# Recommender Systems for Configuration Knowledge Engineering *

**A. Felfernig, S. Reiterer, M. Stettinger, F. Reinfrank, M. Jeran, and G. Ninaus**
Graz University of Technology
Inffeldgasse 16b, A-8010 Graz, Austria
{felfernig,reiterer,stettinger,reinfrank,jeran,ninaus}@ist.tugraz.at

## Abstract

The knowledge engineering bottleneck is still a major challenge in configurator projects. In this paper we show how recommender systems can support knowledge base development and maintenance processes. We discuss a couple of scenarios for the application of recommender systems in knowledge engineering and report the results of empirical studies which show the importance of user-centered configuration knowledge organization.

## 1 Introduction

Product knowledge changes frequently [Soloway, 1987]. Therefore, it must be possible to conduct knowledge base development and maintenance operations efficiently. Since the early developments of configurator applications in the late 1970's and early 1980's [McDermott, 1982], knowledge representations have been improved in terms of (1) *model-based approaches* which allow a clear separation of domain knowledge and problem solving algorithms, (2) *higher-level knowledge representations* which allow a component-oriented representation of configuration knowledge (see, e.g., [Stumptner *et al.*, 1998]), and (3) *graphical knowledge representations* (e.g., [Felfernig *et al.*, 2000; 2001]) which allow a compact representation. In addition to new knowledge representations, intelligent *diagnosis approaches* have been developed which help a knowledge engineer to identify and repair erroneous configuration knowledge [Junker, 2004; Felfernig *et al.*, 2004; 2009; 2013].

Due to diversification strategies of companies, product and service assortments are becoming increasingly large and complex [Huffman and Kahn, 1998]. The complexity of the underlying knowledge bases increases to the same extent which requires additional concepts that help a knowledge engineer to conduct knowledge base development and maintenance operations in an efficient fashion. Furthermore, knowledge bases are often developed by a group of persons with different knowledge, goals, and focuses with regard to development and maintenance operations. This situation requires *adaptive user interfaces* to be integrated into configuration knowledge engineering environments. Adaptive user interfaces for knowledge engineering have the potential to effectively support engineers and domain experts in activities such as *learning* (knowledge base understanding), *finding* (the relevant items in the knowledge base), and *testing & debugging* (removing the source of faulty behavior).

In order to offer more adaptivity in configurator development environments, we propose the application of different types of recommendation technologies [Jannach *et al.*, 2010] which proactively support domain experts and engineers when creating and adapting configuration knowledge. Such technologies should dispose of a basic understanding of cognitive processes when persons develop and maintain configuration knowledge bases. They should support functionalities such as *recommending* relevant items (variables, component types, constraints, diagnoses, etc.) and simultaneously *omitting* specific items that are not relevant. Recommender systems have the potential to provide such a support (see, e.g., [Robillard *et al.*, 2010]).

There are three basic recommendation approaches. *First*, *collaborative filtering* [Konstan *et al.*, 1997] determines recommendations based on the preferences of *nearest neighbors* (users with similar preferences compared to the current user). In this context, items are recommended to the current user which have received a positive rating by the nearest neighbors but are not known to the current user. *Second*, *content-based filtering* [Pazzani and Billsus, 1997] recommends items that are not known to the current user and are similar to items that have already been purchased by her/him. Similarity between items can be determined, for example, on the basis of the similarity of keywords used to describe the item. *Third*, *knowledge-based recommenders* recommend items by using constraints or similarity metrics [Burke, 2000; Felfernig and Burke, 2008].

This paper is organized as follows. In Section 2 we introduce example scenarios for the application of recommender technologies in knowledge engineering. Thereafter, we report results of related empirical studies (see Section 3). In Section 4 we provide a discussion of related work. Conclusions and a discussion of future research issues are given in Section 5.

## 2 Recommenders for Knowledge Engineering

**Collaborative Recommendation of Constraints**. Collaborative filtering (CF) recommender systems have shown to be

---

one of the best choices to achieve *serendipity effects*, i.e., to be surprised (in a positive sense) by item recommendations one did not expect when starting the recommendation process. In situations were knowledge engineers do not know the configuration knowledge base very well, collaborative recommendations can be exploited to support a more focused analysis of the knowledge base. The availability of navigation data from other knowledge engineers is the major precondition for determining recommendations with collaborative filtering. Table 1 shows an example of navigation data that describes in which order knowledge engineers (users) accessed the constraints of a knowledge base. For simplicity we assume that each of the users accessed each constraint (but in different order). Similar applications of collaborative filtering can be imagined for the recommendation of variables (or component types) and instances of a component catalog.

Table 1 stores the information in which order the constraints have been visited by knowledge engineers (users), for example, user 1 analyzed the constraints in the order $[c_5, c_2, c_3, c_1, c_4, c_6]$. Let us assume that the current user has already visited the constraints $c_5$ and (then) $c_2$. The nearest neighbors of the current user (users with a similar navigation behavior) are the users 1, 2, and 4. The majority of these users analyzed constraint $c_1$ in the third step – this one will be recommended to the current user. Note that this recommendation approach is currently under evaluation, therefore no related empirical results will be reported in Section 3.

| user | $c_1$ | $c_2$ | $c_3$ | $c_4$ | $c_5$ | $c_6$ |
|---|---|---|---|---|---|---|
| 1 | 4 | 2 | 3 | 5 | 1 | 6 |
| 2 | 3 | 2 | 5 | 6 | 1 | 4 |
| 3 | 1 | 3 | 2 | 4 | 6 | 5 |
| 4 | 3 | 2 | 4 | 5 | 1 | 6 |
| current | ? | 2 | ? | ? | 1 | ? |

Table 1: Recommending constraints ($c_i$) with *CF*.

**Content-based Clustering of Constraints**. Another possibility to support knowledge engineers is to cluster constraints with the goal to improve the overall clarity of the knowledge base. We will exemplify this on the basis of *k-means clustering* [Witten and Frank, 2005]. Following this approach, we have to generate $k$ initial *centroids* which act as (first) representatives of future clusters. In the following, each object (in our case: constraint) is assigned to the group (cluster) with the closest (most similar) centroid. Thereafter, centroids are recalculated. In our case, a centroid is defined as the object with the highest overall similarity to the other objects in the cluster. The algorithm terminates if the centroids are *stable* (do not change). *k-means clustering* is guaranteed to terminate but is not necessarily optimal since the outcome depends on the initial centroids ([Witten and Frank, 2005]).

For demonstration purposes we introduce the following simple configuration problem which is represented as a basic constraint satisfaction problem (CSP = (V, D, C)) where $V$ represents a set of variables $\{v_1, v_2, ..., v_5\}$, $D$ represents the set of corresponding domains ($dom(v_i) = \{1..5\}$), and $C$ represents the following set of constraints.
$\{c_1 : v_1 = 3 \rightarrow v_2 > 1, c_2 : v_1 = 3 \wedge v_3 = 1, c_3 : v_2 =$

$2 \rightarrow v_3 = 1, c_4 : v_3 = 1 \rightarrow v_1 \neq 1, c_5 : v_3 = 1 \rightarrow (v_4 = 2 \wedge v_1 > v_5), c_6 : v_4 \geq 1 \rightarrow v_5 \leq 4, c_7 : v_5 = 1 \rightarrow v_3 = 2 \vee v_3 = 3\}$.

On the basis of this simple knowledge base, we can calculate the similarities between the individual constraints $(c_a, c_b)$ by using Formula 1. In this formula, $V = variables(c_a) \cup variables(c_b)$, $co-occurrence(v, c_a, c_b) = 1$ if $v$ is contained in both constraints on the same position, $co-occurrence(v, c_a, c_b) = 0.5$ if $v$ is contained in both constraints but on a different position, and $co-occurrence(v, c_a, c_b) = 0$ of no co-occurrence exists. Note that this is *one possible approach to similarity determination*. We also compared this approach with operator-based similarity and a random assignment of constraints to clusters.

$$sim(c_a, c_b) = \frac{\sum_{v \in V} co-occurrence(v, c_a, c_b)}{|V|} \qquad (1)$$

The similarities between the pairs of individual constraints are depicted in Table 2.

| $c_i \in C$ | $c_1$ | $c_2$ | $c_3$ | $c_4$ | $c_5$ | $c_6$ | $c_7$ |
|---|---|---|---|---|---|---|---|
| $c_1$ | 1.0 | - | - | - | - | - | - |
| $c_2$ | 0.33 | 1.0 | - | - | - | - | - |
| $c_3$ | 0.16 | 0.33 | 1.0 | - | - | - | - |
| $c_4$ | 0.16 | 0.5 | 0.16 | 1.0 | - | - | - |
| $c_5$ | 0.1 | 0.25 | 0.1 | 0.37 | 1.0 | - | - |
| $c_6$ | 0.0 | 0.0 | 0.0 | 0.0 | 0.12 | 1.0 | - |
| $c_7$ | 0.0 | 0.33 | 0.33 | 0.16 | 0.12 | 0.16 | 1.0 |

Table 2: Similarities between individual constraints.

On the basis of these individual similarities we are able to determine a set of corresponding clusters ($k = 2$). The determination of such clusters is exemplified in Table 3. First, we (randomly) select two constraints as initial cluster centers (centroids): $c_1$ and $c_5$ (denoted by $cs$). In iteration 2 the center of cluster 1 changes to $c_2$ and we have to re-calculate the cluster assignment. After this iteration, the assignment is stable, i.e., the cluster centers ($c_2$ and $c_5$) remain the same.

| iteration | $c_1$ | $c_2$ | $c_3$ | $c_4$ | $c_5$ | $c_6$ | $c_7$ |
|---|---|---|---|---|---|---|---|
| 1 | 1($cs$) | 1 | 1 | 2 | 2($cs$) | 2 | 2 |
| 2 | 1 | 1($cs$) | 1 | 1 | 2($cs$) | 2 | 1 |

Table 3: k-means clustering of $C = \{c_1, c_2, ..., c_7\}$.

For the visualization of the constraints $\{c_1, c_2, ..., c_7\}$ this means that the knowledge base would be presented in terms of two constraint groups: $\{c_1, c_2, c_3, c_4, c_7\}$ and $\{c_5, c_6\}$.

**Knowledge-based Refactoring Recommendations**. The way in which semantics is expressed has an impact on the understandability of the knowledge base. For example, users need less time to understand the semantics of a knowledge base if implications are expressed in terms of $A \rightarrow B$ compared to the alternative representation of $\neg A \vee B$. Explicit knowledge about the cognitive complexity of constraint representations can be exploited to recommend structural and

semantics-preserving adaptations of knowledge structures. Such recommendations are knowledge-based, since they are explicitly encoded in refactoring rules.

## 3 Empirical Evaluation

For the *content-based clustering of constraints* and *knowledge-based refactoring recommendations* we now present the results of two empirical studies. In the first study, we compared the applicability of three different clustering strategies with regard to knowledge engineering tasks (*find a solution*, *find a minimal conflict*) (see, e.g., [Junker, 2004]).

**Study A: Clustering of Constraints**. For two different configuration knowledge bases ($kba_1$, $kba_2$) we conducted a study based on an within-subjects design (N=40). Each study participant (students of computer science who visited a related course on knowledge engineering) had the task of (1) finding a solution (in $kba_1$) and (2) finding a minimal conflict (in $kba_2$).[1] There were no time limits regarding task completion. Each student was assigned to one type of clustering (one out of variable-based similarity, operator-based similarity, and random clustering), i.e., we did not vary the type of clustering per student. The knowledge bases ($kba_1$, $kba_2$) were defined as CSPs in a domain-independent fashion in order to avoid an additional cognitive complexity related to the understanding of a product domain. The basic properties of the used knowledge bases are summarized in Table 4.

| Knowledge base | $\#(v_i \in V)$ | $v_i$ domain size | $\#(c_i \in C)$ |
|---|---|---|---|
| $kba_1$ | 5 | 5 | 15 |
| $kba_2$ | 10 | 3 | 10 |

Table 4: Knowledge bases used in *Study A*.

The outcome of this experiment is shown in Table 5.

| Grouping approach | $kba_1$: SOL | $kba_2$: CON |
|---|---|---|
| Similar variables | 21.43% | 42.86% |
| Similar operators | 30.77% | 53.85% |
| Random | 38.46% | 76.92% |

Table 5: Error rates for completing the tasks *find a solution (SOL)* and *find a conflict (CON)* depending on clustering approach (variable-based, operator-based, or random).

From the three compared approaches to the clustering of constraints in a configuration knowledge base, variable similarity based clustering clearly outperforms operator-based clustering and random clustering of constraints.

**Study B: Cognitive Complexities**. There are different possibilities to represent equivalent semantics on the basis of a constraint, for example, the *requires* relationship $X \rightarrow Y$ can be represented in terms of $\neg X \vee Y$. The *incompatibility* relationship $\neg(X \wedge Y)$ can be represented as $X \rightarrow \neg Y$. Table 6 depicts five different possibilities to express *requires* and *incompatibility* relationships.

---

[1]We used these tasks to measure knowledge understanding. Further more differentiated tasks are within the scope of future work.

| *Requires* | *Incompatibility* |
|---|---|
| $X \rightarrow Y$ | $X \rightarrow \neg Y$ |
| $\neg X \vee Y$ | $\neg X \vee \neg Y$ |
| $\neg Y \rightarrow \neg X$ | $Y \rightarrow \neg X$ |
| $\neg(X \wedge \neg Y)$ | $\neg(X \wedge Y)$ |
| $Y \leftarrow X$ | $\neg Y \leftarrow X$ |

Table 6: Five different possibilities of representing *requires* and *incompatibility* relationships.

Study B is based on an within-subjects design (N=66) with two configuration knowledge bases. Knowledge base $kbb_1$ consisted of a set of *requires* constraints and $kbb_2$ consisted of a set of *incompatibility* constraints. Each study participant (again, computer science students who visited a related knowledge engineering course) had the task of finding a solution for the given CSP. Each participant was confronted with one version of $kbb_1$ and one version of $kbb_2$ conform the schema depicted in Table 6. For example, if a student received the $X \rightarrow Y$ version of $kbb_1$ then she/he also received the $X \rightarrow \neg Y$ version of $kbb_2$. The knowledge bases $kbb_1$ and $kbb_2$ were (again) defined in a domain-independent fashion (see Study A). The basic properties of the used knowledge bases are summarized in Table 7.

| Knowledge base | $\#(v_i \in V)$ | $v_i$ domain size | $\#(c_i \in C)$ |
|---|---|---|---|
| $kbb_1$ | 5 | 5 | 7 |
| $kbb_2$ | 3 | 3 | 5 |

Table 7: Knowledge bases used in *Study B*.

The outcome of this experiment is shown in Table 8.

| $kbb_1$: SOL | errors | $kbb_2$: SOL | errors |
|---|---|---|---|
| $X \rightarrow Y$ | 21.43% | $X \rightarrow \neg Y$ | 14.29% |
| $\neg X \vee Y$ | 50.0% | $\neg X \vee \neg Y$ | 34.62% |
| $\neg Y \rightarrow \neg X$ | 96.43% | $Y \rightarrow \neg X$ | 50.0% |
| $\neg(X \wedge \neg Y)$ | 73.08% | $\neg(X \wedge Y)$ | 42.31% |
| $Y \leftarrow X$ | 25.0% | $\neg Y \leftarrow X$ | 16.67% |

Table 8: Error rates in solution identification (*SOL*) depending on constraint representation.

A result of the study is that basic implications ($\rightarrow$) should be preferred to other representations in order to maximize understandability. The only type of knowledge representation with a similar performance is the reverse implication, however, when comparing both alternatives, the standard implication seems to be the better choice.

## 4 Related Work

There is a long history of research on the improvement of knowledge engineering processes. Early research focused on model-based knowledge representations that allowed a separation of domain and problem solving knowledge. An example of such a representation are constraint technologies which became extremely popular as a technological basis for industrial applications [Freuder, 1997]. In a next step, graphical

knowledge representations [Felfernig *et al.*, 2000] and intelligent techniques for knowledge base testing and debugging have been developed [Felfernig *et al.*, 2004]. The need of an intuitive access to a corpus of software artifacts is also one of the major requirements for *software comprehension* [Storey, 2006]. In this context, recommender systems [Jannach *et al.*, 2010] have already been identified as a valuable means to provide intelligent support for the navigation in large and complex software spaces (see, e.g., [Robillard *et al.*, 2010]). The application of recommendation technologies for supporting knowledge engineering processes is a new research area. Research contributions in this field have the potential to significantly improve the overall quality of knowledge engineering processes. In [Felfernig *et al.*, 2010] basic knowledge representations are compared, for example, the use of $\rightarrow$ to represent an implication vs. the use of $\neg$ and $\vee$. This work is an important step towards a discipline of *empirical knowledge engineering* with a clear focus on usability aspects and cognitive efforts needed to complete knowledge engineering tasks. The work presented in this paper is a continuation of the work of [Felfernig *et al.*, 2010]. It takes a more detailed look at different alternative representations of *requires* and *incompatibility* relationships and introduces a new concepts related to the content-based clustering of constraints.

## 5   Conclusions

In this paper we showed how recommenders can be exploited to support knowledge engineering tasks. Examples are collaborative filtering of constraint sets, clustering of constraints, and knowledge-based recommendation of refactoring operations. Future work will include the development of further recommendation algorithms, for example, the inclusion of content-based filtering and further clustering algorithms as well as further empirical studies with more differentiated maintenance tasks. Finally, we will focus on an in-depth analysis of existing research in the area of cognition psychology which can further advance the state of the art in (configuration) knowledge engineering.

## References

[Burke, 2000] R. Burke. Knowledge-based recommender systems. *Library and Inf. Systems*, 69(32):180–200, 2000.

[Felfernig and Burke, 2008] A. Felfernig and R. Burke. Constraint-based recommender systems: Technologies and research issues. In *ACM International Conference on Electronic Commerce (ICEC08)*, pages 17–26, 2008.

[Felfernig *et al.*, 2000] A. Felfernig, G. E. Friedrich, and D. Jannach. UML as Domain Specific Language for the Construction of Knowledge-based Configuration Systems. *IJSEKE*, 10(4):449–469, 2000.

[Felfernig *et al.*, 2001] A. Felfernig, G. Friedrich, and D. Jannach. Conceptual modeling for configuration of mass-customizable products. *Artificial Intelligence in Engineering*, 15(2):165–176, 2001.

[Felfernig *et al.*, 2004] A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner. Consistency-based diagnosis

of configuration knowledge bases. *Artificial Intelligence*, 152(2):213–234, 2004.

[Felfernig *et al.*, 2009] A. Felfernig, G. Friedrich, M. Schubert, M. Mandl, M. Mairitsch, and E. Teppan. Plausible repairs for inconsistent requirements. In *21st Intl. Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 791–796, Pasadena, CA, 2009.

[Felfernig *et al.*, 2010] A. Felfernig, M. Mandl, A. Pum, and M. Schubert. Empirical knowledge engineering: Cognitive aspects in the development of constraint-based recommenders. In *IEA/AIE 2010*, pages 631–640, Cordoba, Spain, 2010.

[Felfernig *et al.*, 2013] A. Felfernig, M. Schubert, and S. Reiterer. Personalized Diagnosis for Over-Constrained Problems. In *23rd International Conference on Artificial Intelligence*, Peking, China, 2013.

[Freuder, 1997] E. Freuder. In pursuit of the holy grail. *Constraints*, 2(1):57–61, 1997.

[Huffman and Kahn, 1998] C. Huffman and B. Kahn. Variety for Sale: Mass Customization or Mass Confusion. *Journal of Retailing*, 74:491–513, 1998.

[Jannach *et al.*, 2010] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender Systems*. CUP, 2010.

[Junker, 2004] U. Junker. Quickxplain: Preferred explanations and relaxations for over-constrained problems. In *19th National Conference on Artificial Intelligence (AAAI04)*, pages 167–172, San Jose, CA, 2004.

[Konstan *et al.*, 1997] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Riedl. Grouplens: applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[McDermott, 1982] J. McDermott. R1: A Rule-based Configurer of Computer Systems. *Artificial Intelligence Journal*, 19:39–88, 1982.

[Pazzani and Billsus, 1997] M. Pazzani and D. Billsus. Learning and revising user profiles: the identification of interesting websites. *Mach. Learn.*, 27:313–331, 1997.

[Robillard *et al.*, 2010] M. Robillard, R. Walker, and T. Zimmermann. Recommendation systems for software engineering. *IEEE Software*, 27(4):80–86, 2010.

[Soloway, 1987] E. et al. Soloway. Assessing the Maintainabiliy of XCON-in-RIME: Coping with the Problem of very large Rule-bases. In *Proc. of AAAI-87*, pages 824–829, Seattle, Washington, USA, July 13–17 1987.

[Storey, 2006] M. Storey. Theories, tools and research methods in program comprehension: past, present and future. *Software Quality Journal*, 14:187–208, 2006.

[Stumptner *et al.*, 1998] M. Stumptner, G. Friedrich, and A. Haselböck. Generative Constraint-based Configuration of Large Technical Systems. *AI EDAM*, 12(4):307–320, 1998.

[Witten and Frank, 2005] I. Witten and E. Frank. *Data Mining*. Morgan Kaufman, 2005.