Configuring Multiple Instances with Multi-Configuration

Alexander Felfernig[1] and **Andrei Popescu**[1] and **Mathias Uta**[2] and **Viet-Man Le**[1] and **Seda Polat-Erdeniz**[1] and **Martin Stettinger**[1] and **Müslüm Atas**[1] and **Thi Ngoc Trang Tran**[1]

Abstract. Configuration is a successful application area of Artificial Intelligence. In the majority of the cases, configuration systems focus on configuring one solution (configuration) that satisfies the preferences of a single user or a group of users. In this paper, we introduce a new configuration approach – multi-configuration – that focuses on scenarios where the outcome of a configuration process is a set of configurations. Example applications thereof are the configuration of personalized exams for individual students, the configuration of project teams, reviewer-to-paper assignment, and hotel room assignments including individualized city trips for tourist groups. For multi-configuration scenarios, we exemplify a constraint satisfaction problem representation in the context of configuring exams. The paper is concluded with a discussion of open issues for future work.

1 Introduction

Configuration is a special case of design activity where a product is composed of a *selection of predefined components* that satisfies a set of constraints [4, 11, 14]. Typical configurator applications are based on the assumption that a product (or service) is configured for a *single user*. Related example applications can be found in product domains such as telecommunications [6], automotive [8], and software systems [12]. In contrast to the configuration for an individual user, *group-based configuration* [2] is based on the idea of configuring a product or service for a group of users, i.e., the resulting configuration must take into account as much as possible the individual preferences of group members. An example of a group-based scenario is the configuration and also recommendation of software release plans where software requirements have to be arranged in such a way that the preferences of individual stakeholders are taken into account as much as possible [1, 3].

In this paper, we focus on a scenario where *multiple instances of the same product type* are configured for one user or a group of users. In this context, a set of constraints defines restrictions regarding the possible combinations of individual instances, i.e., *the configured instances are not completely independent*. An application scenario for multi-configuration is the configuration of user-individual exams which have to take into account a set of constraints, for example, *the share of complex questions per student must be below 20%*. Following this representation, we define *multi-configuration* as *a specific type of configuration task where multiple instances of the same type are configured for a single user or a group of users*.

There are various example scenarios which can profit from multi-configuration. (1) When assigning presentation topics to students, the goal could be to identify three different starting paper references per student where, for example, *each reference should be assigned to not more than two students*. (2) When planning room assignments for larger user groups (e.g., for a soccer training camp), a configurator could collect the preferences of individual group members and then generate a room assignment that takes into account as much as possible the preferences of the individual group members. (3) When configuring project teams, a configurator could take care of the optimal configuration of individual teams while keeping in mind the aspect of fairness, i.e., none of the team configurations should be "suboptimal". (4) When designing an apartment/house complex, neighborhood buildings must take into account rules such as *the roof type of all buildings should be the same* or *neighborhood buildings must not reduce the amount of direct sunlight below a specific treshold*. Compared to the former ones, the latter scenario can be regarded as a kind of industrial multi-configuration setting. Finally, the automated generation of test cases [7], for example, in the context of regression testing, can be regarded as a multi-configuration task.

2 Multi-Configuration

In single user and group-based configuration scenarios, one configuration is determined that supports the requirements of the user (or a group of users). In multi-configuration scenarios, a set of configurations is determined for one or a group of users. The following discussions are based on a constraint-based configuration knowledge representation [15]. The following definition of a configuration task (and solution) allows to represent collections of configurations (see Definition 1 and Definition 2). A *multi-configuration task* can be defined as a constraint satisfaction problem (CSP) [15].

Definition 1. A *Multi-Configuration Task* can be defined by a tuple (V, D, REQ, C) where $V = \bigcup\{v_{ij}\}$ is a set of finite domain variables (v_{ij} represents variable j of configuration instance i), $D = \bigcup\{dom(v_{ij})\}$ is a set of corresponding domain definitions, $REQ = \bigcup\{v_{ij} = val_{ij}\}$ represents a set of user requirements, and $C = \{c_1, c_2, .., c_m\}$ is a set of constraints that restrict the way in which individual variable values can be combined with each other.

For simplicity, we assume that user requirements in REQ are represented by simple variable value assignments ($v_{ij} = val_{ij}$). However, if needed, this assumption can be replaced by the more general case of allowing users to specify more complex constraints where each of those constraints is regarded as an individual requirement. Furthermore, we decided not to differentiate between different constraint types, for example, constraints referring to individual configurations and constraints referring to a set of configurations. On the basis of Definition 1, we can introduce the definition of a *multi-configuration* (see Definition 2).

Definition 2. A multi-configuration for a multi-configuration task (V, D, REQ, C) is a set of variable value assignments $CONF =$

[1] Applied Software Engineering & AI, Graz University of Technology, email: {firstname.lastname}@ist.tugraz.at
[2] Siemens Energy AG, email: mathias.uta@siemens.com

$\bigcup\{v_{ij} = val_{ij}\}$ s.t. $CONF \cup C \cup REQ$ is consistent.

We now introduce an example *multi-exam configuration* scenario.[3]

3 Example Multi-Configuration Scenario

In multi-exam configuration, the overall goal is to determine an exam instance for each examinee where a couple of constraints have to be taken into account. The underlying idea is to provide variability models for the purpose of being able to generate examinee-individual exams and thus save time in exam preparation and also avoid cheating due to exam diversity. Furthermore, this approach provides the possibility of a deeper integration of examinees into exam-related decision processes, i.e., students can be allowed to some extend to articulate their preferences regarding an exam. Following Definition 1, we now show how to represent the task of *multi-exam configuration*.

- $V = \{q_{11}..q_{kl}, q_{11}.type..q_{kl}.type, q_{11}.level..q_{kl}.level\}$ where q_{ij} is question j posed to examinee i, $q_{ij}.type$ denotes the question type (category), $q_{ij}.level$ is the question complexity, k the number of examinees, and l the number of questions / examinee.
- $D = \{dom(q_{11})..dom(q_{kl}), dom(q_{11}.type)..dom(q_{kl}.type), dom(q_{11}.level)..dom(q_{kl}.level)\}$, where $dom(q_{ij}) = \{1..p\}$, $dom(q_{ij}.type) = \{1..\overline{q}\}$, and $dom(q_{ij}.level) = \{1..r\}$ (p = number of questions per examinee, $\overline{q}$ = number of question categories, and r = number of question complexity levels).
- $REQ = \{r_1..r_u\}$ where r_α is a requirement identifier and u the number of requirements (defined by examinees and instructors).
- $C = \{c_1..c_v\}$ where v is the number of constraints.

In the following, we introduce a couple of *example constraints* that could be defined in the context of an exam configuration task.

Requirements. REQ includes a set of examinee- (and instructor-) individual constraints – in the context of our example, REQ is a set of examinee-specific constraints that have to be taken into account. The motivation behind this is that we want to make exam generation more flexible in terms of being able to include the preferences of examinees as well as the preferences of instructors.

First, we assume that examinee a prefers to have included $\leq 30\%$ of questions related to the categories $\{\alpha, \beta\}$ (see Formula 1). This way, we want to bring more flexibility into exam configuration, however, students will not always get what they want, i.e., their preferences have to be consistent with constraints defined by instructors. In many scenarios, students will not be allowed to define such constraints, i.e., only instructor requirements are relevant.

$$r_1 : \frac{|\{q_{aj} \in V : q_{aj}.type \in \{\alpha, \beta\}\}|}{|\{q_{aj} \in V\}|} \leq 0.3 \quad (1)$$

Furthermore, we assume that examinee a prefers not to have included questions related to question category γ (see Formula 2).

$$r_2 : |\{q_{aj} \in V : q_{aj}.type = \gamma\}| = 0 \quad (2)$$

Question-level Constraints. These constraints define properties related to the inclusion of specific questions. For example, we assume that a specific question v must not be included in more than x exams (see Formula 3), i.e., $x = 0$ would exclude v from any exam.

$$c_1 : |\{q_{ij} \in V : q_{ij} = v\}| \leq x \quad (3)$$

Furthermore, we assume that each exam configuration includes question u or question v (see Formula 4).

$$c_2 : \bigwedge_{i=1}^{k(\#examinees)} \bigvee_{j=1}^{l(\#questions)} (q_{ij} = u \vee q_{ij} = v) \quad (4)$$

We also assume that the minimum question complexity level is π.

$$c_3 : \forall q_{ij} \in V : q_{ij}.level \geq \pi \quad (5)$$

Global Constraints. We require that no question of type δ must be included in the exam (see Formula 6). A simple reason for formulating such constraints could be that the related topic has not been discussed in detail within the scope of the course.

$$c_4 : |\{q_{ij} \in V : q_{ij}.type = \delta\}| = 0 \quad (6)$$

We want to assure that at least three questions of type γ must be included in each exam u (see Formula 7).

$$c_5 : \forall u \in \{1..k\} : |\{q_{uj} \in V : q_{uj}.type = \gamma\}| \geq 3 \quad (7)$$

Furthermore, the overall estimated duration of each exam must be Δ minutes (constraint formulated for examinee a) (see Formula 8).

$$c_6 : \Sigma_{j=1}^{l(\#questions)}(q_{aj}.duration) = \Delta \quad (8)$$

We also want to assure that the share of complex questions ($level = \phi$) per examinee must be between 16% and 18% (example formulated for examinee a) (see Formula 9).

$$c_7 : 0.16 \leq \frac{|\{q_{aj} \in V : q_{aj}.level = \phi\}|}{|\{q_{aj} \in V\}|} \leq 0.18 \quad (9)$$

Further Constraints. In addition to the mentioned examples, there are many further relevant constraints, for example, *each examinee should be asked a specific question only once* and *the question overlap between each pair of students should be more than 90%*.

4 Conclusions and Future Work

We have introduced the notion of *multi-configuration* which is a specific approach focusing on scenarios where collections of configurations are designed for user groups. In order to better understand the discussed concepts, we have introduced a working example from the domain of *exam configuration*. Future work will include the analysis of the applicability of the presented concepts in the exam configuration domain (e.g., we will identify a complete set of typically relevant domain constraints) as well as in further multi-configuration scenarios. Furthermore, we will analyze new user interfaces and interaction requirements triggered by the application of multi-configuration concepts. The knowledge representation concepts discussed within the context of our exam configuration scenario are currently integrated into the KNOWLEDGECHECKR elearning environment (www.knowledgecheckr.com) [13]. Our major motivation is to increase the flexibility of exam generation but also to counteract cheating in online exams through an increased exam variability. In cases where individual user requirements induce an inconsistency with the exam model constraints, we propose the application of model-based diagnosis concepts [5, 9, 10] which can help to determine minimal conflict resolutions that also take into account aspects such as fairness and representativeness of the remaining questions.

ACKNOWLEDGEMENTS

The presented work has been conducted in the PARXCEL project funded by the Austrian Research Promotion Agency (880657).

[3] For exam configuration, we replace the term *user* with *examinee/instructor*.

REFERENCES

[1] A. Felfernig, 'AI Techniques for Software Requirements Prioritization', in *Artificial Intelligence Methods for Software Engineering*, eds., M. Kalech, R: Abreu, and M. Last, 29–47, World Scientific, (2021).

[2] A. Felfernig, M. Atas, T. Tran, and M. Stettinger, 'Towards group-based configuration', in *ConfWS'16*, pp. 69–72, Toulouse, France, (2016).

[3] A. Felfernig, L. Boratto, M. Stettinger, and M. Tkalcic, *Group Recommender Systems*, Springer, 2018.

[4] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, *Knowledge-based Configuration - From Research to Business Cases*, Elsevier, 2014.

[5] A. Felfernig, M. Schubert, and C. Zehentner, 'An Efficient Diagnosis Algorithm for Inconsistent Constraint Sets', *AI for Engineering Design, Analysis, and Manufacturing*, **26**(1), 53–62, (2012).

[6] G. Fleischanderl, G. Friedrich, A. Haselboeck, H. Schreiner, and M. Stumptner, 'Configuring large systems using generative constraint satisfaction', *IEEE Intelligent Systems*, **13**(4), 59–68, (1998).

[7] A. Gotlieb, B. Botella, and M. Rueher, 'Automatic Test Data Generation Using Constraint Solving Techniques', *ACM SIGSOFT Software Engineering Notes*, **23**(2), 53–62, (1998).

[8] J. Landahl, D. Bergsjö, and H. Johannesson, 'Future Alternatives for Automotive Configuration Management', *Procedia Computer Science*, **28**, 103–110, (2014).

[9] V.M. Le, A. Felfernig, M. Uta, D. Benavides, J. Galindo, and T.N.T. Tran, 'DIRECTDEBUG: Automated Testing and Debugging of Feature Models', in *43rd Intl. Conference on Software Engineering (ICSE'21)*, pp. 81–85, Virtual, (2021). IEEE.

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

[11] D. Sabin and R. Weigel, 'Product Configuration Frameworks - A Survey', *IEEE Intelligent Systems*, **13**(4), 42–49, (1998).

[12] J. Sincero and W. Schröder-Preikschat, 'The Linux Kernel Configurator as a Feature Modeling Tool', in *Workshop on Analyses of Software Product Lines*, pp. 257–260, Limerick, Ireland, (2008). ASPL.

[13] M. Stettinger, T.N.T. Tran, I. Pribik, G. Leitner, A. Felfernig, R. Samer, M. Atas, and M. Wundara, 'KNOWLEDGECHECKR: Intelligent Techniques for Counteracting Forgetting', in *24th European Conference on Artificial Intelligence (ECAI 2020)*, eds., G. DeGiacomo, A. Catala, B. Dilkina, M. Milano, S. Barro, A. Bugarin, and J. Lang, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pp. 3034–3039, Santiago de Compostela, Spain, (2020). IOS Press.

[14] M. Stumptner, 'An overview of knowledge-based configuration', *AI Communications*, **10**(2), 111–125, (1997).

[15] E. Tsang, *Foundations of Constraint Satisfaction*, Academic Press, 1993.