Towards Open Configuration

Alexander Felfernig[1] and **Martin Stettinger**[1] and **Gerald Ninaus**[1] and **Michael Jeran**[1] and **Stefan Reiterer**[2] and **Andreas Falkner**[3] and **Gerhard Leitner** [4] and **Juha Tiihonen**[5]

Abstract. Configuration technologies are typically applied in *closed* settings where one (or a small group of) knowledge engineer(s) is in charge of knowledge base development and maintenance. In such settings it is also assumed that only single users configure the corresponding products and services. Nowadays, a couple of scenarios exist that require more *openness*: it should be possible to cooperatively develop knowledge bases and to jointly configure products and services, even by adding new features or constraints in a flexible fashion. We denote this integration of groups of users into configuration-related tasks as *open configuration*. In this paper we introduce features of open configuration environments and potential approaches to implement these features.

1 Introduction

Configuration [8, 24, 37] is one of the most successful technologies of Artificial Intelligence (AI). It is applied in many domains such as telecommunication [17], furniture [19], and financial services [9]. Most configuration-related functionalities are assuming closed settings where knowledge bases are developed by a single (or a small group of) knowledge engineer(s) and the corresponding configurators are applied by single users. Implementing configurator applications this way entails drawbacks which become manifest in terms of *scalability problems* in knowledge engineering [33] and *suboptimal decisions* if a single user decides for the whole group [16].

Scalability Problems. The transformation of domain knowledge into a configuration knowledge base is an effortful process often characterized by a knowledge acquisition bottleneck [20] that is considered as a major obstacle for a sustainable application of knowledge-based technologies [21, 41]. To tackle this bottleneck, efficient approaches have been developed that support graphical knowledge engineering [7, 22] and intelligent debugging [6, 14, 35].

These approaches help to improve the efficiency of knowledge engineering but still do not solve the problem of *missing scalability*: the increasing amount and complexity of configuration knowledge bases exceeds the resources available for performing the corresponding development and maintenance operations [23, 33]. In order to assure scalability, future configuration technologies have to support a deeper integration of a wider group of users (e.g., product developers, marketing experts, sales representatives, and knowledge engineers) into knowledge engineering. Related solutions should go beyond state-of-the-art approaches that are focusing on experienced knowledge engineers and programmers [24] by allowing the completion of knowledge engineering tasks by the mentioned groups. We denote this approach as *community-based knowledge engineering*.

Suboptimal Decisions. A basic assumption of existing configuration systems is that products and services are typically configured by single users. However, many scenarios exist where not a single user but a group of users is in charge of configuring a product (see Section 3). Existing configuration environments do not take into account such scenarios which often leads to situations where a single user has to "encode" the requirements and preferences of a whole group. This can lead to suboptimal configurations (decisions) that do not reflect the group preferences in an optimal fashion. Future configuration technologies should take into account the fact that groups of users can be engaged in configuration processes and provide group decision mechanisms that help the group to jointly configure a product in a consensual fashion. We denote this type of configuration as *group-based configuration*. Especially in scenarios where multiple stakeholders define and configure products, enhanced flexibility is required: configurator users may request to add or refine product features and constraints which can be seen, for example, in open innovation [4] or postponement scenarios [18, 42]. We subsume such activities under the term *flexible product enhancement*.

The concepts of *community-based knowledge engineering*, *group-based configuration*, and *flexible product enhancement* can be summed up under the notion of *open configuration*. In this paper we sketch functionalities which have to be provided by open configuration environments. In Section 2 we introduce features and potential technological solutions to tackle the issue of scalability in knowledge engineering scenarios. In Section 3 we discuss features of group-based configuration. In Section 4 we discuss aspects of product enhancement in open configuration. With Section 5 we provide a discussion of related work. We conclude the paper with Section 6.

2 Community-based Knowledge Engineering

In the following we will discuss aspects that become relevant if we want to integrate a larger group of users into configuration knowledge engineering. For the sake of simplicity and without loss of generality we assume that a configuration knowledge base is represented in terms of a constraint satisfaction problem (CSP) [27] consisting of a set of variables $V = \{v_1, ..., v_n\}$ with corresponding domain definitions (dom(v_i)), and a set of constraints $C = \{c_1, ..., c_m\}$. We base our discussions on the following simplified financial services configuration knowledge base.

- V = {willingness to take risks (wr), expected return rate (rr), investment period (ip)}
- dom(wr)= {low, medium, high}, dom(rr)={<6%, 6-9%,>9%}, dom(ip) = {shortterm, mediumterm, longterm}

[1] TU Graz, Austria, email: {firstname.lastname}@ist.tugraz.at
[2] SelectionArts, Austria, email: stefan.reiterer@selectionarts.com
[3] Siemens, Austria, email: andreas.a.falkner@siemens.com
[4] University of Klagenfurt, Austria, email: gerhard.leitner@aau.at
[5] Aalto University, Finland, email: juha.tiihonen@aalto.fi

micro task topic	description
variables	definition/evaluation of variables included in V
questions	definition/evaluation of questions related to $v_i \in V$
dialog sequences	definition/evaluation of question sequences
constraints	definition/evaluation of constraints in C
examples	definition/evaluation of test cases in T
diagnoses	evaluation of conflict resolution alternatives for C

Table 1. Community-based knowledge engineering: example micro tasks.

- $C = \{c_1 : wr = medium \rightarrow ip \neq shortterm,$
  $c_2 : wr = high \rightarrow ip = longterm,$
  $c_3 : ip = longterm \rightarrow rr = <6\% \vee rr = 6\text{-}9\%,$
  $c_4 : rr = >9\% \rightarrow wr = high,$
  $c_5 : rr = 6\text{-}9\% \rightarrow wr \neq low \wedge wr \neq medium\}$

In cases where one or a small group of knowledge engineers is in charge of developing and maintaining a configuration knowledge base, attributes (component types), domains, and related constraints are typically formalized on the basis of examples and textual descriptions provided by domain experts [24]. If the product domain knowledge has to be adapted, the whole process is restarted, i.e., domain experts articulate the change requests in an informal fashion and knowledge engineers implement the needed adaptations.

The correctness of changes performed on a knowledge base can be evaluated, for example, on the basis of regression tests where positive and negative test cases are used to figure out whether the knowledge base shows the intended behavior [6]. Positive test cases (examples) are a specification of an intended behavior of the knowledge base and negative test cases exemplify unintended behavior. Existing approaches to configuration knowledge base testing and debugging exploit positive test cases to detect errors/deficiencies by inducing conflicts in the incorrect configuration knowledge base. Such conflicts are minimal sets of constraints that are responsible for the faulty behavior of the knowledge base and therefore have to be adapted by knowledge engineers.

Community-based Knowledge Engineering. Intelligent testing and debugging [6] is an important contribution to the improvement of knowledge engineering processes. However, the growing size and complexity of configuration knowledge bases often makes it hard for individual knowledge engineers to keep track of new developments and adaptations. As a consequence, more time is needed to provide a new production version of the configuration knowledge base and the probability of including erroneous constraints increases. In order to assure scalability, it is important to integrate end-users more deeply into knowledge base development and maintenance and thus to exploit unemployed knowledge engineering potentials.

In the following we discuss issues that have to be taken into account when integrating groups into *community-based knowledge engineering* processes. An in-depth integration of a larger group of users allows knowledge engineers to delegate basic engineering tasks (so-called *micro tasks*). Table 1 provides an overview of micro task topics. For each topic a couple of different concrete micro tasks can be defined, for example, a variable can be defined but also evaluated with regard to the appropriateness of it's domain definition.

In order to figure out variables (component types) relevant for the configuration knowledge base, users should be allowed to enter proposals for variables and component types (including the corresponding domain definitions) on their own. Variables are often associated with questions posed to the user of a configurator application – alternative formulations of such questions and also the sequences in which these questions are posed should be defined and evaluated by users. In addition to structural properties typically defined in terms of variables or component types and their relationships, constraints define additional restrictions on possible combinations of variable values (components).

Especially in community-based scenarios, where a larger number of users interacts with the knowledge engineering environment, engineering practices will change in the sense that users are providing knowledge chunks in a collaborative fashion and the knowledge engineering environment is in charge of aggregating this information. In this context, it is necessary to have mechanisms that automatically distribute knowledge acquisition tasks among users in a systematic fashion (e.g., depending on the workload, knowledge level, and preferences of users). Such tasks can be represented in a more-or-less traditional form of todo-lists but can also be represented in terms of so-called *games with a purpose* [40] which is an upcoming trend also in the knowledge engineering field [36].

A simple example of such a knowledge acquisition interface is depicted in Figure 1. In this example game, the users *Ann* and *Paul* have the task to cooperatively figure out combinations of customer requirements that are incompatible, i.e., induce an inconsistency with the knowledge base. The players have successfully completed their task if they, for example, selected the same set of assignments as candidates for incompatibilities. The underlying assumption of this game is that *Ann* does not know the input of *Paul* and vice-versa.

Further examples of gamification-based interfaces for configuration knowledge acquisition are: cooperative definition of relevant variables (including their domains), the estimation of intuitive dialog sequences (which questions should be asked in which order), the derivation of further constraint types (e.g., filter constraints that match user requirements to corresponding technical product properties), and the estimation of accepted repair rankings in situations where no solution could be found. Such scenarios can be supported by input templates that represent micro-tasks (see Figure 1).

Testing and Debugging. The definition and evaluation of (positive and negative) test cases is a crucial issue since the correctness of a test suite directly influences the correctness of the results determined by a configurator. In [6] positive and negative examples are exploited for debugging knowledge bases on the basis of the concepts of model-based diagnosis [32]. In this context, positive examples are exploited for inducing conflicts in a configuration knowledge base. A negative example is assumed to be integrated in negated form into the knowledge base in the case that it has not been rejected by the knowledge base. On the basis of the following two test cases (examples) we can show how positive examples are used to find errors in the knowledge base. Both test cases are in conflict with constraints

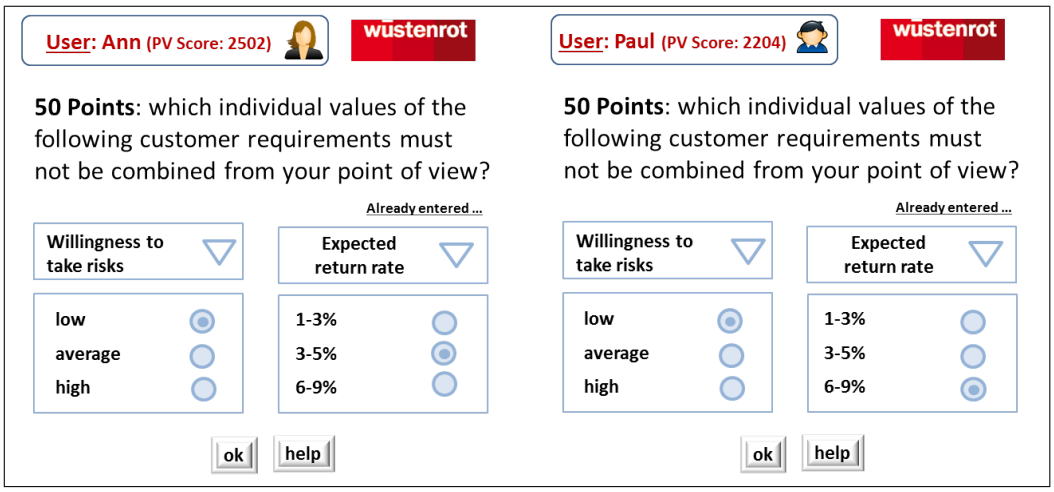

Figure 1. Sketch of a user interface for game-based knowledge acquisition. The overall goal of the game is that both players agree on the set of incompatible value combinations of a given set of variables. This user interface can be regarded as a micro task template for the acquisition of incompatibility constraints.

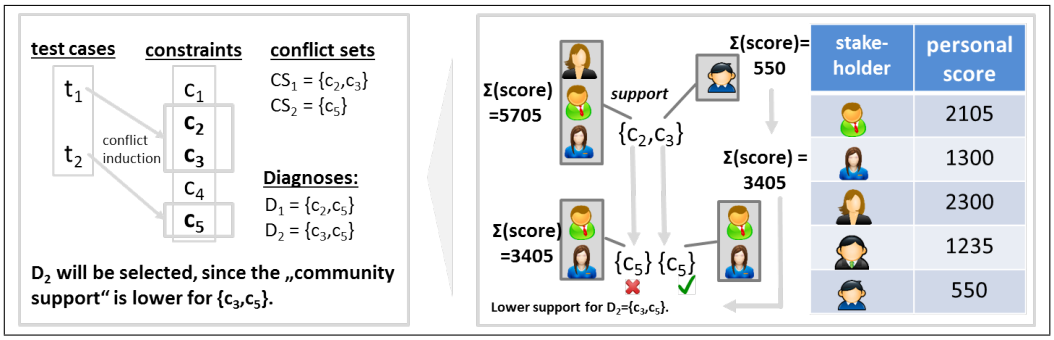

Figure 2. Group-based diagnosis of a faulty configuration knowledge base. Diagnoses are selected by taking into account the expertise of users/knowledge engineers: the higher the *personal score* (value derived from his/her personal contributions), the higher the weight given to his/her opinion.

in the configuration knowledge base introduced in Section 2.

- $t_1 : wr = high \wedge rr = >9\%$
- $t_2 : rr =$ 6-9% $\wedge\ wr = medium$

A conflict between a test case t and a set of constraints in the configuration knowledge base can be defined as a *conflict set* $CS \subseteq C$: $CS \cup t$ inconsistent. Such a conflict set CS is *minimal* if there does not exist another conflict set CS' with $CS' \subset CS$. To resolve a minimal conflict, only one element has to be deleted from CS. In our example, the test case t_1 is in conflict with the constraints c_2 and c_3 and test case t_2 is in conflict with the constraint c_5. Consequently we have two different (and minimal) conflict sets which are CS_1: $\{c_2, c_3\}$ and CS_2: $\{c_5\}$. Resolving these conflicts results in two different diagnoses, namely $D_1 = \{c_2, c_5\}$ and $D_2 = \{c_3, c_5\}$, i.e., a diagnosis is a hitting set [32] which includes at least one constraint from each of the given conflict sets.

Typically, there are many alternative diagnoses and the question has to be answered which of these is acceptable for the users engaged in testing and debugging. Figure 2 depicts a basic approach of integrating knowledge about the users expertise in the determination of a diagnosis. For the conflict $CS_1 = \{c_2, c_3\}$, the majority of users prefers to keep c_2 as-is and to delete or change c_3 to resolve the conflict. Since CS_2 is a singleton, no alternatives exist for resolving the conflict, i.e., c_5 must be selected. Overall, the elements in the diagnosis $D_2 = \{c_3, c_5\}$ have a lower community support and therefore will be changed or deleted by the users in order to restore the consistency with the test-suite $\{t_1, t_2\}$.

3 Group-based Configuration

An assumption of existing configuration environments is that there is no need for additional configuration support in scenarios where groups of users are jointly configuring their preferred product or service. A major consequence of this assumption is that single users are forced to encode the preferences of a group which is often done in a suboptimal fashion.

Within the scope of an industry study with representatives of N=25 companies applying configurators we figured out that none of the existing configuration environments provides technologies that support groups of users in jointly configuring a solution. However, there is a strong agreement on the fact that such technologies have to be included in future configurators. The study participants reported different scenarios for the application of group-based (socially aware) configuration technologies. Social awareness in this context denotes the fact that specific properties of group decision processes are explicitly taken into account by the configuration environment (e.g.,

ID	domain for group-based configuration	components and constraints	decision makers
1	software release plans	requirements, releases, dependencies, preferences	stakeholders in software project
2	product line scoping and open innovation	(new) features, constraints between features, preferences	representatives from different departments, customers
3	bundle configuration (e.g., hotel, flight, tour, etc.)	(new) destinations, hotels, sightseeing tours, (resource) constraints, preferences	travel group
4	stakeholder selection for a new software project	(new) persons, constraints regarding competences and resources, preferences	(initial) team members
5	architectural design in software development	components, interfaces, technologies, constraints between components, preferences	(distributed) software project members
6	financial service configuration	financial services, resource constraints, preferences	family members
7	building configuration (e.g. smart home, office block)	rooms, furniture, light control equipment, constraints between components, preferences	family members, suppliers, company representatives
8	funding decisions	project proposals, resource constraints, preferences	evaluators, consultants, decision makers

Table 2. Application scenarios for *group-based configuration* identified within the scope of a study with N=25 companies applying configuration systems.

the need to achieve consensus among group members). Examples of such scenarios are depicted in Table 2.

In these scenarios a group of users is in charge of *jointly configuring a product or service*, for example, when configuring a *holiday trip* (bundle configuration) for a group of friends [25], the requirements and preferences of all group members should be taken into account. When configuring a *software release plan*, the preferences of individual stakeholders regarding the assignment of requirements to releases have to be taken into account [31].

Taking into account requirements and preferences of group members requires decisions regarding *trade-offs*. In the context of holiday trips such a trade-off could be the acceptance of a lower-quality hotel which is much nearer to the sightseeing destination preferred by a specific user. When configuring software release plans, a trade-off could concern the postponement of a specific requirement to a later release while increasing the importance level of this requirement (to avoid further postponements).

The determination of trade-offs must be based on preference aggregation mechanisms [29] that take into account the preferences of all group members as far as possible. For example, the *least misery* strategy avoids massive discriminations of individual group members by minimizing the maximum number of trade-offs to be accepted by an individual. In contrast, *majority voting* follows the opinions of the majority of the group members which can lead to discriminations against individuals.

An example of the application of the least misery strategy in the context of deciding about a common sightseeing trip is depicted in Table 3. In this simplified example, each person is allowed to select at most two destinations and the corresponding trip must include two destinations. Since Ben and John have similar preferences, majority voting would discriminate Kate. In contrast, least misery tries to find a trade-off that has the potential to create group consensus. For a detailed discussion of preference aggregation mechanisms we refer the reader to [29].

A major issue for future research is the consideration of longer time periods. For example, if a group of friends jointly configures a holiday trip every year, the aggregation mechanisms used by the group-based configuration environment should take into account (as far as possible) the degree to which individuals had to accept trade-offs in the past and use this information for the recommendation of fair trade-offs in future configuration sessions.

On the technical level the above mentioned properties require basic research in the following areas.

First, constraint-based search methods have to be extended with mechanisms that help to predict (partial) configurations which are of relevance for the group. This requires learning methods for search heuristics [34] that help to predict relevant configurations in an efficient fashion. Furthermore, it is important that configurators are able to determine similar and diverse configurations efficiently which could also be achieved on the basis of the mentioned heuristics.

Second, the determination of trade-offs for inconsistent requirements and preferences has to be based on efficient diagnosis methods integrated with intelligent preference aggregation mechanisms [29] that can help to better predict trade-offs acceptable for all group members. These aggregations must take into account the histories stored in interaction logs in order to guarantee decision fairness in the long run.

Third, negotiation and argumentation mechanisms have to be developed which support individuals to express acceptable trade-offs. In our holiday configuration scenario an example of such a statement is ”I accept to visit Greece this year if we agree to organize a trip to Italy next year”. Such arguments cannot be expressed on the basis of existing preference representations.

4 Flexible Product Enhancement

The ability to include additional variables (component types), values (components), and constraints in a flexible fashion is important for the implementation of open configuration.

destination	Lindwurm	Großglockner	Pyramidenkogel	Isonzo Valley
Ben	1	1	0	0
John	1	1	0	0
Kate	0	0	1	1
least misery	1	0	1	0
majority voting	1	1	0	0

Table 3. Example set of tourist destinations (in the Alps-Adriatic area). The assumption in this example is that each person is allowed to articulate at most two preferences and the trip must include at least two destinations.

Product line scoping [26] (in the context of software product line engineering) is in the need of such a flexibility since the features and constraints element of the product line are not completely predefined at the beginning of the engineering process. A larger group of users has to jointly decide which components (features) and constraints should be part of the product line. Thus, product line scoping can be interpreted as open configuration where new alternatives and constraints (and preferences) can be integrated within the scope of the configuration (product line scoping) process.

Open innovation [4] reflects the idea of integrating customer communities into new product development processes of a company. In this context, variability modeling for product lines also requires the support of an easy integration of new component types, components, and constraints which reflect features to be supported by future products. In both scenarios, the integration of new items has to be supported by corresponding group decision processes (see Section 3), for example, before a new feature is integrated into the model, the group has to perform the needed validation steps and decide about the inclusion of the feature. This also holds for the afore mentioned scenarios of release planning and holiday trip configuration.

A further example of the need for flexible enhancements are *postponement strategies* [18, 42]. An example is the automotive industry, where basic car configurations are delivered to dealers who can then integrate additional components such as MP3 players and towbars, i.e., are enabled to integrate their own products and services into the basic configuration delivered by car producers. Conform to the definition given in [18], the mentioned scenario is of *type-III* where customers are allowed to specify additional equipment when they already have a more precise idea of the interior of the car. The corresponding configuration model has to provide flexible interfaces that allow an easy integration of new component types, components, and constraints. A knowledge representation concept that can be exploited in this context are *contextual models* [10] which allow a systematic extension of existing base diagrams with additional items relevant in a specific context (e.g., the car dealer context). In such scenarios, developers of configurator solutions also have to take into account that – depending on the additional items introduced – search heuristics [34] have to be adapted in order to assure efficient search.

5 Related and Future Work

Intelligent testing and debugging methods for configuration knowledge bases have been introduced in [6] where positive test cases can detect errors by inducing conflicts in a configuration knowledge base. Conflicts are then resolved on the basis of model-based diagnosis [32]. In open configuration scenarios, testing and debugging approaches have to be adapted to group-based settings where diagnosis discrimination has to take into account group preferences.

Bessiere et al. [2] introduced basic mechanisms to the learning of constraint sets. In this context, knowledge bases are learned on the basis of positive and negative examples. Generated examples are presented to users who have to decide whether the examples are positive or negative. Learning is based on a so-called *bias* that is a knowledge base generated from a vocabulary (variables, domains, and operators). The bias is systematically reduced on the basis of the information included in the examples, for instance, all conflicts induced in the bias by a positive example have to be resolved. In the case of a negative example, at least one conflict must be preserved which guarantees the rejection of the negative example. Approaches to the application of association rule mining for configuration knowledge discovery are discussed in [23]. An important research issue in this context is to assure the understandability and manageability of the derived configuration knowledge [12].

Human Computation is based on the idea of passing those tasks to humans which are easy to solve for them but are not solvable by computers [39]. Related research has already been conducted in the areas of ontology construction (concept learning) [36] and sentiment analysis in text documents [30]. A major idea of the work presented in this paper is to exploit the concepts of Human Computation as a central mechanism for configuration knowledge base construction and maintenance. These mechanisms go beyond concept learning [36] and include tasks such as diagnosis discrimination, test case classification and evaluation, and configuration dialog design.

Preferences are not known beforehand but are constructed within the scope of a decision process [3, 38]. As a result, biases occur which often lead to suboptimal decisions. Concepts to deal with (group) decision problems in recommender systems are discussed in [11, 15, 25, 28, 31]. A major issue for future research in this context is an in-depth investigation of decision biases in group decision making. An important question is to which extent biases are compensated or become more intense when groups decide.

6 Conclusions

In this paper we introduced central ideas and research questions related to open configuration. Openness in this context is related to the idea of a closer integration of end-users into configuration knowledge base development and maintenance operations and of supporting decision processes in scenarios where groups of users are in charge of configuring a product or service. Furthermore, open configuration is often characterized by the need of being able to integrate new items (e.g., component types, components, and constraints) ”on the fly”. On the basis of the results of a first industry study we reported example application domains and discussed related research challenges. The concepts presented in this paper can be applied in a broad range of scenarios which go beyond *open configuration*. Further example application domains are (constraint-based) scheduling [1], recommender systems [5], and utility evaluation where user groups are in

charge of evaluating alternatives [13].

ACKNOWLEDGEMENTS

The work presented in this paper has been conducted in the research project PEOPLEVIEWS funded by the Austrian Research Promotion Agency (843492).

REFERENCES

[1] P. Baptiste, C. Le Pape, and W. Nuijten, *Constraint-based Scheduling*, Kluwer, 2001.

[2] C. Bessiere, R. Coletta, B. O'Sullivan, and M. Paulin, 'Query-driven constraint acquisition', in *21st International Joint Conference on Artificial Intelligence (IJCAI'07)*, pp. 50–55, Hyderabad, India, (2007).

[3] J. Bettman, M. Luce, and J. Payne, 'Constructive consumer choice processes', *Journal of Consumer Research*, **25**(3), 187–217, (1998).

[4] H.W. Chesbrough, *Open Innovation. The New Imperative for Creating and Profiting from Technology*, Harvard Business School Press, Boston, 2003.

[5] A. Felfernig D. Jannach, M. Zanker and G. Friedrich, *Recommender Systems – An Introduction*, Cambridge University Press, 2010.

[6] A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner, 'Consistency-based diagnosis of configuration knowledge bases', *Artificial Intelligence*, **152**(2), 213–234, (2004).

[7] A. Felfernig, G. E. Friedrich, and D. Jannach, 'UML as Domain Specific Language for the Construction of Knowledge-based Configuration Systems', *International Journal of Software Engineering and Knowledge Engineering*, **10**(4), 449–469, (2000).

[8] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, *Knowledge-based Configuration – From Research to Business Cases*, Elsevier/Morgan Kaufmann Publishers, 2014.

[9] A. Felfernig, K. Isak, K. Szabo, and P. Zachar, 'The VITA Financial Services Sales Support Environment', in *AAAI/IAAI 2007*, pp. 1692–1699, Vancouver, Canada, (2007).

[10] A. Felfernig, D. Jannach, and M. Zanker, 'Contextual Diagrams as structuring mechanisms for designing configuration knowledge bases in UML', in *3rd International Conference on the Unified Modeling Language (UML2000)*, number 1939 in LNCS, pp. 240–254, (2000).

[11] A. Felfernig, W. Maalej, M. Mandl, F. Ricci, and M. Schubert, 'Recommendation and decision technologies for requirements engineering', in *ICSE 2010 Workshop on Recommender Systems in Software Engineering*, pp. 1–5, Cape Town, South Africa, (2010).

[12] A. Felfernig, S. Reiterer, M. Stettinger, F. Reinfrank, M. Jeran, and G. Ninaus, 'Recommender Systems for Configuration Knowledge Engineering', in *Workshop on Configuration*, pp. 51–54, Vienna, Austria, (2013).

[13] A. Felfernig, S. Schippel, G. Leitner, F. Reinfrank, K. Isak, M. Mandl, P. Blazek, and G. Ninaus, 'Automated Repair of Scoring Rules in Constraint-based Recommender Systems', *AI Communications*, **26**(2), 15–27, (2013).

[14] A. Felfernig, M. Schubert, and C. Zehentner, 'An efficient diagnosis algorithm for inconsistent constraint sets', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM)*, **26**(1), 53–62, (2012).

[15] A. Felfernig, E. Teppan, and B. Gula, 'Knowledge-based recommender technologies for marketing and sales', *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, **21**(2), 333–354, (2006). Special issue of Personalization Techniques for Recommender Systems and Intelligent User Interfaces.

[16] A. Felfernig, C. Zehentner, G. Ninaus, H. Grabner, W. Maaleij, D. Pagano, L. Weninger, and F. Reinfrank, 'Group Decision Support for Requirements Negotiation', in *Advances in User Modeling, Springer Verlag*, volume 7138 of *Lecture Notes in Computer Science*, pp. 105–116, (2012).

[17] Gerhard Fleischanderl, Gerhard E. Friedrich, Alois Haselböck, Herwig Schreiner, and Markus Stumptner, 'Configuring large systems using generative constraint satisfaction', *IEEE Intelligent Systems*, **13**(4), 59–68, (1998).

[18] C. Forza, F. Salvador, and A. Trentin, 'Form postponement effects on operational performance: a typological theory', *International Journal of Operations and Production Management*, **28**, 1067–1094, (2008).

[19] A. Haag, 'Sales Configuration in Business Processes', *IEEE Intelligent Systems*, **13**(4), 78–85, (1998).

[20] F. Hayes-Roth, D. Waterman, and D. Lenat, *Building Expert Systems*, Addison-Wesley, 1983.

[21] S. Hoppenbrouwers, P. Lucas, D. Romano, and D. Moffat, 'Attacking the knowledge acquisition bottleneck trough Games-For-Modelling', in *AISB Symposium*, pp. 81–86, (2009).

[22] L. Hotz, A. Felfernig, M. Stumptner, A. Ryabokon, C. Bagley, and K. Wolter, 'Configuration Knowledge Representation & Reasoning', in *Knowledge-based Configuration – From Research to Business Cases*, eds., A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, chapter 6, 59–96, Morgan Kaufmann Publishers, (2013).

[23] Y. Huang, H. Liu, W. Ng, W. Lu, B. Song, and X. Li, 'Automating knowledge acquisition for constraint-based product configuration', *Journal of Manufacturing Technology Management*, **19**(6), 744–754, (2008).

[24] L. Hvam, N. Mortensen, and J. Riis, *Product Customization*, Springer, 2007.

[25] A. Jameson, S. Baldes, and T. Kleinbauer, 'Two methods for enhancing mutual awareness in a group recommender system', in *International Working Conference on Advanced Visual Interfaces*, pp. 447–449, Gallipoli, Italy, (2004).

[26] I. John, J. Knodel, T. Lehner, and D. Muthig, 'A practical guide to product line scoping', in *Software Product Line Conference 2006 (SPLC2006)*, pp. 3–12, (2006).

[27] A. Mackworth, 'Consistency in Networks of Relations', *Artificial Intelligence*, **8**(1), 99–118, (1977).

[28] M. Mandl, A. Felfernig, E. Teppan, and M. Schubert, 'Consumer Decision Making in Knowledge-based Recommendation', *Journal of Intelligent Information Systems (JIIS)*, **37**(1), 1–22, (2010).

[29] J. Masthoff, 'Group Recommender Systems: Combining Individual Models', *Recommender Systems Handbook*, 677–702, (2011).

[30] C. Musat, A. Ghasemi, A., and B. Faltings, 'Sentiment Analysis Using a Novel Human Computation Game', in *3rd Workshop on the People's Web Meets NLP*, pp. 1–9, (2012).

[31] G. Ninaus, A. Felfernig, M. Stettinger, S. Reiterer, G. Leitner, L. Weninger, and W. Schanil, 'IntelliReq: Intelligent Techniques for Software Requirements Engineering', in *21st European Conference on Artificial Intelligence / Prestigious Applications of Intelligent Systems (PAIS 2014)*, p. to appear, Prague, Czech Republic, (2014).

[32] R. Reiter, 'A theory of diagnosis from first principles', *Artificial Intelligence*, **32**(1), 57–95, (1987).

[33] M. Richardson and P. Domingos, 'Building Large Knowledge Bases by Mass Collaboration', in *2nd Intl. Conference on Knowledge Capture (K-CAP03)*, pp. 129–137, (2003).

[34] T. Schrijvers, G. Tack, P. Wuille, H. Samulowitz, and P. Stuckey, 'Search combinators', *Constraint Journal*, **18**(2), 269–305, (2013).

[35] M. Schubert, A. Felfernig, and M. Mandl, 'FastXPlain: Conflict Detection for Constraint-Based Recommendation Problems', in *Trends in Applied Intelligent Systems (proc. of 23rd International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2010)*, eds., Nicolás García-Pedrajas, Francisco Herrera, Colin Fyfe, JoséManuel Benítez, and Moonis Ali, volume 6096 of *Lecture Notes in Computer Science*, 621–630, Springer, Cordoba, Spain, (2010).

[36] K. Siorpaes and M. Hepp, 'Games with a Purpose for the Semantic Web', *IEEE Intelligent Systems*, **23**(3), 50–60, (2008).

[37] M. Stumptner, 'An Overview of Knowledge-based Configuration', *AI Communications*, **10**(2), 111–126, (1997).

[38] E. Teppan and A. Felfernig, 'Asymmetric Dominance- and Compromise Effects in the Financial Services Domain', in *IEEE International Conference on E-Commerce and Enterprise Computing (CEC/EEE2009)*, pp. 57–64, Vienna. Austria, (2009).

[39] L. von Ahn, 'Human Computation', in *Technical Report CMU-CS-05-193*, Carnegie Mellon University, School of Computer Science, (2005).

[40] L. von Ahn, 'Games with a Purpose', *IEEE Computer*, **39**(6), 92–94, (2006).

[41] C. Wagner, 'Breaking the Knowledge Acquisition Bottleneck through Conversational Knowledge Management', *Information Resources Management Journal*, **19**(1), 70–83, (2006).

[42] B. Yang and N. D. Burns, 'Implications of postponement for the supply chain', *International Journal of Production Research*, **41**(9), 2075–2090, (2003).