provide an overview of existing approaches to integrate recommendation and configuration technologies. Finally, we conclude the article with a discussion of future research issues.

Product Configuration

As a basis for the following discussions we introduce the definitions of a *configuration task* and a corresponding *configuration* (solution)—both are based on Felfernig et al. (2004). The definitions are based on the assumption that a configuration task is represented as a basic constraint-satisfaction problem, or CSP (Stumptner 1997). Note that industrial configuration environments are based on advanced constraint-based representations such as *generative constraint satisfaction* (Fleischanderl et al. 1998) where the number of variables (components) is not fixed but generated on demand during search or *dynamic constraint satisfaction* (Haag and Riemann 2011, Mittal and Falkenhainer 1990) where each variable is predefined and has an activation status (only if active, the variable is taken into account during search). For a detailed discussion of different approaches to configuration knowledge representation see Stumptner (1997).

Definition (Configuration Task).

A configuration task can be defined as a CSP (V, D, C) where $V = \{v_1, v_2, \ldots, v_n\}$ represents a set of finite domain variables and $D = \{\text{dom}(v_1), \text{dom}(v_2), \ldots, \text{dom}(v_n)\}$ represents the set of corresponding variable domains. Furthermore, $C = P_{KB} \cup C_R$ represents a set of constraints where $P_{KB} = \{c_1, c_2, \ldots, c_m\}$ represents the *product knowledge* and $C_R = \{c_{m+1}, c_{m+2}, \ldots, c_u\}$ represents a set of requirements. The tuple (V, D, P_{KB}) is denoted as *configuration knowledge base*.

The example of a configuration task in figure 1 stems from the automotive domain. The variables in this simple example are *car type*, availability of a *parking guidance system*, average *fuel consumption* in gallons per 100 miles, *availability of a skibag*, availability of a *4-wheel functionality*, and the *car color*. Users of a configurator can specify their requirements on the basis of C_R.[1]

On the basis of this characterization of a configuration task we introduce the definition of a configuration. A configuration (configuration result or solution) can be interpreted as a bill of materials (BOM), which consists of a list of all components and subcomponents that are part of a product.

Definition (Configuration)

A configuration is an instantiation $I = \{v_1 = i_1, v_2 = i_2, \ldots, v_n = i_n\}$ where i_j is one of the elements of dom(v_j). Users are interested in *valid* configurations (solutions), that is, configurations that are

session	v_1: type	v_2: pdc	v_3: fuel	v_4: skibag	v_5: 4-wheel	v_6: color
s_1	1	0	3	2	0	4
s_2	1	0	0	2	4	3
s_3	2	3	0	1	4	0
s_4	1	4	0	2	0	3
s_5	1	0	0	2	0	0

Table 1. Variables v_j Specified by a User in Session s_k.

The values of s_k x v_j denote the order in which variables have been specified by the user.

complete (each variable has an assigned value) and *consistent* (the assignments are consistent with the constraints in C). *I* = {*type=city*, *fuel=1.7*, *4-wheel=no*, *pdc=yes*, *color=lack*, *skibag=no*} is an example for a valid configuration where *skibag=no* has been selected by the configurator.

Recommendation Technologies for Configurable Products

Users of a configurator application often need advice in order to be able to answer questions such as which variables (features)[2] to specify next? Which values to adapt in situations where no solution can be found? Which values to select—specifically in situations where users lack detailed product knowledge? In the following section we will show how recommendation technologies can be applied for answering these questions.

Recommending Features

Recommendation technologies can help to reduce overheads related to feature selection in different ways: on the one hand features can be explicitly *excluded* if not needed in a certain context; on the other hand they can be *ranked* such that the most relevant ones are easily accessible.

Inclusion and Exclusion of Features

In a car configurator we could exclude the question regarding *skibag* if the user has specified the car type *combi* since cars of *type combi* have a *skibag* included. An example from the financial services domain is the following: if the age of a user is above a certain limit, no pension product related questions should be asked. If the user is primarily interested in low-risk investments, no questions related to the inclusion of foreign currencies and shares should be posed.

The exclusion from a dialogue of features that are not relevant in the current configuration context can be implemented, for example, on the basis of process flows that define in which order which questions are posed to the user (Felfernig and Burke 2008). This process-based selection of relevant questions can be interpreted as a very simple type of knowledge-based recommendation where constraints represent the preconditions for selecting questions. Beside the inflexibility in terms of not being able to focus on questions relevant for a specific user this approach triggers additional knowledge engineering efforts.

Ranking of Features

Besides their explicit inclusion or exclusion, features can be ranked according to their importance for the user. It is a major requirement that state-of-the-art intelligent systems are flexible and proactive in the way they support users in specifying their requirements and selecting relevant features (Pu and Chen 2008). Various approaches to feature recommendation have already been developed—see, for example Mirzadeh and Ricci (2007) and Thompson, Göker, and Langley (2004). We will now sketch the approaches of collaborative, popularity-based, entropy-based, and utility-based feature recommendation.

Table 1 represents a simple interaction log that indicates in which session which variables (features) have been selected by the user in which order. For example, in session s_1 variable v_1 has been selected first, then v_4 and v_3, and finally v_6. The entry 0 denotes the fact that the user did not specify a value due to, for example, missing technical product knowledge or low interest.

Collaborative Feature Selection

One approach to predicting relevant features is to apply the concepts of *collaborative filtering* (Herlocker et al. 2004). If we assume that the user in the current session has already selected and specified the variables v_1 and v_4, the most similar sessions (4-nearest neighbors) would be {s_1, s_2, s_4, s_5} and v_6 would be recommended as the next variable to be specified since it had been selected by the majority of the nearest neighbors.

session	v_1: type	v_2: pdc	v_3: fuel	v_4: skibag	v_5: 4-wheel	v_6: color
s_1	city	no	4.2	no	no	red
s_2	combi	yes	2.6	yes	no	black
s_3	city	yes	2.6	no	no	black
s_4	city	yes	1.7	no	no	black
s_5	combi	yes	2.6	yes	no	black

Table 2. Log of Successfully Completed Configuration Sessions.

Value of the variable v_j in the session s_k.

Popularity-Based Feature Selection

Another feature selection approach is to rank features (v_i) according to their *popularity*, which can be defined as the share of user selections of variable v_i in relation to the total number of variable selections (see equation 1). This measure is simple (it does not require complex calculations) and primarily takes into account features users want to specify.

$$popularity(v_i) = \frac{\#selected(\{v_i\})}{\#selected(\{v_1 \ldots v_n\})} \quad (1)$$

Both collaborative and popularity-based feature selection help to identify questions of relevance for the user but do not take into account minimality in terms of the number of questions needed (Mirzadeh and Ricci 2007). We will now sketch how to reduce the number of questions and at the same time take into account feature relevance.

Entropy-Based Feature Selection

Entropy is used to determine the smallest number of bits needed for transmitting information units with a certain occurrence distribution. The higher the entropy the higher is the degree of information content. The idea of applying entropy for feature selection is to select high-entropy features that minimize the overall number of questions needed to successfully complete a configuration session. The entropy of a feature (variable v_i) can be determined using equation 2 where p_{aj} is the occurrence probability of value $a_j \in \text{dom}(v_i)$.

$$entropy(v_i) = -\sum_{j=1}^{|dom(v_i)|} p_{aj} * \log_2(p_{aj}) \quad (2)$$

For determining the entropy of a feature, we have to exploit the information contained in a log of already successfully completed configuration sessions (see table 2). A configuration log does not include all possible solutions.[3] Still, it is a valuable basis for ranking features with respect to their capability of reducing the number of solutions of interest for the user (Tiihonen and Felfernig 2010). If the goal is to reduce the set of potential solutions, *4-wheel* is not a good candidate for asking a question (entropy: 0.0). The feature *fuel* should be used (entropy: 1.37).

The entropy measure does *not* take into account the relative importance of features, which means that users could be confronted with questions they are not interested in. Consequently, the basic entropy measure should only be applied in situations where we do not have to deal with user preferences regarding the specification of features. Especially in the context of configuration processes, entropy has to be combined with other methods that take into account the aspect of feature relevance (such as collaborative and popularity-based ranking).

Utility-Based Feature Selection

This feature selection approach combines the advantages of popularity-based and entropy-based feature selection (see equation 3).

$$utility(v_i) = \text{entropy}(v_i) * \text{popularity}(v_i) \quad (3)$$

An empirical evaluation by Mirzadeh and Ricci (2007) shows that utility-based feature selection outperforms the entropy and popularity-based approaches in terms of minimizing the number of questions needed in a dialogue.

Recommending Explanations

Quite often users specify requirements (C_R) that are inconsistent with the product knowledge (P_{KB}). One possible way to avoid such situations is to allow the user to enter only one requirement at a time and to prevent the specification of values that do not allow the calculation of a solution. This approach is very limited and does not allow the user to learn about the underlying product assortment and existing trade-offs between product properties (Pu and Chen 2008); furthermore, such approaches prevent the manufacturer from learning new customer requirements.

An alternative is to present *minimal explanations* (*diagnoses*) (Reiter 1987, Felfernig et al. 2009) or *maximal relaxations* (Petit, Bessiere, and Regin 2003; O'Sullivan et al. 2007). Minimal explanations are minimal sets of requirements that have to be adapted or deleted such that a solution can be identified. A maximal relaxation is always the complement of a minimal explanation. The following scenario shows how explanations can support users in a configuration session.

Minimal Explanation

Let us assume that the user has specified a set of requirements (C_R') that are inconsistent with the product knowledge (P_{KB}).[4]

C_R' = { c_7: type=limo, c_8: fuel=1.7, c_9: 4-wheel=yes, c_{10}: pdc=no, c_{11}: color=black}

In such a situation explanations are presented (see table 4) and the user can select a corresponding adaptation (repair). For example, Δ_1={c_7,c_8} has only one associated adaptation, which is (*type=xdrive*, *fuel=4.2*). Repair alternatives can be determined with the help of a constraint solver (Herbrard et al. 2005, Tiihonen and Felfernig 2010), that is, the user is not forced to figure out consistent repairs on his or her own. If many repair alternatives exist and not all of them can be calculated (for performance reasons), the user should also have the option of adapting his or her requirements without being forced to select one of the proposed repairs.

Next we will discuss different approaches to the determination of minimal explanations. We will first introduce the basic concept of *model-based diagnosis* (Reiter 1987) and then explain the concept of *preferred explanations,* which represent *recommendations for explanations*.

Model-Based Diagnosis

Explanations (diagnoses) are based on the resolution of minimal conflict sets calculated by algorithms such as QUICKXPLAIN (Junker 2004). A minimal explanation is a minimal set of constraints that have to be deleted or adapted in order to make C_R consistent with P_{KB}. Resolving a minimal conflict is achieved by deleting at least one constraint from the corresponding conflict set. The definition of a customer requirements explanation problem and the corresponding explanation are based on Felfernig et al. (2004).

Definition (CR Explanation Problem).

A customer requirements explanation problem (C_R, P_{KB}) consists of a set of customer requirements (C_R) and the corresponding product knowledge (P_{KB}).

Definition (CR Explanation).

A customer requirements explanation for a CR explanation problem (C_R, P_{KB}) is a set $\Delta \subseteq C_R$ s.t. $C_R - \Delta \cup P_{KB}$ is consistent. Δ is minimal if there does not exist an explanation Δ' with $\Delta' \subset \Delta$.

ID	Conflict Set
CS_1	{c_7,c_8}
CS_2	{c_7,c_9}
CS_3	{c_8,c_9}
CS_4	{c_7,c_{10}}

Table 3. Minimal Conflict Sets in the Working Example.

ID	Explanation	Repair Alternatives
Δ_1	{c_7, c_8}	(*type=xdrive*, *fuel=4.2*)
Δ_2	{c_7, c_9}	(*type=city*, *4-wheel=no*)
Δ_3	{c_8, c_9, c_{10}}	(*fuel=2.6*, *4-wheel=no*, *pdc=yes*) (*fuel=4.2*, *4-wheel=no*, *pdc=yes*)

Table 4. Explanations Derived from Conflict Sets.

Definition (Conflict Set)

A conflict set $CS \subseteq C_R$ induced by the product knowledge P_{KB} is a set of requirements such that $CS \cup P_{KB}$ is inconsistent. CS is minimal if there does not exist a conflict CS' with $CS' \subset CS$.

Table 3 shows all minimal conflict sets CS_i in C_R' induced by the product knowledge P_{KB}. For a detailed discussion of the determination of minimal conflict sets we refer the reader to Junker (2004).

The minimal explanations for CS_1 ..CS_4 are shown in table 4. Δ_1 and Δ_2 are *minimal cardinality explanations,* which are the first ones returned by algorithms based on breadth-first search (Reiter 1987). Δ_3 is a minimal explanation but not a minimal cardinality explanation (for example, $|\Delta_1| < |\Delta_3|$).

Max-CSP and Max-SAT

An alternative to the determination of explanations using model-based diagnosis (Reiter 1987) are specialized Max-CSP (Petit, Bessiere, and Regin 2003) and Max-SAT (Argelich et al. 2008) solvers. These solvers determine maximum-cardinality constraint sets that allow the calculation of a solution—such sets do not have consistent supersets. The complement of such a constraint set is a min-

c_7:type	c_8:fuel	c_9:4-wheel	c_{10}:pdc	c_{11}:color
40%	20%	15%	13%	12%

Table 5. Relative Importance of Requirements.

ID	Explanation	Utility	Ranking
Δ_1	$\{c_7, c_8\}$	$\{c_7, c_8\} \Rightarrow 1.66$	3
Δ_2	$\{c_7, c_9\}$	$\{c_7, c_9\} \Rightarrow 1.82$	2
Δ_3	$\{c_8, c_9, c_{10}\}$	$\{c_8, c_9, c_{10}\} \Rightarrow 2.08$	1

Table 6. Utility-Based Ranking of Explanations.

ID	c_7	c_8	c_9	c_{10}	c_{11}	Ranking
Δ_1	x	x				3
Δ_2	x		x			2
Δ_3		x	x	x		1

Table 7. Ranking of Explanations with FASTDIAG.

imal cardinality explanation. A detailed discussion of existing Max-CSP and Max-SAT approaches is out of the scope of this article. Further details of the underlying concepts and algorithms are discussed, for example, in Argelich et al. (2008).

Preferred Explanations

Inconsistent customer requirements often entail a large number of alternative explanations (Felfernig et al. 2009). Consequently, techniques are needed that enable the calculation of *preferred explanations*, that is, explanations with a high probability of being accepted by the user. O'Sullivan et al. (2007) propose the concept of *representative explanations* where each constraint contained in the complete set of explanations is also contained in at least one of the explanations shown to the user. DeKleer (1990) introduces a probability-based approach to the determination of preferred explanations (diagnoses), that is, explanations with a high probability of explaining the faulty behavior of a system. A similar approach has been proposed by Felfernig et al. (2009), who show how to apply the similarity between requirements and the configurations stored in an interaction log (see table 2) for the identification of preferred explanations.

A simple example of the application of utility functions (Felfernig, Schubert, and Zehentner 2011) to guide the search for preferred explanations will be sketched in the following. Table 5 shows importance values assigned to each user requirement. Such values can be specified by a user but as well be learned from a user interaction log (Arslan et al. 2002). The function used in our example is shown in equation 4.

$$utility(\{c_1 \ldots c_k\}) = \frac{1}{\sum_{i=1}^{k} \text{importance}(c_i)} \quad (4)$$

The ranking of explanations is sketched in table 6. Note that there is no need for calculating all existing explanations since the utility function (equation 4) can be applied as a best first search node expansion strategy (Felfernig, Schubert, and Zehentner 2011). The function is monotonically decreasing, which guarantees optimality of best first search.

The first explanation presented to the user would be Δ_3 (it has the highest utility). Empirical studies in the domains of computer and financial services configuration have shown that explanation rankings based on the criteria of utility, probability, and similarity clearly outperform prediction approaches based on the criteria of minimal cardinality (Felfernig, Schubert, and Zehentner 2011). If there is a need to incorporate utility functions that are representing interdependences between variables, this can lead to the proposal of nonminimal explanations due to nonmonotonicity.

Another approach to the determination of preferred explanations is FASTDIAG (Felfernig, Schubert, and Zehentner 2011). This algorithm calculates preferred explanations based on a total ordering of the constraints in C_R. Two explanations Δ_a and Δ_b (subsets of $C_R = \{c_\alpha, \ldots, c_\omega\}$) can be compared lexicographically as follows: Δ_a is given preference compared to Δ_b iff $\exists\delta$: $c_\delta \in \Delta_b - \Delta_a$ and $\Delta_a \cap \{c_{\delta-1}, \ldots, c_\alpha\} = \Delta_b \cap \{c_{\delta-1}, \ldots, c_\alpha\}$. A simple example is depicted in table 7 where the assumed total constraint ordering is $c_7 > c_8 > c_9 > c_{10} > c_{11}$, that is, c_7 is a very important requirement and c_{11} is the requirement with the lowest importance. If we compare the explanations $\Delta_2 = \{c_7, c_9\}$ and $\Delta_3 = \{c_8, c_9, c_{10}\}$, Δ_3 is the preferred one since $\{c_7, c_9\} - \{c_8, c_9, c_{10}\} = \{c_7\}$ and $\Delta_a \cap \{\} = \Delta_b \cap \{\}$. Intuitively, the importance of c_7 in Δ_2 is higher than all other constraints in Δ_3, therefore Δ_3 is the preferred explanation. In the given scenario the FASTDIAG algorithm would automatically calculate the explanation Δ_3.

Note that this idea of a preferred explanation perfectly fits industrial requirements, since users of constraint-based applications typically prefer to keep important requirements as is and to change the less important ones (Junker 2004). The predictive quality of FASTDIAG in terms of precision has been analyzed in (Felfernig, Schubert, and Zehentner 2011) with computer and financial service portfolio configuration data sets. The evaluation results do not show significant differences in terms of predictive quality. However, FASTDIAG run times for the first-*n* explanations (n = 1, 5, 10, 20) are significantly lower compared to standard hitting-set-based approaches (Felfernig, Schubert, and Zehentner 2011). Due to these properties the algorithm perfectly supports interactive scenarios with the need of presenting the most relevant (preferred) explanations.

Weighted Max-CSP and Max-SAT

Another alternative for the determination of preferred explanations are specialized *weighted Max-CSP* (Petit, Bessiere, and Regin 2003) and *weighted Max-SAT* solvers (Argelich et al. 2008). Each constraint has an assigned importance value, and the search goal is to identify minimal cost solutions where costs are defined as the sum of the importance values of the constraints part of the explanation. A detailed discussion of weighted Max-CSP and Max-SAT algorithms can be found, for example, in Petit, Bessiere, and Regin (2003) and Argelich et al. (2008).

Recommending Feature Values

Feature value recommendations give users a better understanding of the dependencies between requirements and the possible settings of other variables (Coester et al. 2002). For users who are not experts in the product domain such recommendations provide hints about reasonable instantiations of variables. They also support the idea that users should specify the most important requirements from their point of view and let the configuration system find meaningful completions (complete configurations). Typical questions answered by feature value recommendations are: Which feature values have been selected by users with similar requirements? What are popular completions of partial configurations similar to my requirements? In the following we will discuss related recommendation approaches.

Different types of feature value recommendations can help to improve the quality of user support in configuration sessions. *Static recommendations* do not take into account the context of the current user; for example, the feature value for *skibag* is set to *no* because most of the users are not interested in the equipment. *Rule-based recommendations* are taking into account the context of the current user by interpreting a set of rules; for example, if the selected car *type* is *xdrive*, then the feature value for *skibag* should be set to *yes*. The major difference compared to a constraint in the product knowledge is that recommended feature values do not need to be accepted by the user. Another approach to the recommendation of feature values is to determine the *k*-nearest neighbor configurations (*k*NN—taken from already completed configuration sessions) that are similar to the current set of user requirements and to determine recommendations on the basis of majority voting (Coester et al. 2002). For the identification of *k*-nearest neighbors we can exploit similarity functions that estimate similarity depending on the type of a feature (McSherry 2003).

Examples for such similarity functions are *more-is-better* (for example, the higher the energy efficiency of a car the better), *less-is-better* (for example, the lower the price of a car the better), and *nearer-is-better* (the nearer the size of the steering wheel to the required size the better). The following aspect has to be taken into account when selecting a similarity function: in contrast to basic CSPs more complex representations of a configuration task (Fleischanderl et al. 1998) include connection structures (port connections) and cardinalities of subcomponents. This additional information has to be taken into account by the similarity function. A more detailed discussion of such functions can be found, for example, in Tseng, Chang, and Chang (2005).

Formula 5 shows the basic approach to the determination of a feature value recommendation for the variable v_i on the basis of majority voting where a_j is the j^{th} value in dom(v_i) and v_{ik} denotes the value of v_i of the configuration in the session $s_k \in S$ (table 2).

$$majority(v_i) = \arg\max_{(j=1\ldots|dom(v_i)|)} \left(\sum_{k=1}^{|S|} v_{ik} = a_j \right) \quad (5)$$

If we assume that the current user has already specified values for the variables *type* (*type* = *city*) and *pdc* (*pdc* = *yes*) and we exploit the 2-nearest neighbors (sessions s_3 and s_4 of table 2) for determining a feature value recommendation for the variable *color*, the predicted value would be *black*. An analysis of probability-based approaches to determining personalized feature value recommendations can be found in Coester et al. (2002), Felfernig and Tiihonen (2010), and Haag and Riemann (2011).

In principle, all of the mentioned approaches can be exploited for recommending completions for configurations. However, we want to emphasize that (with the exception of rule-based approaches) feature recommendation approaches cannot guarantee the consistency between a feature value recommendation, the given requirements, and the underlying product knowledge. Before a feature value is recommended, a consistency check is needed (Felfernig and Tiihonen 2010). This situation is well known from case-based reasoning research: very often retrieved cases have to be adapted in order to be consistent with the new requirements (Mantaras et al. 2005). An alternative to the determination of feature value recommendations consistent with the product knowledge is to trigger inconsistency handling by presenting minimal explanations for inconsistencies between requirements and calculated feature value recommendations (Haag and Riemann 2011).

When presenting configurations or subconfigurations to the user, diversity can play an important role (Herbrard et al. 2005, Mantaras et al. 2005). It helps to effectively figure out user preferences by providing an overview of the available configurations instead of presenting a set of very similar ones. Algorithms for calculating diverse configurations during solution search are presented in Herbrard et al. (2005). The authors introduce two basic approaches. *Heuristic search* approximates the most diverse configurations whereas *complete methods* can achieve optimal solution sets (maximum diversity) with the cost of state space explosion as a result of problem reformulation. The major challenge for constraint-based applications is that diversity has to be taken into account *during* search whereas case-based applications can apply diversity metrics on the given set of items (Mantaras et al. 2005).

Issues for Future Research

Issues for future research include recommendation algorithms for testing and debugging, group-based configuration, recommender systems in open innovation, improving knowledge base accessibility, and group-based knowledge engineering. We will discuss each of these issues in turn in the following subsections.

Recommendation Algorithms for Testing and Debugging

In addition to calculating explanations for inconsistent requirements, the concepts of model-based diagnosis (Reiter 1987) can as well be applied to the identification of faulty constraints in the product knowledge (Felfernig et al. 2004). In this context we are interested in minimal sets of faulty constraints that have to be deleted from PKB (or adapted) in order to make the new version of PKB consistent with a given set of test cases. Similar to the scenario of inconsistent customer requirements, knowledge engineers are often confronted with a large number of alternative explanations (diagnoses) and we cannot expect that they are willing to analyze them all. Open research questions to be answered in this context are the following: What are good criteria for determining preferred explanations for faulty constraints in the product knowledge? How to estimate the relevance of a test case?

Group-Based Configuration

Another research issue is how best to support configuration processes for groups of users. Example scenarios are the configuration of holiday trips and the configuration of software release plans. In all these scenarios users are cooperatively developing a configuration. Typical functionalities to be supported in single-user configuration scenarios (for example, recommending features, recommending explanations, and recommending individual feature values) must also be supported in distributed configuration scenarios. However, further research questions have to be answered: How to achieve consensus between the different users in the case of inconsistent requirements? What are the best recommendation algorithms to achieve this goal? To which extent should the preferences of users be made visible to other users? On a more technical level: How can critiquing based recommendation approaches (Pu and Chen 2008) be applied in distributed configuration scenarios (for example, for the critiquing of the preferences of other users)? How can we exploit theories of group decision making to improve the quality of decision processes?

Recommender Systems in Open Innovation

The effective integration of consumer knowledge into a company's innovation process—also known as *open innovation*—is crucial for a successful new product development. Innovation process quality has a huge impact on the ability of a company to achieve sustainable growth. Innovations are very often triggered by consumers who are becoming active contributors in the process of developing new products. Platforms such as sourceforge.net or ideastorm.com confirm this trend of progressive customer integration. These platforms exploit community knowledge and preferences to come up with new requirements, ideas, and products. In this context, the size and complexity of the generated knowledge (informal descriptions of requirements and ideas as well as knowledge bases describing new products on a formal level) outstrips the capability of community members to find solutions consistent with their personal preferences. Example questions to be answered in this context are: Who are the community members with similar preferences, ideas, and requirements? Which ideas should be further developed into product prototypes? What are the preferred product prototypes of the community?

Improving Knowledge Base Accessibility

This is a crucial issue for all processes related to knowledge base development and maintenance. Highly experienced knowledge engineers have a different knowledge base navigation and adaptation behavior and thus should be supported differently compared to less experienced personnel. Research topics in this context are, for example, intelligent knowledge base visualization and navigation support. Example questions to be answered are the following: Which constraints should be additionally inspected after a sequence of already completed adaptation operations? Which information can be omitted in certain maintenance contexts? Similar to the engineering of conventional software systems, knowledge-based systems are in the need of refactoring support. Example questions to be answered are: Which constraints are redundant or will never be active in a configuration session? Which structural changes of the knowledge base should be recommended in order to improve its understandability? In this context we need to have a more detailed look at the underlying cognitive aspects, for example, which constraint structures are more understandable and which type of constraint structuring helps to improve the understanding of the knowledge base?

Group-Based Knowledge Engineering

The support of group-based configuration processes has already been mentioned as an issue for future research. In the same line, the engineering of knowledge bases can be interpreted as a collaborative process between domain experts (product designers, sales representatives, marketing, and so on) and knowledge engineers. Both are often only familiar with specific aspects of the product respectively the corresponding formalization in PKB (for example, the hardware or the software of a telecommunication switch). Furthermore, engaging stakeholders in knowledge engineering processes triggers conflicting interests regarding the product assortment that should be offered to the customer. Example questions to be answered in this context are the following: Which constraints should be part of the knowledge base? Which feature values should be recommended to a specific user in which context? How to achieve consensus between contradicting stakeholder preferences?

Conclusions

In this article we provide an overview of recommendation approaches that help to improve the overall usability of configuration systems. Examples are the recommendation of features that will be of interest for the user, the recommendation of explanations that help the user to solve the "no solution could be found" problem, and the recommendation of feature values. On the basis of a discussion of existing approaches we introduced research questions that should be answered in order to further advance the state of the art in knowledge-based configuration.

Acknowledgments

The work presented in this article has been conducted within the scope of the project WECARE (820128) funded by the Austrian Research Promotion Agency. We want to thank Deepak Dhungana and Stefan Starke from Siemens for their valuable comments that helped us to improve this article.

Notes

1. Note that industrial configurator applications are based on much larger knowledge bases, which include hundreds of variables and constraints. A detailed discussion of the structural properties of such knowledge bases can be found, for example, in Tiihonen (2010).

2. In complex configuration scenarios, *features* represent decision criteria that can be specified by the user—in contrast to other *variables* describing technical details of no interest for the user. For our purposes we use the terms *feature* and *variable* synonymously.

3. To stabilize probability calculations in the case of a small number of entries in a configuration log, an *m-estimate* (Mitchell 1997) can be applied.

4. Note that C_R' does not include a specification of *skibag*, that is, the user has no preference regarding this feature.

References

Ardissono, L.; Felfernig, A.; Friedrich, G.; Goy, A.; Jannach, D.; Petrone, G.; Schaefer, R.; Zanker, M. 2003. A Framework for the Development of Personalized, Distributed Web-Based Configuration Systems. *AI Magazine* 24(3): 93–110.

Argelich, J.; Cabiscol, A.; Lynce, I.; and Manya, F. 2008. Modeling Max-CSP as Partial Max-SAT. In *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing*, Lecture Notes in Computer Science 4996, 1–15. Berlin: Springer.

Arslan, B.; Ricci, F.; Mirzadeh, N.; and Venturini, A. 2002. A Dynamic Approach to Feature Weighting. In *Data Mining III*. Ashurst, UK: WIT Press.

Burke, R. 2000. Knowledge-Based Recommender Systems. *Library and Information Systems* 69(32): 180–200.

Coester, R.; Gustavsson, A.; Olsson, R.; and Rudstroem, A. 2002. Enhancing Web-Based Configuration with Recommendations and Cluster-Based Help. Paper presented at the Workshop on Recommendation and Personalization in ECommerce, Malaga, Spain, 3–4 June.

DeKleer, J. 1990. Using Crude Probability Estimates to Guide Diagnosis. *AI Journal* 45(3): 381–391.

Felfernig, A., and Burke, R. 2008. Constraint-Based Recommender Systems: Technologies and Research Issues. In *Proceedings of the ACM International Conference on Elec-*

tronic Commerce, 17—26. New York: Association for Computing Machinery.

Felfernig, A.; Friedrich, G.; Jannach, D.; and Stumptner, M. 2004. Consistency-Based Diagnosis of Configuration Knowledge Bases. *AI Journal* 152(2): 213–234.

Felfernig, A.; Schubert, M.; Friedrich, G.; Mandl, M.; Mairitsch, M.; and Teppan, E. 2009. Plausible Repairs for Inconsistent Requirements. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 791–796. Menlo Park, CA: AAAI Press.

Felfernig, A.; Schubert, M.; and Zehentner, C. 2011. An Efficient Diagnosis Algorithm for Inconsistent Constraint Sets. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 26(1): 1–10.

Fleischanderl, G.; Friedrich, G.; Haselboeck, A.; Schreiner, H.; and Stumptner, M. 1998. Configuring Large Systems Using Generative Constraint Satisfaction. *IEEE Intelligent Systems, Special Issue on Configuration* 13(4): 59–68.

Haag, A., and Riemann, S. 2011. Product Configuration as Decision Support: The Declarative Paradigm in Practice. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 25(2): 131–142.

Herbrard, E.; Hnich, B.; O'Sullivan, B.; and Walsh, T. 2005. Finding Diverse and Similar Solutions in Constraint Programming. In *Proceedings of the 20th AAAI Conference on Artificial Intelligence*, 372–377. Menlo Park, CA: AAAI Press.

Herlocker, J.; Konstan, J.; Terveen, L.; and Riedl, J. 2004. Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems* 22(1): 5–53.

Junker, U. 2004. QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. In *Proceedings of the 19th AAAI Conference on Artificial Intelligence*, 167–172. Menlo Park, CA: AAAI Press.

Mantaras, R.; McSherry, D.; Bridge, D.; Leake, D.; Smyth, B.; Craw, S.; Faltings, B.; Maher, M.; Cox, M.; Forbus, K.; Keane, M.; Aamodt, A.; and Watson, I. 2005. Retrieval, Reuse, Revision, and Retention in Case-Based Reasoning. *Knowledge Engineering Review* 20(3): 215–240.

McSherry, D. 2003. Similarity and Compromise. In *Case-Based Reasoning Research and Development, Proceedings of the 5th International Conference on Case-Based Reasoning*, Lecture Notes in Computer Science 2689, 291–305. Berlin: Springer.

Mirzadeh, N., and Ricci, F. 2007. Cooperative Query Rewriting for Decision Making Support. *Journal of Applied Artificial Intelligence* 21(10): 895–932.

Mitchell, T. *Machine Learning*. 1997. New York: McGraw Hill.

Mittal, S., and Falkenhainer, B. 1990. Dynamic Constraint Satisfaction Problems. In *Proceedings of the 8th National Conference on Artificial Intelligence,* 25–32. Menlo Park, CA: AAAI Press.

O'Sullivan, B.; Papadopoulos, A.; Faltings, B.; Pu, P. 2007. Representative Explanations for Over-Constrained Problems. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence,* 323–328. Menlo Park, CA: AAAI Press.

Petit, T.; Bessiere, C.; and Regin, J. 2003. A General Conflict-Set Based Framework for Partial Constraint Satisfaction. In *Proceedings of the Ninth International Conference on Principles and Practice of Contraint Programming,* Lecture Notes in Computer Science 2833, 1–14. Berlin: Springer.

Pu, P., and Chen, L. 2008. User-Involved Preference Elicitation for Product Search and Recommender Systems. *AI Magazine* 29(4): 93–103.

Reiter, R. 1987. A Theory of Diagnosis from First Principles. *Artificial Intelligence* 23(1): 57–95.

Stumptner, M. 1997. An Overview of Knowledge-Based Configuration. *AI Communications* 10(2): 111–125.

Thompson, C.; Göker, M.; and Langley, P. 2004. A Personalized System for Conversational Recommendations. *Journal of Artificial Intelligence Research* 21: 393–428.

Tiihonen, J. 2010. Characterization of Configuration Knowledge Bases. Paper presented at the ECAI 2010 Workshop on Configuration, Lisbon, Portugal, 16 August.

Tiihonen, J., and Felfernig, A. 2010. Towards Recommending Configurable Offerings. *International Journal of Mass Customization* 3(4): 389–406.

Tseng, H.; Chang, C.; and Chang, S. 2005. Applying Case-Based Reasoning for Product Configuration in Mass Customization Environments. *Expert Systems with Applications* 29(4): 913–925.

Alexander Felfernig is a professor of applied software engineering at the Graz University of Technology. In his research he focuses on different aspects of the application of recommendation and configuration technologies in industrial environments. Felfernig is a cofounder of ConfigWorks and author or coauthor of more than 150 papers in international journals, conferences, and workshops, including Recommender Systems—An Introduction. For his research he received the Heinz Zemanek Award from the Austrian Computer Society in 2009.

Andreas Falkner holds an MSc and a PhD in computer science. Being affiliated with Siemens AG Austria since 1992, he has developed product configuration frameworks and applications for complex technical systems in various domains such as railway interlocking systems. At present, he is a senior research scientist and program manager in the global technology field constraint-based configurators at Siemens' Corporate Research and Technology department.

Albert Haag is a development architect at SAP AG. He has been involved in designing and implementing the SAP product configurators since 1992 in various roles including project lead and group manager. He received a PhD in computer science from the University of Kaiserslautern and a Dipl. in mathematics from the University of Hamburg. Haag's research interests include product configuration applications and truth maintenance systems.