19th International Configuration Workshop

Proceedings of the 19th International Configuration Workshop

Edited by
Linda L. ZHANG and Albert HAAG

September 14 – 15, 2017
La Defense, France

Organized by

ISBN: 978-2-9516606-2-5

IESEG School of Management
Socle de la Grande Arche
1 Parvis de La Défense
92044 Paris La Défense cedex
France

Linda L. ZHANG and Albert HAAG, Editors
Proceedings of the 19th International Configuration Workshop
September 14 – 15, 2017, La Défense, France

Chairs

Linda L ZHANG, IESEG School of Management, Lille-Paris, France
Albert HAAG, Albert Haag – Product Management R&D, Germany

Program Committee

Michel ALDANONDO, Mines Albi, France
Tomas AXLING, Tacton Systems AB, Sweden
Andres BARCO, Universidad de San Buenaventura-Cali, Colombia
Andreas FALKNER, Siemens AG, Austria
Alexander FELFERNIG, Graz University of Technology, Austria
Cipriano FORZA, Universita di Padova, Italy
Gerhard FRIEDRICH, Alpen-Adria-Universitaet Klagenfurt, Austria
Albert HAAG, Albert Haag – Product Management R&D, Germany
Alois HASELBOECK, Siemens AG, Austria
Petri HELO, University of Vassa, Finland
Lothar HOTZ, HITeC e.V. / University of Hamburg, Germany
Lars HVAM, Technical University of Denmark, Denmark
Dietmar JANNACH, TU Dortmund, Germany
Thorsten KREBS, Encoway GmbH, Germany
Katrin KRISTJANSDOTTIR, Technical University of Denmark, Denmark
Yiliu LIU, Norwegian University of Science and Technology, Norway
Anna MYRODIA, Technical University of Denmark, Denmark
Brian RODRIGUES, Singapore Management University, Singapore
Sara SHAFIEE, Technical University of Denmark, Denmark
Alfred TAUDES, Vienna University of Economics & Business, Austria
Élise VAREILLES, Mines Albi, France
Yue WANG, Hang Seng Management College, Hong Kong
Linda ZHANG, IÉSEG School of Management, France

Organizational Support

Céline LE SUÜN, IÉSEG School of Management, France
Julie MEILOX, IÉSEG School of Management, France

Preface

As a special design activity, product configuration greatly helps the specification of customized products. It has been a fruitful domain for applying and developing advanced artificial intelligence (AI) techniques. Configuration tasks demand powerful knowledge-representation formalisms and acquisition methods to capture the great variety and complexity of configurable product models. In addition, efficient reasoning is required to provide intelligent interactive behavior in contexts such as solution search, satisfaction of user preferences, personalization, optimization, and diagnosis.

The main goal of the workshop is to promote high-quality research in all technical areas related to configuration. The workshop is of interest both for researchers working in the various fields of AI and product design as well as for industry representatives interested in the relationship between configuration technology and the business problem behind configuration and mass customization. It provides a forum for presentation of original methods and the exchange of ideas, evaluations, and experiences.

As such, this year's Configuration Workshop again aims at providing a stimulating environment for knowledge-exchange among academia and industry and thus building a solid basis for further developments in the field.

Furthermore, to encourage the continuous efforts, same as the past several workshops, the workshop this year sets a Best Student Paper Award (applicable to PhD candidates and bachelor and MSc students) and a Best Paper Award (applicable to all other participants than PhD candidates and bachelor and MSc students). The two papers are selected in a two-phase audience vote at the end of the workshop.

Linda L ZHANG AND Albert HAAG

Contents

Learning Constraint Satisfaction Heuristics for Configuration Problems

Giacomo Da Col and **Erich C. Teppan** [1]

Abstract. In this paper, we propose an approach for learning heuristics for constraint satisfaction problems in general and for configuration problems in particular. The genetic algorithm based learning approach automatically derives variable ordering, value ordering and pruning strategies for the exploitation by constraint solvers. We evaluate our approach with respect to the combined configuration problem, which is a generic configuration problem including sub problems such as graph coloring or bin packing. The results show that one of the best performing heuristics identified by our approach performs equally well compared to the expert heuristic defined in cooperation with our project partners from Siemens.

1 Introduction

Configuration problems [14, 18] are classical planning problems where elements have to be connected such that all user requirements are fulfilled and no technical constraints are violated. The configuration of products and services, or more generally the configuration of systems, is an important task and one of the major challenges in many production regimes such as mass customization, configure-to-order, or assembly-to-order [2, 19]. On the one hand, the basic goal is to provide customers with products and services that fulfill all their requirements. On the other hand, these products and services shall be offered at mass production efficiency. In order to fulfill these goals, systems are assembled by pre-designed and pre-fabricated components where such components themselves may be assembled by components.

A solution for a configuration problem is a system description (i.e. a configuration) that satisfies all requirements and contains all the information needed for manufacturing or service provision in an explicit, succinct, and simple to process format. To accomplish this, knowledge-based approaches are highly suitable and, among those, constraint-based approaches have a long and successful history [4, 12, 11].

Configuration problems are most commonly represented as finite discrete constraint satisfaction problems. A constraint satisfaction problem (CSP) is defined as a triple $\langle V, D = \{dom(v) : v \in V\}, C\rangle$ whereby V is a set of variables, D is the set of domains of the variables in V and C is a set of constraints over variables in V. A solution to a CSP is an assignment $v := d \in dom(v)$ for all $v \in V$ such that all constraints $c \in C$ are fulfilled. A CSP comprising only finite domains is called finite. If all domains consist of discrete values (commonly integers), the CSP is called discrete.

The size of the search space of finite discrete CSPs is $O(n^{|V|})$, where n is the maximum domain size. Without the usage of effective heuristics, solutions are out of reach for real-world sized problem instances. In order to effectively traverse the large search space, state-of-the-art constraint solvers offer a set of built-in problem-independent heuristics with which the solvers can be parametrized. Though, for hard real-world problems problem-dependent heuristics engineered by domain experts are typically needed (see for example [22]).

A particularity of configuration problems rooting in the dynamics and flexibility of nowadays production environments is that they often mutate over time. Thus, although not completely changing their nature, configuration problem variants come up from time to time and replace old problem variants. Reasons for that can be changing product portfolios, availability of newer technical components with different technical requirements and/or extended possibilities, change of the machinery in operation or legislation amendments.

A big defiance arising out of this is that already well-established heuristics are not applicable any more and heuristic (re-)engineering for adapting to a new problem variant is costly. Also, it is not sure if the already existing heuristics can be adapted or if a new heuristic must be designed from scratch. In order to cope with this issue, a learning approach for automatically creating effective problem heuristics for configuration problems is highly desirable.

In this paper, we propose a heuristic learning framework for creating problem-dependent heuristics for configuration problems expressed as CSPs. In particular, we describe how to represent the learning problem as a genetic algorithm. Evaluations are made with respect to the combined configuration problem (CCP), which constitutes a hard problem in the Answer Set Competition[2]. The CCP is a generic configuration problem including bin packing, vertex coloring, assignment and path sub problems.

2 Learning Approach

Our approach for learning CSP heuristics, with a special focus on configuration problems, operates on the notion of *variable groups*. A variable group is a set of variables that play the same role in the problem. This can be a single variable, in the simplest case, or an array of variables. Variable groups can be syntactically identified in the formal problem representation.

Example: Given is a set of processes and a set of hosts in a single-hop network. As the type of network is single-hop and the number of built-in network interfaces is predetermined, each host can only be connected to a limited number of other hosts. Furthermore, hosts can run only a limited number of processes in parallel. In order to fulfill their tasks, processes have to communicate with a predefined

[1] Alpen-Adria Universität Klagenfurt, Austria, email: giacomo.da@aau.at, erich.teppan@aau.at

[2] http://aspcomp2015.dibris.unige.it/

set of other processes running on the same or connected hosts. Having the information about which processes have to communicate to each other, the configuration problem consists in deciding for each processes on which host it runs and how hosts are connected to each other in order to ensure all inter-process communications.

Hence, besides some constraints, a CSP model (e.g. in minizinc[3]) would incorporate two variable groups. One for capturing which process runs on which host ($onHost$) and a second group of Boolean variables for expressing which hosts are connected to each other, i.e.:

```
array[1..p] of var 1..h: onHost;
array[1..h,1..h] of var 0..1: conn;
```

Clearly, the number of variables in a variable group (i.e. the number of processes p and the number of hosts h in the above example) is varying in different problem instances. However, the set of variable groups remains the same for every instance. Hence, heuristics operating on variable groups rather than on individual variables are instance-independent, i.e. they can be applied on any instance of a particular problem. This leads us to the notion of *variable group heuristics*:

Definition 1 *Given a CSP* $= \langle V, D, C \rangle$ *and its set of variable groups* $G_P = \{g : g \subseteq V\}$ *with* $\bigcup G_P = V$ *and for all* $g_i, g_j \in G_P : (i \neq j) \rightarrow (g_i \cap g_j = \emptyset)$*, a* ***variable group heuristic*** *specifies:*

- *a basic search algorithm,*
- *a total order of* G_P*,*
- *for each* $g \in G_P$ *a value ordering strategy, and*
- *for each* $g \in G_P$ *a search limit.*

The basic search algorithm specifies the search regime used by the underlying constraint solver. State-of-the-art constraint solvers support a set of different search algorithms such as chronological or non-chronological backtracking search [15], local search methods [7] or clause-learning approaches [16].

The total order of G_P defines a partial ordering of constraint variables, which is the order in which constraint variables should be processed by the constraint solver.

The value ordering strategy for a group $g \in G_P$ specifies for each $v \in g$ in which order domain values are tried. Classic value ordering strategies are ascending, descending or random order.

The search limit for a group $g \in G_P$ specifies for each $v \in g$ when the branch of the search tree below v is cut off instead of being searched.

2.1 Prototype

For learning configuration problem heuristics we have built a genetic algorithm [13] on top of the Jacop[4] constraint solving framework. Conforming to Definition 1, in our prototype a heuristic is represented by four chromosomes (see Figure 1): a single gene chromosome for the basic search algorithm, and three chromosomes with $|G_P|$ many genes each for defining the order number, value ordering strategy and search limit of each group.

As a search algorithm our prototype allows *beam search* or *limited discrepancy search*, which already showed to be effective for large and/or complex problems [5, 8, 9, 10]. Whereas beam search limits the number of guesses for a variable (beam width), limited discrepancy search limits the number of faulty variable guesses along the search path. Depending on the search algorithm, beam width or maximum discrepancies are defined for each variable group by means of the search limit. A search limit is expressed as 2^x with x being a number between 0 and 10. Hence, possible search limits are $1, 2, 4, \ldots, 512, 1024$. This is to avoid using too many possible values in the genetic algorithm whilst still offering a broad choice of parameters. Note that by using high search limits search becomes complete because of search limits are not reached anymore.

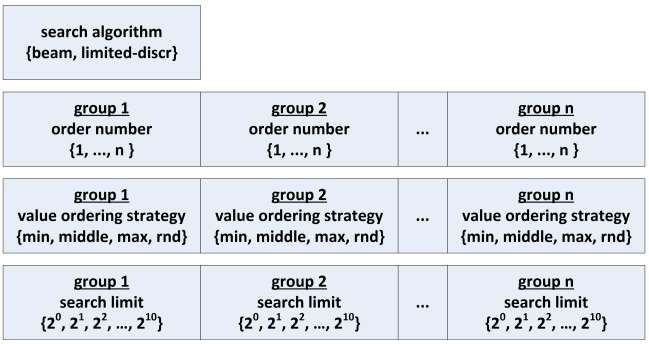


Figure 1. Chromosomes and genes in the genetic algorithm

For the value selection strategies of the variable groups we allow all four strategies already built-in the Jacop constraint solver. This is: minimum value first, maximum value first, the value in the middle of a domain first, and random.

In our prototype, we use the basic genetic algorithm provided by the Jenes library[5]. The fitness function calculates the fitness value $f(h)$ of a heuristic h as the sum of performances measured on a set of test instances, i.e. $\sum_{i=1}^{numInstances} performance(h, i)$. Hereby, $performance(h, i) = 0$ if i could be solved before a timeout occurred. If a timeout occurred, the performance is calculated as the number of wrong decisions measured divided by the number of total decisions made on i, i.e. $performance(h, i) = \frac{wrong(h,i)}{total(h,i)}$. Hence, the best achievable fitness value $f(h)$ is 0, which means that all test instances could solved by applying heuristic h within time restrictions. The worst possible fitness value is equal to the number of instances in the test set, which is produced when every single decision for every instance was wrong.

The learning procedure operating on top of the fitness function is as follows:

1. A first $population$ of heuristics is randomly generated, the fitness value $f(h)$ of every heuristic $h \in population$ is calculated, and the heuristics are ordered with respect to that value.
2. Then, steps (a)-(c) are repeated until a complete new population has been generated:
   (a) Select a pair of parent heuristics from the population. The probability of a heuristic h being selected is $\frac{f(h)}{\sum_{i=1}^{|population|} f(i)}$.
   (b) Two children heuristics are created by performing a single-point crossover on the selected parent heuristics with a predefined probability p_c. If the crossover does not take place, the children are exact copies of the respective parents.

[3] http://www.minizinc.org/
[4] http://jacop.osolpro.com/
[5] http://jenes.intelligentia.it/

(c) Perform a random mutation on the new children with probability p_m. If a mutation is performed, either only a single gene is changed (probability = 0.5) or one gene in each chromosome is changed randomly.

3. The old population is replaced by the new one.
4. If the predefined maximum number of generations is reached the learning procedure terminates, if not, it starts over again with step (2).

By treating the *order number chromosome* as a *permutation chromosome*, the Jenes framework ensures that each group has a unique order number. This is accomplished by using a special *permutation-crossover* operator that restores uniqueness of the order numbers after crossover and a special *permutation-mutation* operator that swaps the values of two genes instead of changing a single gene [17].

3 Evaluation

The problem we address in our evaluation is the combined configuration problem (CCP)[6]. This problem is a good candidate for the evaluation because it is composed of various sub problems often occurring in different configuration problems. It was presented for the first time in the ASP competition 2015[1] and originates in real-world domains like railway safety [6]. The CCP can be defined as follows:

Definition 2 *A CCP instance is an eight-tuple* $\langle G, H, P_1, P_2, w, x, y, z\rangle$ *that consists of:*

- *the bipartite graph* $G = (A, B, E)$*,*
- *the directed acyclic graph* $H = (V \cup B, F)$ *whereby each vertex in* $V \cup B$ *has a positive size* $\in \mathbb{N}$*,*
- *two disjoint paths* P_1 *and* P_2 *in* H*, and*
- *four constants* $w, x, y, z \in \mathbb{N}$*.*

A solution for a CCP instance is a triple $\langle \alpha, \beta, \gamma\rangle$ *with:*

1. *A coloring function* $\alpha : V \cup B \to \{1, \ldots, w\}$ *that assigns to each vertex in* $V \cup B$ *one color from* $\{1, \ldots, w\}$*, such that:*
   - *vertices from different paths must have different colors, and*
   - *for all vertices* v_1 *and* v_2 *with the same color there must exist a path from* v_1 *to* v_2 *(or vice-versa) such that each vertex on the path has the same color.*
2. *A packing function* $\beta : V \cup B \to \{1, \ldots, w * y\}$ *that assigns to every vertex in* $V \cup B$ *a bin in* $\{1, \ldots, w * y\}$*, whereby:*
   - *there are exactly* y *bins for each of* w *possible colors,*
   - *a bin can only contain vertices of the same color, and*
   - *for each bin the total sum of vertex sizes does not exceed the bin size* z*.*
3. *An assignment function* $\gamma : B \to A$ *that assigns to every border element* $b \in B$ *an area* $a \in A$*, such that:*
   - $(a, b) \in E$*,*
   - *no area has assigned more than* z *border elements, and*
   - *all border elements of an area must have the same color.*

[6] The benchmark and the source-code can be found at https://goo.gl/F5ZGpK

To represent the CCP as a CSP, we have two variables for each vertex in $V \cup B$, for capturing its color and its bin respectively. For each vertex in B there is an additional variable for area assignment. These variables naturally form three variable groups.

In cooperation with domain experts from Siemens, an effective heuristic for the CCP was developed by hand. In the following we refer to this solving strategy as the *hand-crafted* heuristic. In the style of Definition 1, the hand-crafted heuristic can be outlined as:

- Limited discrepancy search as search algorithm.
- Variable groups are ordered as follows: First, the area assignment variables are to be processed, followed by the coloring variables, and in the end the bin packing variables are to be processed.
- The value choice is random for coloring and assignment variables. For bin packing variables the minimum value is chosen first.
- The search limit for all the variable groups is 6.

The benchmark for the evaluation consists of 23 smallest instances taken from the ASP competition 2015. The instances are ordered by number of vertices. Typically, an increased number of vertices, which typically reflects the hardness of the instance [6].

The benchmark contains two types of instances. The first type is referred to as *real*, since the directed acyclic graph is based on real-world railway track layouts. The second type is referred to as *grid*, since the directed acyclic graph follows a grid pattern.

We divided the benchmark instances into one learning set and two test sets. The learning set consists of the 11 smallest *real* instances (roughly 50% of all instances).

Our prototype discussed in the last section was run with an initial population of 10 individuals and a generation limit of 15. The probabilities for crossover and mutation were $p_c = 0.8$ and $p_m = 0.4$ respectively.The maximum allowed solving time for the fitness evaluation was 10 seconds.

In the end of the learning phase, 8 out of 10 heuristics were able to solve all the learning instances. With respect to solving time, we tested the best learned heuristics on two test sets against the hand-crafted heuristic and the famous most-constrained-variable heuristic. The maximum allowed solving time before a timeout (t/o) occurred was set to 100 seconds. The first test set consists of the 5 remaining *real* instances. The second test set consists of 7 *grid* instances.

The results are summarized by Tables 1 and Table 2. Table 1 shows the results on the *real* instances. With the most-constrained-variable heuristic it is not possible to solve any of the instances. The hand-crafted heuristic shows good performance, being able to solve all the instances with a maximum runtime of 28 seconds. Half of the learned heuristics, namely *learned 0, 3, 6* and *7* also lead to a solution for all instances. Furthermore, the solving time is comparable with the hand-crafted approach. The other learned heuristics did not perform well as they produced at least one timeout.

Table 2 shows the results on the second test set. In this case the most-constrained-variable heuristic leads to a solution for some of the instances, but loses effectiveness as the number of vertices grows. On the contrary, the hand-crafted heuristic again performs very well such that also all grid instances can be solved within time limits. Among the learned heuristics, *learned 1, 2, 3* and *4* perform well in that all instances can be solved. Moreover, the needed solving time is lower compared to the hand-crafted heuristic.

Taking the performance of both test sets into account, *learned 3* heuristic performed best. Hence, our approach produced an heuristic that performs similarly well as the hand-crafted heuristic and is even slightly better the second test set. The problem-independent most-constrained-variable heuristic performed overall worst.

	r80	r81	r82	r98	r102
most-constr	t/o	t/o	t/o	t/o	t/o
hand-crafted	12	15	13	24	28
learned 0	13	13	11	21	27
learned 1	13	12	t/o	23	26
learned 2	12	13	t/o	22	26
learned 3	12	13	13	22	29
learned 4	13	13	t/o	21	27
learned 5	t/o	13	49	58	29
learned 6	13	12	14	25	36
learned 7	14	12	12	27	31

Table 1. Solving time (secs) on the *real* instances

	g16	g25	g36	g49	g64	g81	g100
most-constr	0	8	2	t/o	t/o	t/o	t/o
hand-crafted	0	0	31	32	36	40	50
learned 0	0	0	1	t/o	t/o	t/o	28
learned 1	0	0	2	3	7	15	29
learned 2	0	0	1	3	7	14	29
learned 3	0	1	2	4	7	14	27
learned 4	0	0	2	3	7	14	31
learned 5	t/o	0	1	3	6	t/o	t/o
learned 6	0	1	t/o	t/o	7	t/o	56
learned 7	0	0	1	3	t/o	14	30

Table 2. Solving time (secs) on the *grid* instances

Analysis of the *learned 3* heuristic revealed some interesting insights. Technically, beam search was used but as the search limit was set to 2^9 actually a complete search was performed. The variable groups were differently ordered compared to the hand-crafted heuristic: First, the bin packing variables are processed with a random value selection strategy, then the area assignment variables are processed with a random value selection strategy, and coloring variables are processed only as the last ones using the value in the middle of a variable domain as the first. Please note that both heuristics, *learned 3* and hand-crafted, use a random value strategy for the first two variable groups.

4 Conclusions

In this paper we presented a genetic algorithm for learning constraint satisfaction heuristics for configuration problems. First evaluations performed on benchmark instances of the generic *combined configuration problem* are highly promising. Our learning approach was able to come up with heuristics that are capable to outperform the famous most-constrained-variable heuristic and can at least keep up with a hand-crafted expert heuristic created in cooperation with project partners from Siemens.

Imposed future works include a more fine-grained grouping of variables (that does not base just on the formal problem representation of the CSP), and the application of the approach to different types of problems like job-shop scheduling (e.g.: [1]). Furthermore, instead of learning heuristics for combinatorial problems, the presented approach might be adapted for the automatic calculation of utilities in knowledge-based recommender systems (e.g.: [3, 21, 20]).

REFERENCES

[1] Giacomo Da Col and Erich C. Teppan, *Declarative Decomposition and Dispatching for Large-Scale Job-Shop Scheduling*, 134–140, Springer International Publishing, Cham, 2016.

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

[3] A. Felfernig, M. Mairitsch, M. Mandl, M. Schubert, and E. Teppan, 'Utility-based repair of inconsistent requirements', in *Proceedings of the 22nd Int. Conf. on Industrial, Engineering and Other Applications of Applied Intelligent Systems: Next-Generation Applied Intelligence*, IEA/AIE '09, pp. 162–171, Berlin, Heidelberg, (2009). Springer-Verlag.

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

[5] David Furcy and Sven Koenig, 'Limited discrepancy beam search', in *IJCAI*, pp. 125–131, (2005).

[6] Martin Gebser, Anna Ryabokon, and Gottfried Schenner, *Combining Heuristics for Configuration Problems Using Answer Set Programming*, 384–397, Springer International Publishing, Cham, 2015.

[7] Michel Gendreau and Jean-Yves Potvin, *Handbook of Metaheuristics*, Springer US, 2010.

[8] M. L. Ginsberg, W. D. Harvey, J. M. Crawford, A. K. Jonsson, and J. C. Pemberton. System and process for job scheduling using limited discrepancy search, May 30 2000. US Patent 6,070,144.

[9] William D. Harvey and Matthew L. Ginsberg, 'Limited discrepancy search', in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, pp. 607–613, San Francisco, CA, USA, (1995). Morgan Kaufmann Publishers Inc.

[10] Abir Ben Hmida, Mohamed Haouari, Marie-José Huguet, and Pierre Lopez, 'Discrepancy search for the flexible job shop scheduling problem', *Computers & Operations Research*, **37**(12), 2192–2201, (2010).

[11] Dietmar Jannach and Markus Zanker, 'Modeling and solving distributed configuration problems: A csp-based approach', *IEEE Trans. on Knowl. and Data Eng.*, **25**(3), 603–618, (March 2013).

[12] Daniel Mailharro, 'A classification and constraint-based framework for configuration', *Artif. Intell. Eng. Des. Anal. Manuf.*, **12**(4), 383–397, (September 1998).

[13] Melanie Mitchell, *An introduction to genetic algorithms*, MIT Press, 1996.

[14] Sanjay Mittal and Felix Frayman, 'Towards a generic model of configuraton tasks', in *11th International Joint Conference on AI - Vol. 2*, IJCAI'89, pp. 1395–1401, San Francisco, CA, USA, (1989). Morgan Kaufmann Publishers Inc.

[15] Francesca Rossi, Peter van Beek, and Toby Walsh, *Handbook of Constraint Programming, 1st Edition*, Elsevier Science, 2006.

[16] J. P. Marques Silva and K. A. Sakallah, 'Conflict analysis in search algorithms for satisfiability', in *Proceedings Eighth IEEE International Conference on Tools with Artificial Intelligence*, pp. 467–469, (Nov 1996).

[17] G. Singh, N. Gupta, and M. Khosravy, 'New crossover operators for real coded genetic algorithm (rcga)', in *2015 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, pp. 135–140, (Nov 2015).

[18] Carsten Sinz, Albert Haag, Nina Narodytska, Toby Walsh, Esther Gelle, Mihaela Sabin, Ulrich Junker, Barry O'Sullivan, Rick Rabiser, Deepak Dhungana, Paul Grunbacher, Klaus Lehner, Christian Federspiel, and Daniel Naus, 'Configuration', *IEEE Intelligent Systems*, **22**(1), 78–90, (January 2007).

[19] Markus Stumptner, 'An overview of knowledgebased configuration', *AI Commun.*, **10**, 111–125, (April 1997).

[20] Erich C. Teppan and Alexander Felfernig, 'Calculating decoy items in utility-based recommendation', in *Proceedings of the 22nd Int. Conf. on Industrial, Engineering and Other Applications of Applied Intelligent Systems: Next-Generation Applied Intelligence*, IEA/AIE '09, pp. 183–192, Berlin, Heidelberg, (2009). Springer-Verlag.

[21] Erich Christian Teppan and Alexander Felfernig, 'Impacts of decoy elements on result set evaluations in knowledge-based recommendation', *Int. J. Adv. Intell. Paradigms*, **1**(3), 358–373, (June 2009).

[22] Erich Christian Teppan, Gerhard Friedrich, and Andreas A. Falkner, 'Quickpup: A heuristic backtracking algorithm for the partner units configuration problem', in *IAAI*, (2012).

Techniques for Solving Large-Scale Product Configuration Problems with ASP

Gottfried Schenner[1] and Richard Taupe[1,2]

Abstract. Answer Set Programming (ASP) is a well-established paradigm for encoding and solving product configuration problems. Unfortunately, generic problem encodings not tuned to specific problem instances do not scale well, i.e. they cannot be used for large-scale problem instances. One reason for this is that classical ASP solvers suffer from the so-called *grounding bottleneck*, i.e. they cannot solve problems whose propositional grounding exceeds given memory limits. In this paper we discuss some pragmatic techniques like lazy grounding, custom encodings, incremental solving, and problem decomposition for making ASP applicable to large-scale product configuration.

1 Introduction

Large-scale product configuration problems frequently arise in industrial settings. These problems typically have a lot of component types (> 100) and instances ($> 1{,}000$), where the exact number of components contained in a solution typically is not known beforehand [6]. Even if constraints in these domains may be simple, size and dynamic nature of such problems make them hard to handle in propositional solving paradigms like answer set programming. In this paper we describe various pragmatic techniques to make answer set programming applicable for large-scale configuration problems. By starting with a generic encoding of a typical industrial hardware configuration example we illustrate the grounding problem and show how to rewrite the encoding based on different configuration scenarios in order to increase the applicability of classical ASP solvers. We then investigate if using a lazy-grounding ASP solver can eliminate the grounding bottleneck for such problems.

2 ASP in a Nutshell

Answer Set Programming (ASP) [1, 11] is a declarative programming paradigm. Instead of stating how to solve a problem, an ASP program is a specification of the problem. An ASP solver then finds a solution to the problem based on the problem specification. Unfortunately writing a correct ASP encoding for a problem is not enough. The grounder must be able to ground the problem and the solver must be able to efficiently find a solution. Performance of both tasks greatly depends on the encoding of the problem. An expert ASP programmer can often find a superior encoding for a problem although the problem definition stays the same.

[1] Siemens AG Österreich, Corporate Technology, Vienna, Austria
firstname.lastname@siemens.com
[2] Alpen-Adria-Universität, Klagenfurt, Austria

2.1 Syntax

An answer-set program P is a finite set of rules of the form

$$h_1; \ldots; h_d \leftarrow b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_n.$$

where $h_1, \ldots, h_d$ and $b_1, \ldots, b_m$ are positive literals (i.e. atoms) and $not\ b_{m+1}, \ldots, not\ b_n$ are negative literals. An atom is either a classical atom or a cardinality atom. A classical atom is an expression $p(t_1, \ldots, t_n)$ where p is an n-ary predicate and $t_1, \ldots, t_n$ are terms. A literal is either an atom α or its default negation $not\ \alpha$. Default negation refers to the absence of information, i.e. an atom is assumed to be false as long as it is not proven to be true. A *cardinality atom* is of the form

$$l\ \{a_1 : l_{1_1}, \ldots, l_{1_m}; \ldots; a_n : l_{n_1}, \ldots, l_{n_o}\}\ u$$

where

- $a_i : l_{i_1}, \ldots, l_{i_m}$ represent *conditional literals* in which a_i (the head of the conditional literal) and all l_{i_j} are classical literals, and
- l and u are terms representing non-negative integers indicating lower and upper bound. If one or both of the bounds are not given, their defaults are used, which are 0 for l and ∞ for u.

$H(r) = \{h_1, \ldots, h_d\}$ is called the *head* of the rule, and $B(r) = \{b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_n\}$ is called the body of the rule. A rule r with $H(r)$ consisting of a cardinality atom is called *choice rule*. A rule r with a head consisting of more than one classical atom (i.e. $|H(r)| > 1$) is called *disjunctive rule*. A rule r with $H(r)$ consisting of at most one classical atom is called a *normal rule*. A normal rule r where $H(r) = \{\}$, e.g. $\leftarrow b.$, is called *integrity constraint*, or simply *constraint*. A normal rule r where $B(r) = \{\}$, e.g. $h \leftarrow .$, is called *fact*.

We allow the typically built-in arithmetic functions ($+$, $-$, $*$, $/$), comparison predicates ($=$,$\neq$,$<$,$>$,$\leq$,$\geq$), and syntactic sugar like $m..n$ which stands for the set of integers $\{m, \ldots, n\}$. Answer set programs in this paper will also contain comments starting with `%`.

2.2 Semantics

There are several ways to define the semantics of an answer-set program, i.e. to define the set of answer sets $AS(P)$ of an answer-set program P. An overview is provided by [12]. Probably the most popular semantics is based on the *Gelfond-Lifschitz reduct* [10]. A variant that applies to choice rules also is presented in [2].

Informally, an answer set A of a program P is a subset-minimal model of P (i.e. a set of atoms interpreted as *true*) which satisfies the following conditions: All rules in P are satisfied by A; and all atoms in A are "derivable" by rules in P. A rule is satisfied if its head is satisfied or its body is not. The disjunctive head of a rule is satisfied if at least one of its atoms is. A cardinality atom is satisfied if $l \leq |C| \leq u$ holds, where C is the set of head atoms in the cardinality literal whose conditions (e.g. $l_{i_1}, \ldots, l_{i_m}$ for a_i) are satisfied and which are satisfied themselves. In the presence of choice rules, the semantics is adjusted to allow non-minimal subsets satisfying the cardinality atom to appear in answer sets.

2.3 Grounding and solving

Most ASP systems split the solving process into grounding and solving. The former part produces the grounding of a program, i.e. its variable-free equivalent. Thereby, the variables in each rule of the program are substituted by constants. The latter part then solves this propositional encoding. This leads to the so-called *grounding bottleneck* which is tackled by lazy grounding [18].

2.4 ASP examples

For better understanding, this section gives some ASP examples. The following program contains a fact and a rule:

```
type(e1,elementA).
element(E) :- type(E,elementA).
```

The one and only answer set to this program is $\{type(e1, elementA), element(e1)\}$. If you extend the program by $type(e1, elementB)$ and a constraint that forbids multiple types for a single element, the program becomes unsatisfiable:

```
type(e1,elementB).
:- type(ID,T1), type(ID,T2), T1 != T2.
```

To illustrate the use of cardinality atoms, let us view an example of a choice rule:

```
0 { type(ID,T):t(T) } 1 :- n(ID).
```

This rule means that for every true atom of the unary predicate n the cardinality atom in the head of the rule must hold, which in turn forces the number of types assigned to the object represented by the former atom to be between 0 and 1. Types are defined by the t predicate and assigned to objects by the *type* predicate. Let P be a program consisting of the choice rule above and the following two facts encoding a problem instance:

```
n(1).
t(elementA).
```

Then P has two answer sets, namely $\{n(1), t(elementA)\}$ and $\{n(1), t(elementA), type(1, elementA)\}$, one representing the solution that object 1 has no type and one representing the solution that object 1 is of type elementA.

3 Running Example

As running example, we use a typical hardware configuration problem. Such problems are often very easy to solve for small instances sizes, but unfortunately most generic encodings will not scale.

We consider an encoding generic for a domain if it is valid for all problem instances of the domain. A non-generic encoding is specific to a subset of all problem instances and only works for specific input data or parameters. Therefore a generic encoding for a domain like our running example will solve all configuration tasks of that domain regardless of the input (e.g. a partial configuration).

3.1 Example domain: racks configuration

In our example domain there may be different types of elements, which are controlled by hardware modules. Each hardware module must be in a frame and a frame must be mounted on a rack. More specifically, the constraints of the domain are:

- There are 4 different types of elements (A-D).
- There are 5 different types of modules (I-V).
- There are 2 different types of racks (Single, Double).
- An ElementA requires one ModuleI, an ElementB requires two ModuleIIs, an ElementC requires three ModuleIIIs and an ElementD requires four ModuleIVs.
- A ModuleV cannot have an element, all other modules must have an element assigned.
- A RackSingle must have exactly 4 Frames, a RackDouble must have exactly 8 Frames.
- A Frame must be mounted on one Rack.

3.2 Example encoding

To encode this example in ASP we represent the objects/components of a configuration with the predicate `type(ID,TYPE)`, e.g. `type(7,rackSingle)` represents a rack with ID 7. Objects with different IDs represent different real-world objects, i.e. we use the unique name assumption.

The relations between the different component types are represented by binary predicates of the form *TYPE1_to_TYPE2*. For example, the relation between racks and frames is represented by the predicate $rack\_to\_frame(RACKID, FRAMEID)$.

The complete schema of the domain looks like this:

```
type(ID,TYPE)
% ID = 1..MAXNROFOBJECTS
% TYPE = elementA, elementB...
element_to_module(ELEMENTID,MODULEID)
rack_to_frame(RACKID,FRAMEID)
frame_to_module(FRAMEID,MODULEID)
```

These predicates suffice to represent a configuration of our simple domain. MAXNROFOBJECTS defines the maximum number of components in a configuration. This constant must

be defined at the beginning of the grounding process. The following represents a possible configuration with MAXNROFOBJECTS=10. Note that the used object IDs are totally arbitrary and some object IDs are unused.

```
type(1,elementA).
type(2,moduleI).
type(3,rackSingle).
type(4..7,frame).
element_to_module(1,2).
rack_to_frame(3,4).
rack_to_frame(3,5).
rack_to_frame(3,6).
rack_to_frame(3,7).
frame_to_module(4,2).
```

4 Reasoning Tasks

ASP programs often have a guess-and-check (generate-and-test) structure. There is a program part to guess all possible solutions and another part to check if a candidate solution is valid [4].

4.1 Checking a configuration

The checking part of the ASP program defines the conditions a valid configuration must satisfy and is expressed mostly with ASP constraints, i.e. rules that filter out invalid answer sets. The following shows some examples:

```
% only one type
:- type(ID,T1), type(ID,T2), T1 != T2.
% elementA can only have modules of type moduleI
:- element_to_module(E1,M1),
   type(E1,elementA), not type(M1,moduleI).
```

If the body of a constraint is satisfied, it will make the entire program unsatisfiable. Therefore, the body of no constraint can be satisfied by an answer set. In the program above, there are two constraints, the first of which forbids assigning two different types (denoted by `T1` and `T2`) to an object (denoted by `ID`) and the second of which forbids assigning modules that are not of type ModuleI to an ElementA.

The checking part does not generate any new components. For checking a configuration the input for the ASP solver consists of the encoding of the configuration (as facts) and the checking program. If the configuration is valid, an answer set is found; otherwise *unsatisfiable* is returned. The size of the grounding of this reasoning task depends on the size of the given configuration, i.e. the number of atoms describing the input configuration.

4.2 Solving a configuration

The checking part of our program alone allows us to check existing configurations. In order to find new configurations or to extend partial configurations the guessing part of the program is needed.

One approach for the guessing part is to write a generic program that can generate all possible configurations. Typically such a program consists of a part that instantiates all objects up to the maximal number of objects and a part that generates all possible associations between these objects.

The following code fragment shows an example[3]:

```
n(1..MAXNROFOBJECTS).           % defines objects
t(elementA). t(elementB). ...   % defines types
% instantiate components
0 { type(ID,T):t(T) } 1 :- n(ID).
% element_to_module
% guess an element for every module (except moduleV).
1 { element_to_module(E,M):element(E) } 1 :-
  module(M), not type(M, moduleV).
%...
```

From a purely declarative standpoint this is all that is needed. The checking part together with the guessing part will specify all valid configurations, which can then be enumerated by a solver. If one wants to find a configuration with certain properties, these properties can be given as additional constraints. A special case are partial configurations. In this case one can give the atoms describing the partial configuration as facts, e.g. `type(1,elementA)` will find all configurations that contain at least one ElementA.

Unfortunately such generic encodings can not be used for large-scale problems due to an explosion of grounding sizes. Additional constraints make the situation worse. If for example we add a constraint that all modules of an element must be in the same frame the grounding needs already more than 10 GB of memory just for 50 components.

Therefore in the following chapter we will discuss some pragmatic techniques to tackle this problem.

5 Strategies to Reduce Grounding

Typically in hardware configuration problems we want to find a configuration that uses a minimal number of components for a given input, i.e. a partial configuration. It does not make any sense to set the maximum number of objects to 1,000 if we want to find a configuration for 9 ElementBs. Therefore if we can find a lower bound for the size of a complete configuration for a given partial configuration the grounding size can be reduced significantly.

Finding the lower bound involves reasoning about component cardinalities. For a domain expert it is relatively easy to see that a minimal configuration for 9 ElementBs contains 18 ModuleIIs, 5 ModuleVs, 1 RackDouble and 8 Frames. The cardinality reasoning required is far from trivial. For instance, due to the constraint that there must be a ModuleV for every ModuleII in a frame one can no longer mount 6 ModuleIIs on a frame. Consequently one needs at least 5 Frames to support all Modules and therefore the solution must contain a RackDouble. This example shows that every additional constraint might affect the number of required components. To automatically derive these bounds, approaches like the one described in [14] can be used, although so far none of these approaches

[3] Note that this encoding uses choice rules to encode the nondeterministic guessing part. Other language constructs could be used instead, e.g. disjunction (cf. [15]). It also uses the unary `element` predicate as a shortcut to all objects whose type is any of the element types. This is achieved by projection rules `element(E) :- type(E,elementA).` etc.

are powerful enough to calculate the effects of arbitrary constraints or can directly be applied to ASP programs.

Another technique for reducing the grounding is to generate the guessing part of the program specifically to the current input. For instance, if all the components of a component type are given, there is no need to generate components of that type in the guessing part. An extreme example is an already completed configuration, where no guessing is required at all.

Unfortunately, depending on the encoding some ASP grounders are not able to detect that situation and still maintain the guessing rules in the program. In these cases it might be necessary to write a domain-specific preprocessing step that removes all guessing rules for components already used.

The following two encodings demonstrate such effects. From the encoding with default negation, rules with default negation are removed by the grounder[4] and only the fact `type(1,rackSingle)` remains.

```
type(1,rackSingle) :- not type(1,rackDouble).
type(1,rackDouble) :- not type(1,rackSingle).
type(1,rackSingle).
```

From the encoding with a choice rule, however, the choice rule is kept by the grounder and therefore all rules depending on `type(1,rackDouble)` are instantiated, which results in a much bigger grounding.

```
1 { type(1,rackSingle);type(1,rackDouble) } 1.
type(1,rackSingle).
```

To automatically generate the guessing part, one can simply filter out the guessing rule for component IDs that are already given as facts in the input program.

5.1 Decompositon

Decomposition of the problem into sub-problems is another way to reduce grounding. One way to decompose a configuration problem is to consider only a subset of the component types and relations in one step. This will create a partial configuration that serves as input for the next subproblem. At the end of the process a complete configuration will be created. The only condition for this approach is that the partial configurations created in each step can be extended to complete configurations. Otherwise backtracking, i.e., recomputing a subproblem is required. This is illustrated in Fig. 1. In our example, instead of solving the configuration problem at once, it can be decomposed into the following sub-problems: First assign elements to modules, then assign modules to frames, then assign frames to racks.

Instead of calling the ASP solver once, now in the best case it is called once for every subproblem. The first subproblem consists of assigning elements to modules. The found answer set (i.e. partial configuration) then serves as input for assigning modules to frames. The resulting partial configuration is then used for assigning frames to racks, which leads to a complete configuration. Problem decomposition can reduce the grounding size significantly.

[4] The first rule can be removed because its head is already satisfied by a fact and the rule thus cannot contribute to a solution, while the second rule can be removed because its body is not satisfied.

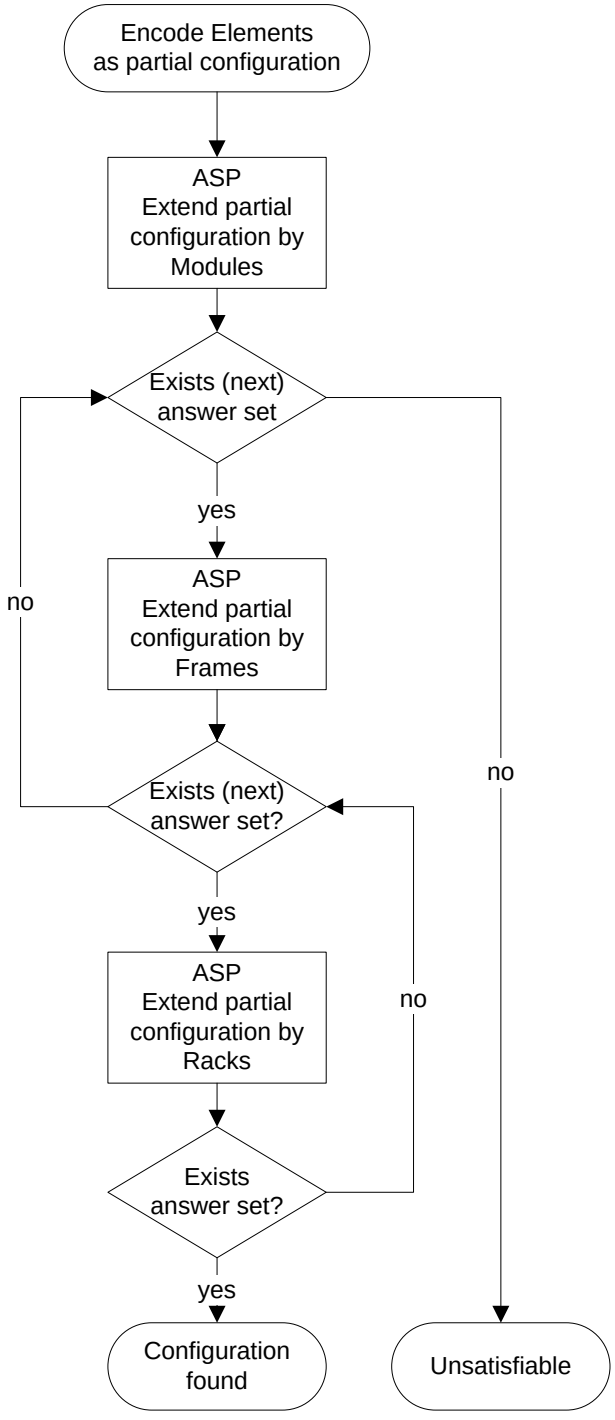


Figure 1. The decomposition approach

The downside of the decomposition approach is that it requires a rewrite of the program. One cannot even reuse the checking part, as the partial configuration considered in the subproblem will violate some constraints of the domain (otherwise it would already be a complete configuration). For example in step one the modules will have no frames assigned yet.

5.2 Incremental solving

It is a desirable property of a product configuration domain that adding an element/functionality to an existing configuration changes the existing configuration only minimally. In the best case the configuration does not have to be modified at all but only extended by components required by the new elements.

If a product configuration domain has this property, grounding size can be reduced by building the configuration incrementally, e.g., instead of adding all elements at once they are added one after the other. At each solver step the required components for one additional component are created and a complete configuration is computed. This is illustrated in Fig. 2. If a configuration is found in a substep this configuration is a valid configuration. Therefore in the incremental approach the checking part of the original program can be reused without modification.

As in each substep only one additional component is added to the configuration, the guessing program must only be able to generate all new components required by the additional component. Therefore the grounding is reduced significantly. At every step the new component either reuses an existing part of the configuration or uses a new component generated by the guessing program.

Again like in Section 5 there is the need for some method to compute the number of required components to support an additional component, e.g. an additional ElementD will require at least 4 ModuleIV instances, 4 Frames and 1 Rack.

5.3 Deriving substructures

Configuration problems often consist of hierarchically organized objects with a predetermined substructure. If the subobjects of a given type are fixed, it is possible to derive the sub-objects directly instead of generating them with a guess-and-check approach.

In our example the relation between elements and modules only depends on the type of the element, therefore it is possible to replace the generative generate-and-test approach with the following implementation, which uses arithmetics to directly derive the necessary modules for elements.

```
% derive modules for elements of type B
type(E+1,moduleII) :- type(E, elementB).
type(E+2,moduleII) :- type(E, elementB).
element_to_module(E,E+1) :- type(E, elementB).
element_to_module(E,E+2) :- type(E, elementB).
```

The disadvantage of this approach is that certain object IDs now have a special semantics and cannot be treated interchangeably anymore, which makes defining partial configurations more complicated. But this is typically the case when symmetry-breaking constructs are defined in a program.

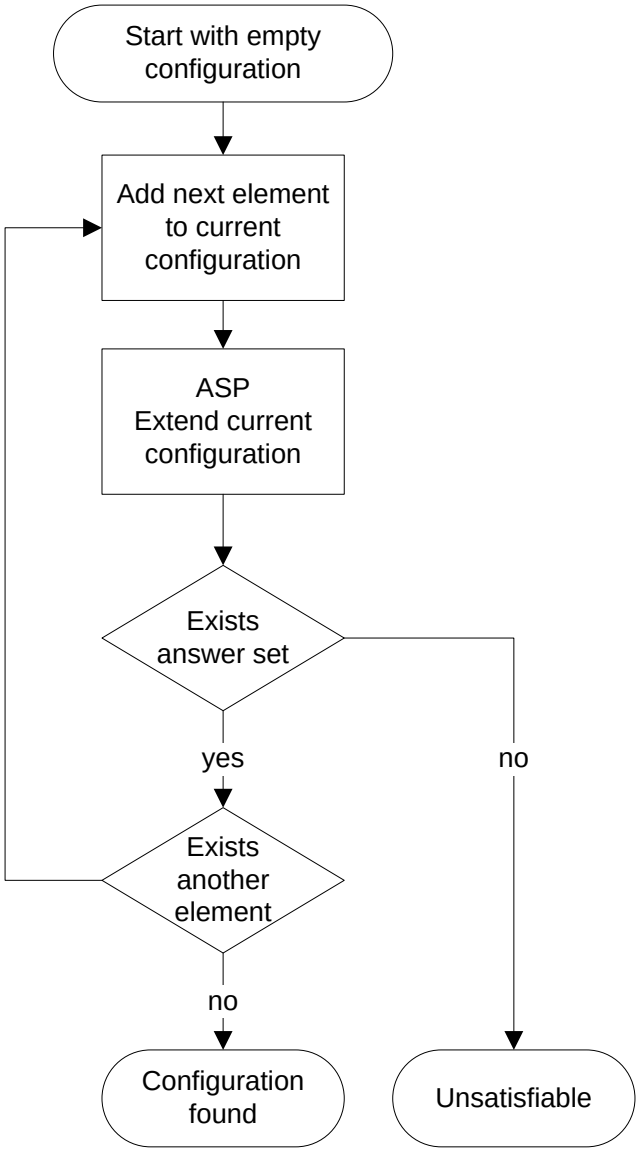


Figure 2. The incremental approach

5.4 Using a lazy-grounding solver

Lazy grounding is an attempt to solve the grounding bottleneck in general, i.e. without the need to adapt input programs. In contrast to traditional ASP systems, which first produce the full grounding for the input program and then solve this propositional program, lazy-grounding systems interleave grounding and search to avoid storing the entire ground program in memory. Current lazy-grounding systems cannot compete with pre-grounding ones in terms of runtime performance, however. We experiment with Alpha, the first lazy-grounding ASP solver to employ conflict-driven nogood learning, one of the major success factors of traditional ASP solvers [18]. However, Alpha (like other lazy-grounding systems) has not yet reached the maturity of traditional ASP systems. One consequence of this is that input programs still have to be adapted because language constructs like disjunction, choice rules, or aggregates are not supported by Alpha yet.

To represent the relations between different component types we use a slot-based representation for every possible relation. For example the relation between racks and frames is represented by the predicate $rack\_to\_frame(RACKID, SLOT(1..8), FRAMEID)$, i.e. a rack has 8 slots to assign a frame to. The slot representation makes it easier to check the cardinality constraints in ASP solvers that do not support aggregates, but makes it necessary to introduce symmetry-breaking constraints if symmetric solutions shall be avoided.

```
type(1,elementA).
type(2,moduleI).
type(3,rackSingle).
type(4..7,frame).
type(8..10,none).
element_to_module(1,1,2).
rack_to_frame(3,1,4).
rack_to_frame(3,2,5).
rack_to_frame(3,3,6).
rack_to_frame(3,4,7).
frame_to_module(4,1,1).
```

One way to improve the performance of a lazy-grounding solver is to employ domain-specific or domain-independent heuristics. Initial research in this area is described in [16].

6 Evaluation

For our experiments, we used the tradtional ASP system clingo[5] 5.2.0 [8] and the lazy-grounding system Alpha[6] [18].

The baseline for our benchmarks is provided by the traditional ground-and-solve approach. Grounding size was measured by producing the grounding in clingo's intermediate format and measuring the size of the resulting text file. For the incremental and the decomposition approaches, grounding size was measured similarly: For the incremental approach, we started with an empty configuration and increased the number of elements step by step, measuring the size of the grounding after each step. For the decomposition approach, we had instances with 1-30 elements, where the maximum number of objects was always set to 15 times the number of elements. The size of the grounding was calculated by adding up the grounding sizes of the three sub-problems per instance.

For the lazy-grounding approach, a different way to estimate the size of the grounding had to be chosen. In this case, the peak total memory usage of the system was measured. From this total memory usage, 64 kB were subtracted to account for the system's basic memory usage. Where no data point for lazy grounding is shown in Figs. 3 to 5, the instance could not be solved by Alpha within 300 seconds. This was the case for all problem instances where a configuration not just has to be checked but also created, which can be seen in Fig. 5.

Results are shown in Figs. 3 to 5. In Fig. 3, we compare the pre-grounding approach with the lazy-grounding approach to check a complete configuration just using the checking part of the program (not the guessing part). In Fig. 4, the guessing part of the program is also included, which makes the grounding larger. In both cases, incremental and decomposition approaches do not make sense because they are only used to find a configuration, not to check an existing one. In Fig. 5 we show numbers for finding a configuration. Here, no results for the lazy-grounding approach can be seen because in our experiments this approach did not yield positive results within a time-out of 300 seconds.

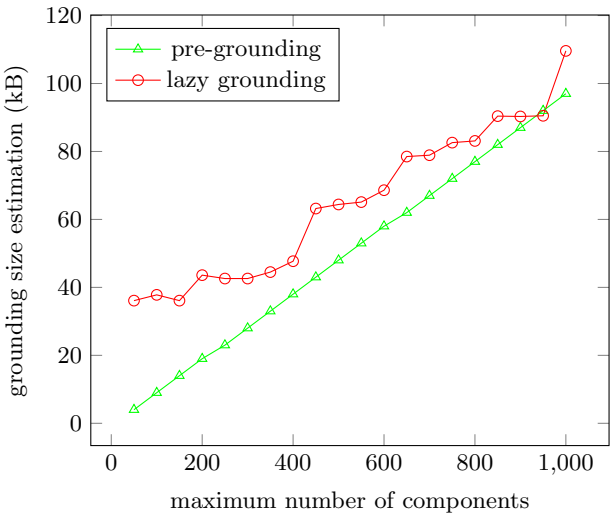


Figure 3. Grounding size for checking a complete configuration

It can be observed that very little grounding is necessary if only the checking part of the program is used to check a configuration. This holds for both the pre-grounding and the lazy-grounding approach (cf. Fig. 3). If the guessing part is included as well, the grounding size of pre-grounding systems rises significantly, while space consumption by the lazy-grounding system Alpha stays very low. Some instances are already too hard computationally for the latter, however (cf. Fig. 4). For creating configurations, both the incremental and the decomposition approach yield smaller groundings than the traditional pre-grounding approach (cf. Fig. 5). The incre-

[5] https://potassco.org/clingo/
[6] https://github.com/alpha-asp

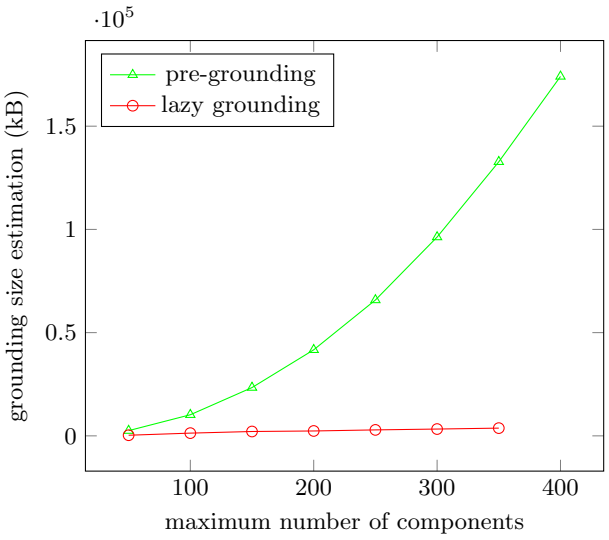


Figure 4. Grounding size for checking a complete configuration (incl. guessing part)

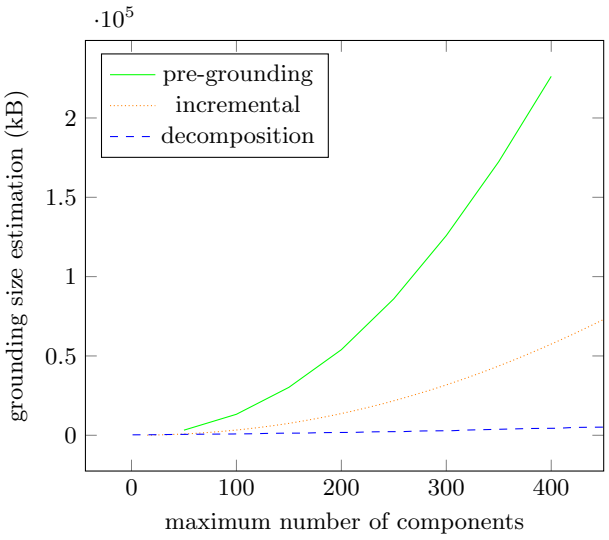


Figure 5. Grounding size for creating a complete and correct configuration

mental approach fares best in this sense, however its runtime performance (which is out of scope of this paper) is not very good.

A qualitative assessment of advantages and disadvantages of each approach is reported in Table 1.

7 Related Work

In this paper we focused on configurations with a large number of components. Another feature of product configurator knowledge bases that frequently leads to an explosion of grounding sizes are integer variables with large domains. Constraint answer set programming (CASP) solvers like clingcon [9] or ASCASS [17] can be used to handle these knowledge bases in ASP efficiently.

In this paper we used a domain-specific problem decomposition to solve the grounding problem. Domain-independent program decomposition approaches have a long history in answer set programming [13] and various applications [5].

Our technique for incrementally building a configuration can be seen as a domain-specific handling of the existential quantifier. A general approach for the problem of model generation can be found in [3]. The technique is also related to incremental solving [7].

8 Conclusion

In this paper we discuss some pragmatic techniques for handling large-scale product configuration problems with ASP. On the upside, we show that using our specialized approaches an off-the-shelf ASP solver like clingo can handle large instances of simple product configuration domains. On the downside, this requires encodings and reasoning tasks specific to the technique used for reducing grounding size.

With further progress of dedicated lazy-grounding solvers it is expected that these specific encodings become obsolete. Unfortunately current lazy-grounding solvers cannot compete with state-of-the-art ASP solvers when it comes to solving performance.

Although the techniques discussed in this paper have been developed primarily to reduce the grounding size, they are also interesting in their own right. Decomposing a large knowledge base frequently occurs in practice. For instance in the case of product configuration it might be desirable to first configure the hardware aspects and then the software aspects of a large system, because otherwise the user of the configurator would be overwhelmed by the complexity of the configuration task.

9 Future Work

In the future we want to investigate how to further automate the techniques discussed in this paper. One key problem that needs to be solved is automatic reasoning about component cardinalities as this is essential for efficient grounding and solving.

Open issues in the area of lazy grounding in general are forgetting of nogoods, learning on the non-ground level, and better search heuristics (cf. [16,18]). Furthermore, we plan to develop domain-specific heuristics to improve the solving performance of lazy-grounding systems. A typical domain-specific

Approach	Pros	Cons
Decomposition	often natural decomposition exists (e.g. hardware/software); user interaction between subproblems possible	rewriting of program necessary; no nogood learning between subproblems
Incremental solving	reuse of checking part; simulates manual configuration	cardinality reasoning necessary; if incremental steps are not independent, solving will deteriorate
Deriving substructures	solving and grounding improved	special semantics of IDs make handling of partial configurations and exchange with other configurators more complicated
Lazy grounding	no rewriting necessary	performance of current solvers; domain-specific heuristics may be required

Table 1. Pros and cons

heuristics in the context of product configuration would be to reuse components that are already part of the configuration.

To make the techniques described in this paper applicable to other ASP solvers as well we did not take advantage of the advanced (python/lua) scripting capabilities of the latest clingo version. As this would give us better control over the solving process, we expect to come up with even better results.

REFERENCES

[1] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński, 'Answer set programming at a glance', *Communications of the ACM*, **54**(12), 92–103, (2011).

[2] Francesco Calimeri, Martin Gebser, Marco Maratea, and Francesco Ricca, 'Design and results of the Fifth Answer Set Programming Competition', *Artificial Intelligence*, **231**, 151–181, (2016).

[3] Broes de Cat, Marc Denecker, Peter J. Stuckey, and Maurice Bruynooghe, 'Lazy Model Expansion: Interleaving Grounding with Search', in *Journal of Artificial Intelligence Research*, pp. 235–286, (2015).

[4] Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer, 'Declarative Problem-Solving Using the DLV System', in *Logic-based Artificial Intelligence*, ed., Jack Minker, The Springer International Series in Engineering and Computer Science, Springer US, (2000).

[5] Thomas Eiter, Michael Fink, and Thomas Krennwallner, 'Decomposition of declarative knowledge bases with external functions.', in *IJCAI*, volume 9, pp. 752–758, (2009).

[6] Andreas A. Falkner and Herwig Schreiner, 'Siemens: Configuration and Reconfiguration in Industry', in *Knowledge-based Configuration*, eds., Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, 199–210, Morgan Kaufmann, Amsterdam, (2014).

[7] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele, 'Engineering an incremental ASP solver', in *International Conference on Logic Programming*, pp. 190–205. Springer, (2008).

[8] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub, 'Clingo = ASP + Control: Preliminary Report', in *Technical Communications of the Thirtieth International Conference on Logic Programming (ICLP'14)*, eds., M. Leuschel and T. Schrijvers, volume arXiv:1405.3694v1, (2014).

[9] Martin Gebser, Max Ostrowski, and Torsten Schaub, 'Constraint answer set solving', in *Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09)*, eds., P. Hill and D. Warren, volume 5649 of *Lecture Notes in Computer Science*, pp. 235–249. Springer-Verlag, (2009).

[10] Michael Gelfond and Vladimir Lifschitz, 'The Stable Model Semantics For Logic Programming', in *Proceedings of the Fifth International Conference and Symposium of Logic Programming*, eds., R. Kowalski and K. Bowen, pp. 1070–1080. MIT Press, (1988).

[11] Vladimir Lifschitz, 'What Is Answer Set Programming?', in *Twenty-Third AAAI Conference on Artificial Intelligence*, (2008).

[12] Vladimir Lifschitz, 'Thirteen Definitions of a Stable Model', in *Fields of Logic and Computation*, eds., Andreas Blass, Nachum Dershowitz, and Wolfgang Reisig, volume 6300 of *Lecture Notes in Computer Science*, 488–503, Springer, Berlin, Heidelberg, (2010).

[13] Vladimir Lifschitz and Hudson Turner, 'Splitting a logic program.', in *ICLP*, volume 94, pp. 23–37, (1994).

[14] Richard Taupe, Andreas A. Falkner, and Gottfried Schenner, 'Deriving Tighter Component Cardinality Bounds for Product Configuration', in *Proceedings of the 18th International Configuration Workshop within CP 2016 conference*, eds., Elise Vareilles, Lars Hvam, and Cipriano Forza, pp. 47–54, Albi, (2016). École des Mines d'Albi-Carmaux.

[15] Richard Taupe and Erich Teppan, 'Influence of ASP Language Constructs on the Performance of State-of-the-Art Solvers', in *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pp. 88–101. Springer International Publishing, (2016).

[16] Richard Taupe, Antonius Weinzierl, and Gottfried Schenner, 'Introducing Heuristics for Lazy-Grounding ASP Solving', in *1st International Workshop on Practical Aspects of Answer Set Programming*, (2017).

[17] Erich Teppan, 'Solving the Partner Units Configuration Problem with Heuristic Constraint Answer Set Programming', in *Proceedings of the 18th International Configuration Workshop within CP 2016 conference*, eds., Elise Vareilles, Lars Hvam, and Cipriano Forza, Albi, (2016). École des Mines d'Albi-Carmaux.

[18] Antonius Weinzierl, 'Blending lazy-grounding and CDNL search for answer-set solving', in *Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings*, eds., Marcello Balduccini and Tomi Janhunen, volume 10377 of *Lecture Notes in Computer Science*, pp. 191–204. Springer, (2017).

Assessing the complexity expressed in a variant table

Albert Haag[1]

Abstract. One statistic commonly associated with a product configuration model is the number of possible solutions it entails. I argue in this paper that this is not a good indicator of complexity, neither for the interactive configuration task, nor for the associated business processes. I focus on the common scenario where the solutions form a set of *variants* that share the same *product properties*, but differ in the *features* that are assigned to the properties. The list of distinct variants then forms a so-called *variant table*.

In the completely unconstrained case, the number of valid variants is the entire solution space: the Cartesian product of the *domains* of the product properties. This set grows exponentially with increasing number of features. However, we may argue that the complexity both in product logistics and configurator interaction then scales only linearly with the the number of features: When a feature is added to the model for some product property, this feature must be handled, and this incurs some "cost" for both interactive configuration and business logistics. But there is no effect on the other product properties (nor, ideally, on the handling of the other features of the property).

Representing the solution space as a Cartesian tuple can be seen as a compressed form of the explicit list of all variants. Based on this observation, I propose that complexity should be linked to the *compressibility* of the variant table. To this end I argue that the compression of a table to a *variant decomposition diagram* (VDD) [8] and/or *Multi-valued Decision Diagram* (MDD) [3] is a better basis for a measure of complexity, both for a configurator and for the business. I also identify subspaces in a variant table that are "constraint free", i.e., pose no decision problem.

The arguments also apply to cases where the overall model has been compiled to some form of *decision diagram*, i.e., to a VDD, MDD, or BDD (*Binary Decision Diagram* [9, 10]). The compiled form can be seen as a conceptual table.

As tables are predominant elements of a product model, and compression techniques further empower them, I discuss additional requirements for making their use universally more appealing in practical settings. I suggest to formally relax the usual finiteness requirement to allow table cells with both unbounded numeric intervals (including real-valued (floating point) bounds) and wild cards matching any Boolean predicate. I call a domain that includes such non-finite elements *quasi-finite*.

1 Introduction

The number of variants of a configurable product a business can offer today may be astronomically large. For example, some car manufactures now offer their product in almost any conceivable color. They can do this because their production technology is such that adding more hues does not add much to production complexity. Similarly, current textile printing technology would allow a vast number of images to be imprinted on a configurable T-shirt. Because the number of variants grows exponentially with the number of offered choices, it is easy to extend models for configurable products so that any given limit on the number of variants is exceeded.

While adding freedom in specifying an image for the T-shirt adds enormously to the number of possible T-shirt variants, it does not necessarily add complexity. Production of imprintable T-shirts would be unaffected, given a suitable textile printer. The complexity of the configuration task is not increased either, regardless if we see this as a decision support or a constraint solving problem[2].

The focus of this paper is on a *mass customization* business that aims to provide *individualized* products[3]. In this setting, the business offers *variants* of a generic product that is described in the product model. All *variants* share the same *product properties*, but differ in the *features* that are assigned to the properties. As pointed out, the business complexity of providing such individualization can be somewhat mitigated by appropriate technology and organization.

The list of distinct offered variants can be captured in a *variant table*. Assuming that k product properties $v_1 \dots v_k$ are given, this table $\mathfrak{T}$ can be represented in explicit relational form as a (possibly huge) matrix:

$$\mathfrak{T} = \left(a_{ij}\right) \tag{1}$$

where a_{ij} is the feature assigned to v_j for the i-th variant.

We may observe that tables are the way businesses capture and represent data in general. Thus, the single variant table $\mathfrak{T}$ would seem to be the preferred and natural way to model variants. Some practical limitations on size and expressibility currently impede this[4]. Whatever way the product is modeled, its complexity must be known in order to determine its effect on the cost of:

- continually maintaining the model,
- providing a quote and sales process for selecting variants, and
- the associated product logistics (procurement, production, shipping, invoicing, etc.)

One aspect of complexity in a product model is related to the underlying decision problem. To this end, I introduce the notions of

[1] Product Management Haag - R&D, Germany, email: albert@product-management-haag.de

[2] Additional freedom does necessitate that a sales configurator for the T-shirt is able to accept an almost arbitrary image as input, and this may make the decision problem of an end customer more difficult in that they need to decide on an image.

[3] We take the term *product configuration* to refer to any kind of configuration tasks for products, including *engineering* and *product design* tasks. *Mass customization* refers to the common but more special setting that a product variant can be defined by assigning values to a fixed given set of product properties.

[4] The limitations on size can be addressed by compression as long as the underlying product complexity is managed. This is the main topic of this paper. Practical extensions of expressibility (*quasi-finite* domains) are briefly discussed at the end of this paper.

a *free property* and a *free feature*. Let us assume that choosing an imprint for a T-shirt has no effect whatsoever on any of the other features of the T-shirt[5]. I call such a property a *free property*. Any feature of a free property is a *free feature*, i.e., it does not interact with any features of any of the other properties. More precisely, a feature is *free* in a given configuration state, if it will remain validly choosable regardless of the further choices for other properties. And if all remaining valid features for a property are free then the property is free in that state.

I postulate that free features contribute only a fixed increment to both configuration and business complexity. I argue this in Section 6.2 for the examples in Section 2, but it requires further validation in the field, which has not yet been performed.

The examples in Section 2 assume a model of a T-shirt as a single variant table $\mathfrak{T}$. This is the setting I focus on here.

The setting of a single overall variant table $\mathfrak{T}$ also conceptually applies, when a complex model can be compiled to a *decision diagram*, notably a VDD (*Variant Decomposition Diagram* [8, 7]), an MDD (*Multi-valued Decision Diagram* [3, 1]), or a BDD (Binary Decision Diagram [10, 9]). The compilate can conceptually be seen as equivalent to an overall variant table $\mathfrak{T}$. It is beyond the scope of this paper to provide a detailed treatment of these approaches. But I give a brief characterization:

- A BDD is a generic approach at finding a very compact representation for a given set of logical propositional formulae (only Boolean variables) as a decision diagram [10].
- An MDD is a decision diagram more directly tuned to the setting here. A node in the diagram maps to a product property, the edges of the diagram emanating from it correspond to the valid domain of features at that point. MDDs can be mapped to BDDs and vice versa [2].
- A VDD is based on a "column oriented" decomposition of a table. In certain circumstances, which we may take as given here, a VDD can be mapped to an MDD and vice versa [8].

The above observation does point to one conclusion: Complexity in a product model can be split into "complexity of compilation" and "complexity of execution". The "complexity of compilation" can be prohibitive, which then may be an immediate impediment to the approach followed here. However, in practice, compilation often succeeds. In particular, the heuristics for constructing VDDs discussed in [8] have so far been practically viable as far as they were tested on practical examples. I am only interested in "complexity of execution" in the sequel, and assume that the overall list of variants $\mathfrak{T}$ is either explicitly given or obtained through a compilation approach. When $\mathfrak{T}$ is given explicitly as in (1) then compiling $\mathfrak{T}$ to a VDD (or an MDD) is referred to as *compression* of $\mathfrak{T}$.

The strong focus here on the table paradigm is motivated by the following: We observe that tables are a preferred way for businesses to model constraints on the variants they offer[6], as long as they remain manageable. When they become unmanageable, spreadsheets using various informal work-arounds are often employed to extend the table paradigm. These can include:

- wild card expressions,
- compression using *c-tuples* (allowing multiple values in one cell)[7],
- real-valued (floating point) intervals[8],
- alternatively listing the exclusions (invalid variants), and
- normalization: splitting up a big table into several smaller ones

The first two are briefly discussed in Section 10.2. The last two are of practical importance, but out-of-scope here.

2 Examples: Configureable T-shirts

In [6, 7] a very simple T-shirt model is used as an example. This is based on three product properties: *Size*, *Color*, *Imprint*, and two constraints that can be easily expressed verbally:

1. One of the *Imprints*, *'MIB'* ("Men In Black") is only available in black
2. The other *Imprint*, *'STW'* ("Save The Whales") is not available in a small size.

In [8] the running example used is a T-shirt with 120120 valid variants, based on eight properties: *Style*, *Fabric*, *Size*, *Color*, *Imprint*, *ImprintColor*, *Price*, and *Dye*.

For the discussion here, it is opportune to have a series of T-shirt models expressed as variant tables of increasing complexity. These merge the ideas of the examples in [6, 7, 8]. We want the unconstrained case to make business sense. To this end the eight properties of the example in [8] are modified as follows: The product property *Price* is replaced by $Donation$ (representing a voluntary donation to a charity associated with the chosen imprint) and the property *Dye* is replaced by *CO2Offset* (a voluntary surcharge to offset the environmental impact of the CO_2 emitted in producing the T-shirt).[9] We add the two imprints *'MIB'* and *'STW'* from the simple T-shirt in [6, 7][10]. A further property *ImprintSize* is added for potentially reasoning about a relation between *ImprintSize* and *Size*.

All in all, we take an order for a T-shirt to be fully described by the nine product characteristics: *Style*, *Fabric*, *Size*, *Color*, *Imprint*, *ImprintColor*, *ImprintSize*, *Donation*, and *CO2Offset* with the finite domains for each of the properties as depicted in Tables 1 and 2.[11]

The examples are numbered, so that we may refer to them later.

Example 1. Unconstrained T-shirt

The T-shirt model as described above with the nine properties with their domains as specified in Tables 1 and 2, but without any additional constraints. There are 120324096 (120 million) possible variants[12].

Example 2. T-shirt with the constraint that *'MIB'* implies *'Black'*

The T-shirt model as in Example 1, but with the constraint that an imprint *'MIB'* implies the imprint color *'Black'*. Any *'MIB'* variant coupled with any non-black *ImprintColor* is then invalid. There are 884736 such invalid variants. 119439360 valid variants remain.

[5] Section 2 contains more detailed examples.

[6] Simple numeric formulas (perhaps linked to a table) would be another preferred form. However, they are less prevalent than tables.

[7] A *c-tuple* is a tuple of spreadsheet cells, where each cell can contain multiple values. A cell represents the disjunction of these values. The entire tuple represents the Cartesian product of these sets of values.

[8] A numeric interval represents (a possibly infinite) disjunction.

[9] We may assume that both of these features in some way affect the final product: as part of the T-shirt label or as a certificate added to the delivery.

[10] These properties and their two constraints were omitted from the T-shirt model given in [8] in order not to clutter up the discussion there.

[11] We may imagine a web-shop that takes an order for a personalized T-shirt as input and then delivers based on this specification. Pricing would here be a separate issue that might be handled during the checkout process.

[12] 120324096 (120 million) variants is a large number − a list of this size is probably not feasible without compression. There is no apparent model complexity, except that the configurator has to be able to either provide fast interactive filtering over all 120324096 variants or be clever about an alternate presentation.

Example 3. Adding the constraint that *'STW'* needs large shirts

The constraint that any *'STW'* variant with a small size (*'3T'*, *'4T'*, *'S'*, or *'XS'*) is considered to be invalid is additionally added to Example 2. There are 589824 such invalid variants. Because this set does not overlap that of the invalid *'MIB'* variants, we can simply subtract both 884736 and 294912 from the total number of all variants. This still yields a large number (close to 120 million): 118849536 valid variants.

What is the complexity of a variant table $\mathfrak{T}$ representing the valid variants in Example 3? This is discussed in more detail in subsequent sections. But, even without a detailed analysis we can see that there are three disjoint states (three subproblems from the business point of view − each of them large, but unconstrained):

1. Both *'MIB'* and *'STW'* are excluded externally (by user choice). The problem is again simple, because the other 100 imprints are completely unconstrained.
2. *'MIB'* is selected. Then *ImprintColor* is forced to *'Black'*. All other properties are completely unconstrained
3. *'STW'* is selected. Then the sizes *'3T'*, *'4T'*, *'S'*, and *'XS'* are excluded. All other properties are completely unconstrained.

By adding further constraints in the further examples one by one, we obtain an increasing problem complexity coupled with a decreasing size in the number of valid variants. For these examples, the number of valid variants is best determined by actually constructing the variant table and counting the variants. These results are summarily given in Section 9.

Table 1: Domains for four T-shirt product properties

Style	Fabric	Imprint	ImprintColor
FullSleeve	Cotton	MIB	Black
HalfSleeve	Mixed	STW	Blue
NoSleeve	Synthetic	100 *other imprints*	Red
			Green

Table 2: Domains for remaining five T-shirt product properties

Size	Color	ImprintSize	Donation (EUR)	*CO2Offset* (EUR)
3T	Black	Baby	0.00	0.00
4T	Blue	Big	0.99	0.99
L	Green	Cute	1.99	1.99
M	Pink	ExtraBig	5.00	2.99
S	Purple	Full	9.99	3.99
XL	Red	Medium	15.00	4.99
XS	White	Small	20.00	5.99
XXL	Yellow	Tiny	25.00	10.00

Example 4. Constraining *'Style'*, *'Fabric'*, and *'Size*

The T-shirt model as in Example 3, but with the additional constraint expressed in Table 3 taken from [8][13]. This states that the fabric *'Cotton'* does not come in all styles, and that non-cotton fabrics are not available in all sizes. "*" is a wild card symbol that matches any value in the domain.

[13] Tables 3 and 4 use c-tuples as a pragmatic form of compression and might be maintained in exactly this way by a product manager using a spreadsheet.

Table 3: Compressed relation of Style, Fabric, and Size

Style	Fabric	Size
{HalfSleeve, FullSleeve}	Cotton	*
*	{mixed, synthetic}	{XS, S, M, L, XL, XXL}

Example 5. Constraining *'Color'* and *'ImprintColor'*

The T-shirt model as in Example 4, but with the additional constraint that only certain combinations of the properties *'Color'* and *'ImprintColor'* are valid. This relation is given in Table 4. "*" is a wild card symbol that matches any value in the domain.[14]

Table 4: Relation of Color and ImprintColor

Color	ImprintColor
Black	{'Blue', 'Red', 'Green'}
Blue	{'Black, 'Red', 'Green'}
Red	{'Black', 'Blue', 'Green'}
White	*
Green	{'Black', 'Blue', 'Red}
Pink	*
Purple	*
Yellow	*

3 Filtering of variant tables

Given a product model in the form of a single variant table $\mathfrak{T}$ and an external restriction $\mathbf{R}$ on the desired features (e.g., by the "user"), the basic operation a configurator will have to perform is determining the valid variants in $\mathbf{R}$ via the intersection in (2). This will be the case regardless of whether $\mathfrak{T}$ is unconstrained or not.

We may take $\mathbf{R}$ to be a *c-tuple*[15]: $\mathbf{R} = R_1 \times \ldots \times R_k$.

$$\mathfrak{R} := \mathfrak{T} \cap \mathbf{R} \tag{2}$$

$\mathfrak{R}$ can be calculated by a *filtering query*, which is the most central operation on the table that must be efficient. In [7, 8] it is pointed out that (2) can be realized as an SQL query when $\mathfrak{T}$ is in a relational database:

`SELECT * FROM` $\mathfrak{T}$ `WHERE` v_1 `IN` R_1 `AND` $\ldots v_k$ `IN` R_k; (3)

If $\mathfrak{T}$ is compressed to a VDD[16] $\mathfrak{V}$, then the intersection can be equivalently calculated from $\mathfrak{V}$ and $\mathbf{R}$, which I denote by:

$$\mathfrak{R} := \mathfrak{V} \sqcap \mathbf{R} \tag{4}$$

Following the database paradigm and its terminology, I shall call the intersection $\mathfrak{R}$ in (2), (4) (or the result of the equivalent query (3)) a *variant result set* (VRS). The performance of determining the

[14] It is tempting to formulate the content of this constraint as an inequality *Color* ≠ *ImprintColor*. But using a table is often preferred in practice, because:

- It is more precise. The inequality would compare "apples with oranges",
- It is not dependent on a perhaps proprietary modeling language, and
- It is the natural paradigm used for the other constraints

[15] Whereas it is conceivable that more general restrictions might be formulated, this is currently not supported in most configurators, and we do not consider it here.

[16] VDDs are detailed in [8]. I review them in brief in Section 4.

VRS $\mathfrak{R}$ in (4) is linearly dependent on the number of nodes in $\mathfrak{V}$ [8], not the number of rows in $\mathfrak{T}$.

A VRS can be used as the basis for further filtering. Thus, (3) can be generalized to further filter any VRS:

$$\texttt{SELECT} \ * \ \texttt{FROM} \ \mathfrak{R} \ \texttt{WHERE} \ v_1 \ \texttt{IN} \ R_1 \ \texttt{AND} \ \ldots v_k \ \texttt{IN} \ R_k; \quad (5)$$

This is needed when the external restriction is successively narrowed by the user. In reality, (5) is the central operation that a configurator must support efficiently.

4 Brief review: Variant Decomposition Diagram

A *Variant Decomposition Diagram* (VDD) is the table compression method referred to in the sections on compression. I give a brief characterization using the example of the very simple T-shirt model in [6, 7], which is listed in table 5 here. The resulting VDD is depicted in Figure 1.[17]

In Table 5 choose the first value in the first column (*'MIB'*). Table 5 is then decomposed into three parts:

1. the boxed cells (*Imprint* = *'MIB'*),
2. the *slanted* cells (rows with *Imprint* = *'MIB'*, but without boxed cells), and
3. the **boldface** cells (rows with *Imprint* ≠ *'MIB'*)

Table 5: Variant table $\mathfrak{T}_S$ for the simple T-shirt

Imprint	Size	ImprintColor
MIB	*Large*	*Black*
MIB	*Medium*	*Black*
MIB	*Small*	*Black*
STW	**Medium**	**Black**
STW	**Large**	**Black**
STW	**Medium**	**White**
STW	**Large**	**White**
STW	**Medium**	**Red**
STW	**Large**	**Red**
STW	**Medium**	**Blue**
STW	**Large**	**Blue**

A root node representing the entire table is created and labeled with the selected feature and column (*'MIB'* in column 1). The root node has two outgoing arcs (links). One, the *HI*-link, points to a node representing the subtable of slanted cells (right subtable with one less column). The other, the *LO*-link, is drawn with a dotted line and points to the subtable of the subtable of boldface cells (left subtable). The subtables formed by the slanted and boldface cells, respectively, can be recursively decomposed in like manner by always choosing the first value in the first column. For the right subtable (slanted cells) this is the feature *'Large'* in column 2. For the left subtable (boldface cells) this is the feature *'STW'* in column 1. A *HI*-link to an empty table points to the *sink* $\top$ (*T*). A *LO*-link to an empty table points to the *sink* $\bot$ (*F*).

[17] Each node in Figure 1 is labeled in the form $\langle m : (j, val)|n\rangle$. The labels are artifacts of the current VDD implementation. (j, val) designates the assignment of *val* to product property v_j (a product feature). m is the variable number assigned to this feature (i.e., it encodes (j, val)), and n is the node number assigned during construction of the VDD (a unique identifier for the VDD-node).

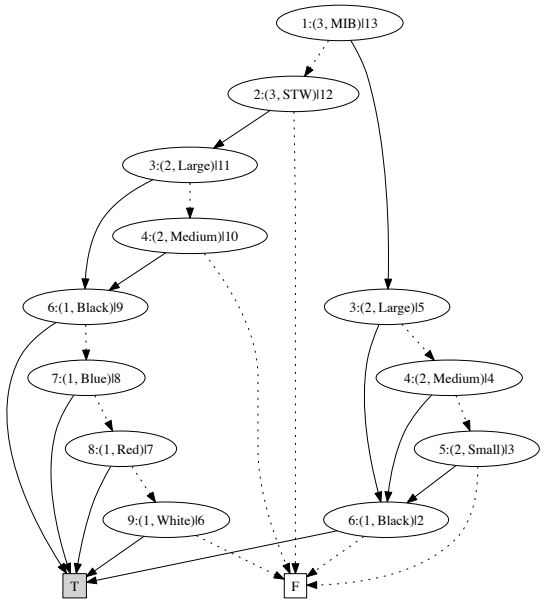


Figure 1: VDD of simple T-shirt

Each path in a VDD from the root node to a sink $\top$ represents a variant. If a feature is part of a solution, then the *HI*-link is followed from all nodes that refer to that feature. Otherwise the *LO*-link is followed. For example, *'STW'*, *'Medium'*, and *'Black'* are the chosen features on the path

$$\textit{'MIB'} \rightsquigarrow \textit{'STW'} \rightarrow \textit{'Large'} \rightsquigarrow \textit{'Medium'} \rightarrow \textit{'Black'} \rightarrow \top \quad (6)$$

where "$\rightsquigarrow$" denotes a *LO*-link and "$\rightarrow$" denotes a *HI*-link.

4.1 L-chains

Let $\mathfrak{V}$ denote a VDD and $\nu \in \mathfrak{V}$ one of its nodes. Then a set of nodes whose *HI*-links all point to the same child node and are linked together through *LO*-links is called an *l-chain*. In Figure 1 there are three l-chains:

- the nodes for *'Large'* and *'Medium'* on the left
- the nodes for *'Black'*, *'Blue'*, *'Red'*, and *'White'* on the left
- the nodes for *'Large'*, *'Medium'*, and *'Small'* on the right

Some individual nodes that are not part of a larger l-chain are considered as a singleton l-chain. In Figure 1 these are

- the root node for *'MIB'*,
- the node for *'STW'* on the left, and
- the node for *'Black'* on the right

An l-chain can be labeled by the set of the collected values it refers to. The l-chains, above would thus be labeled as

$$\{\textit{'Large'}, \textit{'Medium'}\}, \{\textit{'Black'}, \textit{'Blue'}, \textit{'Red'}, \textit{'White'}\} \ldots \{\textit{'Black'}\}$$

4.2 Solutions of a VDD as c-tuples

The solution (6) is comprised of the nodes for *'STW*, *'Medium'*, and *'Black'*. Let us refer to these nodes by node number n as ν_2, ν_{10}, and

ν_9.[18] Each of these nodes is part of an l-chain. Let C_2, C_{10}, and C_9 denote the labels of these l-chains:

- $C_2 = \{$*'STW'*$\}$ for ν_2,
- $C_{10} = \{$*'Large'*, *'Medium'*$\}$ for ν_{10}, and
- $C_9 = \{$*'Black'*, *'Blue'*, *'Red'*, *'White'*$\}$ for ν_9.

This means that the Cartesian set $\mathbf{C}_2 := C_2 \times C_{10} \times C_9$ is a subset of Table 5. A Cartesian product that is a subset of the overall variant solution space will be called a *c-tuple*. A c-tuple that contains only valid variants (i.e., is a subset of the variant table $\mathfrak{T}$) identifies an area of some regularity (when it is not trivially a single variant).

The compression of a variant table $\mathfrak{T}$ to a VDD $\mathfrak{V}$ can be seen as one way of determining c-tuples (via l-chains) in $\mathfrak{T}$. Figure 1 shows that Table 5 is the disjoint union of two c-tuples[19]:

$$\mathbf{C}_1 := \{\textit{'MIB'}\} \times \{\textit{'Large'}, \textit{'Medium'}, \textit{'Small'}\} \times \{\textit{'Black'}\}$$

$$\mathbf{C}_2 := \{\textit{'STW'}\} \times \{\textit{'Large'}, \textit{'Medium'}\} \times \{\textit{'Black'}, \textit{'Blue'}, \textit{'Red'}, \textit{'White'}\}$$

The added benefit of a VDD over a mere list of c-tuples is that if several c-tuples have common tails (ends) then these tails can be represented by a single common node. Thus a VDD offers more compression potential than finding a disjoint decomposition of a table as c-tuples. (There is no example of this in Figure 1.)

The compression to a VDD (and also any compression directly to c-tuples) is subject to heuristics, i.e. it is usually not possible to guarantee a "best" compression.

5 Free product properties

In Example 1 the T-shirt model is completely unconstrained, i.e., the entire solution space of over 120 million variants is valid. In this situation no choice of any feature for a product property will ever constrain any other product property. In other words, an external agent (the user) can "freely" choose any feature of any property, without affecting any other choices[20].

If the T-shirt example in Section 2 allowed an arbitrary text (perhaps of a certain limited length) to be imprinted on the T-shirt, this text could be modeled as a *free property* with an infinite domain. Each ordered or produced variant would have a value assigned to this property, but the configurator would only need to solicit the arbitrary input, quite separate from the rest of the configuration problem.

In Example 1 all properties are free properties. This is not necessarily a typical setting. However, it is more typical after an external restriction. If the external restriction is a subset of $\mathfrak{T}$, i.e. $\mathbf{R} \subset \mathfrak{T}$, then $\mathbf{R} = \mathfrak{T} \cap \mathbf{R}$, and all choices in $\mathbf{R}$ are free properties under $\mathfrak{T}$ and $\mathbf{R}$. In Example 2 of a T-shirt with only the constraint between the imprint *'MIB'* and the imprint color *'Black'*, all product properties become free properties, once it is clear that either *'MIB'* has been excluded, or equivalently, some other imprint has been chosen. Thus, product properties that are not globally free properties can become free, given a certain external restriction $\mathbf{R}$.

6 Complexity given only free properties

We could arbitrarily consider a completely unconstrained problem to be of *zero* complexity. However, a configurator must still be able to respond to filtering queries (2), (4) against this (large) set of variants. The complexity for the configurator may be seen as related to the complexity of these queries. This is discussed in Subsection 6.1. The more speculative, related issue of how business complexity is affected, is the topic of Subsection 6.2.

The general form of a VDD representing a c-tuple is given in Figure 2 for the example of the unconstrained simple T-shirt.

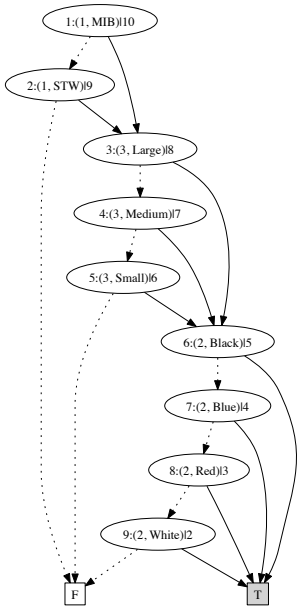


Figure 2: VDD of unconstrained simple T-shirt

6.1 Configurator complexity given only free properties

Somewhat arbitrarily, in (7) the complexity $cplx(\mathbf{R})$ of a c-tuple $\mathbf{R} = R_1 \times \ldots \times R_k$ is defined to be the number of distinct features in $\mathbf{R}$. We will subsequently see why this makes sense.

$$cplx(\mathbf{R}) := s_{\mathbf{R}} = \sum_{j=1}^{k} |R_j| \tag{7}$$

$s_{\mathbf{R}}$ is also the minimal number of nodes in a VDD for $\mathbf{R}$.[21] [22] The response-time for a filtering query to a VDD can be guaranteed based on the number of its nodes. Therefore, $cplx$ directly relates to the guaranteed performance for any filtering query and thus makes sense as a complexity measure for the interactive configuration.

$cplx$ increases with the size of the c-tuple $\mathbf{R}$, but only linearly with the number of its features, not exponentially, as does the number of tuples in $\mathbf{R}$.

6.2 Business complexity given only free properties

Let us make the following assumption: For a free property v_j, technology can be utilized so that production/delivery is affected only by

[18] See footnote 17.

[19] This would be seen more directly by constructing the merged form, $\mathfrak{V}^*$, of $\mathfrak{V}$ (see [8]).

[20] Of course, only one feature can be assigned to a given property at a time

[21] This section uses some results about VDD compression detailed in [7, 8]. We assume these to be true without the reasoning being reproduced here.

[22] This is illustrated in Figure 2. Each component C_j of a c-tuple $\mathbf{C}$ is represented by an l-chain consisting of a node for each element of C_j

a coefficient ω_j for the property and an additional weight w_x for the feature $x \in D_j$. The plausibility of this is established by a thought experiment: Adding a dyeing capability for additional colors of T-shirts increases the complexity of that work center, and it may be more difficult to provide some colors over others, but the rest of the T-shirt production/delivery is not affected (if there are no technical constraints). There would be a basic cost associated with the dyeing process (ω_4 for the property *'Color'*), and difficult colors would have higher weights, e.g., it could be that

$$0 = w_{\text{'White'}} < w_{\text{'Black'}} < w_{\text{'Blue'}} = w_{\text{'Red'}} < w_{\text{'Green'}} < \dots$$

Somewhat arbitrarily, but based on this observation, (8) proposes a measure for the business complexity $cplx_b(\mathbf{R})$ of a c-tuple:

$$cplx_b(\mathbf{R}) := \sum_{j=1}^{k} \sum_{x \in R_j} \omega_j w_x \tag{8}$$

Whether (8) is indeed a meaningful measures of the business complexity incurred by offering a given c-tuple of variants needs evaluation and verification in the field.

7 Complexity of a variant table $\mathfrak{T}$

A variant table $\mathfrak{T}$ can be disjointly decomposed into q c-tuples, and this can be seen as a decomposition into q disjoint subproblems, each of them consisting only of free properties. The compression of $\mathfrak{T}$ to a VDD implies such a decomposition as a side effect [7, 8]. As already pointed out, a main conceptual difference between the VDD representation and the representation as a disjoint list of c-tuples, is that the VDD has a more compact representation, because common tails to the c-tuples are represented only once in a VDD.

A query (3) based on a VDD (or a query based on a subsequent resulting VRS (5)) can exploit this gain. As could a business set-up. The complexity of the query depends only on the number of VDD nodes. Hence, I propose to define this node number as the overall configurator complexity for $\mathfrak{T}$, given as a VDD $\mathfrak{V}$.

$$cplx(\mathfrak{V}) := |\mathfrak{V}| \tag{9}$$

By analogy to (8), the associated business complexity would be:

$$cplx_b(\mathfrak{V}) := \sum_{\nu \in \mathfrak{V}} \omega_{col(\nu)} w_{feature(\nu)} \tag{10}$$

where $col(\nu)$ designates the product property that the node ν refers to and $feature(\nu)$ the associated assigned product feature.

While it would have seemed natural to define the complexity of $\mathfrak{T}$ to be the minimal complexity that can be achieved, the decomposition of a variant table $\mathfrak{T}$ to a VDD (and to the c-tuples it implies) is governed by heuristics, and a best compression is not readily found[23]. The same would be true for a measure based on a direct decomposition to c-tuples.

A point of speculation: A business set-up for processing variants would also be based on heuristics, and insights of how to compress the configuration model may lead insights into how to organize this business set-up.

[23] A VDD produced with a *column heuristic* [7] can be mapped to and from a *Multi-Valued Decision Diagram* (MDD) [8]. In this sense, the complexity observations here carry over to product models compiled into MDDs as well.

8 Free features and negation

A feature is defined to be *free* in a given configuration state, if it will remain validly choosable regardless of the further choices for other properties. And if all remaining valid features for a property are free then the property is free in that state. This is detailed in Subsection 8.1. Negation of a table allows identifying free features. Negation is reviewed in Subsection 8.2.

8.1 Free product features

Even for a property that is not a free property, it can happen that certain of its features are unaffected by constraints. In Example 2 the product property *'ImprintColor'* is no-longer a free property. However, the choice of *'Black'* as an *ImprintColor* is free, as it will always be choosable. A product feature that can never be a cause of inconsistency is called a free feature.

8.2 Brief excursion on negation

Table negation is discussed in some detail in [6]. I review it here only briefly, as far as needed to motivate the notion of a *free feature*.

Let $\mathfrak{T}$ be a variant table and let $\mathbf{D} = D_1 \times \ldots \times D_k$ (where D_j is the domain of product property v_j) be the c-tuple denoting the overall solution space. We assume that $\mathbf{D}$ contains only features referenced in $\mathfrak{T}$, i.e., that it is the smallest c-tuple that contains all features in $\mathfrak{T}$.

Let $\overline{\mathfrak{T}} := \mathbf{D} \setminus \mathfrak{T}$ be the complement of variant table $\mathfrak{T}$. $\overline{\mathfrak{T}}$ lists all invalid variants that can be formed using the features in $\mathbf{D}$. Let $\pi_j(\overline{\mathfrak{T}})$ be the set of values occurring in the j-th column of $\overline{\mathfrak{T}}$. $\pi(\overline{\mathfrak{T}}) := \pi_1(\overline{\mathfrak{T}}) \times \ldots \times \pi_k(\overline{\mathfrak{T}})$ is the smallest c-tuple that contains all features in $\overline{\mathfrak{T}}$.

Let $\overline{D_j} := D_j \setminus \pi_j(\overline{\mathfrak{T}})$. $\pi(\overline{\mathfrak{T}})$, and $\overline{D_j}$ are well-defined, i.e., they do not depend on heuristics. Define $\mathfrak{Q}$ as the set of valid variants that are outside of $\pi(\overline{\mathfrak{T}})$. Let us consider only cases where $\pi(\overline{\mathfrak{T}})$ is smaller than $\mathbf{D}$, i.e., $\mathfrak{Q}$ is non-empty:

$$\mathfrak{Q} := \mathbf{D} \setminus \pi(\overline{\mathfrak{T}}) \neq \emptyset \tag{11}$$

It is shown in [6] Proposition 3 that $\mathfrak{Q}$ in (11) can be decomposed into k c-tuples and the following c-tuple (12) lies in $\mathfrak{Q}$, i.e., will always contain only valid variants:

$$\overline{D_1} \times D_2 \times D_3 \times \ldots \times D_k \tag{12}$$

This means that all features in $\overline{D_1}$ are free features. However, as any product property can be chosen as v_1, the features in $\overline{D_j} := D_j \setminus \pi_j(\overline{\mathfrak{T}})$ are free for any product property. Thus, negation is one way of determining free features.

9 Complexity results for the T-shirt examples

Table 6 lists the number of variants, the associated number of c-tuples, and the complexity *cplx* (number of VDD nodes[24]) for the examples in Section 2. As stated, the compression to VDDs and/or c-tuples depends on heuristics. The variant tables for the examples are too large to apply the usual preferred decomposition heuristic explicitly in my current implementation. They were constructed directly from a compressed c-tuple representation and should be taken

[24] We could also give the complexity in guaranteed run-time. For sake of argument, we might suppose one microsecond per node. Then all listed complexities would translate to less than one millisecond.

as "hand crafted"[25]. Nevertheless, they give an indication of what can be expected. Further optimization can only produce smaller VDDs for the examples.

Table 6: Complexity results for examples

Table	#Variants	#C-tuples	cplx
Example 1	120324096	1	152
Example 2	119439360	2	172
Example 3	118849536	3	262
Example 4	85934080	6	312
Example 5	83228672	18	936

10 Business considerations

How product individualization may affect a business is discussed below. Also, we observe a tendency of businesses to express product data, including constraints, in tabular form. This requires being able to deal with properties with non-finite domains. I propose to extend variant table domains to be *quasi-finite* (10.2).

10.1 Business complexity in mass customization

We assume that the business is producing, delivering, and invoicing the individualized product variant $\vec{p}$, based on the procurement of suitable components. If a procured item $\vec{i}$ is itself *individualized*, a corresponding individualized specification must be attached to the *procurement order* for $\vec{p}$, and $\vec{i}$ must be linked to $\vec{p}$ throughout the production process.[26]

If the overall variants can be subdivided into only a few c-tuples, each of which is unconstrained, then this may be an indication to also structure production that way. In other cases, production faces a configuration problem of its own. In the case of the T-shirt this problem might be identical to the original sales configuration problem. However, production often involves additional features (e.g., *FabricDye* and *PrintDye*) with constraints of their own. We call such an additional configuration a (*manufacturing*) *completion*. The complexity analysis would apply to these *completions* in a like manner.

10.2 Quasi-finite solution spaces

Several extensions to the tabular paradigm are already in common informal use and/or are already supported by proprietary environments like the *SAP Variant Configurator* [5]:

- compression of Cartesian sets of valid variants to *c-tuples*,
- numeric intervals in table cells with and without an underlying finite domain, and
- wildcard symbols[27] in table cells with and without an underlying finite domain

Standardizing the definition of a variant table to include the above capabilities is desirable to allow transparent and non-proprietary product modeling, which is important to businesses. (A configurator would then be expected to deal with such tables.) I propose to call the resulting domains *quasi-finite*, as they are a controlled relaxation of the usual finiteness requirement.[28]

[25] VDD construction directly from c-tuples taking free properties into account is current work in progress.

[26] The problem of inter-business communication about product models is a challenging topic, out of scope here. The complexity of managing the link between $\vec{i}$ and $\vec{p}$ is not considered here, either.

[27] This generalizes to Boolean predicates.

11 Summary and Outlook

The main observation in this work is that the compressibility of a variant table is also a reasonable measure of the product complexity, both from the configuration and from the business viewpoint. Further insights are the notions of *free properties* and *free features*. A feature that can no-longer lead to a conflict in a given configuration state is a free feature in that state. If a property domain consists only of free features, this is a free property. When all variants in the remaining configuration state are valid, then all properties are free: the remaining problem is unconstrained. Dividing a variant table into q disjoint c-tuples is diving the problem into q unconstrained sub problems.

A current focus of work is to investigate how free properties and free features might be used to improve the configurator experience. It also seems promising to investigate, whether free properties and free features can be used to influence the compression heuristics themselves. This is ongoing work. As are alternate ways of determining the free features (beyond negation).

The actual complexity faced by a business is not yet well-understood. I proposed linking business complexity to the variant table complexity via some additional property coefficients and weights for the individual features. But, this requires verification in the field. As would any kind of guidance on reasonable limits on complexity that should be observed, both from the point of view of maintainability by a product manager and from useful deployment in the business.

Finally, in order to match practical expectations, I suggested relaxing finite table domains to *quasi-finite* domains, allowing both numeric intervals and wild cards. Mass customization businesses would also benefit by more standardization of modeling capabilities, particularly concerning variant tables, their expressiveness, and exchange formats in compressed form.

ACKNOWLEDGEMENTS

I would like to thank my daughter Laura and the reviewers for their comments, which helped improve this paper considerably

REFERENCES

[1] Henrik Reif Andersen, Tarik Hadzic, John N. Hooker, and Peter Tiedemann, 'A constraint store based on multivalued decision diagrams', in *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, ed., Christian Bessiere, volume 4741 of *Lecture Notes in Computer Science*, pp. 118–132. Springer, (2007).

[2] H.R. Andersen, T. Hadzic, and D. Pisinger, 'Interactive cost configuration over decision diagrams', *J. Artif. Intell. Res. (JAIR)*, **37**, 99–139, (2010).

[3] Rüdiger Berndt, *Decision diagrams for the verification of consistency in automotive product data*, Hochschulschrift, dissertation, thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2016. Zusammenfassung in deutscher Sprache.

[4] Rüdiger Berndt, Peter Bazan, Kai-Steffen Jens Hielscher, Reinhard German, and Martin Lukasiewycz, 'Multi-valued decision diagrams for the verification of consistency in automotive product data', in *2012 12th International Conference on Quality Software, Xi'an, Shaanxi, China, August 27-29, 2012*, eds., Antony Tang and Henry Muccini, pp. 189–192. IEEE, (2012).

[5] U. Blumöhr, M. Münch, and M. Ukalovic, *Variant Configuration with SAP, second edition*, SAP Press, Galileo Press, 2012.

[28] Support for quasi-finite domains in VDDs is described in [6, 8], although the term *quasi-finite* is not itself used there yet.

[6] Albert Haag, 'Arc consistency with negative variant tables', In Tiihonen et al. [11], pp. 81–87.
[7] Albert Haag, 'Column oriented compilation of variant tables', In Tiihonen et al. [11], pp. 89–96.
[8] Albert Haag, 'Managing variants of a personalized product', *Journal of Intelligent Information Systems*, 1–28, (2016).
[9] Tarik Hadzic, 'A bdd-based approach to interactive configuration', in *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, ed., Mark Wallace, volume 3258 of *Lecture Notes in Computer Science*, p. 797. Springer, (2004).
[10] D.E. Knuth, *The Art of Computer Programming*, volume 4A Combinatorial Algorithms Part 1, chapter Binary Decision Diagrams, 202–280, Pearson Education, Boston, 2011.
[11] Juha Tiihonen, Andreas A. Falkner, and Tomas Axling, eds. *Proceedings of the 17th International Configuration Workshop, Vienna, Austria, September 10-11, 2015*, volume 1453 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.

ICONIC
Interactive CONstraint-based Configuration

Élise Vareilles[1] and **Hélène Fargier**[2] and **Michel Aldanondo**[1] and **Paul Gaborit**[1]

Abstract. Constraint satisfaction problems or CSP are very often used to formalize product configuration problems in both research and industry. CSP formalize relevant knowledge through variables, each one associated to a definition domain, linked by constraints, limiting the combinations of their permissible values. Thus, CSP makes it possible to describe exhaustively the solution space, corresponding to a set of all possible products. Two different methods of processing CSP allow to exploit the generic models in an interactive way: problem filtering methods (reasoning directly on the CSP network and removing inconsistent values) and solution filtering methods (reasoning on a representation of the solution space in the form of a compiled graph). Both of the methods have advantages and drawbacks in online product configuration. This paper aims at putting the first ideas on the joint use of these two methods in the same interactive configuration problem.

1 Introduction

Who has never wanted to own a particular product, such as shoes, smart-phone, cosmetic, car, etc., specially designed for him/her, perfectly suited to his/her desires, and affordable ? For several decades now, customers want to bring a personal touch to their products to make them special and unique. To meet this demand of personalization, companies nowadays no longer only offer standard products, but more and more personalizable ones. Thanks to the Web technologies, this personalization is done directly and interactively online. Customers can play with the range of choices and options offered by companies: they can assemble, cut, color, choose, ... visualize the result of their desires and ultimately order it, in a few minutes with a few clicks.

Enabling consumers or customers to personalize their products (glasses, shoes, computers, cars, etc.) is one of the current concerns of companies, whatever their size or activity sector. From the consumers' point of view, this customization has to be simple and fast (a few clicks on a web-page) while allowing them to obtain the product corresponding to their desires and their budget. From a business perspective, this customization is based on the definition of configurable products, represented by catalogs of predefined components and their relationships, as well as the implementation of interactive configuration systems or configurators.

Consequently, configuration systems have to cope with high combinatorial problems. To do so, they exploit a generic model [37] [34] [35] gathering knowledge about:

- customers requirements and desires on product definition,
- product components including their compatibility and / or incompatibilities (defining the generic bill-of-material for a product family),
- product production or manufacturing process.

Constraint satisfaction problems or CSP are very often used to formalize product configuration problems in both research and industry [7] [19]. CSP formalize relevant knowledge through variables, each one associated to a definition domain, linked by constraints, limiting the combinations of their permissible values [29]. Thus, CSP makes it possible to describe exhaustively the configuration problem and its solution space, corresponding to a set of all possible products. The users interact with the configuration system by progressively giving a value to or limiting the domain of the variables of their choice until all the variables have a unique value and the product is completely configured. The job of the configuration system is to guarantee that:

1. all choices are consistent with each other, at each step of the configuration process,
2. they can lead to the configuration of the desirable product and
3. the relevant indicators, such as price, delivery time and so on, are maintained up-to-date.

Two different methods of processing CSP allow to exploit the generic models in an interactive way:

- problem filtering methods which reason directly on the CSP network and remove inconsistent values. These methods use the constraints to make deductions on the problem by detecting locally inconsistent values. They guarantee interaction with the users but not the withdrawal of all values leading to non-solutions (they are therefore "incomplete").
- solution filtering methods which reason on a representation of the solution space in a form of a compiled graph. This compilation takes place off-line before the query phase, which relaxes the constraints on their temporal complexity but repels the difficulty in space: the compiled form can have an exponential theoretical spatial complexity.

Knowing this context, the aim of this article is therefore to give the first idea on a joint use of filtering and compilation methods in the same configuration problem to exploit the best of both approaches and mitigate their identified limits in the interactive product configuration problems.

The paper is divided as follows. In section 2, the motivation of our proposal, the constraint background and a simple illustrative example are presented. In sections 3 and 4, a focus is made respectively on problem filtering methods and solution filtering methods as well as

[1] Université de Toulouse, Mines d'Albi, Route de Teillet Campus Jarlard, 81013 Albi Cedex 09, France, email: firstname.lastname@mines-albi.fr
[2] Université de Toulouse, IRIT, avenue de l'étudiant, 31400 Toulouse, France, email: fargier@irit.fr

their their advantages and disadvantages in product configuration. In section 5, the main idea of the proposed hybrid method is exposed as well as some discussions.

2 Motivation, Background and Example

In interactive configuration problems, it is the user and not the machine that solves a combinatorial problem of optimizing preferences. Product configuration problems are ones of the typical examples (interactive configuration of a car, of a computer, etc.). By allowing the customers to explore the solution space, the online configuration system allows them to maximize their satisfaction.

Constraint satisfaction problems or CSP are very often used in product configuration problems, both in research and in industry [7] [19]. Many authors such as [40], [34], [27] have shown that configuration could be efficiently modeled and aided when considered as a CSP (Constraints Satisfaction Problem). A CSP is a triplet $\{\mathcal{X}, \mathcal{D}, \mathcal{C}\}$ where $\mathcal{X}$ is a set of variables, $\mathcal{D}$ is a set of finite domains (one for each variable) and $\mathcal{C}$ a set of constraints linking the variables [29]. The variables can be either discrete or continuous. The constraints can either be of compatibility, when defining the possible or forbidden combinations of values for a set of variables (lists of compatible values, mathematical expressions), or of activity, when allowing the activation of a subset of variables and constraints [27] [20] [37].

The constraint-based modeling makes it possible to easily formalize the generic product family by a set of variables, each one associated to its definition domain and linked to the others by constraints that limit the combinations of allowed values. Constraints make it possible to describe exhaustively the solution space, i.e. the set of possible products. CSP have several advantages:

- a great freedom of knowledge modeling: compatibility between components, mathematical formulas for product evaluation, activation of optional components, etc.,
- non-orientation of reasoning: any variable present in the problem is both an input variable (which may be restricted by the user) or an output variable (resulting from a calculation, for example). It is therefore quite possible to constrain the price and then identify all the corresponding products,
- a clear separation between knowledge models and their exploitation (algorithmic processing),
- the possibility of combinations with other knowledge based approaches, and more particularly with case-based reasoning or CBR [26] [22] [1] [41], data-mining [18] [21] [2] and ontologies [38] [30] [39].

Constraint-based configuration systems allow to the users to browse the CSP solution space (the set of all possible products) by offering them the ability to:

- visualize the current solution being configured in a relevant way by presenting only the components, variants and options which are actually part of the solution,
- express preferences on components, variants, and options, such as selecting a single value, choosing a set of values, excluding a set of values, or expressing explicitly preferences explicitly between values,
- estimate the current solution being configured according to several criteria, sometimes antagonistic, such as its cost, performance or delivery time,
- express constraints on evaluation criteria, such as limiting the cost / performance / delivery time of the current solution, which limit the choice of options and variants, or optimize solutions on one of these criteria [32] [33].

Technically, product configuration is an iterative process of removing solutions (products or components) from the solution space that are no longer consistent with the choices made by a user (typically, the potential customer) and the generic model. Through an iterative process, the user gradually specifies his/her needs and gradually converges towards a solution or a set of solutions satisfying his/her needs and desires.

Two concurrent methods of CSP processing allow us to reason on the generic model interactively: the methods reasoning on the problem described by a network of constrained variables (object of section 3), and those reasoning on a solution space represented as a compiled graph (object of section 4).

We illustrate our proposal on a very simple example of car configuration, coming from [4] and presented in Fig. 1.

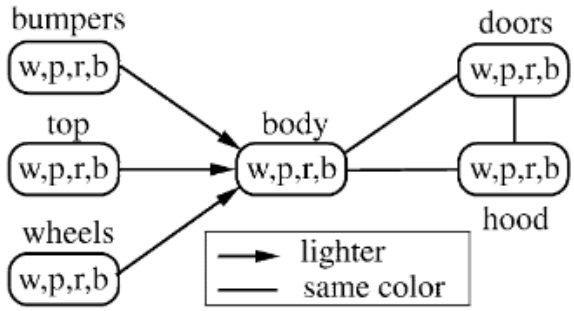


Figure 1. Car configuration problem [4]

This very simple example is composed of:

- six components: $\mathcal{X}$ = {*bumpers*, *top*, *wheels*, *wheels*, *body*, *hood*, *doors*} with all the variable sharing the same initial domain: $\mathcal{D}$ = { white, pink, red, black }
- six constraints of $\mathcal{C}$:
  - $V(C_1) = \{body, doors\}$ with $\{(white, white), (pink, pink), (red, red), (black, black)\}$ as allowed combinations,
  - $V(C_2) = \{hood, doors\}$ with $\{(white, white), (pink, pink), (red, red), (black, black)\}$ as allowed combinations,
  - $V(C_3) = \{body, hood\}$ with$\{(white, white), (pink, pink), (red, red), (black, black)\}$ as allowed combinations,
  - $V(C_4) = \{bumpers, body\}$ with $\{(white, pink), (white, red), (white, black), (pink, red), (pink, black), (red, black)\}$ as allowed combinations,
  - $V(C_5) = \{top, body\}$ with $\{(white, pink), (white, red), (white, black), (pink, red), (pink, black), (red, black)\}$ as allowed combinations,
  - $V(C_6) = \{wheels, body\}$ with $\{(white, pink), (white, red), (white, black), (pink, red), (pink, black), (red, black)\}$ as allowed combinations.

3 Problem Filtering Methods

The problem filtering methods use the constraints locally to detect the values which are no more consistent with the current problem.

One of the most widely used methods is the one of arc consistency [29]. Arc consistency verifies that any value d of a domain $\mathcal{D}$ of a variable v of $\mathcal{V}$ is compatible with each constraint taken one by one. Dedicated filtering methods based on the arc consistency exist for each type of CSP: k consistency techniques (arc consistency and path consistency) used mainly for discrete or mixed CSP [24] [11] [13], arc continuous consistency [16], 2B-consistency [23] or Box-consistency [10] [9] for continuous CSP.

All the difficulty of the problem filtering methods lies in the fact that a perfect filtering is generally an NP-complete problem. Therefore, CSP filtering algorithms are limited to local, approximate, but polynomial reasoning. In interactive configuration, the use of problem filtering methods ensures the interactivity with users but does not guarantee the pruning of all values leading to non-solutions.

- Advantages of problem filtering methods:
  Reasoning on any type of configuration problem (discrete, continuous or mixed with or optional variables, etc.).
- Limitations of problem filtering methods:
  Probable conservation of not realizable solutions and use of the backtrack mechanism to restore coherence.

Let's have a look at the use of problem filtering methods on our simple example. The corresponding constraints network is presented in Fig. 2.

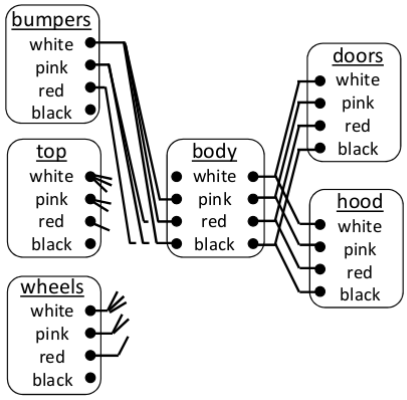


Figure 2. Car configuration problem Network

On such an example, there is no problem of filtering: all the inconsistent values are easily removed as the CSP is discrete and quite simple. But on much more complex configuration problem with discrete and continuous variables, the pruning of all values leading to non-solutions cannot be guaranty [12].

4 Solution Filtering Methods

The solution filtering methods are based on the transformation, by compilation, of a formalized configuration problem such as a CSP into a finite state automaton which therefore exhaustively represents the space of solutions [42]. This type of method has the advantage of avoiding subsequent backtracks after the compilation of the automaton and solves the filtering quality problem [5].

The compilation of a discrete problem into a finite state automaton is NP-complete, but it is done off-line, not online as it is for problem filtering methods. It has already proved its relevance and efficiency on actual industrial applications [5]. The online use of the compiled automaton guarantees that the users' choices lead to a solution in linear time which is completely consistent with interactivity in configuration.

- Advantages of solution filtering methods:
  Guarantee of achieving a solution without backtracks.
- Limitations of solution filtering methods:
  Inability to efficiently compile certain types of problems (especially with continuous variables) and spatial explosion on problems with optional variables.

Let's have a look at the use of solution filtering methods on our simple example. The corresponding automaton is presented in Fig. 3.

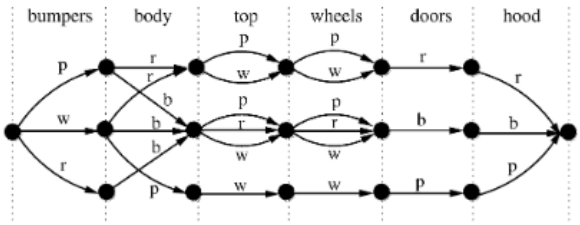


Figure 3. Car configuration problem Network

On such an example, there is no problem of compilation: the automaton is very quickly generated. But on much more complex configuration problem with discrete and continuous variables, it can be very difficult, and sometimes impossible, to build the finite state automaton.

5 Hybrid Proposed Method & Discussions

Product configuration is a topic that emerged some twenty years ago and which is developing significantly at the present time, driven by industrial applications and scientific results [7] [19].

With respect to constraint-based configuration problems, problem filtering methods [29] [12] have already proved their worth for discrete problems (core of many configuration systems, including those of ILOG, Pros); The problem filtering methods have been recently enriched (global inverse consistency [12], filtering and alternative values [8]). Nowadays, the main difficulty is to efficiently integrate continuous variables without discretizing their domain (essentially filtering by bound-consistency [23] [9] [16]) and secondly, to effectively take into account optional components without adding a specific value in domains [3] [43] [25].

The solution filtering methods [5] have also been extended by taking into account price information, which the problem filtering methods have difficulty to handle [6], by developing particularly efficient data structures [15], and finally by the definition of methods of learning user's preferences on compiled structures [14]). Some works have extended the solution filtering methods to continuous or mixed CSP [28] [31] [17], but rather on scheduling problems than on product configuration ones.

The objective of the article is to discuss on the design, development and testing of an interactive configuration algorithm that combines the problem filtering and solution filtering methods. The development and testing of this new hybrid filtering method would allow the best of both approaches to be exploited by mitigating their limits identified in interactive product configuration problems.

Scientifically, the first idea is to compile the sub-components of the product family and to use problem filtering algorithms to propagate users' choices from one automaton to another. This first joint use could avoid the pitfall of the spatial explosion due to optional components. The second working line deals with a proposed data structures more suited to continuous variables than the conventional automatons and exceeding the bound consistency filtering approaches. Rather than adapting the structures designed for discrete variables to continuous ones (which is possible but ineffective by discretizating the continuous domains), the approaches resulting from the work on continuous domains could be used, such as Q-trees and R-trees [36].

Compared to existing work, the proposed hybrid method is therefore both logical and innovative: how to combine problem filtering and solution filtering methods together in the same configuration problem. The future results seem very promising for interactive configuration problems.

A PhD subject is actually waiting for a good candidate and future PhD student. The PhD thesis is conducted between two teams: the ORKID research team of the Industrial Engineering Lab of Mines Albi France and the ADRIA research team of IRIT Toulouse France. All the proposals will be validated on several industrial cases mainly from the automotive sector.

REFERENCES

[1] A. Aamodt and E. Plaza, 'Case-based reasoning: Foundational issues, methodological variations, and system approaches', *AI Commun.*, **7**(1), 39–59, (March 1994).

[2] R. Agrawal, T. Imieliński, and A. Swami, 'Mining association rules between sets of items in large databases', *SIGMOD Rec.*, **22**(2), 207–216, (June 1993).

[3] J. Amilhastre, *Reprsentation par automate d'ensemble de solutions de problmes de satisfaction de contraintes*, Ph.D. dissertation, Universit de Montpellier, 1999.

[4] J. Amilhastre, H. Fargier, and P. Marquis, 'Consistency restoration and explanations in dynamic CSPs-Application to configuration', *Artificial Intelligence*, **135**(1-2), 199–234, (février 2002). bb modif 28/02/02.

[5] J. Amilhastre, H. Fargier, and P. Marquis, 'Consistency restoration and explanations in dynamic csps application to configuration', *Artificial Intelligence*, **135**(1 2), 199 – 234, (2002).

[6] H.R. Andersen, T. Hadzic, and D. Pisinger, 'Interactive cost configuration over decision diagrams', *CoRR*, **abs/1401.3830**, (2014).

[7] J.M. Astesana, L. Cosserat, and H. Fargier, 'Constraint-based vehicle configuration : a case study (regular paper)', in *International Conference on Tools with Artificial Intelligence (ICTAI), Arras, 27/10/2010-29/10/2010*, pp. 0–1, http://www.computer.org, (octobre 2010). IEEE Computer Society.

[8] C. Becker and H. Fargier, 'Maintaining alternative values in constraint-based configuration', in *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pp. 454–460, (2013).

[9] F. Benhamou, 'Heterogeneous constraint programming', in *5th international conference on algebraic and logic programming*, ed., Springer Verlag, pp. 62–76, Aachen, Allemagne, (Mars 1996).

[10] F. Benhamou, D. Mc Allester, and P. Van Hentenryck, 'Clp(intervals) revisited', in *ILPS'94*, pp. 1–21, (1994).

[11] C. Bessière and M. Cordier, 'Arc-consistency and arc-consistency again', in *AAAI*, pp. 108–113, Cambridge MA, (1993).

[12] C. Bessière, H. Fargier, and C. Lecoutre, 'Computing and restoring global inverse consistency in interactive constraint satisfaction', *Artificial Intelligence*, **vol. 241**, pp. 153–169, (September).

[13] C. Bessière and J. Régin, 'An arc-consistency algorithm optimal in the number of constraint checks', in *ECAI Workshop on Constraint Processing*, (1994).

[14] A. Choi, G. Van den Broeck, and A. Darwiche, 'Tractable learning for structured probability spaces: A case study in learning preference distributions.', in *IJCAI*, volume 15, pp. 2861–2868, (2015).

[15] A. Darwiche, 'Sdd: A new canonical representation of propositional knowledge bases.', in *IJCAI*, ed., Toby Walsh, pp. 819–826. IJCAI/AAAI, (2011).

[16] B. Faltings, 'Arc consistency for continuous variables', in *Artificial Intelligence*, volume 65, pp. 363–376, (1994).

[17] H. Fargier, F. Maris, and V. Roger, 'Temporal Constraint Satisfaction Problems and Difference Decision Diagrams: A Compilation Map. (regular paper)', in *IEEE International Conference on Tools with Artificial Intelligence, Vietry sul mare, Italie, 09/11/2015-11/11/2015*, pp. 429–436, http://www.ieee.org/, (novembre 2015). IEEE.

[18] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, *Advances in Knowledge Discovery and Data Mining*, American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[19] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, *Knowledge-based Configuration: From Research to Business Cases*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1 edn., 2014.

[20] E. Gelle and R. Weigel, 'Interactive configuration based on incremental constraint satisfaction', in *IFIP*, pp. 117–126, (1995).

[21] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edn., 2011.

[22] J. Kolodner, *Case-based Reasoning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[23] O. Lhomme, 'Consistency techniques for numeric CSP', in *International Joint Conference on Artificial Intelligence*, pp. 232–238, Chambéry, France, (8 1993).

[24] A.K. Mackworth, 'Consistency in networks of relations', in *Artificial Intelligence*, volume 8(1), pp. 99–118, (1977).

[25] K. McDonald and P. Prosser, 'A case study of constraint programming for configuration problems', Technical report, APES Research Group, (2002). APES-45-2002.

[26] M. Minsky, 'A framework for representing knowledge', Technical report, Cambridge, MA, USA, (1974).

[27] S. Mittal and B. Falkenhainer, 'Dynamic constraint satisfaction problems', in *AAAI*, pp. 25–32, Boston, US, (1990).

[28] J.B. Møller, Jakob Lichtenberg, H.R. Andersen, and H. Hulgaard, 'Difference decision diagrams', in *Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings*, pp. 111–125, (1999).

[29] U. Montanari, 'Networks of constraints: fundamental properties and application to picture processing', in *Information sciences*, volume 7, pp. 95–132, (1974).

[30] B. Neumann, 'Configuration expert systems: a case study and tutorial', in *SGAICO Conference on Artificial Intelligence in Manufacturing, Assemblu=y and Robotics*, pp. 27–68, Munich, Germany, (1988).

[31] A. Niveau, H. Fargier, C. Pralet, and G. Verfaillie, 'Knowledge Compilation Using Interval Automata and Applications to Planning (regular paper)', in *European Conference on Artificial Intelligence (ECAI), Lisboa, 16/08/2010-20/08/2010*, pp. 459–464, http://www.iospress.nl/, (aot 2010). IOS Press.

[32] P. Pitiot, M. Aldanondo, and E. Vareilles, 'Concurrent product configuration and process planning: Some optimization experimental results', *Computers in Industry*, **65**(4), 610–621, (2014). *WoS**.

[33] P. Pitiot, M. Aldanondo, E. Vareilles, P. Gaborit, M. Djefel, and S. Carboneel, 'Concurrent product configuration and process planning, towards an approach combining interactivity and optimality', *International Journal of Production Research*, **51**(2), 524–541, (2013). *WoS**.

[34] D. Sabin and E.C. Freuder, 'Configuration as composite constraint satisfaction', in *Artificial Intelligence and Manufacturing Research Planning Workshop*, pp. 153–161, (1996).

[35] D. Sabin and R. Weigel, 'Product configuration frameworks - a survey', *IEEE Intelligent System and their Applications*, (1998).

[36] J. Sam-Haroud, *Constraint consistency techniques for continuous domains*, Ph.D. dissertation, Ecole Polytechnique Fdrale de Lausanne, 1995.

[37] T. Soininen and E. Gelle, 'Dynamic constraint satisfaction in configuration', in *American Association for Artificial Intelligence, Workshop on Configuration*, Orlando, US, (1999).

[38] T. Soininen, J. Tiihonen, T. Mannisto, and R. Sulonene, 'Towards a general ontology of configuration', *AI EDAM-ARTIFICIAL INTELLIGENCE FOR ENGINEERING DESIGN ANALYSIS AND MANUFACTURING*, **12**(4), 357–372, (1998).

[39] S. Staab and R. Studer, *Handbook on Ontologies*, Springer Publishing

Company, Incorporated, 2nd edn., 2009.
[40] E. Tsang, 'Foundations of constraints satisfaction', in *Academic Press*, London, (1993).
[41] E. Vareilles, M. Aldanondo, A. Codet De Boisse, T. Coudert, P. Gaborit, and L. Geneste, 'How to take into account general and contextual knowledge for interactive aiding design: Towards the coupling of CSP and CBR approaches', *Engineering Applications of Artificial Intelligence*, **vol. 25**, pp. 31 – 47, (2012).
[42] N.R. Vempaty, 'Solving constraint satisfaction problems using finite state automata', in *Swartout*, pp. 453–458, (1992).
[43] M. Veron, *Modlisation et rsolution du problme de configuration industrielle : utilisation des techniques de satisfaction de contraintes*, Thse de doctorat, Institut National Polytechnique de Toulouse, France, 2001.

Features of 3D graphics in sales configuration

Petri Helo[1] and **Sami Kyllönen**[2] and **Samuli Pylkkönen**[2]

Abstract. Advanced 3D graphics is becoming increasingly important part of sales configuration systems. Availability of high performance graphics processing enables distribution of product models in web browsers and AR/VR platforms. The challenge is that in addition to traditional sales, product, manufacturing, and documentation related rules, handling 3D graphical objects present some new requirements for the configuration task. This paper discusses the modelling of 3D sales configuration. Finally, a framework for 3D graphics based configuration is presented.

1 INTRODUCTION

Product configuration systems [1] are generic purpose software packages aiming to solve the configuration task by using a combination of rule processing, CSP and object-oriented modelling. The new challenges include web based 3D graphics of product visualization [2], seamless integration to supply chain [3], support for collaborative configuration [4], and use of cloud based technologies [5].

Today, three-dimensional graphics is becoming an essential part of sales configuration systems or CPQ (Configure Price Quote) systems. The ability to visualize complex product systems is an important aspect in sales communication. Engineering related processes of complex products require consideration of visualization and use of 3D graphics [6].

For configuration modelling task the demand to apply extensive visualisations in the process gives additional work and considerations. Based on user selections, a product configuration system should produce multiple outcomes for example, a valid product variant, bill-of-variants, pricing, visualizations or even immersive 3D models. Very often the development of configuration models and graphical models are conducted by different individuals and departments. Integration of the models require careful planning.

The purpose of this paper is to identify and discuss specific features, such as position and constellation related rules, viewpoints, collision detection and kinematics in the context of sales configuration modelling. From methodological perspective, a proposed framework was developed based on an interview and discussion with five persons experienced on building and maintaining product configuration models which include 3D graphics in a central role. Two case examples of actual configuration system implementations are presented to show a comparison how 3D can be handled. The result is by no means complete, but aims to open discussion what type of challenges 3D graphics based product configuration models are presenting.

Three main elements were identified to be specific characteristics and demanding in use of 3D graphics by the modellers: (1) systems level configuration modelling, (2) graphics related rules and constraints, (3) requirements coming from immersive technologies. The following part discusses these aspects.

2 SYSTEMS LEVEL CONFIGURATION

Many 3D configurators aim to capture modelling of larger and more complex systems. Instead of modelling an instance of a product family, the focus of modelling is on the system level which comprises of several product variant instances sharing the same environment and interacting with each other [6].

Examples of system level configurations could include modelling of buildings, power plants, ships. Number and type of product instances are defined by higher level configuration parameters and rules. For example, produced power parameter may define the number and types of generators needed. On a product configurator level, separate configuration rules may be applied.

This type of modelling requires object-oriented approach. Product variant instances are encapsulated objects, which can be added (constructor) or removed (destructor) from the common space, objects may be communicate by using methods; some methods may be public for others to interact or restricted to be used only within the product model (private).

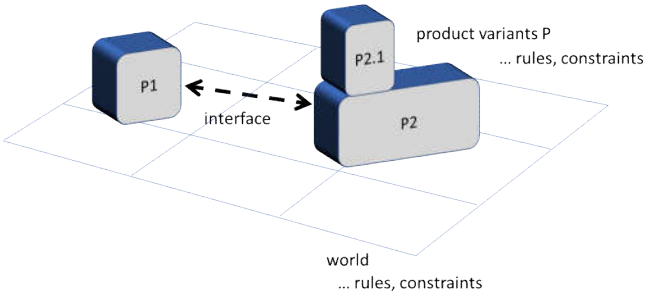


Figure 1. Configuration system consisting of configurable product families P1, P2, and P3 in a 3D world.

Another typical systems level configuration feature is the engineer-to-order process (ETO). ETO refers to configuration where product structures are not completely defined initially and final configuration is completed in phases as further information is

[1] Networked Value Systems, University of Vaasa, Finland email: petri.helo@uva.fi
[2] Wapice Oy, Finland, email: sami.kyllonen@wapice.com
[2] Wapice Oy, Finland, email: samuli.pylkkonen@wapice.com

gained [6]. There are several possibilities for reasons for phased configuration:

- Incomplete or changing product parameters [6]
- Transition from customer requirements to design parameters or physical building blocks require engineering work [6]
- Interdependency caused in later phase conflicting with initial parameter values [7]
- Co-creation presents an additional challenge [8]. This could mean that there is a distributed team maintaining the system level configuration rules, but also the possibility that end users are working in a team and sharing the configuration model.

Standardization of ETO process and adaptation of mass customization principles has been reported to be an interesting topic for many project based industries including construction and machine building [9] [10] [11].

In order to support some ETO requirements, product structure modelling for reusing the elements between projects, has been presented by Brière-Côté et al [12]. Another aspect is that on system level, product lines develop and evolve during the life-cycle. Pleuss et al [13] have presented a model-driven approach to describe feature models and analyse the development paths.

Systematic processes have been introduced to support modular product family development [14] and product family architecture [15] to support mass customization and CPQ. However, systematic system level processes for managing ETO offering has not been presented.

3 GRAPHICS RELATED FEATURES

Use of graphics in configuration system is becoming quite standard. However, very often the full potential of visualization is not taken, as graphical representation is just one configuration outcome among the others. For example, Forza and Salvador [16] have presented very well the role of geometrics and graphic models in product configuration. They demonstrate how many products can be modelled by using a library of objects and graphic configuration result is produced as an output together with code, cost, BOM and other traditional outcomes. Shafiee at el [17] analysed visual representation techniques in product configuration.

Pleuss and Botterweck [18] have presented an S2T2 configurator tool to visualize the configuration possibilities and variability space. Visualization of elements and rules in modelling has been presented too. For example, Tiihonen et al [19] have used this kind of approach in WeCoTin system.

Seeing the actual visualization of the product may have interesting results for users. Deng et al [20] studied empirically how users combine colours in product configuration. They concluded that users are likely to combine colours according to visual coherence and total number of actually used colours was smaller than expected based on combinatorics.

Configuration system can be integrated from two perspectives with 3D graphics: (1) user interface may be built in 3D world and selections are done in this space; (2) the result of configuration input from traditional form type of user interface is visualized as 3D graphics. Both configuration approaches require special attention in terms of positioning, kinematics, animation and rules.

3.1 Positioning of objects

Position of objects is depending on several object properties, rules and constraints, including:

- Dimensions and shape of each product variant
- Positioning in the world coordinates (absolute position, position relatively to other objects, - left, right, top, bottom).
- Rotation directions of each product
- Objects should be aligned in correct direction with each other
- Objects should be positioned in the same edge line with each other. In case of different lengths, also possibility to align on centreline.
- Kinematic direction, for example latches and doors should open towards out from the object group centre.
- Stacking rules of objects, maximum number of stacked objects or dynamic stacking constraints

Navigation in 3D world - zooming, viewing, panning may be controlled by the following parameters:

- Product configuration may be observed from various positions, angles and distances
- Maximum distances and angles may be set for each axis
- Navigating inside of objects may be restricted
- Objects may be dragged, dropped and moved within the allowed space
- Objects may be grouped together for rules
- Taking screenshots from camera points

Auto-snap of objects – users can move objects in common space and auto-snap rules may be used:

- Objects should attach to each other if the distance is short enough.
- Objects should not be locked to each other if the distance is short enough.

Figure 2 shows an example of electrical cabinet configuration, where among traditional configuration parameters, product layout and space allocations are visualized as an output of configuration. Dynamic rules may be used to calculate resource consumption as well as discrete additions of cabinet space.

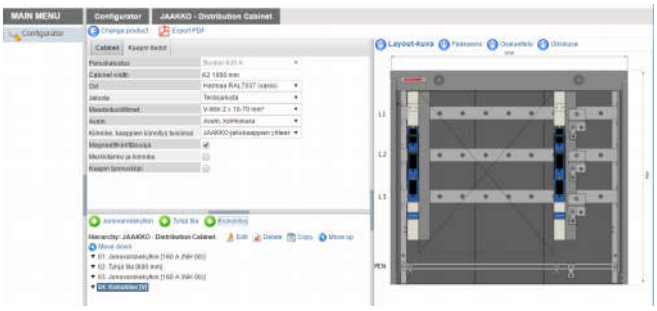


Figure 2. Electrical cabinet space allocation in sales configuration system.

3.2 Kinematics

User selections trigger processing of rules and changes in 3D graphics presentation of the product. Product variant parameter

value changes can interact with other models' interfaces. Interfaces can be physical objects or "virtual planes" within the 3D space.

Objects may be moved in relation with each other. Objects which are attached to each other may have kinematic rules. For example, one object may rotate around another object by using kinematic rule type axis, constrained by 25 degree allowed space. This kind of rules are used for example in door opening directions or lifting arms.

3.3 Animation

Animation is a method used to describe functionality, connections or location of elements of a product configuration. For presentation purposes, user may select product and objects within the configuration space and get further information about each clicked object.

Animations may be generated by using camera flights between predefined camera positions. Completed configurations may yield an animation, which visualizes the use of product configuration in a real operating environment. This outcome may be distributed part of the offer documentation as an appendix.

3.4 Rules and constraints

Features, functions and physical building blocks in sales configuration system needs to relate to 3D graphical models. In addition to interfaces between these, special rules and constraints may be introduced:

- Position and constellation related rules
- Animation – dynamic rules in terms of changing parameter values at certain times and events
- Collision detection related rules
- Dynamic rules for space, weight and pressure related resources

These rules may be related to correct visualisation only and handled in 3D model. In many cases the rules are related to product configuration side as well and should restrict users to choose impossible variants for further processing.

4 VR/AR and gamification

Virtual Reality and Augmented reality (VR/AR) technologies are used to visualize product configuration in a realistic operations environment. Virtual reality technology creates an immersive user interface for navigation and use the actual system. For configuration purposes this approach gives benefits:

(1) Augmented reality embeds 3D visualization of the product model into the real-world. Well known examples are sofa configurators where furniture layout may be embedded into a photo of a real room and house configurators where builds are visualized in real 3D space.

(2) Seeing product system in real environment is expected to improve hit rate of offers and this approach is used in applications where product configuration connects with ambient environment visually.

(3) Gamification refers to game like use of product configuration. Simulating the use of product system from operator perspective could be an example of gamification. Computer simulation could validate the behaviour of the configuration model and the performance of the system in close to actual environment.

Figure 3 presents an example of virtual reality model of a power plant, where a maintenance task may be completed in realistic environment.

Figure 3. Virtual reality model of a power plant configuration.

5 CASE EXAMPLES

Two illustrative examples of real CPQ implementations are presented: (1) a waste processing plant configuration, and (2) piping system configuration for industrial applications. Both cases have been implemented by using Summium CPQ configurator system.

5.1 Case: Plant level configuration

First example presents a model of waste processing plant, which is a system level configurable model. The geometry of products has been processed from PDM to produce "light models" for operations. The number of polygons have been reduced to increase the graphical performance. Geometry models have been linked with configurator models.

The first task is to create a production line by adding machinery and connecting these with each other. Each machine (product variant) has kinematics and connecting interfaces, which define compatibility with other products. A system level optimal solution is presented and the entire system is presented in a world view. The

next step in the process is to modify the proposed layout to support building shapes and dimensions.

Detailed configuration for each machinery can be processed at this point and finally a request is generated for CAD server API, which will process the actual visualisation. The system generates both sales related documents as well as actual CAD model of the plant. The result can be used for production and sourcing.

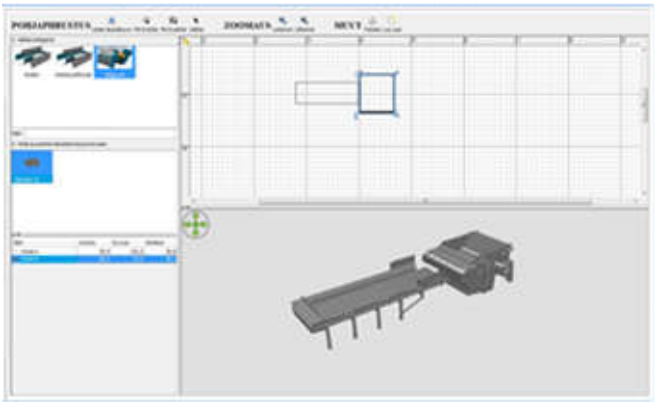


Figure 4. Layout planning and 3D visualization of building blocks.

The process steps in this structured approach are as follows:

(1) System level modelling in 3D
(2) Layout modifications in 3D
(3) Product level configuration of variants
(4) Final visualisations in CAD system

5.2 Case: Piping system configuration

The second case example is a configuration system for generating a piping system in 3D space and validation of the solution. The system is used to design a piping routing for a building or machinery. The configuration system proposes a solution. When geometry is checked and collision testing are completed, a list of bill-of-materials may be generated for processing pricing calculation and required information for the production.

The user input is the coordinate points for piping system entry and exit points, diameters of pipes and flanges. A 3D model is generated to satisfy the constraints and minimize the pipe length. The outcome is a 3D model, which may be rotated and studied visually. For each piping part, manufacturing phases may be simulated to visualize the bending process and requirements for the machinery.

The process steps in case 2 were as follows:

(1) Definition of piping connection points and pipe types
(2) System level optimized solution
(3) Visual evaluation, e.g. collision test
(4) BOM and part generation
(5) Production simulation for parts

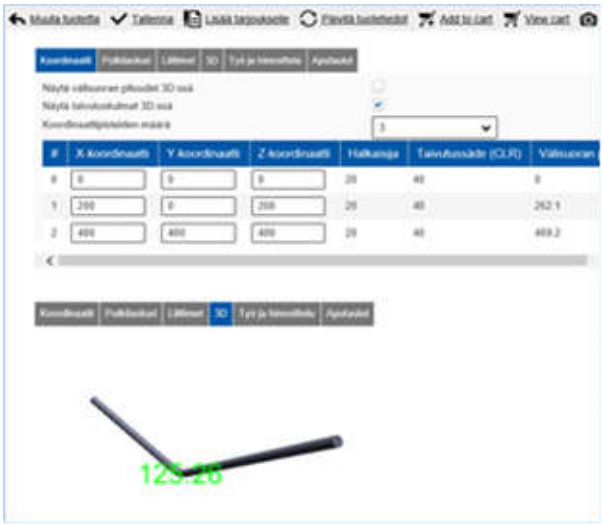


Figure 5. Generating 3D piping system based on input points.

5.3 Comparison

Two described configurator examples use 3D models as a central part of the CPQ functionality. Both product systems have system level engineering perspective, but have differences from each other (Table 1).

Table 1. Comparison of plant and piping cases.

	Case 1	**Case 2**
	Plant	Piping
World	Construction area available	Common parameters for each product variant: materials, rating
Layout	Building a line by adding machinery connected to each other, mostly 2D layout	3D space of interfaces where product variants should connect
Building blocks	Configurable machinery with connecting interfaces; independent configuration models	Parametric CAD models are the main output, each variant produces a separate file
User interactions	Visualisation of the plant, free navigation and pre-set camera points	Visual inspection, collision detection
System level constraints	Construction area, performance metrics of the plant	Spatial connections, minimizing pipe length

6 CONCLUSIONS

Use of 3D graphics can add value for many product configuration tasks especially in CPQ. As product configuration model and 3D visualization models are separate, the interfaces and process structures should be well design to support the configuration task. 3D visualisation may have three main purposes (Table 2): (1) to support input data entry in visual form, (2) to provide a playground for testing or inspection of configuration result, or (3) generate documentation for offers, orders or production.

As increasing number of users are familiar with basic 3D navigation, the use of 3D models for data entry is becoming a feasible option. Functional testing of the product variant is an application related task. It enables connecting sales phase modelling with computer aided engineering and other complex tasks. Using the 3D models or VR models to support documentation is an entry level solution.

Recent studies have shown that immersive technologies have potential. Górski et [21] developed a product configurator of a bus for product training purposes by using VR. There is also recent empirical evidence to show that gamification has impact on certain types of tasks [22], which could be relevant for CPQ systems as well.

Table 2. Functionality framework of 3D configuration systems.

User interaction	Input	Output
Adding products	Adding and moving products in 3D space	• Layout and space allocation • System level performance
Functional testing	Operational use of product or system within 3D space	• Collision detection • Checking dimensions • VR/AR • Training
Documentation generation	Configuration selections	• 3D visualisation • Offer supplements • Installation support

Table 3 summarizes the specific features of 3D configuration, which were anticipated based on discussions within the expert teams. The list probably very limited but highlights some challenges what 3D graphics set as requirements for future product configurators and CPQ systems. System level analysis of product, graphics modelling integration and immersive technologies present interesting possibilities to expand the scope of use of configurators and present challenges for product modelling task.

Table 3. Features specific for 3D configuration systems.

Perspective	Modelling considerations
System level analysis	• Interaction between components • World environment • Engineer-to-order design process
Graphics model related issues	• Positioning of objects • Kinematics • Animation • Rules and constraints
Immersive presentation and gamification	• VR/AR specific rules • Dynamic event based game rules

We believe that use of 3D models become a standard in CPQ systems. Improving the connections between the domains of sales, pricing, BOM, parts and graphical models, further work is needed especially from configuration modelling side. As each of the domains are somewhat separated from each other and experts are typically at different parts of organisations, proper models for distributed modelling and version control is needed.

REFERENCES

[1] Zhang, L. L. (2014). Product configuration: a review of the state-of-the-art and future research. International Journal of Production Research, 52(21), 6381-6398.

[2] Rolland, R., Yvain, E., Christmann, O., Loup-Escande, E., & Richir, S. (2012, March). E-commerce and Web 3D for involving the customer in the design process: the case of a gates 3D configurator. In Proceedings of the 2012 Virtual Reality International Conference (p. 25). ACM.

[3] Helo, P. T., Xu, Q. L., Kyllönen, S. J., & Jiao, R. J. (2010). Integrated Vehicle Configuration System—Connecting the domains of mass customization. Computers in Industry, 61(1), 44-52.

[4] Shamsuzzoha, A., Kyllönen, S., & Helo, P. (2009). Collaborative customized product development framework. Industrial Management & Data Systems, 109(5), 718-735.

[5] Yip, A. L., Corney, J. R., Jagadeesan, A. P., & Qin, Y. (2013, June). A product configurator for cloud manufacturing. In ASME 2013 International Manufacturing Science and Engineering Conference collocated with the 41st North American Manufacturing Research Conference (pp. V002T02A010-V002T02A010). American Society of Mechanical Engineers.

[6] Kristianto, Y., Helo, P., & Jiao, R. J. (2015). A system level product configurator for engineer-to-order supply chains. Computers in Industry, 72, 82-91.

[7] Helo, P. T. (2006). Product configuration analysis with design structure matrix. *Industrial Management & Data Systems*, *106*(7), 997-1011.

[8] Prahalad, C. K., & Ramaswamy, V. (2004). Co-creation experiences: The next practice in value creation. *Journal of interactive marketing*, *18*(3), 5-14.

[9] Haug, A., Ladeby, K., & Edwards, K. (2009). From engineer-to-order to mass customization. *Management Research News*, *32*(7), 633-644.

[10] Jensen, P., Lidelöw, H., & Olofsson, T. (2015). Product configuration in construction. *International Journal of Mass Customisation*, *5*(1), 73-92.

[11] Kristjansdottir, K., Hvam, L., Shafiee, S., & Bonev, M. (2017). Identification of Profitable Areas to Apply Product Configuration Systems in Engineer-To-Order Companies. In *Managing Complexity* (pp. 335-350). Springer International Publishing.

[12] Brière-Côté, A., Rivest, L., & Desrochers, A. (2010). Adaptive generic product structure modelling for design reuse in engineer-to-order products. *Computers in industry*, *61*(1), 53-65.

[13] Pleuss, A., Botterweck, G., Dhungana, D., Polzer, A., & Kowalewski, S. (2012). Model-driven support for product line evolution on feature level. *Journal of Systems and Software*, *85*(10), 2261-2274.

[14] Bonev, M., Hvam, L., Clarkson, J., & Maier, A. (2015). Formal computer-aided product family architecture design for mass customization. *Computers in Industry*, *74*, 58-70.

[15] Pakkanen, J., Juuti, T., & Lehtonen, T. (2016). Brownfield Process: A method for modular product family development aiming for product configuration. *Design Studies*, *45*, 210-241.

[16] Forza, C., & Salvador, F. (2006). Product information management for mass customization: connecting customer, front-office and back-office for fast and efficient customization. Springer.

[17] Shafiee, S., Kristjansdottir, K., Hvam, L., Felfernig, A., & Myrodia, A. (2016, December). Analysis of visual representation techniques for product configuration systems in industrial companies. In Industrial Engineering and Engineering Management (IEEM), 2016 IEEE International Conference on(pp. 793-797). IEEE.

[18] Pleuss, A., & Botterweck, G. (2012). Visualization of variability and configuration options. *International Journal on Software Tools for Technology Transfer*, *14*(5), 497-510.

[19] Tiihonen, J., Heiskala, M., Anderson, A., & Soininen, T. (2013). WeCoTin–A practical logic-based sales configurator. *AI Communications*, *26*(1), 99-131.

[20] Deng, X., Hui, S. K., & Hutchinson, J. W. (2010). Consumer preferences for color combinations: An empirical analysis of similarity-based color relationships. *Journal of Consumer Psychology*, *20*(4), 476-484.

[21] Górski, F., Buń, P., Wichniarek, R., Zawadzki, P., & Hamrol, A. (2015). Immersive city bus configuration system for marketing and sales education. *Procedia Computer Science*, *75*, 137-146.

[22] Hamari, J., Koivisto, J., & Sarsa, H. (2014, January). Does gamification work?--a literature review of empirical studies on gamification. In *System Sciences (HICSS), 2014 47th Hawaii International Conference on* (pp. 3025-3034). IEEE.

Increased accuracy of cost-estimation using product configuration systems

Jeppe Bredahl Rasmussen and **Lars Hvam** and **Niels Henrik Mortensen** [1]

Abstract. This article describes an approach for utilizing Product Configuration Systems (PCS) for quantifying project costs in project-based companies. It presents a case study demonstrating a method of quantifying costs in a way that makes it possible to configure cost- and time estimates. Piecework costs, material costs and sub-supplier costs are used as principle cost elements and linked to structural and process elements to facilitate configuration. The cost data are used by the PCS to generate fast and accurate cost-estimates, quotations, time estimates and cost summaries. The described cost quantification principles have been used in a Scandinavian SME (Small and Medium-sized Enterprise) since the 90's, but have since 2011 been adopted to be used in a configuration system. A longitudinal case study was conducted to compare cost and time-estimation accuracy before and after implementation. We conclude that the proposed method for grouping costs, combined with a PCS, can be used in project-based construction industries to make more accurate estimates of project costs. Reasons for improved accuracy are, according to company experts, the increased documentation and visibility of cost-estimates, dynamic allocation of variable costs, version control of cost-agreements and the ability to handle an increased level of cost details.

1 Introduction

Cost-estimation accuracy in project-based companies can be a challenge that often results in cost overruns of construction projects[1]. To respond to these challenges, a wide range of cost-estimation techniques have been developed to increase accuracy, ranging from simple estimation techniques to applied artificial intelligence. However, the most recently developed methods have not been adopted to the extent that would be expected, partly due to lack of understanding of new methods, but also by lack of trust in the benefits of such methods [2]. Product configuration systems have proven useful to improve time performance, error rates and profitability in a wide range of companies. [3–7] Some use has been made of the generation of cost-estimates by means of rule-based expert systems within the field of product configuration. Examples of cost-estimates generated by PCS are catamaran-type leisure boats in Korea [8], and optimization of the cost and scheduling of heavy earthmoving operations [9]. A PCS was developed by Chan [10] to predict the price and manufacturability of six commonly used component designs. The component designs generated by the PCS were afterwards validated by sourcing prices from real manufacturers and confirmed the reliability of predictions from the expert system[10]. Cost-estimation of metal casts has been developed by use of fuzzy reasoning systems[11]. Cost-accuracy has been reported as an observed benefit in industry by use of PCS[12]. This article is a case study investigating an implementation of a PCS to improve cost-estimation accuracy in a project-based construction company. In order to investigate the effects of a PCS, we followed a case company using a PCS to generate cost-estimates and quotations. Based on the believe that PCS can improve cost-estimation accuracy in the construction industry, the following proposition was tested:

Proposition
Implementation of PCS can improve cost-estimation accuracy in project-based construction companies

To test the propositions a collaboration with a case company that had changed from a traditional cost-estimation approach to a PCS was followed in a longitudinal case study. Access to the content of the PCS calculation principles and domain experts for clarifying questions was during the period in order to provide us with an understanding of the most important reasons behind any changes in cost-estimation accuracy. We sought to increase understanding of how a PCS adds value to a company and what reasons might be behind increases in cost-estimate accuracy. The paper is structured as follows: (1) Literature review of current cost-estimation practices in project-based industries; (2) Research methods; (3) Description of how to model cost-elements in a PCS; (4) Case describing the use of a PCS for cost-estimation, its impact on cost-accuracy and possible explanations; (5) Discussion of the results; and finally (6) Conclusions.

2 Literature review of cost-estimation techniques

Cost estimation is important to project management as it provides information for resource management, decision-making and cost scheduling [13]. Cost over-runs are a common problem in project-based companies when cost-estimates lack accuracy [1]. Numerous methods have been proposed for cost-estimation and numerous textbooks are readily available on the topic. Often the focus is on the principles and processes involved in cost estimation. The general suggestion is to break costs down into such elements as labour, materials and plant costs and add some percentage for contingency [14,15]. The process of estimation is to produce a statement of the approximate quantity of material, time and cost to perform construction work. In 1989 Carr [13] identified a need to

[1] Section of Engineering Design and Product Development, Technical Univsersity of Denmark.email: jbrras@mek.dtu.dk

establish cost-estimating principles and stipulated that a proposal in the construction industry must include an estimate that is close to reality, a suitable level of detail, and all relevant items, without adding extra and use quality documentation as a basis for business decisions. Furthermore, the cost-estimate should distinguish between direct and indirect costs and variable and fixed costs. Additionally some way of handling contingency should be in place to mitigate unforeseen circumstances [13]. Multiple methods for cost-estimation exist; they can be divided into Bottom-up and Top-down approaches. Bottom-up estimating (or resource-driven estimating) includes breaking down a project to its distinct parts in a 'work breakdown structure'. The aim is to reach a level of detail where costs are relatively stable and most costs are included. It is generally agreed that the bottom-up approach is quite accurate, but it is also time consuming [16]. Top-down (or parametric estimating) relies on past projects and reviews and modifies earlier projects by scaling and estimating expected costs [17]. Advanced methods have been developed and classified in four types of cost estimation modelling: Experience based (algorithms, heuristics, expert system programming), simulation (heuristics, experts models, decision rules), parametric (regression, Bayesian, statistical models, decision rules) and discrete state (Linear programming, classical optimization, network, PERT, CPM) [18]. Much research has been conducted within Case Based Reasoning (CBR) as it allows recall and reuse of knowledge from prior projects [19]. Rule-based experts systems have failed to meet the need that construction managers have to handle complexity and CBR has emerged as an alternative[20]. A system that integrates CBR and rule based expert systems was developed for cost-estimation of refurbishing of houses and it was concluded that the combinatorial approach is beneficial but not commercially viable due to the complexity of such an approach [21]. In practice, cost-estimation methods depend on the nature and type of organization and are not very standardized [22]. A survey of 84 very small to large firms in the UK [2] were asked about current cost estimating practices and the study concluded that the most used methods were of a relatively simple nature, such as estimation of standard procedure, comparison with past projects and comparison with finished parts projects. Intuition and simple arithmetic formulas are also widely used. Most of the advanced cost-estimation methods have not been adopted by industry. Reasons listed for lack of adaptation are that the companies are not familiar with recent methods, companies lack time and knowledge and they doubt whether the new techniques can be of benefit to the construction industry. The study also concluded that companies mainly use cost-estimation for construction planning and not for construction evaluation [2]. This article seeks to add a case to the evidence that a PCS can offer benefits to the construction industry both by offering opportunities for increased cost-accuracy, but also by making it easier to use cost-estimate data for construction evaluation.

3 Research Method

This research was based on a case study of a project based construction company that generates cost-estimates and quotation letters. It was a longitudinal case study that observed changes in cost-estimate accuracy occurring in a company that have changed from a standard cost-estimate approach to a PCS. The longitudinal case study was chosen due to the ability for the researcher to watch a changes unfold in real time [23]. Data on cost-estimations and actual costs were provided by the company. The data were analysed by researchers by comparing pre-project cost-estimations with realized costs in order to test the proposition. The cost-estimates from 2009 were generated by standard methods and those from 2014, which were generated with PCS, were compared to the actual costs of the given projects. The possible reasons for the results were investigated qualitatively in interviews with two different company experts who had both used the system and taken part in the development. The interviews were performed individually to prevent interviewees from offering the same explanation or affecting each other. Published studies of cost-estimation in construction industry were reviewed in order to identify best practices, to document that the principles used in the case company resemble current practice and to identify similar use of computer aided cost-estimation, in order to provide context.

3.1 Cost-estimation model based on configuration

The proposed principles for cost-estimation by means of PCS resemble standard cost estimation processes as described in text books on cost-estimations by breaking down cost elements into smaller cost elements such as labour, materials and plant costs [15]. This approach resembles the bottom-up approach to cost-estimation, which is believed to be accurate and complete but also time consuming [16]. The time taken for a detailed bottom-up approach is acceptable for mass-produced products and the effort invested in making detailed estimations is justified, since they can be reused. Multiple cases of knowledge based configurations of bills of materials and processing times exists in make-to-order companies [11,24,25] However, few accounts have been published of knowledge based product configuration systems designed to configure entire projects, including detailed costing information. No relevant reports were found on configuration of cost summaries in the research databases SCOPUS and Web of Science. The key words searched were "expert system" ,"configuration" ,"decision support", "reasoning system" in combination with "cost summary", "cost overview", "Cost accounting". Cost-accounting, among other activities, is used to take decisions on pricing and on the introduction of new products and discontinuing of products [26]. The detailed level of cost-information influences product cost decisions. The more complex the product the more difficult it is to include product costing feedback, so more accurate costing information provides benefits in forecasting [27]. The currently proposed cost-estimation model for projects divides cost elements into three different categories; piecework cost (salary), materials costs and subcontractor costs. The piecework costs represent the agreed cost for a worker to perform a given piece of work. The cost of having the worker perform the work corresponds to the time expenditure for a construction process. The material cost represents the costs of materials for a given project. The subcontractor costs are fixed price agreements with subcontractors to solve a given task. These costs are believed to be enough to give a complete picture of a cost-estimate and are in line with current practice [14,15]. In PCS the costs are assigned to parts or process descriptions that can be selected in the configuration system in order to configure a project. Additionally, parts and processes contain account descriptions designated according to cost-type and supplier information. The account descriptions can be used to

generate a cost summary of all expenditure in a project with a description of supplier and the expected total sum. The cost summary enables companies to compare cost-estimates with actual costs at a detailed level, with little effort. The cost-summary enables the company to use the cost-estimations for evaluation, which is currently not standard practice [2]. Evaluations of cost data and accurate cost databases are believed to be a key factor for success in the improvement of cost-estimations in building projects and firms will have to find some means of retaining the knowledge and experience from past projects [28].

4 Background of the case company

The case company in this study was a Scandinavian company that sourced construction components and provided system deliveries as service installations. The company was classed as a SME and in 2015 it had a turnover of 34 million € and approximately 130 employees. In 2015 the company bid on 1319 projects and won 229 projects which in total represents production, sourcing and assembly of 3001 individual products. The customers are typically a group of people buying installations in a community where the customers buy the product individually but share the costs of installation. The average project cost was 148.471 € and the average cost per product was 11.329 €. An average of project costs in 2013 were distributed between assembly workers (25%), materials (52%), subcontractors (11%) and additional costs for setup and removal of each construction site (12%). The ratio of expenses had not changed much since then. Since 2015 the company had used a configuration system to generate cost-estimates and quotations for projects. The projects were all deliveries of similar products, but in many customer specific variants from a few different product families. The configuration system was based on component selection with assigned salary costs, materials costs and subcontractor costs. (Section 3.2) The cost-estimation techniques used by the case company were roughly the same before and after implementation of a PCS. The main difference was in the visibility and documentation of cost-estimates, automation of changes in quotations and a slightly improved detail level in cost contributions.

4.1 Configuration of cost-elements

A schematic representation of the proposed PCS shows a system overview including user inputs, PCS knowledge and generated outputs. (Figure 1). The user inputs was an interface with a drop-down menu on which the salesman could select elements to specify product design and work process. The knowledge of the configuration system was represented by parts or processes to be selected connected with a group of cost-elements; piecework cost, material cost and subcontractor costs. Every part or process element in the configuration system could hold one or more of the cost-elements dependent on the characteristics of the chosen element, i.e. a chosen component could include information on both piecework-costs and materials costs. This was because some parts of the construction project included both a work process to be performed and a material to be used for the process. The knowledge about the processes, materials and subcontractor costs was handled in the PCS and a finite solution space could be defined and handled by an inference engine.

The PCS could handle changing project costs by adding or removing project elements according to changes in the required product and thereby easily create revisions and changes in cost-estimates and output documents. In order to handle the complexity of construction projects special open entry fields were used in the configurator with the possibility to describe non-standard elements. Non-standard elements might consist of any of the three types of costs and was a flexible way of adding non-standard process and costing knowledge. The total sum of piecework-, material- and subcontractor costs was used to generate the output of the PCS. For internal use, the case company generated time-estimates (total salary cost estimate divided by hourly fee gives an approximate assembly time) and cost summaries according to expected expenses from specific suppliers and subcontractor agreements. The cost summary helped to evaluate accuracy and identify billing mistakes. For external use, quotation letters were generated for customer, each containing a fixed price based on a configured cost-estimate. The time-estimate and the time-schedule were based on the estimated salary cost, so the accuracy of the configuration was of great importance for overall project cost accuracy. An under-estimate in salary and thus time-estimates could result in increased expenses due to overtime rent of machinery and other very variable costs.

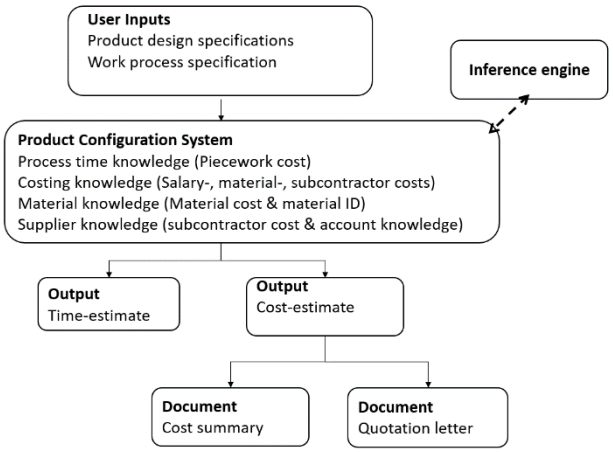


Figure 1 Overview of PCS and outputs delivered

4.2 Analysis of cost-estimate accuracy before and after implementation of a PCS

The case company performed an analysis of the cost accuracy of the major cost elements of 55 cases in 2009, corresponding to 12 months of operations, in order to review and improve the current cost-estimation process. The deviations were calculated per major cost element, as defined in (1).

$$\textbf{Cost deviation} = \textbf{Actual cost} - \textbf{Estimated cost} \quad (1)$$

If the actual cost of a project is higher than the estimated cost, the cost deviation will be negative. If the actual cost is lower than the estimated the result is a positive deviation. If a project exceeds the cost estimate it shows a negative deviation on the graph and in case of a lower price than estimated a positive deviation. In 2009 fluctuations in the deviations in cost-estimates could be observed and only few projects where completed at a cost close to the

estimation (Figure 2). It can be seen that the fluctuations move in both positive and negative directions but when deviating the different cost elements generally move in the same direction. This indicates a tendency to over-estimate or under-estimate a complete project and not just parts of it. Furthermore, the tendency is that most deviations are negative meaning that the cost-estimators most likely to have underestimated project costs when there are deviations. The conclusion from the investigation was that increased cost accuracy was identified as an area that must be improved. Based on the analysis it was decided by the case company to invest in a PCS to generate quotations, in order to improve accuracy. (Section 4.1)

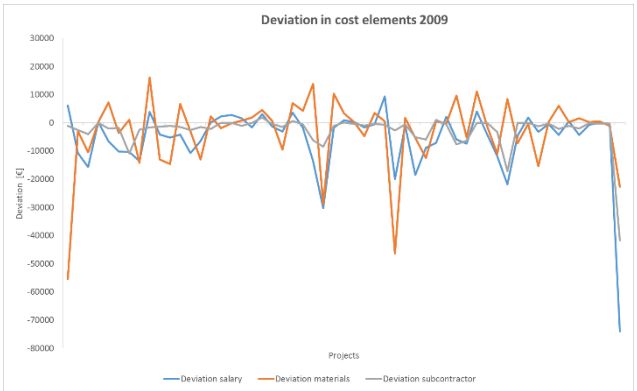


Figure 2 Deviations in cost elements 2009

In 2014 another analysis of 42 cases corresponding to 4 months of operations were performed to evaluate the effect of the PCS. Less fluctuation in the deviations of cost estimates were observed in 2014, resulting in better accuracy (Figure 3). The line had straightened around zero indicating that the deviations had been reduced. There were still three major outliers in salary and subcontractor categories. In order to understand them, expert interviews were conducted to clarify the cause. In those particular cases the company was experiencing a shortage of workers to complete the projects and was forced to complete the projects by using subcontractors. The deviations in salary and sub-contractor costs equalized each other and the consequence was therefore not negative to the company's profit.

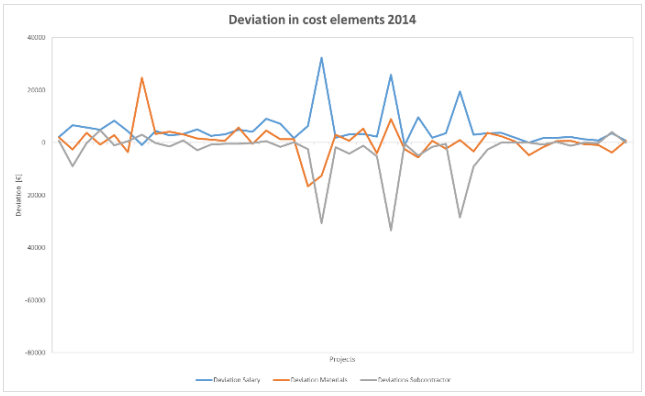


Figure 3 Deviations in cost elements 2014

An overview of the sum of the actual costs and estimated costs can be seen in Table 1. Note that the total sum of salary and subcontractor costs does not hit the target very precisely, which is related to the prior explanation of the outliers. In the rest of the article the data set has been corrected to exclude the three cases to make a better representation of the actual distribution of the deviations. From this point in the article only 39 cases are included in the 2014 analysis.

	estimated costs		actual costs	
	2009	2014	2009	2014
# projects	55	42	55	42
Sum of salary	1963	714	2224	501
Sum of materials	5004	1749	5173	1726
Sum of subcontractor	410	199	592	331
Total Sum	7377	2662	7989	2558

Table 1 Sum of total cost elements in 1000 € (2009 & 2014)

Reason for deviations in 2009 were according to the company a lack of standardized solutions, too little detail on cost elements and lack of control of expenses in relation to external use of consultants for gaining approval for products. Reasons for deviations in 2014 were according to the company late changes in the order resulting in a change in price. Positive deviations in the materials category were explained by a change in product design resulting in a positive deviation due to a lower final price.

4.4.1 Comparison of individual cost elements accuracy

All of the cost-element deviations were plotted in a column diagram and rank-ordered from the greatest negative deviation to the greatest positive deviation on identical scales per cost-element. A reduction in under-estimated cases was observed across all cost elements in 2014. Most notable are the salary and materials estimates, which showed substantial reductions in under-estimates. The subcontractor category still suffered from a tendency to underestimate costs.

4.4.2 Comparison of salary costs

In 2009 the deviations were significantly more likely to be negative (39 negative projects) than the estimates made supported by a PCS in 2014 with 3 negative projects (Figure 4). In 2014 deviations continued to occur but with positive deviations and with a significantly smaller magnitude. The greatest negative deviation in 2009 was approximately 75.000 €, while the greatest negative deviation in 2014 was approximately 1.000 €. This is a significant difference in miscalculations and of great importance to the profitability of the case company, as it will help to avoid losing money, but also to calculate correct time-schedules and subcontractor costs that are dependent on the number of days needed to complete the work.

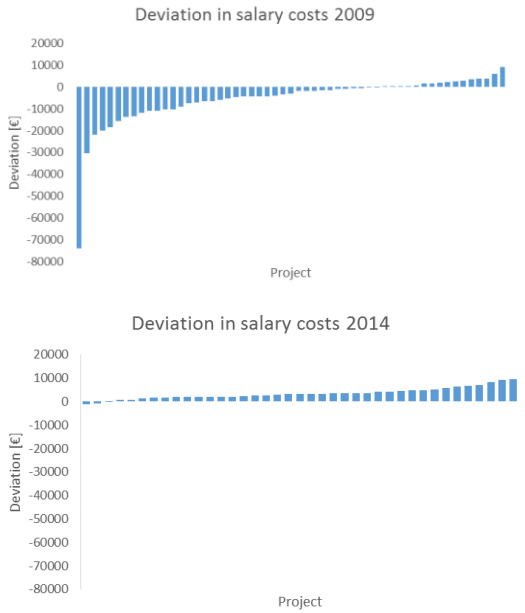


Figure 4 Comparison of salary cost-estimates 2009 and 2014

4.4.3 Comparison of material costs

In 2009 significantly more negative cost-estimates were made than in 2014 when they were supported by the PCS. In 2014 negative deviations continued to occur, but the magnitude of the misestimates was much smaller than in 2009. In 2014 the distribution was evenly distributed around zero deviation, indicating that the estimates were closer to the target than before. The deviation graphs reveal greater accuracy and process control.

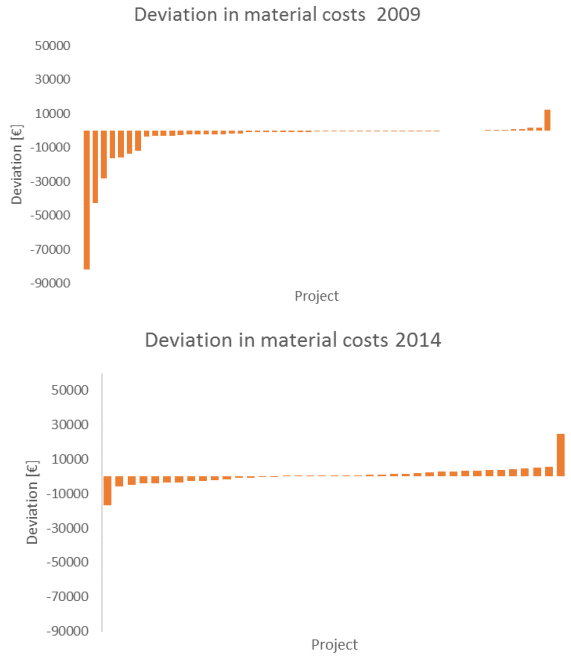


Figure 5 Comparison of material cost-estimates 2009 and 2014

4.4.4 Comparison of subcontractor costs

In 2009 some negative deviations occurred and the tendency was to underestimate subcontractor costs. In 2014 fewer deviations occurred but there was still a tendency to underestimate subcontractor costs. Experts at the company suggested that one reasonable explanation was that the PCS cannot handle all subcontractor costs as they are not as standardized as the salary and materials category. Another reasonable explanation offered was that the subcontractor costs are often variable costs that depend on the time-schedule, so an incorrect salary estimate would lead to an incorrect time schedule, resulting in increased sub-contractor costs. This means that the improved sub-contractor costs might be a "knock-on" effect from improved salary-cost estimation.

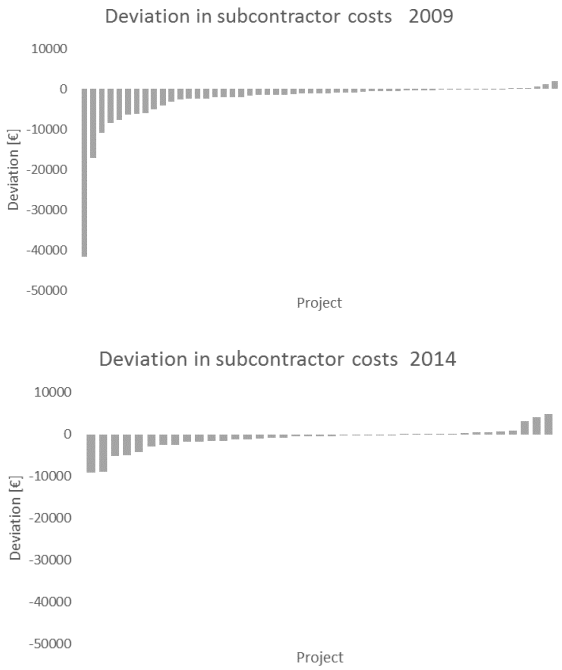


Figure 6 Comparison of subcontractor cost-estimates 2009 and 2014

4.4.5 Summary of cost deviations

Sections 4.4.2, 4.4.3 and 4.4.4. summarize evidence that the cost-estimate accuracy improved significantly within all cost-elements. Table 2 gives an overview of the percentage of under-estimates of projects from before and after implementation of the PCS. Most notable is the increased accuracy in the salary-cost and materials-cost categories. As the salary costs and materials costs together constitute 72% of average total expenses in a typical project, the gains in cost-estimate accuracy contribute to the case company's profitability.

	Salary cost		Material Costs		Subcontractor Cost	
	2009	2014	2009	2014	2009	2014
Under-estimate	71%	8%	76%	38%	89%	67%

Table 2 Percentage of project-costs under-estimated 2009 & 2014

The total amount of money in the two categories of under-estimates and over-estimates can be seen in Table 3. In 2009, the financial loss due to under-estimates were significant, with a total loss of €693.000. In 2014 the financial gain on improved accuracy and compensation by over-estimations was €122.000. It is important to note that the absolute sums are not based on the same number of projects of comparable sizes, so they are not directly comparable. A comparison of the positive and negative deviations from Table 3 was compared to the actual cost of projects from Table 1 and can be seen in Table 4. The data show an improvement moving from a tendency to lose money on under-estimates to earning money on over-estimates. This had a significant impact on profitability, assuming that the company was still competitive at the new cost-estimates. The number of ingoing orders in the case company had in fact increased during the time period investigated.

	Salary cost		Material Costs		Subcontractor Cost	
	2009	2014	2009	2014	2009	2014
Under-estimate	-351.5	-1.9	-249.7	-53.4	-156.5	-53.9
Over-Estimate	42.6	137.1	18.4	78.9	3.6	15.2
Total Sum	-308.9	135.2	-231.3	25.4	-152.8	-38.7

Table 3 Sum of total deviations in 1000 € (2009 & 2014)

	Salary cost		Material Costs		Subcontractor Cost	
	2009	2014	2009	2014	2009	2014
Under-estimate	-15,8%	-0,4%	-4,8%	-3,1%	-26,4%	-16,3%
Over-Estimate	1,9%	27,4%	0,4%	4,6%	0,6%	4,6%

Table 4 Percentage of over- and under-estimated deviations in relation to sum of actual cost-elements in all projects

4.3 Reasons for improved accuracy according to case company

In order to understand the reasons behind the improved accuracy in the different cost elements, semi-structured interviews were conducted and the graphs from Section 4.4 were presented with an open question asking *"what are your explanation for the difference in accuracy between 2009 and 2014?"* Two different interviews were conducted with the head of sales and the head of R&D. The head of sales stated that there was no doubt the PCS had helped to increase the accuracy of cost-estimations by standardizing product solutions. He added that before the PCS was implemented the head of economy had routinely reduced the expected actual cost by 4% on any quotation for certain products due to a clear tendency to deviate in a negative direction. The increased visibility of cost elements created by a cost summary page were identified as a tool that enabled evaluation of costs and updates of prices and had helped to reduce deviations and make sure correct prices were used. It was also pointed out that the level of detail of the prices in the different cost elements had been improved due to the ability to handle prices automatically. The interview with the head of sales credited the following three critical features of the PCS with its success:

- Increased visibility and documentation of cost elements in cost-estimates, by means of cost summaries
- Version control of cost agreements maintained in a single system
- Increased level of detail in cost elements

The head of R&D stated that the old calculation system was tedious and it was difficult to handle cost updates from suppliers and version control. The result was that cost-estimates were often calculated on the basis of different price agreements and resulted in incorrect cost-estimates. He also pointed out that when a project changed in the old system, it was a major task to change all variable aspects of the project, and that this often resulted in mistakes being made. He stated that another reason for the improved accuracy was the visual and easy overview of possible standard solutions, which helped the sales representative to sell products that were already registered as a standard product with known and agreed costs. Yet another reason was the possibility to handle and maintain a higher level of detail in the cost-elements. The interview with the head of R&D credited the following three critical features of the PCS for its success:

- Dynamic allocation of variable costs
- Version control of cost agreements maintained in a single system
- Increased level of detail in cost elements

Afterwards, when the head of sales and head of R&D were brought together to discuss the data on accuracy they agreed that all aspects mentioned were important reasons for the increased accuracy and refinement of cost-estimations.

5 Discussion

The focus of this work was to investigate how a PCS can be used to quantify project costs in project based construction companies. A method for grouping of costs has been presented that respects best practice as documented in the literature but adds a way to calculate time-estimates and cost summaries. The method used by the company made it possible to configure cost-estimates and generate quotations, time-estimates and cost summaries in a single PCS. The automatic generation of documents proved to be a useful way to improve cost accuracy. These findings complement the existing literature on automation in the construction industry by adding a case of successful implementation of rule-based expert system with tangible benefits. The possible reasons mentioned by company experts indicate that a PCS might be a new and viable way to improve cost-estimation evaluation. Another finding was that the PCS can help increase the level of detail and thereby obtain a suitable level of detail as described by Carr [13] without obscuring the user's over-view. The cost-estimation principles used by the case company resembled standard procedures for the construction industry and so the results are believed to be replicable in similar project based companies. However, the presented case study was of a single case company, which clearly limits the generalizability of the study. The case company operated within a defined product solution space which made the use of a rule based expert system feasible. The analysis was based on a limited number of projects and the sample size from 2009 was larger than from 2014. It might be that some outliers occurred

among the cases in 2014 that were not considered, and that this might alter the conclusions. However, the data in combination with expert interviews strongly indicates that there is a connection between the implementation of a PCS and cost-estimation accuracy. Future studies should seek to implement similar solutions in other companies in the construction industry to validate the present results.

6 Conclusion

The purpose of this case study was to investigate cost-estimation accuracy in a longitudinal study and assess the impact of the implementation of a PCS on cost accuracy. It was concluded that the cost estimations did improve quantitatively, showing fewer and smaller deviations and fewer negative under-estimations among all cost-elements. The reasons for these improvements were investigated qualitatively in open interviews with company experts who considered their implementation of a PCS was the main reason for improved cost-accuracy. The reasons behind the improved accuracy could according to these company experts be explained by the increased documentation and visibility of cost-estimates, dynamic allocation of variable costs, version control of cost-agreements and the ability to handle an increased level of costing details.

REFERENCES

[1] H.K. Doloi, Understanding stakeholders' perspective of cost estimation in project management, Int. J. Proj. Manag. 29 (2011) 622–636.

[2] A. Akintoye, E. Fitzgerald, A survey of current cost estimating practices in the UK, Constr. Manag. Econ. 18 (2000) 161–172.

[3] A. Trentin, E. Perin, C. Forza, Overcoming the customization-responsiveness squeeze by using product configurators: Beyond anecdotal evidence, Comput. Ind. 62 (2011) 260–268.

[4] L. Hvam, S. Pape, M.K. Nielsen, Improving the quotation process with product configuration, Comput. Ind. 57 (2006) 607–621.

[5] N.H. Mortensen, MAKING PRODUCT CUSTOMIZATION PROFITABLE, Int. J. Ind. Eng. Appl. Pract. 17 (2010).

[6] T. Männistö, H. Peltonen, T. Soininen, R. Sulonen, Multiple abstraction levels in modelling product structures, Data Knowl. Eng. 36 (2001) 55–78.

[7] C. Forza, F. Salvador, Product information management for mass customization, 2007.

[8] D.K. Oh, W.J. Oh, D.K. Lee, Study of Shipbuilding Cost Estimation for Catamaran-type Leisure Boats Using Product Configuration Model, Trans. Korean Soc. Mech. Eng. A. 38 (2014) 911–916.

[9] N. Markiz, A. Jrade, An expert system to optimize cost and schedule of heavy earthmoving operations for earth- and rock-filled dam projects, J. Civ. Eng. Manag. 23 (2017) 222–231.

[10] D.S.K. Chan, Expert System for Product Manufacturability and Cost Evaluation, Mater. Manuf. Process. 17 (2002) 855–865.

[11] A. Maciol, Knowledge-based methods for cost estimation of metal casts, Int. J. Adv. Manuf. Technol. (2016) 1–16.

[12] L. Hvam, Observed benefits from product configuration systems, Int. J. Ind. Eng. 20 (2013).

[13] R.I. Carr, Cost-estimating principles, J. Constr. Eng. Manag. 115 (1989) 545–551. doi:10.1061/(ASCE)0733-9364(1989)115.

[14] M. Brook, Estimating and Tendering for Construction Work, Routledge, 2008.

[15] J.I.W. Bentley, Construction Tendering and estimating, E. & F. N. Spoon, 1987.

[16] M. Jahangir, Costing R & D Projects : A Bottom-Up Framework, 45 (2003) 12–17.

[17] P.G. Pugh, Working top-down: cost estimating before development begins, Arch. Proc. Inst. Mech. Eng. Part G J. Aerosp. Eng. 1989-1996 (Vols 203-210). 206 (1992) 143–151.

[18] C.A. Ntuen, A.K. Mallik, APPLYING ARTIFICIAL INTELLIGENCE TO PROJECT COST ESTIMATING, Vol. 29 (1987) 8–13.

[19] T.W. Liao, P.J. Egbelu, B.R. Sarker, S.S. Leu, Metaheuristics for project and construction management - A state-of-the-art review, Autom. Constr. 20 (2011) 491–505.

[20] J.H.M. TAH, V. CARR, R. HOWES, An application of case-based reasoning to the planning of highway bridge construction, Eng. Constr. Archit. Manag. 5 (1998) 327–338.

[21] F. Marir, I. Watson, CBRefurb : Case-Based Cost Estimation, IEE Colloq. Case Based Reason. Prospect. Appl. 5 (1995) 1–3.

[22] T.T. Choon, K.N. Ali, A Review Of Potential Areas of Construction Cost Estimating and Identification of Research Gaps, J. Alam Bina. 11 (2008) 61–72.

[23] C. Karlsson, Research methods for operations management, Routledge, 2016.

[24] A. Gayretli, H.S. Abdalla, An object-oriented constraints-based system for concurrent product development, 15 (1999).

[25] E.M. Shehab, H.S. Abdalla, Manufacturing cost modelling for concurrent product development, 17 (2001) 341–353.

[26] L.H. Boyd, J.F. Cox, Optimal decision making using cost accounting information, Int. J. Prod. Res. 40 (2002) 1879–1898.

[27] M. Gupta, R.R. King, An experimental investigation of the effect of cost information and feedback of cost decisions, Contemp. Account. Res. 14 (1997) 99–127.

[28] A.A. Aibinu, T. Pasco, The accuracy of pre-tender building cost estimates in Australia, Constr. Manag. Econ. 26 (2008) 1257–1269.

Configuration and Response to calls for tenders: an open bid configuration model

Delphine Guillon[1,2] and **Abdourahim Sylla**[1,3] and **Élise Vareilles**[1] and **Michel Aldanondo**[1] and **Éric Villeneuve**[2] and **Christophe Merlo**[2] and **Thierry Coudert**[3] and **Laurent Geneste**[3]

Abstract. During bidding process, bidders have to submit offers which will suit the customers' requirements. The OPERA project aims at building a decision support tool to help bidders to design offers using CSP and compare them on orignial indicators. The objective is (1) to help bidder to have the same routine for bid answerings (2) to help them to design more accurate responses and more efficiently. One of the major tasks during bidding process is offer elaboration, which is in our case, 90% a configuration problem and 10% an innovative design one. Four industrial partners are part of the OPERA project: two in the secondary sector and the two others in tertiary one. This paper presents the first results of this project for open bidding configuration. Therefore, we have built a first version of an open generic bidding model which gathers three types of offers data: (1) context characterization data, (2) data defining the product or service and (3) data defining its delivery process, in case of success. Context data allow to characterize the customer profile, the call for tender characteristics, the bidder profile and the environmental factors. The product is decomposed on subsystems and components using a bill of materials and we propose some tracks to extend our model to services. The process is composed of activities, characterized by a couple (resources, workload). This model has been tested on one use case for each industrial partner. This paper is illustrated by a generic instance of a bike open bidding configuration.

1 INTRODUCTION

The response time to calls for tenders has been greatly reduced in the last decades. In a more and more global and competitive environment, companies have now to bid very quickly and very efficiently to calls for tenders. Their bids must be competitive both in quality and selling price if they want to have a chance to win. In such a context and with the increasing number of calls for tenders, companies cannot afford to spend time and resources to study in details their bids. They have now to rationalize, systematize and make more reliable bids definition.

Our aim is to design a tool dedicated to the bidding process in order to help bidders to respond quickly and efficiently to call for tenders. The tool will help a team of consultants to have the same routine for responding to bids. Therefore we propose to build a knowledge-based system or KBS, to help companies to define their bids in such a way they suit the best customers' requirements. We consider that defining a bid corresponds to partially configure at the same time a product or a service and its delivery process [25]. In this paper, we propose a generic bid decomposition and an open generic model for both products and services to support the bidding process. The KBS will be based on this model.

This work is part of a French project, named OPERA, which aims at tackling this problem of response to calls for tenders. OPERA is a French acronym for "Outils logiciels et ProcEssus Pour la Réponse à Appel d'offres", which means "Software tools and Processes for Bids". The OPERA project has started in November 2016 and involves four industrial partners which are daily confronted to this problem of response to call for tenders. In this panel, two of the companies are from the secondary sector while the two others are from tertiary one.

After an initial phase of 6-month interviews, we have found out that the definition of a bid corresponds for 90% to a configuration problem and 10% to a design one: this means that we are mostly in a configuration problem (Assemble-to-order situation), with a small part of new items to be designed (Engineer-to-order situation) [21]. Therefore, we call this particular problem an open bid configuration problem. In addition, a general bid structure as well as a generic bidding process have started to emerge from these interviews. These initial findings lead us to propose the first version of a generic model for open bid configuration.

The rest of the paper is structured as follows: first in section 2, we sketch the OPERA project scope : the four companies involved in the OPERA project are introduced as well as the interview process. We conclude this section by presenting the OPERA tender response process in the light of the literature review. Second, in section 3, the general bid structure is drawn and leads us, in section 4, to the definition of the generic model for open bid configuration dedicated to products. In section 5, a focus is specifically made on the extension of this open product model to services. A discussion and some perspectives conclude our article in section 6. An example of an open bike configuration illustrates our proposals throughout the article.

2 OPERA PROJECT SCOPE

In the past six months, the four companies involved in the OPERA project were interviewed about their tendering process. Two main findings have led us to consider the tendering problem as an open bid configuration one. Let us start by describing the four companies and the interview process we have followed.

2.1 OPERA Companies

The four companies involved in the OPERA project are daily confronted to the response to call for tenders problem. They answer more

[1] IMT Mines Albi-Carmaux, France, email: lastname@mines-albi.fr
[2] ESTIA, Bidart, France, email: f.lastname@estia.fr
[3] ENI Tarbes, France, email: f.lastname@enit.fr

than 100 calls for tenders per year without any guarantee to win.

Two of the companies are from the secondary sector: one of them designs and produces computer numerical control (CNC) machines, and the other one designs and assembles control systems for harbor cranes. The two other companies are from the tertiary sector: one is a professional consulting and training firm, specialized in Supply Chain, Lean Management and industrial methods and the other one is the global leader in innovation and high-tech engineering. Three out of the four companies are SMEs and all of them are present on the world market.

Regarding the tender response process, it mainly relies on some human expertise and know-how. But there are very few experts in that field within the companies and they all have their own way to answer to calls for tenders. None of the companies has a dedicated knowledge-based system able to support this process. Only one company has an Excel file dedicated to the financial part of the tender response: this Excel file helps them quoting the bid by estimating the financial risks incurred. The four companies really want to improve their tender response process in order to be more confident in the proposed bid by capitalizing on good practices, by standardizing their ways of doing things and by assessing risks.

2.2 OPERA Interview Process

In order to capitalize on the companies' know-how on tender response process, we conducted interviews during the last 6 months. We had a monthly one-day meeting with each of them.

Our questionnaire included the following sections:

1. Definition of a bid: which data are needed ? Which documents are produced ? Which decisions are made ?
2. Description of the tender response process: which steps are followed ? Which activities are carried out ? Which people are involved ? etc.
3. Description of the product or service: which items are necessary ? How is the technical part designed ? etc.
4. Description of the risk management: does a risk management process exist in the company ? How are risks taken into account ? What is their impact on the bid solicitation ?
5. Description of the bid knowledge management: is the knowledge capitalized ? How is it shared between bid experts ?

This first round of answers have allowed us to identify a first generic structure of response to tender and to consider this problem as an open configuration problem. Indeed, 90% of the response correspond to a configuration problem, i.e. the selection of relevant items in the item catalog, whereas 10% correspond to a design problem, i.e. the design of some specific items to fulfill some specific customer's requirements. Answering to a call for tenders is therefore a mix of configuration and design.

2.3 OPERA Tender Response Process Definition

[4], [5] and [7] have studied the bidding process. According to [7], it includes four activities: analysis of opportunity, design of technical offer, calculation of selling price and proposal to the client. [13] describes this very competitive environment.

Two types of call for tenders can be distinguished: public ones and private ones [5]. Public tenders are clearly specified. The final customer has to put companies in direct competition on very strict conditions. Private calls for tenders are less formal. Three out of our companies are used to submit only to private tenders. Only one is used to public one. Thus our model aims at being as more generic as possible to be used for the both types of call for tenders.

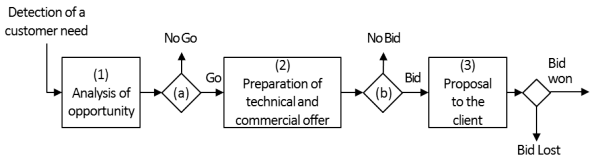


Figure 1. Bidding process, adapted from [7]

The OPERA bidding process, as described in Fig. 1, is a combination of the literature review and the results of the 6-month interviews. The bidding process starts when a company detects a customer's need. It can be a formal invitation to tender or the knowledge that a particular customer needs something new. The company has then to analyze the opportunity to bid. It is the first step of the bidding process. We have identified two major decision steps:

Go/No go decision: A decision is made to answer or not to the call for tenders. In the literature [7], this decision is called "Bid/No Bid".

Bid/No bid decision: A decision is made to submit or not a bid to the customer. For instance, if the bid is not ready on time or if finally, it seems complicated to propose a solution in the bidder's scope or if the offer does not generate enough margin, the bidder can decide at the end "No bid".

For the four companies involved in the OPERA project, the *Go/No Go* decision is made after a macroscopic financial analysis. If the call for tenders is financially promising or strategic, the bidding process is launched. It appears that only less than 5% of responses are not submitted in the *Bid/No Bid* decision step. The first *Go/No Go* decision point is therefore critical for the companies but is out of the scope of the OPERA project.

Our work focuses on the activities between the *Go/No Go* and *Bid/No bid* decision points, as detailed in Fig. 2.

First, the product or service as well as the delivery process have to be defined. Second, a partial risks analysis is carried out to provide more realistic costs and due date. We have to point out that the definition of the bid is an iterative process. There can be several round trips in order to clearly define the offer. These loops can have several reasons: either the cost or due date do not fit the customer's requirements (too expensive or too long), or the project is too risky for the company (there is a big risk to loose money), or the specifications have changed due to the fact that the customer has changed his mind. The KBS should help the companies to build different bids on the same specifications, to compare them following relevant criteria (selling price, due date, risk, confidence) [25] and to select the one which suits the best both customer's requirements and companies' skills.

3 GENERIC BID STRUCTURE FOR PRODUCTS

In this section, we firstly clarify the concept of bid (see subsection 3.1). Then, the identified data and knowledge needed to define a bid have been structured in four different sets. The first one characterizes mainly the context of the call for tenders (see subsection 3.2), the second one, the bill of material (see sub-section 3.3) and the third one, the delivery process and the potential risks incurred (see subsection 3.4).

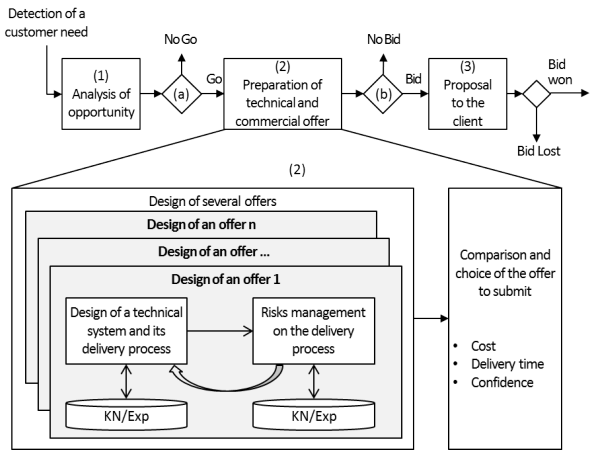


Figure 2. Part of the OPERA Bidding Process

3.1 Bidder and Customer Bids

From the interviews, two kinds of bids have emerged and have to be distinguished:

- The **bidder bid** gathers all the information, documents, data, work, that have been done or used during the bidding process. All the data, information and knowledge used to define the bidder bid are collected and stored in the knowledge bases in order to be reused for future bids.
- The **customer bid** corresponds to an extract from the bidder bid and is submitted to the customer after the *Bid/No Bid* decision point. Indeed, a large part of the work carried out by the company during the bidding process is simply not provided to the customer. Most of the time, the customer bid always contains a description of the bill of material (or BOM) and the selling price, with sometimes the provisional schedule of the project.

In this paper, we only focus on the definition of the bidder bid. The bidder bid is divided into two parts: the first one corresponds to the bill of material and the second one to its delivery process, taking into account the key resources and the major risks. A partial analysis of the risks is mandatory first, to better evaluate the delivery time, second to post the associated cost on the project cost and third, to be aware of the potential hard points during the project.

3.2 Bid Context

The context of the response to call for tender has a strong impact on the bid, for both the BOM and the delivery process [2]. We have started to identify the key elements of the context of the invitation to tender which have an influence on the bid.

Four types of key elements seem to stand out and characterize (1) the customer's profile, (2) the call for tender characteristics, (3) the bidder's characteristics and (4) the environmental factors. These elements will allow the bidder to propose an offer which will fit the customer's requirements observing the whole call for tender context.

Customer's profile. First, the customer's profile can have a very strong impact on the definition of the bidder bid. For instance, a regular or a strategic customer can involve some specific resources or item high quality or Technical Readiness Level (TRL). Therefore the company has to identify a list of features characterizing potential customers. For each invitation to tender, the potential customer has to be described thanks to all the selected features.

Call for tender characteristics. Second, the characteristics of the call for tenders can also have an important impact on the offer. For instance, the bidder can choose to submit different offers for a public market or for a private one.

Bidder's characteristics. Third, the bidder characteristics, i.e. context within the company responding to the call for tenders, can have an impact on the bid. For instance, the workshop load or the backlog state (at the time of the bid) might have to be considered while defining the bid. A bad estimate can be catastrophic and may result in non-compliance with the commitments made in the bid, which is especially true for the delivery time.

Environmental factors. Then, the environmental factors, i.e. context which is external to the company, can also play an important role on the choice of some items of the bid. For instance, identified competitors or detecting the emergence of a new market can have an impact on the company's financial margin. In some cases, the season or the weather can affect the delivery process. It is therefore important to take these kinds of information into account from the beginning.

3.3 Product decomposition

3.3.1 Bill of materials

The second part of a bid corresponds to the bill of material or BOM. This BOM relies on a generic model of the product catalog. The definition of the bid is based on this generic model but not only. As previously mentioned, the interviews have revealed that 90% of the bid BOM correspond to a configuration problem [19], [26] whereas the remaining 10% to a design one. The generic model has therefore to cope with the 10% of design.

In the OPERA project, we have restricted the number of levels for the BOM to three: at the top, the final product, in the middle, the sub-items composing it and at the bottom, the components. In Fig. 3, an instance of BOM for a bike is presented.

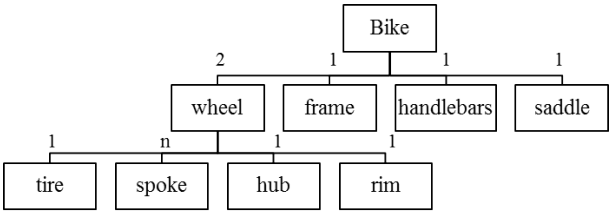


Figure 3. Instance of Bill of material for a bike

The BOM is gradually building up by decomposing the final product into sub-items and components. This decomposition is based on concepts that are linked to each BOM item [8]. In the OPERA project, we link to each concept of the ontology [24], a constraint satisfaction problem. Two types of concepts of the ontology have been identified:

- those corresponding to already known items and for which a generic model exists, i.e. representing the set of all possible combinations. These concepts are mainly used for the configuration problem (90% of the bidder bid BOM).
- and those which are new and have to be designed. These concepts are mainly used for the design problem (10% of the bidder bid BOM).

An existing concept gathers some knowledge about itself: features, definition domain and relations between features. For instance, in our

bike example, one item associated to the *Wheel* concept, is characterized by a diameter, a spoke number and a price, one item associated to the *Bike* concept is characterized by a size and a price, one item associated to the *Tire* concept has a diameter and a price, as shown on table 1, whereas an item associated to a *New* concept brings together no features other than price.

Table 1. Item Knowledge Concepts Example

Concept	Features	Definition domain
Wheel	Wheel Diameter	[12, 29] inch
	Spoke Number	[16, 20, 28, 32, 36] spokes
	Wheel Price	[100, 1600]€
Bike	Size	[16, 25] inch
	Bike Price	≥ 0€
Tire	Tire Diameter	[12, 29] inch
	Tire Price	[100, 600]€
New	Price	≥ 0€

Knowledge embedded in concepts (existing or new) can be enhanced during the bidding process: some features with their definition domain can be added as well as their relations within a concept.

3.3.2 Key Performance Indicators

Usually the bidder design more than just one offer to respond to a call for tender. We propose to use some key performance indicators (KPI) to characterize each offer and to help the bidder to compare them and choose the one which will suit both customer and bidder requirements in the best way. Each item of the BOM is characterized by relevant KPI. Those are at least the cost and the confidence.

Cost. First, we obviously have to compare the different designed offers on the cost. The aggregation of the cost of bottom item of the BOM (mainly components) and each integration will allow to calculate the cost of the product. Depending on the company and on what is key for it, we can also use price or margin.

Confidence. We also propose to use confidence as defined by [25] to evaluate the confidence of the bidder on the ability of the company to design and deliver a product, aggregating the confidence for each components

Depending on the product and the company, one can take into account weight, speed or energy consumption of the product in the KPIs. These KPIs are aggregated bottom-up and help the decision-maker to compare different offers and make the best decision about the one to choose and submit to the client.

3.4 Delivery Process for Products

The third part of the bid is dedicated to the delivery process. This delivery process is composed of the key activities that have to be carried out inside company [16], [23].

3.4.1 Why configuring the delivery process during the bidding process ?

In order to be able to correctly respond to a call for tender, companies cannot just stop after the definition of the product or service. They also have to think about the delivery process in order to evaluate more accurately:

- the delivery time proposed to the customer,
- the cost of the delivery process,
- the risks which might occur during the delivery process.

Delivery time. The delivery time is most of the time the major criteria to respect as it is part of the agreement. In case of delay, some late fees can have to be paid by the company. It is therefore critical for companies to evaluate it as accurately as possible.

Cost. Sometimes, the cost of the delivery process has to be included in the whole bid selling price, for instance the cost of a delivery in a foreign country. Don't include this cost could induce an important gap between the evaluated cost and the real cost, meaning that the company's margin can dramatically drop.

Risk analysis. A partial risk analysis should also be carried out to identify as early as possible the major problems which can occur during the project (in case of a success). The analysis of the major risks' impact (on cost and time) is critical to better evaluate the delivery time, the bid's cost (and therefore the company's financial margin) in both an optimistic and pessimist situations. Indeed, an answer to a call for tenders is the first commitment between a customer and a potential supplier. It seems quite difficult to radically change the price or delivery time after a first bid on the same specifications.

3.4.2 Proposition of a generic delivery process for products

In the OPERA project, for the secondary sector companies, the generic delivery process is a sequence of activities. The activities are the most important ones regarding the bidding process, and have been identified by the bid experts. From the interviews, a generic delivery process has been proposed, as presented in Fig. 4. This generic delivery process is composed of five activities:

Design and Scheduling studies. In this activity, all the new items have to be designed and integrated to the bill of material. We also plan the Gantt chart of the project.

Procurement. All the components and raw material have to be supplied.

Manufacturing. The components have to be manufactured to build the final product,

Assembling and Testing. The components have to be assembled and the final product has to be tested and must comply with the specifications.

Delivery and Commissioning. The final product is delivered to the customer.

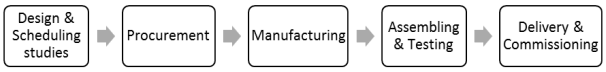


Figure 4. Instance of Product Delivery Process

3.4.3 Key features identification

For each activity, the bid experts have to identify the key features. Those are the ones which have emerged from our interviews:

- duration: each activity may be shorter or longer, depending on several facts, such as allocated resources, risks, complexity, etc.,
- key resources and workload: each resource is characterized by a type (human or machine), a level (slow, fast, junior, senior, etc.) and a workload (under-loaded, loaded, overloaded),

- key risks and impacts: each major risk which could occur on the delivery process has to be described and its impacts on the project (cost and time) have to be defined. It will help the bidder to anticipate possible inconvenient events and propose a bid with a cost and a due date including these possible events.

We will also use for the delivery process the same key performance indicators (KPI) as for the product part (see section 3.3.2): we evaluate the cost of each activity and resource as well as its duration. We can also use the confidence define by [25].

For each bid, this delivery process is configured with respects to the bid context and the BOM.

4 OPEN BID CONFIGURATION MODEL FOR PRODUCTS

In this section, we first present how an open bid configuration model can be formalized as a constraint satisfaction problem. This problem is built in parallel with the BOM: at each decomposition, a new CSP is added and integrated to the current problem thanks to concepts and their relationships. Then, we illustrate our proposals on the open bike example and make a synthesis of our proposals.

4.1 Open Bid Configuration for Products & CSP

We have chosen to model the open bid configuration problem as a Constraint Satisfaction Problem (CSP) as the open bid configuration problem is for 90% a configuration problem [12]. CSP allows to model knowledge and to reason on it to find all the solutions consistent with the current problem.

CSP has been defined by [20] as a triplet $(\mathbb{V}, \mathbb{D}, \mathbb{C})$ where :

- $\mathbb{V} = \{v_1, v_2, ..., v_k\}$ is a finished set of variables,
- $\mathbb{D} = \{d_1, d_2, ..., d_k\}$ is a finished set of definition domains of variables,
- $\mathbb{C} = \{c_1, c_2, ..., c_m\}$ is a finished set of constraints on variables where a constraint describes allowed or forbidden combinations of variables' values.

Constraints allow:

1. to prune the solution space by limiting the value combinations that the variables can simultaneously take (compatibility constraints)
2. to modify the structure of the solution space by adding or removing elements (variables or constraints) to the current problem (activation constraint) [18].

The open bid configuration model relies on the construction of the final constraint bid model by the combination of several CSPs. At the end of the bidding process, the final bid model gathers in the same model, the CSP describing the BOM and the one describing the delivery process.

As previously said in sub-section 3.3, the BOM is building up by decomposing the final product into sub-items and components, each one associated to a specific concept. Each concept is described as an unattached CSP, allowing to configure it. Even if the concept is new, it has an impact on the final product. Constraints between the concepts can also be defined by bid expert in order to propagate the valuation of a variable in a specific concept to the variables of the other ones.

Concerning the delivery process, each activity is formalized as a CSP and linked to the activity network thanks to end-to-end relationships. The association of a concept to the final product creates the links between the bill of material and the generic delivery process [28] [22]. This association activates the constraints between the product and the process in order to propagate the choices made on one side to the other one, and vice-versa.

At the end of the bidding process, the internal bid is completely defined, meaning that all the variables are valuated in such a way that all the constraints are consistent.

4.2 OPERA Application

In OPERA, the bid is building up following the open bid configuration model.

First, the bid context has to be described. For instance, the workshop load (inside context) can have a big impact on the ability of the plant to produce the product on time. If the workshop is over-loaded, the risk of being late has a high probability to occur whatever the product. This impacts, for instance, the duration of the manufacturing and assembly activity which will see its lower bound increase. Therefore, the delivery date will be impacted.

Concerning the bill of material, let's consider that the final product is a well-known bike, composed of two wheels and two tires, also known, as presented in Fig. 3. In that case, we are in a pure configuration problem. First, when building-up the bill of material, the higher item is associated to the concept *Bike*. This association has two main impacts:

- the first brick of the bid model is laid: the associated CSP is instantiated,
- the generic delivery process is linked to the bill of material.

Second, the bike is decomposed into two sub-items with the same concept, that of *Wheel*. This association has two main impacts:

- the second brick of the bid model is laid: the associated CSP is instantiated twice and linked to the previous one by the constraints between concepts. In our case, we assume that there exists a constraint between the *Bike* concept and the *Wheel* one. This constraint specifies the allowed combinations of values for the size of a bike and the diameter of a wheel, as illustrated in table 2.
- the generic delivery process is updated regarding these new variables (not explained here).

Third, the wheels are decomposed into components, one of which is associated to the *Tire* concept. This association has two main impacts:

- the third brick of the bid model is laid: the associated CSP is instantiated and linked to the previous one by the constraints between concepts. In our case, we assume that there exists a constraint between the *Wheel* concept and the *Tire* one. This constraint specifies the diameter of the wheel equals the one of the tire, as presented by the eq. 1.
- the generic delivery process is updated regarding these new variables (not explained here).

$$Wheel :: Diameter = Tire :: Diameter \tag{1}$$

Now, let's consider that product is a unknown bike, meaning that at least, one of its item is associated to the *New* concept. For instance, let us consider that the rim is the new component. This association (rim, *New*) has two main impacts:

Table 2. Bike inter-Concepts Constraint Example

Variable 1	Variable 2
Bike::Size	Wheel::Diameter
16	$[13, 14]$
$[17, 19]$	$[14, 17]$
$[20, 22]$	$[18, 22]$
≥ 22	≥ 23

- a new brick of the bid model is laid: the associated CSP is instantiated but not linked to the previous one by a constraint between concepts, as no relation can be established in advance.
- the generic delivery process is updated regarding these new variables. As one of the items is associated to a *New* concept, the finalization of the design activity lasts longer than for a pure configuration problem and there can be a risk of integration of the new component in the existing bill of material.

4.3 Open Bid Configuration Model for Products: Synthesis

The open bid configuration relies on four sets of information, allowing to characterize the context of the call for tenders, the bill of material, the delivery process and the potential risks incurred, as presented in Fig. 5.

The **context of the call for tenders** is useful to characterize the customer, the call for tender itself, the bidder and the environment, as they have a potential impact on the product and its delivery process.

The **bill of material** is top-down building up by the decomposition of the product into sub-items and components. For each item, we associate a concept gathering its knowledge formalized as an unattached CSP, if any. Each time a concept is added, the open bid configuration model is upgraded with the new CSP.

These concepts have a strong impact too on the **delivery process** features, such as duration, key resources and major risks. The delivery process is directly associated to the final product and is completely configured during the bidding process.

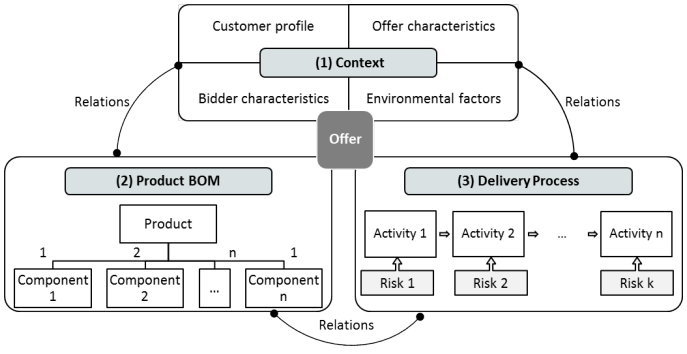


Figure 5. Offer modeling for bidding process in the secondary sector

5 MOVING TO AN OPEN GENERIC BIDDING MODEL FOR SERVICE PROVISION

In this section, we first highlight the main differences between product and service configuration in bidding process. Then, we discuss how our open bidding model can be extended to services and conclude with further works.

5.1 From Product to Service Bids

[17] describes the problem of modularization and configuration of services. But since our work deals with bid configuration during call for tenders process, we focus on Business-to-Business (BtoB) services, and not Business-to-Customer (BtoC) ones. We also restrict our study to pure services (and not product service systems). For both service provision companies we are working with, and it is usually the case during bidding process, each business is different because of changes in the customer's needs. Some parts might be the same as on a past case, but past offers cannot be exactly reused. As explained by [6], these companies "need to balance meeting the needs of individual customers with ensuring a satisfactory degree of efficiency in the deployment of services". This arises the need for service configuration.

[12] and [3] have studied service configuration. As reported in [12], service configuration seems similar to that of physical products but the results of research on mass customization of goods may not be directly applicable. There is relatively little research on configurable services and on developing suitable configurators. [15] highlights the gap dealing with mass customization of services, configurable services, and configured in services based on a review of product mass customization and configuration. Three main differences have been identified [9]:

- Products in manufacturing organizations are highly tangible ; services and especially the service delivery process are less so;
- Related to this, production flows are transparent in manufacturing and less transparent in services. The same holds for problems and irregularities;
- Finally, the customer is much less involved in the production process in the manufacturing domain than in services. The interaction with the customer determines the quality of the service.

In the context of bidding process, we can extend a part of our model in a trivial way: companies still have to identify the bid context, define the nomenclature of items and characterize the delivery process. Thus part (1), (2) and (3) of Figure 5 stay the same. The important difference for service provision offer is the link between everything: How to move from an open product configuration model to an open service configuration model for bidding process ? (Fig. 6).

[11] and [27] define service as a process. Can a service be resumed by its delivery process ? We think not and we try to define how we can decompose a service in a kind of nomenclature, such as a deliverable breakdown structure (DBS).

In [14], Goldstein et al. explain that "From the service organizations perspective, designing a service means defining an appropriate mix of physical and non-physical components". They precise that "service components are often not physical entities, but rather are a combination of processes, people skills, and materials".

For [1], a service can be decomposed into *service elements*. These service elements "represent what a supplier offers to its customers". They precise that "a service element can be decomposed into smaller service elements, as long as these smaller elements can be offered to customers separately, possibly by different suppliers." Thus the criteria to decompose a service could be this one: each element can be offered separately.

In this sub-section, we voluntarily use the generic term of *nomenclature* in contrast to *bill of material or BOM* which is more specific to products.

Indeed, we have to point out that the links between the item nomenclature and the delivery process differ between the companies

of the secondary and tertiary sectors. Two main differences have been identified:

- Firstly, for the secondary sector companies, the delivery process correspond to the one carried out to produce the final product (higher level item of the nomenclature), whereas for the tertiary sector companies, it seems that it is not always the case. A service is composed of several deliveries composed of several work-packages which have all their own delivery process, i.e. there potentially exist as many delivery processes as the number of the lower level items.
- Secondly, for the secondary sector companies, the activities of the delivery process are always carried out inside the company. A the end of the process, the product is ready to be delivered. In contrario, some of the activities of the tertiary sector companies, are carried out outside the company, directly on customers' premises. Therefore, the delivery process is decomposed into two types of activities: those carried out in the company and those carried out in the customer's premises. In this section, we only present the delivery process for the secondary sector companies. Nevertheless, a special focus is made in section 5 on the tertiary sector companies.

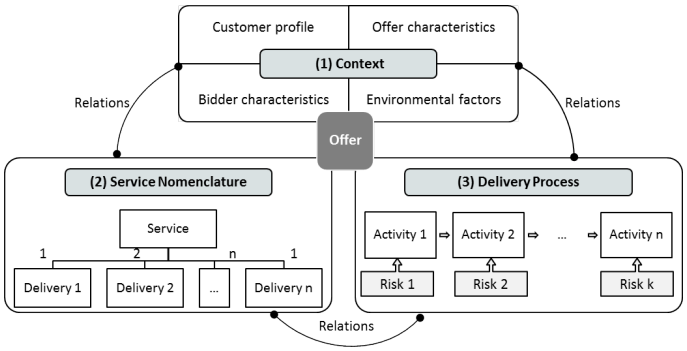


Figure 6. Offer modeling for bidding process in the tertiary sector

5.2 Service Model: Discussions

Tertiary companies mainly differ by the type of deliverables they produce. For the two companies of the tertiary sector, we have identified two types of deliverables:

1. tangible deliverables,
2. intangible deliverables.

For the tangible deliverables, the customer is mostly interested by the results of the delivery process. The customer expects a very specific deliverable and is not especially interested in the delivery process. For instance, one out of the OPERA industrial partners respond to call for tenders for Computational Fluid Dynamics (CFD) studies. We can't decompose the deliverable in deliveries. Indeed, the deliverable is not material as the way that it is not an assembly of components. Here, the deliverable is the result of calculations. This result is what the potential customer is interested in. (S)He is not interested in the delivery process, as long as (s)he has the result of the asked calculation. Only the quality and the compliance with the commitments matter.

For intangible deliverables, there is no physical deliverable. The potential customer is more interested in the delivery process itself. In training for instance, the customer is mostly interested in the fact that people who are trained reach a specif skill level. To define such an offer, the bid has to specify the given courses, the addressed topics in the training and how the training will go on, which is actually a part of the delivery process (activities carried out on customer's premises). A large part of the configuration of the bid consists in deciding which courses to give, which topics to discuss, how much time to give to each topic and who will be the trainer. This kind of bid definition is quite closed to the one for products in the sense that it consists in selecting and picking the relevant courses for a course catalog, as studied in [10].

6 DISCUSSION AND FUTURE RESEARCH

In this paper, we have presented our first result to define an open generic model for bid definition. We consider that defining a bid corresponds for 90% to a configuration problem and for 10% to a design problem. We consider two types of bids: the bidder bid, which is built up in companies and the customer bid, which is submitted to the customer. Our work focuses on bidder bids. We have identified a generic internal bid structure which is decomposed of three different sets of knowledge: the one characterizing the bid context (customer profile, call for tender characteristics, bidder profile and environmental factors), the one characterizing the item nomenclature and the last one, characterizing the delivery process.

We have instantiated this model for products and show the interest of our proposals. A use-case dedicated to products is actually in progress. We then have discussed about its applicability to services and highlighted the fact that depending on the types of the deliverables, the open model for products can easily be used. We have seen that the open service bid model seems quite similar to the one for products when the deliverables is intangible and when the items can be chosen in a item catalog.

We still have to work on how to define an open bid configuration model for both products and services, independent of the type of deliverables. This open configuration model will also integrate a new metrics to characterize bids: the confidence of the bidder in the customer offer [25]. This new metrics is partially based on the notion of risks which are partially analyzed during the bidding process.

ACKNOWLEDGEMENTS

We would like to thank all the industrial partners of the ANR OPERA Project[4] for their implication in the project.

[4] `http://gind.mines-albi.fr/en/projet/opera`

REFERENCES

[1] Hans Akkermans, Ziv Baida, Jaap Gordijn, Nieves Peña, Ander Altuna, and Iñaki Laresgoiti, 'Value webs: Using ontologies to bundle real-world services', *IEEE Intelligent Systems*, **19**(4), 57–66, (2004).

[2] Ziv Baida, Jaap Gordijn, Hanne Sæle, Andrei Z Morch, and Hans Akkermans, 'Energy services: A case study in real-world service configuration', in *International Conference on Advanced Information Systems Engineering*, pp. 36–50. Springer Berlin Heidelberg, (2004).

[3] Jörg Becker, Daniel Beverungen, Ralf Knackstedt, and Martin Matzner, 'Configurative service engineering - A rule-based configuration approach for versatile service processes in corrective maintenance', *Proceedings of the 42nd Annual Hawaii International Conference on System Sciences, HICSS*, (2009).

[4] Anne-Lise Benaben, *Méthodologie d'identification et d'évaluation de la sûreté de fonctionnement en phase de réponse à appel d'offre*, Ph.D. dissertation, 2009.

[5] Juan Diego Botero, Cdrick Bler, and Daniel Noyes, 'Bprm methodology: Linking risk management and lesson learnt system for bidding process.', in *APMS (1)*, eds., Bernard Grabot, Bruno Vallespir, Samuel Gomes, Abdelaziz Bouras, and Dimitris Kiritsis, volume 438 of *IFIP Advances in Information and Communication Technology*, pp. 233–240. Springer, (2014).

[6] Per Carlborg and Daniel Kindström, 'Service process modularization and modular strategies', *Journal of Business & Industrial Marketing*, **29**(4), 313–323, (2014).

[7] Rachid Chalal and A R Ghomari, 'An Approach for a Bidding Process Knowledge Capitalization', *World Academy of Science, Engineering and Technology*, **13**(7), 293–297, (2006).

[8] M. J. Darlington and S. J. Culley, 'Investigating ontology development for engineering design support', *Advanced Engineering Informatics*, **22**(1), 112–134, (2008).

[9] Jeroen de Mast, 'Six sigma and competitive advantage', *Total Quality Management & Business Excellence*, **17**(4), 455–464, (2006).

[10] Regine Dörbecker and Tilo Böhmann, 'The concept and effects of service modularity - A literature review', *Proceedings of the Annual Hawaii International Conference on System Sciences*, 1357–1366, (2013).

[11] Bo Edvardsson, Anders Gustafsson, and Inger Roos, 'Service portraits in service research: a critical review', *International Journal of Service Industry Management*, **16**(1), 107–121, (feb 2005).

[12] Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, *Knowledge-based Configuration: From Research to Business Cases*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1 edn., 2014.

[13] Lawrence Friedman, 'A Competitive-Bidding Strategy', *Operations research*, **4**(1), 104–112, (1956).

[14] Susan Meyer Goldstein, Robert Johnston, JoAnn Duffy, and Jay Rao, 'The service concept: The missing link in service design research?', *Journal of Operations Management*, **20**(2), 121–134, (2002).

[15] Mikko Heiskala, Kaija-Stiina Paloheimo, and Juha Tiihonen, 'Mass customization of services: benefits and challenges of configurable services', in *Frontiers of e-Business Research (FeBR)*, (2005).

[16] P.T. Helo, Q.L. Xu, S.J. Kyllnen, and R.J. Jiao, 'Integrated vehicle configuration systemconnecting the domains of mass customization', *Computers in Industry*, **61**(1), 44 – 52, (2010).

[17] Thorsten Krebs and Aleksander Lubarski, 'Towards modularization and configuration of services–current challenges and difficulties', in *18 th International Configuration Workshop*, p. 77, (2016).

[18] S. Mittal and B. Falkenhainer, 'Dynamic constraint satisfaction problems', in *AAAI*, pp. 25–32, Boston, US, (1990).

[19] S. Mittal and F. Frayman, 'Towards a generic model of configuration tasks', in *proceedings of the Eleventh International joint Conference on Artificial Intelligence*, pp. 1395–1401, (1989).

[20] U. Montanari, 'Networks of constraints: fundamental properties and application to picture processing', in *Information sciences*, volume 7, pp. 95–132, (1974).

[21] Jan Olhager, 'Strategic positioning of the order penetration point', *International Journal of Production Economics*, **85**(3), 319–329, (2003).

[22] P. Pitiot, M. Aldanondo, E. Vareilles, P. Gaborit, M. Djefel, and S. Carboneel, 'Concurrent product configuration and process planning, towards an approach combining interactivity and optimality', *International Journal of Production Research*, **51**(2), 524–541, (2013). *WoS**.

[23] K. Schierholt, 'Process configuration: Combining the principles of product configuration and process planning', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM*, **15**(5), 411–424, (2001).

[24] S. Staab and R. Studer, *Handbook on Ontologies*, Springer Publishing Company, Incorporated, 2nd edn., 2009.

[25] A. Sylla, E. Vareilles, M. Aldanondo, T. Coudert, L. Geneste, and P. Pitiot, 'Concurrent configuration of product and process : moving towards eto and dealing with uncertainties.', in 18^{th} *International Configuration Workshop*, (2016).

[26] J. Tiihonen, T. Lehtonen, T. Soininen, A. Pulkkinen, R. Sulonen, and A. Riitahuhta, 'Modeling configurable product families', in *4th Workshop on Product Structuring (WDK)*, (1998).

[27] Alan. Wilson, Valarie A. Zeithaml, Mary Jo. Bitner, and Dwayne D. Gremler, *Services marketing : integrating customer focus across the firm*, McGraw-Hill Higher Education, 2012.

[28] Linda Zhang, E. Vareilles, and M. Aldanondo, 'Generic bill of functions, materials and operations for sap2 configuration', Technical report, HAL, (2013).

Automated Question Generation from Configuration Knowledge Bases

Amal Shehadeh[1] and **Alexander Felfernig**[1] and **Müslüm Atas**[1]

Abstract. Knowledge-based configuration systems support decision-making processes by helping customers (users) to effectively identify products and services that suit their wishes and needs. These systems have the potential to significantly improve the underlying business processes by reducing error rates and time efforts related to advisory services. Despite the successful application of configuration technologies in different sectors, there are still issues related to the transfer of configuration-related knowledge to employees. In this paper, we discuss a new approach to automatically generate questions related to configuration knowledge from knowledge bases, to be exploited in training and educating sales representatives. Furthermore, we present an evaluation of the question generation approach with regard to performance.

1 Introduction

Knowledge-based configuration systems support users in the identification of solutions (configurations) that fulfill their wishes and needs [6, 10]. Domain examples are cars, financial services, digital cameras, and elevators [6]. Typically, users specify their requirements and the system tries to identify solutions (configurations) that satisfy all the requirements. These systems exploit deep product domain knowledge in order to determine the solutions. If no solution could be found, users have to be supported in finding a way out of this situation.

Knowledge-based configuration systems have the potential to improve the underlying business processes in various dimensions, for example, by reducing error rates and time efforts related to customer advisory services and, as a consequence, increasing customers trust and satisfaction with the solutions. Furthermore, customer domain knowledge can be improved by configuration technologies through the interaction with these systems, where customers gain a deeper understanding of the product domain and, as a consequence, less efforts are needed related to the explanation of basic domain aspects.

Despite the successful application of configuration technologies in different sectors, there are still issues related to the transfer of configuration-related knowledge to employees. In dialogs with customers, sales representatives should not only rely on solutions and related explanations provided by the configuration environment but should also dispose of the needed domain knowledge [5]. The goal of this paper is to show how different types of questions can be automatically generated from a configuration knowledge base, to provide e-learning support to sales representatives with regard to the mentioned knowledge transfer. Examples of application domains can be financial services, telecommunication, computers, cars, mobiles, TVs, and digital cameras.

A number of researches have been conducted in the context of automatic question generation from different types of knowledge sources. In the remainder of this section, we will shortly analyze existing approaches to question generation and then focus on our question generation approach.

Gütl et al. [9] introduced an approach to the automated creation of different types of test items. They described the design and development of the Enhanced Automatic Question Creator (EAQC), which is designed to deal with multilingual learning material. EAQC extracts the most important concepts out of textual learning content and creates single choice, multiple-choice, completion exercises, and open ended questions on the basis of these concepts. Their approach combines statistical, structural, and semantic methods of natural language processing as well as a rule-based solution for concept extraction. Results of their study show that the extracted concepts and created questions are relevant with respect to the learning content.

Yllias et al. [17] proposed a topic-to-question generation approach that generates all possible questions from a topic of interest. They considered that each topic is associated with a body of texts containing useful information. Questions are generated by exploiting the named entity information and the predicate argument structures of the sentences present in the body of texts. The generated questions are ranked by considering both their importance and syntactic correctness. They evaluated their approach and found that it can significantly outperform all other considered systems in terms of topic-relevance and syntactic correctness.

Mazidi et al. [14] described an approach to automatic question generation from natural language understanding (NLU) to natural language generation (NLG). Their approach analyzes first the central semantic content of each independent clause in each sentence, then question templates are matched to what the sentence is communicating in order to generate higher quality questions. They evaluated their approach and found that it generates a higher percentage of acceptable questions than prior state-of-the-art systems.

Agarwal et al. [1] presented an automatic question generation system (GFQG) that can generate gap-fill questions from a document. Gap-fill questions are fill-in-the-blank questions with multiple choices (one correct answer and three distractors) provided. The system selects the most informative sentences of the chapters and generates gap-fill questions by first blanking keys from the sentences and then determining the distractors for these keys. Syntactic and lexical features are used in this process without relying on any external resource apart from the information in the document.

Majumder et al. [13] presented a system that generates multiple

[1] Institute for Software Technology, Graz University of Technology, Austria, email: ashehade@ist.tugraz.at, alexander.felfernig@ist.tugraz.at, muatas@ist.tugraz.at.

choice questions automatically from an input corpus. The proposed technique selects informative sentences based on topic modeling and parse structure similarity. The selected sentences are used then in the key selection and distractor generation using a set of rules and name dictionary. Experimental results demonstrated that the proposed technique is quite accurate.

Alsubait et al. [2, 16] discussed an approach to ontology-based multiple-choice question generation, and proposed a design of protocol to evaluate the usefulness and difficulty of the generated questions. They presented an empirical evaluation of their approach and examined its feasibility to be applied by educators with no prior experience in ontology building. They found out that their approach can result in a reasonable number of educationally useful questions with good predictions about their difficulty levels.

In this paper we introduce an approach that has commonalities with the question generation approach introduced in [2, 16]. A *configuration knowledge base* can be used for question generation where questions do not only refer to the *solution space* defined by the knowledge base but also to situations where *no solution can be found* for a given set of customer requirements [5]. Compared to the approach presented in [2, 16], our question generation approach is based on a configuration knowledge base represented as a Constraint Satisfaction Problem (CSP) [3]. In this paper, we introduce a new approach to automatically generate multiple-choice questions out of a constraint-based configuration knowledge base. To the best of our knowledge, no similar researches have been conducted before.

The remainder of this paper is organized as follows. In Section 2, we present a working example from the domain of financial services, giving the needed background related to configuration concepts. In Section 3, we introduce our approach to the automated question generation from a constraint-based configurator knowledge base. An evaluation of the question generation approach with regard to performance is presented in Section 4. In Section 5, we discuss issues for future work. Finally, the paper is concluded in Section 6.

2 Working Example: A Simple Financial Services Configuration Knowledge Base

Knowledge-based configuration systems support users in the identification of relevant solutions. Typically, these systems try to find a solution (configuration) that satisfies all user requirements; and if no solution could be found, such systems propose explanations to support the user in finding a way out of this dead-end. Constraint-based configurators are a specific type of knowledge-based configurators that are based on explicit configuration rules (constraints) and retrieve solutions (configurations) that fulfill these rules. The knowledge base of a constraint-based configurator is defined in terms of a constraint satisfaction problem (CSP).

Definition (Constraint Satisfaction Problem - CSP). A constraint satisfaction problem *(CSP)* can be defined by a triple (V, D, C) where *V* is a set of domain variables $\{v_1, v_2, ..., v_n\}$ describing potential user (customer) requirements and product properties, D represents variable domains $\{dom(v_1), dom(v_2), ..., dom(v_n)\}$, and C is a set of constraints defining restrictions on the possible combinations of variable values $\{c_1, c_2, ..., c_n\}$.

On the basis of such a configuration knowledge base and a given set of customer requirements, we are able to calculate solutions (configurations). The task of identifying a solution that satisfies the customers requirements is denoted as a configuration task.

Definition (Configuration Task). A configuration task can be defined as a CSP (V, D, C, R) - see the previous definition of *CSP* with the corresponding representation of V, D, and C - where R represents a set of user (customer) requirements (i.e. a set of requirement variable assignments from *V*).

On the basis of this configuration task definition, we can now introduce the definition of a concrete configuration (i.e. solution for a configuration task).

Definition (Configuration). A configuration (CSP solution) for a given configuration task is a complete set A of variable assignments $v_i = a$ to the variables $v_i \in V$ ($v_i = a \rightarrow a \in$ domain(v_i)) with *consistent* ($A \cup C \cup R$).

Consequently, configurations (solutions) identified for configuration tasks can be considered as candidate solutions for a customer (user). Alternative configurations (solutions) can be ranked according to their utility for the customer. This ranking can be based on the *multi-attribute utility theory* (MAUT), which evaluates each solution with regard to its utility for the customer [10].

Now we introduce a simple example of a configuration task from the *financial services* domain that will be used as working example throughout the paper. The variables in V are *willingness to take risks* (wr), *duration of investment* (di), *expected return rate* (rr), and $itemname$ which represents the name of a financial service.

- $V = \{wr, di, rr, itemname\}$
- D = {dom(wr) = {low, medium, high}, dom(di) = {shortterm, mediumterm, longterm}, dom(rr) = {low, medium, high}, dom($itemname$) = {equityfund, investmentfund, bankbook}}
- $C = \{c_1 : wr = low \rightarrow itemname = bankbook$, $c_2 : wr = medium \rightarrow itemname \neq equityfund$, $c_3 : di = shortterm \rightarrow itemname = bankbook$, $c_4 : di = mediumterm \rightarrow itemname \neq equityfund$, $c_5 : rr = high \vee rr = medium \rightarrow itemname \neq bankbook$, $c_6 : \neg(wr = low \wedge rr = high)$, $c_7 : \neg(di = shortterm \wedge rr = high)$, $c_8 : \neg(wr = high \wedge rr = low)\}$
- $R = \{r_1 : wr = low, r_2 : di = shortterm, r_3 : rr = low\}$

A configuration (solution) for the given configuration task is the following set of variable assignments $A = \{wr = low, di = shortterm, rr = low, itemname = bankbook\}$. In this case, the given set of customer requirements defined in R is consistent with the constraints in C. However, even a slight change in customer requirements (R) makes it inconsistent with the constraints in C. For example, if we change the customer requirements to $R = \{r_1 : wr = low, r_2 : di = shortterm, r_3 : rr = high\}$, R becomes inconsistent with C and no solution (configuration) can be identified.

In situations where no solution can be found for a given set of customer requirements, concepts of *model-based diagnosis* (MBD) [15] can help to identify a minimal set of requirements that has to be deleted or adapted such that a solution can be identified. For the identification of such minimal changes, we introduce the following definitions.

Definition (Conflict Set). A conflict set is a set $CS \subset R$ s.t. $C \cup CS$ is inconsistent. A conflict set CS is said to be minimal iff $\nexists$ a conflict set CS': $CS' \subset CS$.

Minimal conflict sets can be determined on the basis of conflict detection algorithms, such as QUICKXPLAIN [11] that calculates one conflict set at a time for a given set of constraints, and follows a divide-and-conquer search strategy that helps to significantly accelerate the performance compared to other approaches.

For example, the identified minimal conflict sets for the previous set of customer requirements ($R = \{r_1 : wr = low, r_2 : di = shortterm, r_3 : rr = high\}$) are $CS_1 = \{r_1, r_3\}$ and $CS_2 = \{r_2, r_3\}$.

Definition (Diagnosis Task). A diagnosis task is defined as a tuple (C, R) where C is a set of constraints, R is a set of customer requirements, and $R \cup C$ is inconsistent.

On the basis of this diagnosis task definition, we can introduce the definition of a diagnosis.

Definition (Diagnosis). A set $\Delta \subseteq R$ for a given diagnosis task (C, R) is a diagnosis if $R - \Delta \cup C$ is consistent, i.e., Δ is a set of requirements that has to be deleted from R or adapted such that the remaining customer requirements are consistent with the constraints in C. Furthermore, Δ is minimal iff $\nexists$ a set Δ': $\Delta' \subset \Delta$.

The standard approach to calculate minimal diagnoses is to resolve all minimal conflict sets existing in the constraint set. The derivation of the corresponding diagnoses is based on the calculation of hitting sets, as in the HSDAG [15] algorithm (hitting set directed acyclic graph), which follows a breadth-first search strategy. Another approach to calculate minimal diagnoses is to identify it directly from the inconsistent constraint set without the need to calculate minimal conflict sets, as in FASTDIAG [4] algorithm which follows a divide-and-conquer search strategy.

For example, the identified minimal diagnoses for the previous set of customer requirements (R = $\{r_1 : wr = low, r_2 : di = shortterm, r_3 : rr = high\}$) are $\Delta_1 = \{r_3\}$ and $\Delta_2 = \{r_1, r_2\}$.

In case of large sets of diagnosis alternatives, it will not always be clear which diagnosis should be selected or in which order alternative diagnoses should be displayed to the user. Therefore, approaches to reduce the number of diagnosis alternatives and for the determination of *personalized diagnoses* (to identify diagnoses that are more relevant to users) have been introduced. Felfernig et al. [7] presented an approach to rank diagnoses based on multi-attribute utility theory, where they assume that customers (users) provide weights for each individual requirement which represent its importance to them (users preferences). The higher the importance of a requirement, the lower the probability that it will be included in a diagnosis displayed to the user. Also Felfernig et al. [7, 8] presented a personalized diagnosis approach that integrates collaborative configuration techniques (similarity-based, utility-based, probability-based, ensemble-based) with diagnosis search to increase the prediction quality (in terms of precision) of diagnoses. Other existing approaches only focus on *minimal cardinality diagnoses* [8] (i.e. minimal diagnoses with the lowest possible number of included constraints) which are determined on the basis of breadth-first search strategy, assuming that repair alternatives with low-cardinality changes are favored compared to alternatives including a higher number of changes, but it cannot be guaranteed that minimal cardinality diagnoses leads to the most relevant diagnoses to users.

After having identified the set of possible minimal diagnoses, *repair actions* [10] for each of those diagnoses should be proposed to the user so that at least one solution can be identified, i.e., possible adaptations to the existing set of requirements R so that the user can find a solution. For the identification of repair actions, we introduce the following definition.

Definition (Repair Task). Given a set of customer requirements R inconsistent with C and a corresponding diagnosis $\Delta \subseteq R$ ($\Delta = \{ra_1, ra_2, ..., ra_n\}$), the corresponding repair task is to determine an adaption to the requirements included in the diagnosis RA = $\{ra'_1, ra'_2, ..., ra'_n\}$, such that $R - \Delta \cup RA$ is consistent with C.

All the aforementioned approaches can be exploited in the context of question generation from knowledge bases (see the following section).

Types of configuration knowledge. Knowledge-based configuration systems include knowledge in different forms. First, given a set of customer requirements (R), a configurator can determine products/items that can be recommended to the customer (*filter knowledge*). Second, given a product/item, a configurator can determine customer requirements that are consistent with this product/item (*product knowledge*). Third, given a configuration (solution), a configurator can determine a set of customer requirements (R) that are inconsistent with this solution; or given a set of inconsistent customer requirements (R), a conflict detection algorithm [11] can determine a minimal subset of R that triggers an inconsistency with C (*inconsistency knowledge*). Fourth, in situations where no solution can be identified for a given set of customer requirements (R), diagnosis algorithms [15, 4] can determine the needed minimal changes to help the user out of the *no solution could be found dilemma* (*analysis knowledge*). Fifth, in situations where no solution can be identified for a given set of customer requirements R, concrete repair actions for the requirements included in a diagnosis can determine what changes in (R) lead to the identification of a solution (*repair knowledge*). In the following section, we discuss how configuration task definitions can be exploited for the generation of questions to be used, for example, for educating sales representatives.

3 Generating Questions from Configurator Knowledge Bases

Questions and corresponding answers can be automatically generated from a configuration task definition. On the basis of our working example, we now introduce an approach to automatically generate questions (and related answers) from a knowledge base for the aforementioned types of configuration knowledge. The generated questions can be exploited for training and supporting sales representatives. The overall goal of these questions is to increase the personal level of sales knowledge and, as a consequence, to make advisory processes more efficient.

3.1 Filter knowledge related question generation

(1) The underlying task is to figure out which products/items (P = $\{itemname = p_1, itemname = p_2, ..., itemname = p_n\}$ where $p_i \in domain(itemname)$) fit a given set of predefined customer requirements (R). More formally, filter knowledge related questions can be generated on the basis of a configuration task definition (V, D, C, R). On the basis of such a definition, a constraint solver (configurator) is able to calculate configurations (solutions) that satisfy $R \cup C$ i.e. all possible instantiations of customer requirements (R) and corresponding products/items (P). The set of customer requirements (R) can then be the basis of the generated question, and the corresponding products/items (P) can be the correct answer(s), while faulty answers are the remaining products/items i.e. all possible instantiations of $itemname$ that satisfy $\{itemname\ != p_1\ and\ itemname\ != p_2 ...\ and\ itemname\ != p_n\}$

Example. Given the configuration task definition of our working example (financial services configurator), a set of customer requirements could be R = $\{r_1 : wr = low, r_2 : di = shortterm, r_3 : rr = low\}$, the set of corresponding correct answers is P= $\{itemname = bankbook\}$ and the set of corresponding faulty answers is $\{itemname = equityfund, itemname = investmentfund\}$. The corresponding generated question would be: ***Given the following customer requirements ..., which products/items to recommend?***

(2) The underlying task is to figure out which products/items, other than a predefined one ($itemname = p$), fit a given set of customer requirements (R) without any change in R. Formally, this type of questions can be generated on the basis of a configuration task definition (V, D, C, R). A constraint solver (configurator) is able to calculate configurations that satisfy $R \cup C \cup \{itemname! = p\}$ i.e. all possible instantiations of customer requirements (R) and corresponding products/items (P = $\{itemname = p_1, itemname = p_2, ..., itemname = p_n\}$ where $p_i \in domain(itemname)$ and $p_i ! = p\}$). The set of customer requirements (R) with the predefined product/item ($itemname = p$) can then be the basis of the generated question, and the corresponding products/items (P) can be the correct answer(s), while faulty answers are the remaining products/items i.e. all possible instantiations of $itemname$ that satisfy $\{itemname ! = p_1$ and $itemname ! = p_2 ...$ and $itemname ! = p_n\}$.

Example. Given the configuration task definition of our working example, a set of customer requirements could be R = $\{r_1 : wr = high$, $r_2 : di = longterm$, $r_3 : rr = high\}$ and a predefined product/item $\{itemname = equityfund\}$, the set of corresponding correct answers is P = $\{itemname = investmentfund\}$, and the set of corresponding faulty answers is $\{itemname = bankbook, itemname = equityfund\}$. The corresponding generated question would be: ***Given the following customer requirements ..., which products/items other than... would you recommend without any changes in the requirements?***

(3) The underlying task is to figure out what additional requirements can be added to a given incomplete but consistent set of customer requirements (R) so that a configuration can still be identified. Formally, this type of questions can be generated on the basis of a configuration task definition (V, D, C, R). A constraint solver (configurator) is able to calculate all sets of requirement variable assignments (RV) that satisfy $R \cup C$ ($RV = \{RV_1, RV_2, ..., RV_m\}$: $RV_i = \{v_1 = a_1, v_2 = a_2, ..., v_n = a_n\}$ $where$ $v_i \in V, a_i \in domain(v_i), v_i = a_i \notin R,$ and $RV \cup R \cup C$ is $consistent$). The set of defined customer requirements (R) can then be the basis of the generated question, and the calculated sets of requirement variable assignments (RV) are the correct answers, while faulty answers can be derived from other instantiations of the requirement variables in RV i.e. possible sets of requirement variable assignments that satisfy $\neg RV = \{\neg RV_1$ and $\neg RV_2 ...$ and $\neg RV_m\}$[2] where $\neg RV \cup R \cup C$ is inconsistent.

Example. Given the configuration task definition of our working example, a set of customer requirements could be R = $\{wr = low$, $di = shortterm\}$. The set of corresponding correct answers is RV=$\{rr = low\}$, and faulty answers can be $\neg RV$=$\{\{rr = medium\}, \{rr = high\}\}$. The corresponding generated question would be: ***Given the following customer requirements ..., which additional requirement(s) can be added such that at least one configuration (solution) can still be identified?***

3.2 Product knowledge related question generation

The underlying task is to figure out which sets of customer requirements (R) fit a given product/item ($itemname = p$). In this context, a constraint solver (configurator) is able to calculate configurations that satisfy $C \cup \{itemname = p\}$ i.e. all possible sets of customer requirements R =$\{R_1, R_2, ..., R_n\}$ ($R_i = \{r_1, r_2, ..., r_m\}$) where $R \cup C \cup \{itemname = p\}$ is consistent. The given product/item ($itemname = p$) can then be the basis of the generated question, and the corresponding sets of customer requirements R represent the correct answer(s). Faulty answers can be represented by other instantiations of customer requirements that satisfy $\neg R = \{\neg R_1$ and $\neg R_2 ...$ and $\neg R_n\}$[3] where $\neg R \cup C \cup \{itemname = p\}$ is inconsistent.

Example. Given the configuration task definition of our working example, the product/item could be $\{itemname = bankbook\}$. The corresponding collection of consistent customer requirements is R= $\{R_1 : \{r_1 : wr = low, r_2 : di = shortterm, r_3 : rr = low\}$, $R_2 : \{r_1 : wr = medium, r_2 : di = shortterm, r_3 : rr = low\}$, $R_3 : \{r_1 : wr = low, r_2 : di = mediumterm, r_3 : rr = low\}$, $R_4 : \{r_1 : wr = medium, r_2 : di = mediumterm, r_3 : rr = low\}$, $R_5 : \{r_1 : wr = low, r_2 : di = longterm, r_3 : rr = low\}$, $R_6 : \{r_1 : wr = medium, r_2 : di = longterm, r_3 : rr = low\}\}$, i.e., in this example there exist *6* sets R_i of customer requirements that are consistent with the selected product/item $bankbook$. Examples of faulty answers are: $\{wr = high, di = longterm, rr = medium\}$, $\{wr = medium, di = longterm, rr = high\}$, $\{wr = high, di = longterm, rr = high\}$. The corresponding generated question would be: ***Given the following product/item ..., which sets of customer requirements are consistent with this product/item?***

3.3 Inconsistency knowledge related question generation

(1) The underlying task is to figure out which sets of customer requirements (R) trigger an inconsistency with a preselected product/item ($itemname = p$). More formally, inconsistency knowledge related questions can be generated on the basis of a configuration task definition (V, D, C, R) where all combinations of customer requirements have to be determined that never entail $\{itemname = p\}$. A constraint solver (configurator) is able to calculate configurations that satisfy $C \cup \{itemname = p\}$ i.e. all possible sets of customer requirements R =$\{R_1, R_2, ..., R_n\}$ ($R_i = \{r_1, r_2, ..., r_m\}$) where $R \cup C \cup \{itemname = p\}$ is consistent. The given product/item ($itemname = p$) can then be the basis of the generated question, and the corresponding sets of customer requirements R represent the faulty answer(s), while correct answers are other instantiations of customer requirements that satisfy $\neg R = \{\neg R_1$ and $\neg R_2 ...$ and $\neg R_n\}$ where $\neg R \cup C \cup \{itemname = p\}$ is inconsistent.

Example. In our working example, we can preselect the product/item $\{name = equityfund\}$. Combinations of customer requirements that do not entail $\{name = equityfund\}$ are all possible combinations with the exception of the following two combinations (faulty answers): R=$\{R_1$=$\{wr = high$, $di = longterm$, $rr = medium\}$, R_2=$\{wr = high, di = longterm, rr = high\}\}$. The corresponding generated question would be: ***Given the following product/item ... which combination(s) of customer requirements does not lead to this product/item?***

(2) Assuming that the set of customer requirements R is inconsistent with C, the underlying task is to figure out which minimal sets of R are inconsistent with C. More formally, this type of questions can be generated on the basis of a conflict detection task (C, R). The

[2] If $RV_i = \{v_1 = a_1, v_2 = a_2, ..., v_n = a_n\} \rightarrow RV_i = \{v_1 = a_1 \wedge v_2 = a_2 \wedge ... \wedge v_n = a_n\} \rightarrow \neg RV_i = \{\neg v_1 = a_1 \vee \neg v_2 = a_2 \vee ... \vee \neg v_n = a_n\}$

[3] If $R_i = \{r_1, r_2, ..., r_m\} \rightarrow R_i = \{r_1 \wedge r_2 \wedge ... \wedge r_m\} \rightarrow \neg R_i = \{\neg r_1 \vee \neg r_2 \vee ... \vee \neg r_m\}$

conflict detection task (C, R) can be used for question representation, and related correct answers are represented by the conflict sets CS_i (calculated on the basis of a conflict detection algorithm, such as QUICKXPLAIN algorithm [11]). Faulty answers can be derived from the calculated minimal conflict sets CS_i by taking subsets or supersets of it, since a subset of a minimal conflict set is not a conflict and a superset of a minimal conflict set is not a minimal conflict. For example, if CS_i = $\{r_a, r_b\}$ is a minimal conflict set, then $\{r_a\}$ and $\{r_b\}$ are non-conflicts and $\{r_a, r_b, r_c\}$ is not a minimal conflict set.

Example. Given the configuration task definition of our working example with the following inconsistent set of customer requirements R = $\{r_1 : wr = low, r_2 : di = shortterm, r_3 : rr = high\}$. Alternative minimal sets of customer requirements that are inconsistent with C are: $\{CS_1 = \{r_1, r_3\}\}$ and $\{CS_2 = \{r_2, r_3\}\}$. Examples of faulty answers derived from the conflict set CS_1 are $\{r_1\}$, $\{r_3\}$, and $\{r_1, r_2, r_3\}$. The corresponding generated question would be: ***Given the following configuration task definition... which one is a minimal set of requirements from which no corresponding configuration (solution) can be identified?***

(3) Given an incomplete but consistent set of customer requirements $R = \{r_1, r_2, ..., r_n\}$, the underlying task is to figure out what additional requirements induce inconsistency if have been added to R. More formally, this type of inconsistency knowledge related questions can be generated on the basis of a configuration task definition (V, D, C, R). On the basis of such a definition, a constraint solver (configurator) is able to calculate all sets of requirement variable assignments (RV) that satisfy $R \cup C$ ($RV = \{RV_1, RV_2, ..., RV_m\} : RV_i = \{v_1 = a_1, v_2 = a_2, ..., v_n = a_n\}$ $where$ $v_i \in V, a_i \in domain(v_i), v_i = a_i \notin R,$ and $RV \cup R \cup C$ is $consistent$). The set of defined customer requirements (R) can then be the basis of the generated question, and the calculated sets of requirement variable assignments (RV) are the faulty answers, while correct answers can be derived from other instantiations of the requirement variables in RV i.e. possible sets of requirement variable assignments that satisfy $\neg RV = \{\neg RV_1$ and $\neg RV_2 ...$ and $\neg RV_m\}$ where $\neg RV \cup R \cup C$ is inconsistent.

Example. Given the configuration task definition of our working example, and a set of customer requirements R = $\{r_1 : wr = low, r_2 : di = shortterm\}$. The set of corresponding faulty answers is RV=$\{rr = low\}$, while correct answers can be $\neg RV$=$\{\{rr = medium\}, \{rr = high\}\}$. The corresponding generated question would be: ***What additional requirements trigger an inconsistency (lead to no solution) if have been added to the following set of requirements ....?***

3.4 Analysis knowledge related question generation

(1) Assuming that the set of customer requirements (R) is inconsistent with C, the underlying task is to figure out which minimal sets of R that have to be deleted or adapted such that consistency can be restored (a solution can be found). More formally, analysis knowledge related questions can be generated on the basis of a diagnosis task definition (C, R). The diagnosis task definition can be used for question representation, and the related correct answers are represented by the determined minimal diagnoses Δ_i (calculated on the basis of a diagnosis detection algorithm such as FASTDIAG [4]). Faulty answers can be identified based on the calculated minimal diagnoses by taking subsets or supersets of it, since a subset of a minimal diagnosis is not a diagnosis and a superset of a minimal diagnosis is not a minimal diagnosis. For example, if Δ_i = $\{r_a, r_b, r_c\}$ is a minimal diagnosis, then $\{r_a, r_b\}$ is a non-diagnosis and $\{r_a, r_b, r_c, r_d\}$ is not a minimal diagnosis.

Example. Given the configuration task definition of our working example with the following set of customer requirements R = $\{r_1 : wr = low, r_2 : di = shortterm, r_3 : rr = high\}$. The corresponding alternative minimal sets of customer requirements (diagnoses Δ_i) that have to be deleted from R or adapted such that a configuration can be identified, are the following: $\{\Delta_1 = \{r_1, r_2\}, \Delta_2 = \{r_3\}\}$. For example, deleting (or adapting) the requirement r_3 restores consistency between R and C, i.e., allows the calculation of a solution. Examples of faulty answers derived from the diagnosis Δ_1 are $\{r_1\}$, $\{r_2\}$, and $\{r_1, r_2, r_3\}$. The corresponding generated question would be: ***Given the following configuration task definition ... which one is a minimal set of requirements that has to be deleted or adapted such that a configuration (solution) can be identified?***

(2) Assuming that the set of customer requirements (R) is inconsistent with C, the underlying task is to figure out which maximal sets of requirements from R guarantee the identification of a solution. More formally, this type of questions can be generated either (I) on the basis of a diagnosis task definition (C, R) where the related correct answers are represented by R - Δ where Δ is a minimal cardinality diagnosis or (II) on the basis of finding a *maximal satisfiable subset* (MSS) - see the next definition - where the correct answers are represented by the MSS. Faulty answers are represented by subsets or supersets of the MSS.

Definition (Maximal Satisfiable Subset "MSS"). A subset $M \subset R$ is an MSS iff $M \cup C$ is consistent and $\forall r_i \in \{R - M\} \Rightarrow M \cup \{r_i\} \cup C$ is inconsistent.

Example. Given the configuration task definition of our working example with the same inconsistent set of customer requirements used in the previous example: R = $\{r_1 : wr = low, r_2 : di = shortterm, r_3 : rr = high\}$ and the calculated minimal cardinality diagnosis $\Delta_2 = \{r_3\}$. The corresponding maximal set of requirements from R that allows the calculation of a solution is $\{R$ - Δ_2=$\{r_1 : wr = low, r_2 : di = shortterm\}$, which represents also the MSS. Examples of faulty answers are $\{wr = low\}$, $\{di = shortterm\}$, and $\{wr = low, di = shortterm, rr = high\}$. The corresponding generated question would be: ***Given the following configuration task definition ... which one is a maximal set of requirements that allows the identification of a configuration (solution)?***

(3) Assuming that the set of customer requirements (R) is inconsistent with C, the underlying task is to find out what minimal diagnoses can be derived from predefined minimal conflict sets (with which its deletion or adaptation, a solution can be found). More formally, this type of questions can be generated on the basis of a conflict detection task (C, R). The conflict detection task with the calculated minimal conflict sets CS_i can then be used for question representation, and the correct answers are represented by resolving the conflict sets using a proper algorithm such as HSDAG [15], which leads to the determination of the minimal diagnoses Δ_i. Faulty answers can be identified based on the calculated minimal diagnoses by taking subsets or supersets of it, since a subset of a minimal diagnosis is not a diagnosis and a superset of a minimal diagnosis is not a minimal diagnosis. This type of questions has the same goal of the analysis knowledge related questions defined in *(1)*, which is finding out the minimal diagnoses. The difference is that in *(1)*, the task is to find out the minimal diagnoses directly from an inconsistent set

of customer requirements (R), while here the task is to find out the minimal diagnoses from predefined conflict sets (CS_i).

Example. Given the configuration task definition of our working example with the following inconsistent set of customer requirements $R = \{r_1 : wr = low, r_2 : di = shortterm, r_3 : rr = high\}$ and the following conflict sets $\{CS_1 = \{r_1, r_3\}, CS_2 = \{r_2, r_3\}\}$. The corresponding derived diagnoses Δ_i that have to be deleted from R or adapted such that a solution can be identified, are the following: $\{\Delta_1 = \{r_1, r_2\}, \Delta_2 = \{r_3\}\}$. Examples of faulty answers derived from the diagnosis Δ_1 are $\{r_1\}$, $\{r_2\}$, and $\{r_1, r_2, r_3\}$. The corresponding generated question would be: ***Given the following configuration task definition ... what minimal diagnoses can be derived from the following minimal conflict sets ...?***

3.5 Repair knowledge related question generation

Assuming that the set of customer requirements (R) is inconsistent with C, the underlying task is to figure out what changes should be applied on the requirements so that at least one configuration (solution) can be identified, i.e. to define concrete repair actions for the requirements included in a diagnosis. The identification of repair actions starts with the calculation of minimal diagnoses, assuming that repair alternatives with low-cardinality changes are favored compared to the ones with higher number of changes (repairs on minimal cardinality diagnoses). More formally, repair knowledge related questions can be generated on the basis of a repair task definition (C, R). The repair task definition can be used for question representation, and the related correct answers are represented by the repair actions (RA) calculated from the determined minimal cardinality diagnosis Δ where $R - \Delta \cup RA$ is consistent with C (i.e. allows the identification of a configuration). Faulty answers can be derived from other instantiations of the requirement variables in RA ($\neg RA : R - \Delta \cup \neg RA \cup C$ *is inconsistent*).

Example. Given the configuration task definition of our working example with the following inconsistent set of customer requirements $R = \{r_1 : wr = low, r_2 : di = shortterm, r_3 : rr = high\}$. The corresponding alternative minimal sets of customer requirements (diagnoses Δ_i) that have to be adapted such that a configuration (solution) can be identified, are the following: $\{\Delta_1 = \{r_1, r_2\}, \Delta_2 = \{r_3\}\}$. The corresponding repair action that restores consistency is, based on the minimal cardinality diagnosis Δ_2, $RA = \{r'_3 : rr = low\}$ i.e., $\{R - \Delta_2 \cup RA\}$ allows the identification of a configuration. Example of a faulty answer is $\{rr = medium\}$. The corresponding generated question would be: ***Given the following configuration task definition ... which one is a minimal change in the requirements that leads to the identification of a configuration (solution)?***

4 Evaluation

We implemented our question generation approach, based on CHOCO solver [4], and tested it on five different knowledge bases including the knowledge base of our working example (the financial services knowledge base). The tests have been executed on a standard desktop computer Intel(R) Core(TM) i5-2320 CPU 3.00GHz with 8GB RAM, and included all types of generated questions with a performance evaluation for the question generation approach (the average time needed to generate the questions). The knowledge bases we ran our tests on are relatively small, but we will do more tests and evaluations for our question generation approach on bigger and more complex knowledge bases in the future. Now, we will present a performance evaluation for some types of the generated questions.

In our implementation, we used for the calculation of minimal diagnoses both algorithms (for performance comparison), the standard hitting set directed acyclic graphs algorithm HSDAG [15] and FASTDIAG algorithm [4]. FASTDIAG complexity in terms of the number of needed consistency checks for calculating one minimal diagnosis is $O(log_2(\frac{n}{k}) + 2k)$ in the best case (all elements of the diagnosis are contained in one path of the search tree) and $O(2k.log_2(\frac{n}{k}) + 2k)$ in the worst case (each element of the diagnosis is contained in a different path of the search tree) where k is the minimal diagnosis set size and n is the number of constraints in R.

The calculation of minimal conflict sets is based on QUICKXPLAIN algorithm [11]. QUICKXPLAIN complexity in terms of needed consistency checks for calculating one minimal conflict set is $O(log_2(\frac{n}{k}) + 2k)$ in the best case, and $O(2k.log(\frac{n}{k}) + 2k)$ in the worst case where k is the minimal conflict set size and n is the number of constraints in R. So the number of consistency checks needed for calculating a conflict set (QUICKXPLAIN) and the number of consistency checks needed for calculating a diagnosis (FASTDIAG) fall into a logarithmic complexity class.

We measured the runtime performance of calculating the first minimal diagnosis on different sets of user requirements (of cardinality 3, 5, 7, and 9 requirements) using FASTDIAG algorithm – for diagnosis related questions. The results are depicted in Figure 1. Also the results of a performance evaluation of diagnoses (analysis) related questions (runtime depending on the number of calculated diagnoses) are depicted in Figure 2, where we did a runtime performance comparison between FASTDIAG and HSDAG algorithm for calculating number of diagnoses between 2 and 6. The results show that FASTDIAG outperforms HSDAG in the time needed to calculate diagnoses.

The used knowledge base is a simple web hosting services knowledge base with a total of 16 constraints, 10 variables, and a varying number of conflict sets (of cardinality 1–2) and corresponding diagnoses (between 1–6 diagnoses). Each test has been conducted 10 times with a differing constraint ordering. The average runtime is measured in milliseconds (ms).

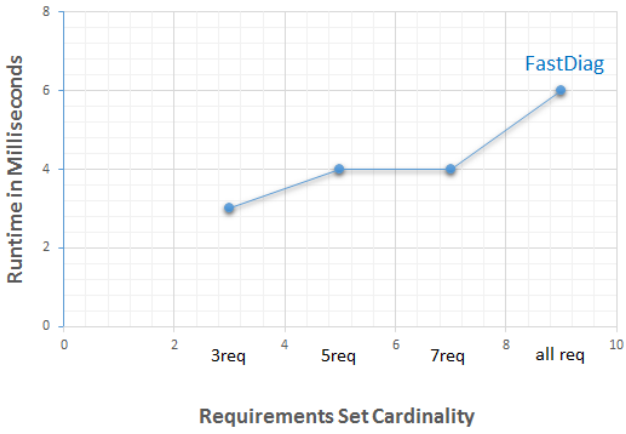


Figure 1. RUNTIME EVALUATION: The average runtime in milliseconds (ms) needed to calculate the first minimal diagnosis with FASTDIAG for 3, 5, 7, and 9 user requirements (req) – for diagnoses related questions.

In the same way, we evaluated the performance of the conflicts

[4] www.choco-solver.org

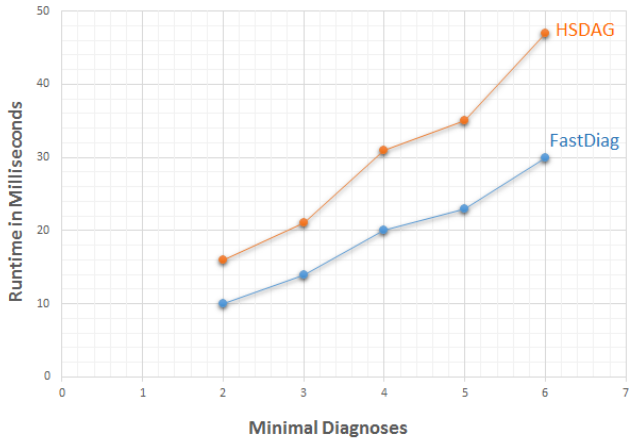


Figure 2. RUNTIME EVALUATION: The average runtime in milliseconds (ms) needed to calculate 2–6 minimal diagnoses with FASTDIAG vs. HSDAG on 9 requirements set – for diagnoses related questions.

(inconsistency) related questions (runtime depending on the number of calculated conflict sets) using QUICKXPLAIN algorithm. We measured the runtime performance of calculating the first minimal conflict set on different sets of user requirements (of cardinality 3, 5, 7, and 9 requirements), then we measured the runtime performance of calculating number of conflict sets between 2 and 6. The results are depicted in Figure 3 and Figure 4.

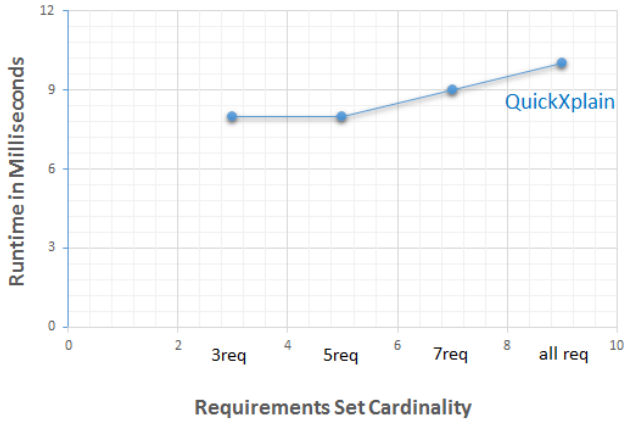


Figure 3. RUNTIME EVALUATION: The average runtime in milliseconds (ms) needed to calculate the first minimal conflict set with QUICKXPLAIN for 3, 5, 7, and 9 user requirements (req) – for conflicts related questions.

Two major factors would influence the performance of QUICKXPLAIN [6]. ***(1)*** The size of conflict sets – the more elements in the conflicts, the more consistency checks are needs for determining one minimal conflict set. ***(2)*** The ordering of the constraints in C – the more constraints are spread over C, the more consistency checks can be expected since less constraints can be omitted in early phases of QUICKXPLAIN execution.

Our experiments showed a very good performance for the question generation approach we presented in this paper but as we mentioned before, all our tests have been conducted on small knowledge bases. More tests and evaluations will be conducted, in future, on bigger and more complex knowledge bases. The evaluations will not include runtime performance only but also evaluation of the quality of the generated questions.

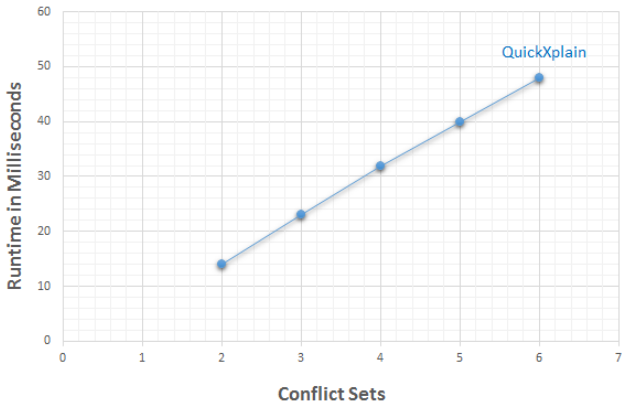


Figure 4. RUNTIME EVALUATION: The average runtime in milliseconds (ms) needed to calculate 2–6 minimal conflict sets with QUICKXPLAIN on 9 requirements set – for conflicts related questions.

5 Future Work

Especially in the context of educating sales representatives, automated question generation becomes an essential and required functionality, since it reduces the overheads of manual question generation and management, besides its added benefits related to supporting sales representatives in strengthening their product and sales knowledge, and hence having better advisory services.

In future, we will test and evaluate our approach on larger and more complex knowledge bases, where the number of calculated diagnoses and repair actions could become very large, which makes the identification of acceptable diagnoses and repair actions a difficult task for the user. For this, we will exploit the approaches we mentioned in Section 2 to reduce the number of calculated diagnoses, and to personalize diagnoses, in order to identify only the diagnoses that are more relevant to the user. Consequently, new question types will be added such as (for example):

- Given the following configuration task definition ... identify the minimal cardinality sets of requirements that have to be deleted or adapted such that a solution can be identified.
- Given the following configuration task definition ... which one is a minimal diagnosis set with a higher probability to be selected by the user?

Also with the growth of knowledge bases, the number of identified solutions (configurations) might become very large, which makes it difficult for the user to select the most relevant solution(s). For this, approaches to rank solutions according to their relevance to the user (such as utility-based ranking mechanisms [10]), and approaches to identify a representative set of solutions to be displayed to the user, will be studied and used in our question generation approach. Besides that, the number of generated questions could become very high that selecting a representative set from it to be displayed to the user will be needed. In this context, we will analyze potential synergies with existing approaches to reduce test cases in the context of regression testing. We will provide a brief look at some of these approaches in the remainder of this section, but we will analyze it in depth in future.

Regression testing [12] in software engineering is carried out to ensure that any changes made in the software (in the fixes or any enhancement changes) don't influence the previously working functionality. Usually it is done by re-running existing test cases on the

modified code to determine whether the changes affect anything. This process is costly and time-intensive. Instead of re-running all the existed test cases, a number of different approaches were studied to reduce the number of selected test cases (selecting an optimal subset of test cases from the initial test suite to minimize the testing time, cost and effort).

The three major studied approaches for reducing the number of test cases include *test suite minimization*, *test case selection*, and *test case prioritisation* [18]. Test suite minimization seeks to eliminate redundant test cases in order to reduce the number of tests to run, and test case selection seeks to identify the test cases that are relevant to some set of recent changes, while test case prioritization seeks to order test cases in such a way that early fault detection is maximized (finding the optimal permutation of the sequence of test cases).

We will provide in future also coverage metrics that indicate the quality of the generated questions. In this context, we will analyze coverage metrics used in test case generation in software engineering, such as, *Test Coverage* and *Test Case Effectiveness* metrics. Test coverage provides an indication of the completeness of the test suite. The coverage can be with respect to the requirements, functionality, use cases, etc (in our case, it can be in terms of configuration coverage, variable domains coverage, generated questions coverage, etc); and test case effectiveness provides an indication of the effectiveness of the test cases and the stability of the software (in our case, effectiveness of the generated questions).

Finally, we will do an evaluation of the effectiveness and applicability of our question generation approach in a real life scenario.

6 Conclusions

In this paper, we introduced an approach for automated question generation from configuration knowledge bases. To the best of our knowledge, this approach hasn't been discussed before. We showed how configuration task definitions can be exploited for the generation of different types of questions, to be used for training and education purposes such as educating sales representatives. We evaluated our approach with regard to runtime performance of question generation. Finally, we pointed out further additions and improvements as issues for future work.

REFERENCES

[1] M. Agarwal and M. Prashanth, 'Automatic gap-fill question generation from text books', in *Proceedings of the 6th Workshop on Innovative Use of NLP for Building Educational Applications*, IUNLPBEA '11, pp. 56–64, Stroudsburg, PA, USA, (2011). Association for Computational Linguistics.

[2] T. Alsubait, B. Parsia, and U. Sattler, 'Generating multiple choice questions from ontologies: Lessons learnt', in *Proceedings of the 11th International Workshop on OWL: Experiences and Directions (OWLED 2014) co-located with 13th International Semantic Web Conference on (ISWC 2014), Riva del Garda, Italy, October 17-18, 2014.*, pp. 73–84, (2014).

[3] A. Felfernig and R. Burke, 'Constraint-based recommender systems: Technologies and research issues', in *Proceedings of the 10th International Conference on Electronic Commerce*, ICEC '08, pp. 3:1–3:10, New York, NY, USA, (2008). ACM.

[4] A. Felfernig, M. Schubert, and C. Zehentner, 'An efficient diagnosis algorithm for inconsistent constraint sets', *Artif. Intell. Eng. Des. Anal. Manuf.*, **26**(1), 53–62, (February 2012).

[5] A. Felfernig, A. Shehadeh, M. Jeran, C. Gütl, T. Tran, M. Atas, S. Polat, M. Stettinger, A. Akcay, and S. Reiterer, 'Studybattles: A learning environment for knowledge-based configuration', in *Proceedings of the 18th International Configuration Workshop*, pp. 109–116, Toulouse, France, (September 2016).

[6] Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, *Knowledge-based Configuration: From Research to Business Cases*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1 edn., 2014.

[7] Alexander Felfernig, Monika Schubert, Gerhard Friedrich, Monika Mandl, Markus Mairitsch, and Erich Teppan, 'Plausible repairs for inconsistent requirements', in *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pp. 791–796, Pasadena, California, USA, (2009).

[8] Alexander Felfernig, Monika Schubert, and Stefan Reiterer, 'Personalized diagnosis for over-constrained problems', in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pp. 1990–1996. AAAI Press, (2013).

[9] C. Gütl, K. Lankmayr, J. Weinhofer, and M. Höfler, 'Enhanced automatic question creator eaqc: Concept, development and evaluation of an automatic test item creation tool to foster modern e-education', *Electronic Journal of e-Learning*, **9**, (2011).

[10] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich, *Recommender Systems: An Introduction*, Cambridge University Press, New York, NY, USA, 1st edn., 2010.

[11] Ulrich Junker, 'Quickxplain: Preferred explanations and relaxations for over-constrained problems', in *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, pp. 167–172. AAAI Press, (2004).

[12] Passant Kandil, Sherin Moussa, and Nagwa Badr, 'A study for regression testing techniques and tools', *International Journal of Soft Computing and Software Engineering [JSCSE]*, **5**(4), 64–84, (2015).

[13] Mukta Majumder and Sujan Kumar Saha, 'A system for generating multiple choice questions: With a novel approach for sentence selection', in *Proceedings of the 2nd Workshop on Natural Language Processing Techniques for Educational Applications*, pp. 64–72, Beijing, China, (July 2015). Association for Computational Linguistics.

[14] K. Mazidi and P. Tarau, *Automatic Question Generation: From NLU to NLG*, 23–33, Springer International Publishing, Cham, 2016.

[15] R. Reiter, 'A theory of diagnosis from first principles', *AI Journal*, **23**(1), 57–95, (1987).

[16] A. Tahani, B. Parsia, and U. Sattler, *Knowledge Engineering and Knowledge Management: EKAW 2014 Satellite Events, VISUAL, EKM1, and ARCOE-Logic, Linköping, Sweden, November 24-28, 2014. Revised Selected Papers.*, chapter Generating Multiple Choice Questions From Ontologies: How Far Can We Go?, 66–79, Springer International Publishing, Cham, 2015.

[17] C. Yllias and H.A. Sadid, 'Towards topic-to-question generation', *Comput. Linguist.*, **41**(1), 1–20, (March 2015).

[18] Shin Yoo and Mark Harman, 'Regression testing minimisation, selection and prioritisation - a survey', Technical Report TR-09-09, King's College London, (October 2009).

ASP-based Knowledge Representations for IoT Configuration Scenarios

Alexander Felfernig[1] and **Andreas Falkner**[2] and
Müslüm Atas[1] and **Seda Polat Erdeniz**[1] and **Christoph Uran**[1] and **Paolo Azzoni**[3]

Abstract. The purpose of this paper is to introduce basic application scenarios for configuration technologies in Internet of Things (IoT) product domains. We show how to represent configuration knowledge in the domain of *smart homes* on the basis of Answer Set Programming (ASP). In this context, we introduce different configuration model elements and constraint types and show their corresponding ASP representation in a way that is also useful for ASP beginners. We conclude the paper with a discussion of open issues for future work.

1 Introduction

Configuration is one of the most successfully applied Artificial Intelligence technologies [6, 19]. It is a specific type of design activity where a product is configured on the basis of a set of already defined component types and corresponding constraints that restrict the way in which component instances can be combined. A *configuration task* is defined in terms of a generic product structure, a corresponding set of constraints, and a set of requirements (often also denoted as customer requirements) that additionally restrict the set of possible solutions. A solution (configuration) for a configuration task is represented by a set of component instances, their connections and attribute settings which altogether are consistent with the constraints and requirements included in the configuration task definition.

There is a multitude of application domains for knowledge-based configuration – example domains are the automotive sector, financial services, operating systems, software product lines, and railway interlocking systems [6]. Configuration technologies nowadays become increasingly popular in different kinds of Internet of Things (IoT) [1] scenarios. The Internet of Things is an emerging paradigm that envisions a networked infrastructure which enables the interconnection of devices (things) anyplace and anytime. In the IoT context, configurators can be applied, for example, to the identification of ramp-up configurations (self-configuration), i.e., to figure out which components (potential software and hardware) are needed in a certain IoT setting. Configuration technologies can also be applied during runtime where, for example, a configurator helps to identify pareto-optimal configurations of communication protocols with regard to criteria such as *performance* and *cost of data transfer*.

The size and complexity of configuration problems in the IoT domain often does not allow the application of basic configuration knowledge representation and reasoning such as constraint satisfaction [23]. Smart homes often consist of hundreds or even thousands of different components and constraints – such scenarios are in the need of a component-oriented knowledge representation that is easy to use and maintain [6, 13]. Open source constraint-based approaches do not support such a representation and existing component-oriented commercial environments are based on proprietary knowledge representations with limitations also in terms of standardization. An alternative to constraint-based knowledge representations especially useful for large and complex configuration domains is Answer Set Programming (ASP) [12, 18]. ASP supports the definition of component hierarchies and related constraints in a declarative way (which is not possible with basic CSP-based configuration environments). Potential component instances have to be pre-defined, i.e., in its basic form ASP does not support pure component generation during runtime.

There exist a couple of research contributions related to the application of answer set programming in the configuration context. Soininen and Niemelä [18] can be considered as pioneers who first showed the application of ASP to represent and solve configuration tasks. A resulting configuration environment is presented, for example, in [6, 22]. An object-oriented layer to answer set programs has been introduced by [4]. In this work, configuration tasks can be represented on an object-oriented level without being forced to take into account specific details of ASP-based configuration knowledge representations. Thus, this work can be seen as a contribution to improve the applicability of ASP technologies especially in terms of reducing efforts related to knowledge base development and maintenance. Feature model related ASP representations are introduced in [16]. An approach to the testing of object-oriented models on the basis of ASP is introduced in [5]; in this context it is shown how UML-based configuration knowledge representations can be represented in ASP and how positive and negative test cases can be represented and included for the purpose of supporting unit tests on knowledge bases. Friedrich et al. [8] introduce an approach to re-configuration in ASP – in this context, a reconfiguration can be considered as a set of changes to an already existing configuration such that new requirements are taken into account. Finally, Teppan et al. [21] introduce a hybrid approach that integrates constraint solving with ASP. A major advantage of this integration is that the grounding bottleneck[4] in answer set programming can be transformed into a more efficiently solvable search problem in constraint programming.[5] The major fo-

[1] Institute for Software Technology, Graz University of Technology, Inffeldgasse 16b/2, A-8010 Graz, email:{a.felfernig, muesluem.atas, spolater, christoph.uran}@ist.tugraz.at
[2] Siemens AG Österreich, email:andreas.a.falkner@siemens.com
[3] Eurotech Group, Italy, email: paolo.azzoni@eurotech.com

[4] See Section 4.
[5] Although CSPs often do not support flexible (component-oriented) knowledge representations, they have the potential to support ASP-based reasoning processes (e.g., in terms of increasing efficiency).

cus of this paper are ASP-based knowledge representations. For an overview of different further approaches to configuration knowledge representation we refer to [13].

The contributions of this paper are the following. First, we introduce ASP-based configuration knowledge representations in the context of Internet Of Things (IoT) scenarios. Second, our aim is to provide easy to understand examples (for ASP newbies) of how to represent configuration knowledge in ASP and also to show limitations of ASP knowledge representations. Third, we discuss different issues for future research that will help to accelerate a broad application of ASP technologies in knowledge-based configuration.

The remainder of this paper is organized as follows. In Section 2 we discuss IoT-related configuration domains and introduce a smart home configuration model which is used as working example throughout the paper. In Section 3 we show how to translate individual model elements into a corresponding ASP-based representation. In this context we also sketch how ASP solvers operate to determine a solution for a configuration task (Section 4). In Section 5 we sketch the role of ASP solving in our IoT-related research project. The paper is concluded with a discussion of open research issues (Section 6).

2 IoT Domains and Configuration Models

In the following we provide an overview of IoT domains where the application of ASP-based configuration technologies is reasonable. In this context, we introduce a simplified configuration model from the domain of smart homes in order to show different facets of ASP-based configuration knowledge representations.

Air Pollution Monitoring. Air pollution monitoring systems help to ensure healthy living conditions, for example, in cities. An issue in this context is the distribution of sensors in a city topology that assures a representative collection of measurement data. This data is analyzed on the basis of different types of learning algorithms that help to figure out in which contexts which actions have to be triggered. Examples of related actions are a general warning to leave the house, reduced speed limits on highways, recommendations to groups (e.g., schools) in terms of the maximum time that should be spent outdoors, and warnings regarding the malfunctioning of filter equipments in industrial production. In air pollution monitoring, configuration technologies can be used to select the type and placement of sensors given a specific topology (e.g., a topology of a city) and also to select the types of algorithms that should be used for data analysis in certain contexts.

Health Monitoring. Health monitoring solutions can be based on different types of data that can be used to determine recommendations related to factors such as eating behavior, sports activities, sleeping times, and also data about the body condition. Many tools already allow the manual entering of consumed food, however, in future scenarios such information will be available on the basis of standardized data exchange protocols. Measurement of sports activities and sleeping time is already included in many commercial solutions. Finally, detailed information about the physical condition of a person is not taken into account in many of the existing tools. In such scenarios, configuration technologies can be used to parametrize the underlying algorithms, for example, recommended heart rates when doing physical practices depend on the age, gender, and weight of a person (and further physical parameters). Whether specific food items can be recommended or not depends, for example, on potential allergies of a person. Finally, especially in group sports (e.g. football or tennis), the type of training also depends on the participating persons. For example, if three persons are participating in a tennis training session and one person has a bad physical condition, this has an impact on the selection of exercise units for this group.

Energy Production and Management. Energy production is in the need of configuration technologies in various scenarios, for example, wind turbines must be configured and parametrized in order to be able to maximize energy production in a certain environment. Knowledge about where and when a higher amount of energy will be needed can trigger a corresponding reconfiguration of the load factor of water reservoirs. In the context of private energy production, configuration technologies can help to rearrange energy consumption times, for example, when to recharge the electric car or when to activate the washing machine. Especially in the context of energy management in buildings, reconfiguration technologies can play a role by supporting the change of building parameters depending on given environmental data such as temperature, weather conditions, and forecasts.

Enhanced Retail Services. In-store shopping is based on specific distributions of sensors and other devices such as information displays. During the ramp-up phase of such an application it has to be assured that customer location sensors are distributed in a reasonable fashion and information displays are positioned in such a way that the information can be easily seen by customers. In such scenarios, configuration technologies can be applied in order to determine the amount of sensors needed, the positioning of information displays, and also to determine the layout of the whole shop depending on the product assortment that should be offered to a customer.

Animal Monitoring. There exist a couple of scenarios where information about animal locations and information about the physical condition of animals is important. For example, in wildlife scenarios where animals are spread over huge and not accessible areas, it is important to provide an infrastructure for animal monitoring that is not based on physical presence of human administrators. In such contexts, for example, different types of drones can be used to support data collection. Depending on the region size and topography and requirements regarding the amount of data to be collected, drones have to be configured in a way that optimizes the trade-off between energy consumption, range, and support of the defined data collection requirements. In such scenarios, configuration technologies can be useful to support the complete configuration of the needed data collection equipment.

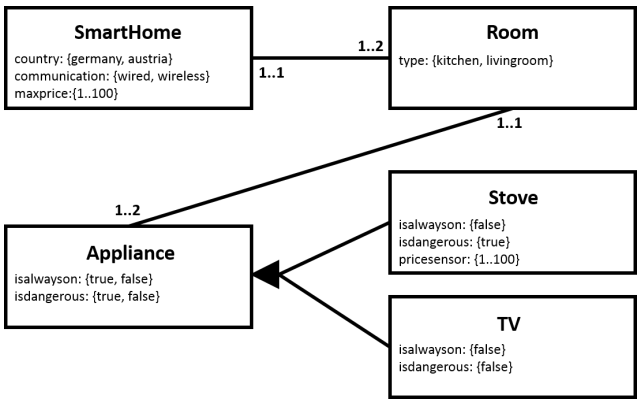


Figure 1. Simplified configuration model (reduced #component types, #attributes, domains, and multiplicities) of a *smart home* used for demonstration purposes. Additional constraints are presented in our discussion of ASP-based configuration knowledge representations.

Smart Homes. A simplified smart home configuration model is depicted in Figure 1. Smart homes [15] include functionalities for actively supporting persons in their daily life. Examples thereof are *intelligent light management* that allows to (semi-automatically) adapt the illumination of rooms depending on the time of the day and season, *energy management* that supports intelligent air conditioning for whole buildings, *security management* (e.g., when nobody is at home), and functionalities related to the support of *ambient assisted living* scenarios with related functionalities such as automated fall detection. In such scenarios, configuration technologies can be used to design in detail which smart home hardware and software components (e.g., sensors, actuators, and apps) have to be installed in which part of the building. The smart home model depicted in Figure 1 will serve as working example in this paper to demonstrate ASP-based configuration knowledge representations.

3 Configuration Knowledge Representation in ASP

In the following we will show how to represent configuration knowledge for a simplified configuration model from the domain of smarthomes (see Figure 1).[6]

(a) *Potential Component Instances*. In the ASP context, all decision variables have to be pre-specified. In our example model shown in Figure 1, three different component types are included which are represented by the predicate names *smarthome*, *room*, and *appliance* (with the subtypes *stove* and *room*). For these component types we have to specify the maximum amount of instances that can be part of a related smarthome configuration (see Figure 2), i.e., one *smarthome* (e.g., *psmarthome*(1) denotes a potential instance of *smarthome* with id 1), two instances of *room*, and 4 instances of each subtype of *appliance*. The integer number arguments (e.g., *proom(2;3)*) in the ASP facts serves as a unique key to distinguish the (potential) instances.

```
psmarthome(1).
proom(2;3).
pstove(4;5;6;7).
ptv(8;9;10;11).
```

Figure 2. Definition of potential smart home component instances using Answer Set Programming (ASP) notation. For example, *proom(2;3)* denotes two potential instances of type *room*.

(b) *Component Types and Instances*. For a configuration, it has to be decided which of the potential instances shall be included. Therefore we establish an association of the aforementioned definitions of potential instances with the "real" component instances that can be included in a configuration (see Figure 3). For each potential instance it has to be decided whether to include this as an instance in a configuration or not, for example in our configuration knowledge base, each potential room instance can be part of a configuration or not (this is specified by the corresponding lower and upper bounds of the corresponding rule).

```
0{ smarthome(X) }1 :- psmarthome(X).
0{ room(X) }1 :- proom(X).
0{ stove(X) }1 :- pstove(X).
0{ tv(X) }1 :- ptv(X).
```

Figure 3. Definition of component types (in ASP). For example, each potential instance of type room (i.e., *proom(X)*) can be part of a configuration as *room(X)*.

[6] For demonstration purposes we use the syntax of the *clingo* environment (see potassco.sourceforge.net).

(c) *Generalization Hierarchies*. Generalization hierarchies allow to further categorize different component types (see Figure 4). For demonstration purposes, we represent *stove* and *tv* as (disjunctive) subtypes of the component type *appliance* – an alternative to introducing a *type* attribute similar as for *room*. We also include a rule that assures that instances of appliances are instantiated to *stove* or *tv*. Note that disjunctiveness between subtypes is assured by definition (see Figures 2 and 3).

As we are not interested in incomplete configurations (e.g., instances of *appliance* that are not refined to *stove* or *tv*), we add a constraint that ensures that each *appliance* instance is either refined to a *stove* instance or a *tv* instance. This is only necessary when users are allowed, for example, to include component instances represented by corresponding facts (e.g., *appliance(4)*).

```
appliance(X) :- stove(X).
appliance(X) :- tv(X).
:- appliance(X), not stove(X), not tv(X).
```

Figure 4. Defining generalization hierarchies (in ASP). For example, each *stove* is an *appliance* and each *tv* is an *appliance*, and vice-versa, each *appliance* is either a *stove* or a *tv*.

(d) *Attributes*. For the defined component type attributes, we have to introduce attribute domain definitions (see Figure 5).

```
dommaxprice(1..100).
domcountry(germany;austria).
domcommunication(wired;wireless).
domtype(kitchen;livingroom).
domisalwayson(true;false).
domisdangerous(true;false).
dompriceofsensor(60).
```

Figure 5. Attribute domain definitions (in ASP). For example, *dommaxprice* represents an (integer) attribute that will be used to specify the maximum price of a *smarthome* solution. For simplicty, we only include price information related to *stoves* – see also Figure 6.

Attribute domain definitions have to be associated with the corresponding component types, for example, the domain definition *domcountry* of the attribute *country* has to be associated with the component type *smarthome* (see Figure 6). With ASP choice rules we enforce that each instance has exactly one domain value for each of its attributes. Generalization is covered in a natural way: see the right-hand-side of the last two rules in the figure. As attributes are created only for existing instances (but not for potential instances), spurious solutions (such as arbitrary attribute settings for unused potential instances and their combination) are avoided.

```
1{country(X,Y):domcountry(Y)}1:- smarthome(X).
1{communication(X,Y):domcommunication(Y)}1:- smarthome(X).
1{maxprice(X,Y):dommaxprice(Y)}1:-smarthome(X).
1{type(X,Y):domtype(Y) }1:- room(X).
1{isalwayson(X,Y):domisalwayson(Y)}1:- appliance(X).
1{isdangerous(X,Y):domisdangerous(Y)}1:- appliance(X).
1{priceofsensor(X,Y):dompriceofsensor(Y)}1:- stove(X).
```

Figure 6. Associating attributes with component types (in ASP). For example, *country* is an attribute associated with *smarthomes*. On an instance level, attribute instances are only generated if corresponding component instances exist, i.e., attribute instances are only created when necessary.

If domain definitions are reduced in subcomponent types, this can be expressed in a corresponding ASP rule, for example, *isdangerous(X,false):- tv(X).* expresses the fact that a *tv* set is not considered as a dangerous appliance (see Figure 7).

(e) *Associations and Multiplicities*. Associations between component types on the model level are represented in terms of binary predicates on the ASP level (see Figure 8). We use ASP rules to define

```
isalwayson(X,false) :- stove(X).
isdangerous(X,true) :- stove(X).
isalwayson(X,false) :- tv(X).
isdangerous(X,false) :- tv(X).
```

Figure 7. Reducing ASP attribute domain definitions, for example, in generalization hierarchies. For example, the *isdangerous* attribute of type *appliance* is reduced to *true* if the corresponding component is a *stove*.

potential links, e.g. *smarthomeroom*, with the allowed minimum and maximum multiplicities of the association.

```
1{ smarthomeroom(X,Y): room(Y) }2 :- smarthome(X).
1{ smarthomeroom(Y,X): smarthome(Y) }1 :- room(X).
1{ roomappliance(X,Y): appliance(Y) }2 :- room(X).
1{ roomappliance(Y,X): room(Y) }1 :- appliance(X).
```

Figure 8. Defining associations and corresponding multiplicities (in ASP). For example, each *smarthome* has 1-2 associated components of type *room*.

(f) *Incompatibility Constraints*. Such constraints typically specify incompatibilities regarding the combination of specific component types or simply combinations of incompatible attribute values. An example of an incompatibility is represented by the following constraint that expresses the fact that a *tv* should not be situated in a room of type *kitchen* (see Figure 9).

```
:- roomappliance(X,Y), type(X,kitchen), tv(Y).
```

Figure 9. Defining incompatibility constraints (in ASP). For example, no *tv* should exist in a *kitchen*.

(g) *Requires Constraints*. Requirements relationships describe situations where the integration of a certain component or the selection of a certain attribute value also requires the integration/selection of further component types/attribute values. An example constraint is the following: *smarthomes* in *Austria* must have two rooms (the corresponding ASP representation is shown in Figure 10).

```
2{ smarthomeroom(X,Y): room(Y) }2 :-
    smarthome(X), country(X,austria).
```

Figure 10. Defining requires constraints (in ASP). For example, *smarthomes* in *Austria* include at least two *rooms*.

(h) *Resource Constraints*. Resource constraints specify producer and consumer relationships, for example, a resource (*producer*) could be the money available for the smarthome (*maxprice* specified by the customer) and the consumers could be the installed sensors (represented by the attribute *priceofsensor*). A corresponding resource constraint could indicate that the sum of the prices of all sensors must not exceed the upper price limit specified by the customer (attribute *maxprice*). An implementation of a resource constraint in ASP is shown in Figure 11.

```
sensorprice(T) :- T = #sum{ PR, A : priceofsensor(A, PR)};
:- smarthome(X), maxprice(X,Y), sensorprice(P), P > Y.
```

Figure 11. Defining resource constraints (in ASP). For example, the price of a *smarthome* (represented by *sensorprice* only) must not exceed the *maxprice* defined by the customer.

(i) *Navigation Constraints*. ASP allows to represent complex constraints which require navigation between instances. For example, a stove in one room excludes further stoves in other rooms of the same smarthome (see Figure 12). This can also be interpreted as a further example of an incompatibility constraint (see Figure 9). Even recursively defined predicates can be used in such constraints such that transitive closures (e.g. reachability in graphs) and constraints on them can be expressed. This is a modeling advantage compared to standard constraint solvers.

```
:- stove(A1), roomappliance(R1,A1), smarthomeroom(H,R1),
   R1 != R2,
   smarthomeroom(H,R2), roomappliance(R2,A2), stove(A2).
```

Figure 12. Defining navigation constraints (in ASP). For example, two different *rooms* with a *stove* must not be part of the same *smarthome* configuration. In this context, R1 != R2 assures that two different *rooms* are analyzed with regard to the inclusion of a *stove*.

(j) *Example Customer Requirements*. Having defined the whole configuration knowledge base, customers can specify their requirements with regard to a corresponding smarthome configuration as facts and even more generally as constraints (see Figure 13). Examples of such customer requirements are: *the smarthome installation will be located in Austria* and *no dangerous appliances should be installed* (see the following constraints).

```
smarthome(1).
country(1,austria).
:- appliance(X), isdangerous(X,true).
```

Figure 13. Defining requirements (in ASP). For example, the *smarthome* should be in *austria* and no dangerous *appliances* should be included.

4 ASP Solving and Limitations

Classical ASP solvers [9] work in two steps: (1) *grounding* which translates the ASP program to a variable-free format and (2) propositional (SAT) *solving* of the grounded program. Figure 14 shows a reduced version of our *smarthome* configuration knowledge base.

```
% potential instances
psmarthome(1). proom(2;3;4).

% attribute domain definitions
domcountry(germany;austria). domtype(kitchen;livingroom).

% definition / generation of instances
0{ smarthome(X) }1 :- psmarthome(X).
0{ room(X) }1 :- proom(X).

% associating attributes with component types
1{ country(X,Y): domcountry(Y) }1 :- smarthome(X).
1{ type(X,Y): domtype(Y) }1 :- room(X).

% definition / generation of associations
1{ smarthomeroom(X,Y): room(Y) }2 :- smarthome(X).
1{ smarthomeroom(Y,X): smarthome(Y) }1 :- room(X).

% further constraints
:- smarthome(X), country(X,austria),
   not 2{ smarthomeroom(X,Y): room(Y) }2.

% customer requirements
room(2). type(2,kitchen).
```

Figure 14. Restricted version of example *smarthome* configuration model.

The knowledge base in Figure 14 is a subset of the class diagram in Figure 1. It includes a component type *smarthome* with the attribute *country* and a component type *room* with the attribute *type*. A *smarthome* can have 1 or 2 *rooms* but Austrian *smarthomes* must have two rooms (this is defined in terms of an additional constraint). One potential instance is defined for *smarthome* and three potential instances are defined for *room*. The requirements specify the inclusion of a *room of type kitchen*.

The *grounding* results are shown in Figure 15 – the relationship to the knowledge base of Figure 14 is explained in terms of comments. *ASP facts* of the original knowledge base are also represented as facts

in the grounded knowledge base. Variables in rules are removed and the rules are duplicated accordingly, for example, three rules are generated for the three possible room instances: i.e., *proom(X)* in the second rule for generation of instances is replaced with each of the three facts for potential instances and the head of the rule is replaced accordingly. Furthermore, the *count* aggregate is represented in its standard form instead of just curly brackets.

```
% potential instances
psmarthome(1). proom(2). proom(3). proom(4).

% attribute domain definitions
domcountry(germany). domcountry(austria).
domtype(kitchen). domtype(livingroom).

% definition / generation of instances
0<=#count{1,0,smarthome(1):smarthome(1)}<=1.
0<=#count{1,0,room(2):room(2)}<=1.
0<=#count{1,0,room(3):room(3)}<=1.
0<=#count{1,0,room(4):room(4)}<=1.

% associating attributes with component types
1<=#count{1,0,country(1,germany):country(1,germany);
          1,0,country(1,austria):country(1,austria)
          }<=1:- smarthome(1).
1<=#count{1,0,type(2,kitchen):type(2,kitchen);
          1,0,type(2,livingroom):type(2,livingroom)
          }<=1:- room(2).
1<=#count{1,0,type(3,kitchen):type(3,kitchen);
          1,0,type(3,livingroom):type(3,livingroom)
          }<=1:-room(3).
1<=#count{1,0,type(4,kitchen):type(4,kitchen);
          1,0,type(4,livingroom):type(4,livingroom)
          }<=1:-room(4).

% definition / generation of associations
1<=#count{
    1,0,smarthomeroom(1,2):smarthomeroom(1,2):room(2);
    1,0,smarthomeroom(1,3):smarthomeroom(1,3):room(3);
    1,0,smarthomeroom(1,4):smarthomeroom(1,4):room(4)
          }<=2:-smarthome(1).

1<=#count{1,0,smarthomeroom(1,2):
            smarthomeroom(1,2):smarthome(1)}<=1:-room(2).
1<=#count{1,0,smarthomeroom(1,3):
            smarthomeroom(1,3):smarthome(1)}<=1:-room(3).
1<=#count{1,0,smarthomeroom(1,4):
            smarthomeroom(1,4):smarthome(1)}<=1:-room(4).

% further constraints
:- smarthome(1); country(1,austria); not 2<=#count
  {1,0,smarthomeroom(1,2):smarthomeroom(1,2):room(2);
   1,0,smarthomeroom(1,3):smarthomeroom(1,3):room(3);
   1,0,smarthomeroom(1,4):smarthomeroom(1,4):room(4)}<=2.

% customer requirements
room(2). type(2,kitchen).
```

Figure 15. Grounded version of restricted *smarthome* configuration model.

In general, grounding can lead to extremely large knowledge bases, especially if rules or constraints entail many variables with a large domain. The domain sizes are multiplied which leads to exponential growth in the number of variables in the worst case. This shows one of the weaknesses of answer set programs - they are not well suited for problems with *large integer domains or floating point numbers*. Ways to deal with this issue is to combine answer set programming with constraint solving techniques (see, e.g., [11, 21]) and lazy grounding (see, e.g., [3]).

Solving an ASP results in answer sets (solutions). Each solution must be well-founded (i.e., derived from the given facts) and consistent (i.e. not violating any constraint). Figure 16 shows all solutions (answer sets) derived from the example in Figure 14.

As the customer prefers *room 2* and *country austria* requires 2 rooms, there is only one answer set (*Answer: 1*) with one room - its type is *kitchen* (as specified by the customer requirements) and the *country* is *germany*. Sketch of the reasoning steps: *room(2)* can be seen as a propositional variable with truth value *TRUE*. It appears in the body of the second rule for generation of associations and that body contains no other variables. Therefore the head is evaluated and requires exactly one variable in the count aggregate to be set to *TRUE*. The only one is *smarthomeroom(1,2)* and it is founded if *smarthome(1)* is generated by the first count aggregate for generation of instances. Therefore, it derives the facts *smarthomeroom(1,2)* and *smarthome(1)*. For *country*, there are exactly two alternatives, but only *germany* does not lead to a constraint violation. Therefore *country(1,germany)* is added as a fact (i.e. assigned truth value *TRUE* in the SAT view of the reasoning process). As no other variables need to be set (all count aggregates are fulfilled), we have the first solution.

All other solutions derive a second room. This allows all combinations of the alternatives for *type (kitchen, livingroom)*, *country (germany, austria)* and *identifier (3, 4)*. One can imagine that the corresponding multiplication of alternatives can lead to tremendously many solutions. At least concerning the identifiers, the solutions are equivalent (i.e. there is no relevant difference between answer sets 2 to 5 and answer sets 6 to 9). In order to reduce the search space for such unneeded solutions, symmetry breaking techniques [2] and search heuristics [10] can be used.

```
- Answer: 1 -
smarthome(1) country(1,germany)
room(2) type(2,kitchen)
smarthomeroom(1,2)
- Answer: 2 -
smarthome(1) country(1,germany)
room(2) type(2,kitchen) room(4) type(4,livingroom)
smarthomeroom(1,2) smarthomeroom(1,4)
- Answer: 3 -
smarthome(1) country(1,austria)
room(2) type(2,kitchen) room(4) type(4,livingroom)
smarthomeroom(1,2) smarthomeroom(1,4)
- Answer: 4 -
smarthome(1) country(1,germany)
room(2) type(2,kitchen) room(4) type(4,kitchen)
smarthomeroom(1,2) smarthomeroom(1,4)
- Answer: 5 -
smarthome(1) country(1,austria)
room(2) type(2,kitchen) room(4) type(4,kitchen)
smarthomeroom(1,2) smarthomeroom(1,4)
- Answer: 6 -
smarthome(1) country(1,germany)
room(2) type(2,kitchen) room(3) type(3,kitchen)
smarthomeroom(1,2) smarthomeroom(1,3)
- Answer: 7 -
smarthome(1) country(1,austria)
room(2) type(2,kitchen) room(3) type(3,kitchen)
smarthomeroom(1,2) smarthomeroom(1,3)
- Answer: 8 -
smarthome(1) country(1,germany)
room(2) type(2,kitchen) room(3) type(3,livingroom)
smarthomeroom(1,2) smarthomeroom(1,3)
- Answer: 9 -
smarthome(1) country(1,austria)
room(2) type(2,kitchen) room(3) type(3,livingroom)
smarthomeroom(1,2) smarthomeroom(1,3)
```

Figure 16. Solutions for *smarthome* knowledge base of Figure 15.

Figure 17 shows symmetry breaking constraints which force the solver to use identifiers from lowest to highest. In the knowledge base defined by the ASP entries of Figure 2–13, the number of answer sets would be reduced from 8.400 to 2.800 if the symmetry breaking constraints are taken into account. Adding constraint *:- room(X), proom(Y), X>Y, not room(Y).* to Figure 14 would reduce the number of answer sets from 9 to 5 in Figure 16.

```
:- room(X), proom(Y), X>Y, not room(Y).
:- stove(X), pstove(Y), X>Y, not stove(Y).
:- tv(X), ptv(Y), X>Y, not tv(Y).
```

Figure 17. Symmetry breaking constraints for the entries of Figures 2–13.

5 AGILE Configuration Technologies

The configuration knowledge representations discussed in this paper are applied within the scope of the European Union project AGILE[7] that focuses on the development of recommendation and configuration technologies for IoT gateways. Within AGILE, configuration technologies are applied to support different kinds of ramp-up scenarios, i.e., initial setups of IoT gateways infrastructures entailing the needed hardware and software components. Furthermore, AGILE supports runtime configuration and reconfiguration, for example, in terms of optimizing the usage of data exchange protocols with regard to optimality criteria such as economy and efficiency. The basis for AGILE configuration solutions in ramp-up domains is the *clingo* environment. In AGILE, we are especially focusing on improving the performance of constraint-based reasoning and model-based diagnosis that are both supporting technologies also in the context of answer set programs [7, 17, 21].

6 Research Issues

There are still a couple of research challenges to be tackled to make ASP-based configuration more applicable and performant. Graphical configuration knowledge representations and a corresponding automated translation into ASP-based representations will help to improve knowledge engineering processes. An automated generation of ASP knowledge bases from object-oriented product topologies has already been proposed in [4]; translation routines for standard constraints such as *requires*, *excludes*, and *resources* would help to further increase knowledge engineering efficiency. Improving constraint answer set programming or finding new ways of integrating ASP and CSP could lead to a full exploitation of the different advantages of both paradigms. In order to solve large-sized problems, lazy grounding and heuristics must be combined and improved (related work is reported, e.g., in [24]). All means to reduce problem sizes of ASPs should be exploited, for example, precise estimation of the number of needed instances (see [20]). Finally, domain-specific constraints such as discussed in [14] have to be analyzed with regard to their representation in ASP.

7 Conclusions

In this paper we give an overview of basic applications of configuration technologies in Internet of Things (ioT) scenarios. In this context we show how to apply answer set programming (ASP) techniques to represent and solve configuration problems. ASP is a logic-based approach and well-suited for a component-oriented representation of configuration tasks. This capability is extremely useful especially in large and complex product domains. In order to provide a basic reference for ASP beginners, we show how to represent most representative constraints in ASP.

ACKNOWLEDGEMENTS

The work presented in this paper has partially been conducted within the scope of the Horizon 2020 Project AGILE (Adoptive Gateways for dIverse MuLtiple Environments, 2016–2018).

[7] agile-iot.eu.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, 'The Internet of Things: A Survey', *Computer Networks*, **54**(15), 2787–2805, (2010).
[2] C. Drescher, O. Tifrea, and T. Walsh, 'Symmetry-breaking answer set solving', in *ICLP10 Workshop on Answer Set Programming and Other Computing Paradigm*, (2010).
[3] T. Eiter, T. Kaminski, C. Redl, and A. Weinzierl, 'Exploiting Partial Assignments for Efficient Evaluation of Answer Set Programs with External Source Access', in *25th International Joint Conference on Artificial Intelligence (IJCAI-16)*, pp. 1058–1065, New york, NY, USA.
[4] A. Falkner, A. Ryabokon, G. Schenner, and K. Shchekotykhin, 'OOASP: Connecting Object-Oriented and Logic Programming', in *13th International Conference on Logic Programming and Nonmonotonic Reasoning*, pp. 332–345, Lexington, KY, USA, (2015).
[5] A. Falkner, G. Schenner, G. Friedrich, and A. Ryabokon, 'Testing Object-Oriented Configurators With ASP', in *ECAI 2012 Workshop on Configuration*, pp. 21–26, Montpellier, France, (2012).
[6] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, *Knowledge-based Configuration: From Research to Business Cases*, Elsevier/Morgan Kaufmann Publishers, 1st edn., 2014.
[7] A. Felfernig, M. Schubert, and C. Zehentner, 'An Efficient Diagnosis Algorithm for Inconsistent Constraint Sets', *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing (AIEDAM)*, **26**(1), 53–62, (2012).
[8] G. Friedrich, A. Ryabokon, A. Falkner, A. Haselböck, G. Schenner, and H. Schreiner, '(Re)configuration using Answer Set Programming', in *Workshop on Configuration*, pp. 17–25, Barcelona, Spain, (2011).
[9] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, *Answer Set Solving in Practice*, Morgan & Claypool Publishers, 1st edn., 2012.
[10] M. Gebser, B. Kaufmann, J. Romero, R. Otero, T. Schaub, and P. Wanko, 'Domain-Specific Heuristics in Answer Set Programming', in *27th AAAI Conf. on AI*, pp. 350–356, Bellevue, WA, USA.
[11] M. Gebser, A. Ryabokon, and G. Schenner, 'Combining heuristics for configuration problems using answer set programming', in *13th International Conference on Logic Programming and Nonmonotonic Reasoning*, pp. 384–397, Lexington, KY, USA, (2015).
[12] M. Gelfond and V. Lifschitz, 'The stable model semantics for logic programming', in *5th International Conference of Logic Programming (ICLP'88)*, pp. 1070–1080, (1988).
[13] L. Hotz, A. Felfernig, M. Stumptner, A. Ryabokon, C. Bagley, and K. Wolter, 'Configuration Knowledge Representation and Reasoning', in *Knowledge-Based Configuration: From Research to Business Cases*, 41–72, (2014).
[14] L. Hotz and K. Wolter, 'Smarthome Configuration Model', in *Knowledge-based Configuration – From Research to Business Cases*, eds., A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, chapter 10, 157–174, Morgan Kaufmann Publishers, (2013).
[15] G. Leitner, A. Fercher, A. Felfernig, K. Isak, S. Polat Erdeniz, A. Akcay, and M. Jeran, 'Recommending and configuring smart home installations', in *Workshop on Configuration*, pp. 17–22, (2016).
[16] V. Myllärniemi, J. Tiihonen, M. Raatikainen, and A. Felfernig, 'Using Answer Set Programming for Feature Model Representation and Configuration', in *Workshop on Configuration*, pp. 1–8, (2014).
[17] K. Shchekotykhin, 'Interactive Query-Based Debugging of ASP Programs', in *29th AAAI Conf. on AI*, pp. 1597–1603, Austin, Texas, USA.
[18] T. Soininen and I. Niemelä, 'Developing a declarative rule language for applications in product configuration', in *PADL*, pp. 305–319, (1998).
[19] M. Stumptner, 'An Overview of Knowledge-based Configuration', *AI Communications*, 111–125, (1997).
[20] R. Taupe, A. Falkner, and G. Schenner, 'Deriving tighter component cardinality bounds for product configuration', in *18 th International Configuration Workshop*, p. 47, (2016).
[21] E. Teppan and G. Friedrich, 'Heuristic Constraint Answer Set Programming', in *ECAI 2016*, pp. 1692–1693, (2016).
[22] J. Tiihonen, T. Soininen, I. Niemelä, and R. Sulonen, 'A practical tool for mass-customizing configurable products', in *14th International Conference on Engineering Design*, pp. 1290–1299, (2003).
[23] E. Tsang, *Foundations of Constraint Satisfaction*, Academic Press, London, 1993.
[24] A. Weinzierl, 'Blending lazy-grounding and CDNL search for answer-set solving', (2017).

Cluster-Based Constraint Ordering for Direct Diagnosis

Muesluem Atas[1] and **Alexander Felfernig**[1] and **Seda Polat Erdeniz**[1] and **Stefan Reiterer**[1] and **Amal Shehadeh**[1] and **Thi Ngoc Trang Tran**[1]

Abstract. Prediction quality and runtime performance are important performance indicators for diagnosis algorithms. In this paper, we propose a new method (ClusDiag *Cluster-Based Constraint Ordered Direct Diagnosis*) which can improve both indicators. ClusDiag has a learning phase to find a constraint ordering heuristic. After the learning phase, a diagnosis is found by applying the direct diagnosis algorithm FastDiag on an inconsistent constraint set where the constraints are reordered with respect to the constraint ordering heuristic.

Keywords— Configuration Systems; Diagnosis; Constraint Satisfaction; Variable and Value Ordering Heuristics; Clustering; Performance Optimization

1 Introduction

When a constraint set is inconsistent, there is no solution for the corresponding constraint satisfaction problem (CSP) / configuration problem [5]. In this case, a diagnosis is required to make the CSP solvable (s.t. at least one solution can be found). Direct diagnosis algorithms find diagnoses without need of finding corresponding the minimal conflict sets. In diagnosis identification, the most important aspects are the *runtime performance* of the algorithm and the *prediction quality* of the result [4].

We propose a new method called ClusDiag which can find diagnoses using cluster-specific [12] constraint ordering heuristics that are exploited by direct diagnosis search (in our case by the FastDiag algorithm). These heuristics help to increase the efficiency of diagnosis determination in two different aspects: runtime performance and prediction quality. Time-efficient heuristics find diagnoses faster, whereas prediction quality-efficient heuristics help to find a diagnosis that can be used to provide solutions which are more likely accepted by the user.

Prediction quality-efficient heuristics can be learned based on user interaction logs that contain information on which solution trade-offs were acceptable in the case of inconsistent user requirements. In order to determine the prediction quality of heuristics during the learning phase, we use user interaction data collected within the scope of a user study. Entries in this dataset consist of specified user requirements inconsistent with the knowledge base and corresponding solutions finally selected by the user. In the mentioned user study, we first asked users to specify the customer requirements (or constraints) and then to select alternative solutions if their requirements could not be fulfilled by the underlying configuration knowledge base (represented as a product table).

In our approach we differentiate between two phases: (1) in the offline phase, clustering and learning of cluster-specific heuristics for direct diagnosis (in our case FastDiag) is performed, (2) in the online phase, cluster specific heuristics are used to support direct diagnosis search.

In order to evaluate the *performance* of ClusDiag, we applied the algorithm on the dataset collected within the scope of our user study. In order to evaluate the *prediction quality* of the algorithm in combination with a specific constraint ordering heuristic, we analyzed whether a predicted diagnosis leads to a solution selected by the user.

The contributions of our paper are the following. First, the prediction quality of direct diagnosis algorithms can be improved based on our clustering-based learning approach for constraint ordering. Second, our approach can also improve the runtime performance of direct diagnosis search.

The remainder of this paper is organized as follows. In Section 2, we introduce a working example to show the basic ClusDiag approach. In Section 3, we report the results of evaluating the approach with regard to runtime performance and prediction quality. The paper is concluded with a discussion of related and future work.

2 Working Example

To show the basic ClusDiag approach, we introduce a simple working example. In this example, there are five products available in the product catalog of a bike shop and each bike is further characterized by values related to the three features $f1 = motorsize$, $f2 = price$, and $f3 = rating$ as shown in Table 1. [2] We know about six previous customer requirement specifications that lead to an inconsistency, i.e., no solution could be found (see Table 2). Inconsistency in this context means that there does not exist a bike in the product table that supports the properties specified by the requirements. Note that for simplicity we assume the existence of equality constraints that specify the relationship between customer requirements and bike properties (e.g., $cs1.f1.val = p1.f1.val$). Furthermore, we assume that the set of features describing customer requirements and bike properties is equivalent.

Finally, we know which products were selected by the customers after changing their initial requirements in such a way that the new set of requirements is consistent with the product table, i.e., at least one product could be found (see Table

[1] Software Technology Institute, Technical University of Graz, Austria, email {muesluem.atas, alexander.felfernig, spolater, stefan.reiterer, amal.shehadeh, ttrang} @ist.tugraz.at

[2] Our working example includes a set of explicitly enumerated configurations. However, our approach can be applied the same way in the context of implicitly enumerated configurations described, for example, in terms of variables, domains, and constraints.

3).

	bike1	bike2	bike3	bike4	bike5
f1 (motor size in cc)	1000	1000	600	600	1200
f2 (price)	1400	1200	1000	1000	1600
f3 (rating)	1200	870	1450	720	1100

Table 1. Product table with five types of bikes where each has three features as motor size, price, and rating.

cust.1 cs1	cust.2 cs2	cust.3 cs3	cust.4 cs4	cust.5 cs5	cus.6 cs6
f1=1400 f2=1000 f3=1000	f1=1000 f2=700 f3=700	f1=1400 f2=900 f3=900	f1=900 f2=700 f3=700	f1=900 f2=700 f3=700	f1=500 f2=300 f3=300

Table 2. Inconsistent constraint sets of six past customers.

cust.1	cust.2	cust.3	cust.4	cust.5	cus.6
bike1	bike1	bike2	bike5	bike4	bike2

Table 3. Bikes finally selected by customers (after their specified requirements given in Table 2).

2.1 Clustering Phase

In the clustering phase, we cluster similar customer requirements using the former user interaction data (in our working example, this data is represented by Tables 2 and 3). We apply k-means clustering on the customer requirements.

K-means clustering creates k clusters where it minimizes the sum of squares of distances between cluster elements [9] as shown in Formula 1. In the context, k is the number of target clusters, S is a cluster set, μ_{i} is the average value of cluster elements in S_{i}, and x is a cluster element in S_{i}.

$$min \sum_{i=1}^{k} \sum_{x \in S_{\mathrm{i}}} \|x - \mu_{\mathrm{i}}\|^2 \quad (1)$$

In k-means clustering, the difference (or distance) between two cluster elements x and y with multiple attributes can be calculated based on the *Euclidean Distance (n-dimensional)* [2] as shown in the Formula 2 where x_{j} is the j^{th} attribute of x and y_{j} is the j^{th} attribute of y.

$$x - y = \sqrt{\sum_{j=1}^{n} (x_{\mathrm{j}} - y_{\mathrm{j}})^2} \quad (2)$$

In our working example, we set the parameters of the k-means clustering algorithm as follows: k=2, i.e., two clusters, maximum number of iterations of k-means clustering: 3. In the example, k-means clustering is applied to the data mentioned in Table 2.

First, we create the mentioned two clusters and assign the two items with the highest pairwise distance (see Formula 2) as initial cluster elements. The remaining items are now examined in sequence and assigned to the cluster to which they are closest, in terms of Euclidean distance to the cluster mean (or centroid). The mean vector is recalculated each time a new member is added to a cluster and after five steps we established the two clusters as shown in Table 4.

After having created the clusters, we cannot yet be sure that each item has been assigned to the right cluster. So, we compare each item's distance to its own cluster mean and to that of the other cluster. Only item *cs*2 is nearer to the mean of the other cluster (cluster2) than to its own cluster (cluster1).

	cluster1		cluster2	
	items	centroid	items	centroid
s.1	cs1	[1400,1000,1000]	cs6	[500,300,500]
s.2	cs1,cs2	[1200,850,850]	cs6	[500,300, 500]
s.3	cs1,cs2,cs3	[1266,866,866]	cs6	[500,300,500]
s.4	cs1,cs2,cs3	[1266,866,866]	cs6,cs4	[700,500,500]
s.5	cs1,cs2,cs3	[1266,866,866]	cs6,cs4,cs5	[766,566,566]

Table 4. Clustering customer requirements in five steps.

The iterative relocation would now continue up to defined number of maximum iterations. However, in this example each item is now nearer its own cluster mean than to that of the other cluster and the iteration stops since no further adaptations are needed.

	cluster1		cluster2	
	items	centroid	items	centroid
i.1	cs1,cs2,cs3	[1266,866,866]	cs6,cs4,cs5	[766,566,566]
i.2	cs1,cs3	[1400,950,950]	cs6,cs4,cs5,cs2	[825,600,600]

Table 5. Iterations of relocating constraint sets.

2.2 Learning Constraint Orderings

After clustering the user requirements specified in the user interaction log (see Table 2), we apply a genetic algorithm for learning two different constraint ordering heuristics: one for optimizing the prediction quality of direct diagnosis and the other one for optimizing the runtime performance per cluster.[3]

In the working example, the genetic algorithm learns constraint ordering heuristics for improving the prediction quality. Prediction quality in our case is measured in terms of precision ([8]), i.e., the overall share of correct diagnosis predictions (diagnoses that lead to a product that also has been selected by the user) in relation to the total number of diagnosis predictions (determined by FastDiag on the basis of the constraint ordering within a cluster). We initialize the genetic algorithm with the parameters population size = 2, maximum number of generations = 2, mutation rate = 0.015, uniform rate = 0.5, and target fitness value = 1.

In the first generation, for cluster-1, we create two (since population size is set to two) random constraint orderings (individuals): [c2,c1,c3], [c3,c2,c1]. Then, one by one we use these constraint orderings as constraint orderings in the direct diagnosis algorithm FastDiag [6]. The prediction quality of the [c2,c1,c3] for cs1 is 0.5, which means the average of the diagnosis algorithm's correct predictions (where the constraint ordering heuristic [c2,c1,c3] is applied) is 0.5.

We calculate the precision values for all constraint orderings in the populations of each generation and select the best individuals (individual is constraint ordering in our case) in the populations.

After 2 generations (maximum number of generations is set to 2), the genetic algorithm stops. Then we can use the best constraint ordering learned by the genetic algorithm. As shown in Table 6, the best constraint ordering for cluster1 is [c3, c2, c1] whereas [c1, c3, c2] for cluster2.

We apply the same algorithm for learning constraint ordering heuristics for runtime performance. In this case we set the

[3] Combined heuristics allowing tradeoffs between runtime performance and prediction quality will be analyzed within the scope of our future work.

	cluster1		cluster2	
	ordering	fitness	ordering	fitness
gen.0	[c2, c3, c1]	0.4	[c3, c2, c1]	0.3
gen.1	[c3, c2, c1]	0.5	[c3, c2, c1]	0.3
gen.2	[c3, c2, c1]	0.5	[c1, c3, c2]	0.6

Table 6. Learning constraint ordering heuristic for each cluster.

target fitness value to 0 because this time the fitness value is the runtime which we aim to minimize this time.

2.3 Application Phase

In the offline phase, we focus on cluster generation and learning of heuristics for direct diagnosis. Now, in the online phase, we apply our heuristics to a new inconsistent set of customer requirements.

	c1	c2	c3
cs_new	f1=1200	f2=1000	f3=1000

Table 7. A new set of customer requirements (represented in terms of constraints).

In the working example, we need to find a diagnosis with a high prediction quality since our aim is to find a product of high relevance for the customer.

First, we find a cluster that is closest to the new set of customer requirements cs_new[1200, 1000, 1000]. This is cluster1 with centroid [1400, 950, 950]. Then, we apply the identified optimal constraint ordering for cluster1 (which is [c3, c2, c1]) to cs_new. The reordered constraint set is cs_new_reordered = {(f3=1000), (f2=1000), (f1=1200)}.

When we apply the FastDiag algorithm to cs_new_reordered, we find the diagnosis {c1, c3}. After removing the diagnosis from the constraint set, we have cs_new_reordered_diagnosed = {(f2=1000)}. According to this new constraint set, there are two solutions available in the product table which are bike3 and bike4.

Applying the diagnosis means to remove elements from the customer's inconsistent constraint set (set of requirements). In our example, we can find two products which can be recommended.

3 Evaluation

In order to evaluate ClusDiag, we applied our approach to a dataset collected within the scope of a user study - the dataset comprises 264 different inconsistent requirements specifications and the corresponding products users (study participants) selected after changing their requirements. In the mentioned user study, the product table consisted of 30 digital cameras which were described by 10 features.

We compared the mentioned two versions of ClusDiag with each other and with the FastDiag algorithm with randomized constraint ordering (baseline version). For randomized constraint ordering, we calculated the average performance and prediction quality values from 10 random constraint orderings. Figure 1 shows a comparison of ClusDiag-enhanced FastDiag (FastDiag that exploits the cluster-based constraint ordering heuristics) and a basic FastDiag version (constraints are ordered randomly) with regard to *prediction quality*. In Figure 2, we show the comparison of the runtime performance of FastDiag with the corresponding clustering-supported version of the algorithm (in this case, heuristics were optimized with regard to algorithm runtime performance).

4 Related Work

Most widely known algorithms for the identification of minimal diagnoses are QuickXplain [10] for predetermining minimal conflict sets which are used by a hitting set based approach such as HSDAG [13] to compute minimal diagnoses. FastDiag [6] also determines minimal diagnoses but does this without the need of identifying minimal conflict sets.

De Kleer et al. [3] introduce a probability-based approach to estimate relevant diagnoses. This approach is based on traditional hitting set based diagnosis determination, i.e., is in the need of predetermining conflict sets.

Methods to find approximate solutions for diagnosis tasks are introduced, for example, in [1, 14]. In contrast to our approach, approximate solutions do not guarantee the minimality of diagnoses. Since the initial version of HSDAG introduced by [12], a couple of algorithms have been proposed to increase the efficiency of the inital version – see, for example, [15].

The authors of [11] focus on interactive settings where users of constraint-based applications are confronted with situations where no solution can be found. In this context, the authors propose an approach to determine representative sets of diagnoses, i.e., diagnoses that cover as much as possible all potential faulty elements of the solution space. Direct diagnosis as used in our work focuses on constraint orderings that help to increase the probability of finding diagnoses relevant for the user. Focusing on such leading diagnosis is not the central focus of the work presented in [11].

Authors of [7] present an algorithm for computing a generalization of conflict-based explanations of inconsistency for the QCSP (Quantified Constraint Satisfaction Problem) which is a generalization of the classical CSP in which some of the variables can be universally quantified.

Direct diagnosis is an approach to omit conflict detection and directly determine diagnoses without the need to identify the corresponding conflict sets. FastDiag [6] is a diagnosis algorithm that supports direct diagnosis. It is based on a divide-and-conquer strategy with a number of consistency checks similar to QuickXplain [10]. QuickXplain [10] itself is a conflict detection algorithm that helps to identify minimal conflict sets. Similar to FastDiag, the algorithm relies on a linear ordering of the constraints. It is often used in combination with HSDAG to determine diagnoses [13].

5 Conclusions

In this paper, we have introduced a new diagnosis approach called ClusDiag that increases the efficiency of direct diagnosis algorithms with respect to prediction quality and runtime by applying cluster-specific constraint ordering heuristics.

As future work, we will test our approach with different clustering techniques such as hierarchical clustering. Furthermore we will include further optimization criteria for learning constraint ordering heuristics. For example, we will learn constraint ordering heuristics that support trade-offs between runtime performance and prediction quality.

Acknowledgement

The work presented in this paper has been conducted within the scope of the EU Horizon 2020 project AGILE (http://agile-iot.eu/).

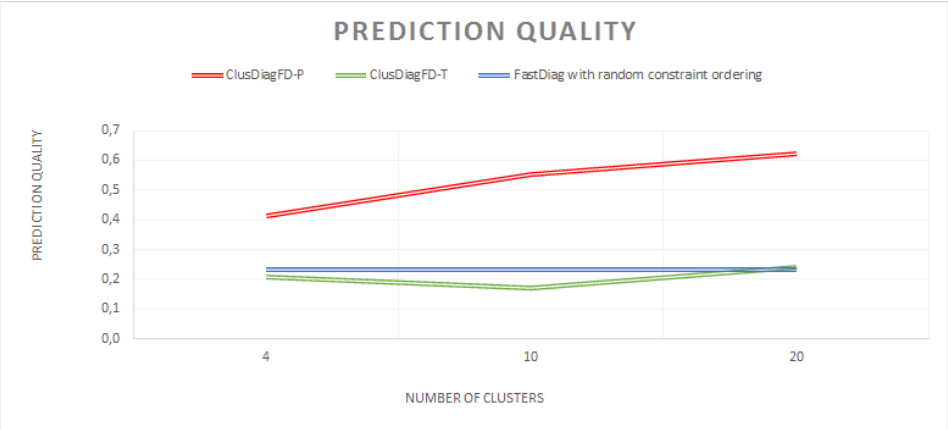


Figure 1. Comparison of the prediction quality (measured in terms of precision [0..1]) of the different variants of FastDiag (ClusDiagFD-P = FastDiag with constraint orderings optimized for high prediction quality, ClusDiagFD-T = FastDiag with heuristics focusing on runtime performance, and FastDiag with random constraint ordering = basic version with randomized constraint orderings). We ran the algorithms over 264 inconsistent CSPs (dataset collected within the scope of a user study). We observed that increasing the number of clusters also helps to increase the prediction quality.

Figure 2. Runtime performance comparison of diagnosis algorithms. We ran the algorithms over 264 inconsistent CSPs (from user study data). Then we have taken the average of the runtimes (over 264 CSPs) needed for diagnosis identification. Performance denotes the number of diagnoses found per millisecond. When we increase the number of clusters, diagnosis speed is increasing which means performance of ClusDiag increases.

REFERENCES

[1] Rui Abreu and Arjan JC Van Gemund, 'A low-cost approximate minimal hitting set algorithm and its application to model-based diagnosis.', in *SARA*, volume 9, pp. 2–9, (2009).

[2] Per-Erik Danielsson, 'Euclidean distance mapping', *Computer Graphics and image processing*, **14**(3), 227–248, (1980).

[3] Johan de Kleer, 'Using crude probability estimates to guide diagnosis', *Artificial Intelligence*, **45**(3), 381–391, (1990).

[4] Alexander Felfernig, Gerhard Friedrich, Monika Schubert, Monika Mandl, Markus Mairitsch, and Erich Teppan, 'Plausible repairs for inconsistent requirements.', in *IJCAI*, volume 9, pp. 791–796, (2009).

[5] Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, *Knowledge-based Configuration: From Research to Business Cases*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1 edn., 2014.

[6] Alexander Felfernig, Monika Schubert, and Christoph Zehentner, 'An efficient diagnosis algorithm for inconsistent constraint sets', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **26**(01), 53–62, (2012).

[7] Alex Ferguson and Barry O'Sullivan, 'Quantified constraint satisfaction problems: From relaxations to explanations.', in *IJCAI*, pp. 74–79, (2007).

[8] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich, *Recommender systems: an introduction*, Cambridge University Press, 2010.

[9] Xin Jin and Jiawei Han, *K-Means Clustering*, 563–564, Springer US, Boston, MA, 2010.

[10] Ulrich Junker, 'Quickxplain: Conflict detection for arbitrary constraint propagation algorithms', in *IJCAI'01 Workshop on Modelling and Solving problems with constraints*, (2001).

[11] Barry O'Sullivan, Alexandre Papadopoulos, Boi Faltings, and Pearl Pu, 'Representative explanations for over-constrained problems', in *AAAI*, volume 7, pp. 323–328, (2007).

[12] Seda Polat Erdeniz, Alexander Felfernig, and Muesluem Atas, 'Cluster-specific heuristics for constraint solving', *International Conference on Industrial, Engineering, Other Applications of Applied Intelligent Systems*, (2017).

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

[14] Staal Vinterbo and Aleksander Øhrn, 'Minimal approximate hitting sets and rule templates', *International Journal of approximate reasoning*, **25**(2), 123–143, (2000).

[15] Franz Wotawa, 'A variant of reiter's hitting-set algorithm', *Information Processing Letters*, **79**(1), 45–51, (2001).

Modeling and configuration for Product-Service Systems: State of the art and future research

Daniel Schreiber [1] and **Paul Christoph Gembarski** and **Roland Lachmayer**

Abstract. Regarding that boundaries between products and services vanish, the number of Product-Service Systems (hereinafter referred as PSS) rises continuously. One of the biggest advantages of a PSS is at the same time one of the biggest challenges: Upgrade, adaptation and modernization of the product itself during the life cycle, which was already suggested 15 years ago. Since then, modeling and configuration of PSS has been discussed, but the approaches that can be identified in literature remain mainly vague and conceptual.
In this paper modeling and configuration approaches for PSS are firstly discussed and assessed with respect to their knowledge-modeling capabilities and data models. It will be shown that constraint based approaches with parametric representation of product and service components of a PSS would be beneficial for a general PSS design approach. An useful procedure to develop such a data model could be the property-driven-development approach invented by Weber. Therefore, it is examined to what extent an existing approach to transfer Webers property-driven-development approach to PSS is a basis for this.

1 INTRODUCTION

The international assimilation of standards and the rapid dissemination of knowledge makes it more and more difficult for companies to distinguish themselves from their competitors by means of technical product characteristics.
One way of addressing this challenge is to expand their offer into an integrated problem solution, which consists of a combination of products and services. Such Product-Service Systems (PSS) exist, but in most cases as an additional service for an existing product. For manufacturers, an additive ratio of products and service is not desirable since this only leads to discount requirements. In order to use the advantages of PSS, an integrative relationship between product and service has to be created, which has a real added value [28].
Literature agrees that the setup of the development process has a decisive influence on the quality of the PSS. For the integrated design of PSS, service and product components must be treated equally in the development process. At the same time, PSS must be seen as solutions which fulfill customer requirements. Whether the value proposition is primarily achieved by the product or the service components, is only a secondary aspect [29].
To fulfill individual customer requirements a specific solution is needed and so an individual integrated development process. Literature agrees on that, but the approaches remain vague and conceptual. The transfer to other use cases is difficult, because the approaches are partly discussed on very simple or special examples [9].

[1] Leibniz Universität Hannover, Institut für Produktentwicklung und Gerätebau, email: schreiber@ipeg.uni-hannover.de

The lack of evaluation and insufficient detailing of the existing approaches makes it impossible to use one of them as the definition of a generally accepted and standardized approach for the development of PSS [11]. Annarelli et al. [2] emphasize that looking at the existing literature in the field of PSS research, PSS design is one of the most attractive areas. Nevertheless, the number of papers, which is focusing exclusively on design, is very limited. Commonly, the authors consider the development of PSS as a secondary field of interest, beneath their actual research question. This leads to the fact that PSS design cannot be considered as a research stream per se [2].
In this paper, firstly the necessary basic information on PSS and the existing approaches to the development and configuration of PSS will be presented. In the following section, the need for a constraint-based approach is elaborated and existing approaches to their potential and possible extensions are presented. Finally, an overview of the next steps and future research is given.

2 PSS

There are several definitions of PSS in literature. This section provides a brief overview about existing characterizations of PSS and documented requirements on PSS is given.

2.1 Characterization of PSS

Mont [20] described the concept of PSS as a system of products, services, supporting networks and infrastructure. This system has a lower environmental impact than traditional business models, it is competitive and satisfies customer needs. She accentuates the benefit of PSS for manufacturers and assumes an extension of the product life-cycle. An additional value for the customer is based on the possibility of upgrading and modernization of the product.
In his classification of PSS, Tukker [30] categorizes eight different types of PSS. The main categories, which are product-oriented, use-oriented and result-oriented, are shown in figure 1.
For Morelli [21] PSS is a description of solution oriented partnerships with material and immaterial components, which satisfies the requirements of each stakeholder. He mainly sees the use of PSS in business to business relationships, not between companies and consumers [21].
Meier [19] also focuses on industrial PSS as business to business applications. He defines PSS as a knowledge-intensive socio-technical system, which is characterized by the joint development, provision and use of product and service parts. He also identifies the adaptability of the solution to changing customer requirements in the product life-cycle as a core aspect of PSS. Thereby he draws attention on future reconfiguration and adaptation of already deployed PSS.

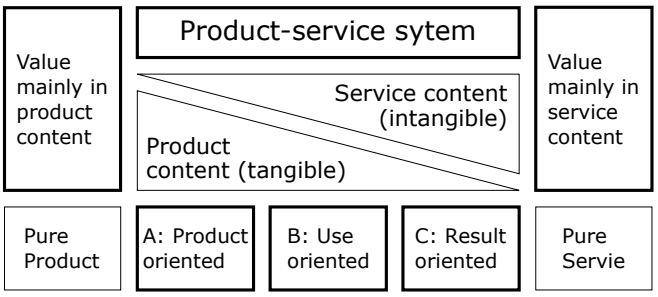


Figure 1. Main Categories of PSS (acc. to Tukker [30])

Müller [22] defines PSS also as a socio-technical system which is life-cycle, sustainability and customer-oriented. Customer and vendor are integrated in the resulting business relationship which provides the functionality that meets the customer needs. He highlights that the success of development and implementation of PSS depends on the ability to adapt quickly to changing customer requirements and to anticipate these changes in the early stages of PSS development. This requires efficient recording and monitoring of customer requirements.
According to Tukker [31], the most common concepts of PSS were defined in the period just after the year 2000. In recent literature authors still come up with their own definitions of PSS, but these do not differ fundamentally from the existing concepts.
From the authors' point of view, the fact that there are repeated attempts to redefine PSS and the terminologies involved shows that there is no general definition accepted by all researchers. This is partly due to the very heterogeneous composition of the research community with members from various disciplines. In the forthcoming years, proposed terminologies will increase. The goal is to develop a common ontology for the PSS community for further research. Nevertheless, in the current literature it is emphasized that further research is needed to develop an efficient PSS design methodology. [33].
Based on the existing concepts of PSS and the former postulated theses about PSS design research, three major characteristics can be named, which are the basis for the understanding of PSS in this paper:

- coequal development of product and service components
- integration and addressing of individual customers in the development process
- monitoring and addressing of the customer requirements during the whole life-cycle of the PSS

2.2 Requirements on PSS

Like mentioned before, a general accepted characteristic of PSS is the fulfillment of individual customer requirements and needs [21, 19, 9]. This leads to a large number of requirements, which must be summarized and abstracted at a rational level.
Ryynänen et al. [24] aggregated about 200 requirements and condensed them to 20 key requirements in five categories. For this paper relevant points are documented and discussed hereinafter. Ryynänen et al. describe a platform to fulfill the requirements, that should allow the designer to manage product requirements, track design changes and versions, manage product architecture and logical structure for product configuration and use intelligence in the design based on previous models, rules and design methods.
Furthermore, the platform should enable a conversion between different file formats (CAD files, product models, visualization models, manufacturing models) and manage results of tests and simulations of the digital prototype [24].

3 DEVELOPMENT AND CONFIGURATION OF PSS

In this section an overview about existing approaches of PSS development is given. Beginning with general approaches for the PSS design process followed by approaches of PSS configuration and approaches about Computer-Aided-Engineering (CAE) for PSS. This section is completed with approaches of knowledge-based modeling of PSS.

3.1 Design Processes for PSS

One of the basic statements named in section 2 is the integrated development of PSS. Most of the existing publications are restricted to partial aspects of the development process or to one component of the PSS, either the product or service part [3, 26, 33].
Steinbach introduced an approach based on the idea of Webers Characteristics-Properties Modeling/ Property-Driven Developments (CPM/PDD) [27]. Weber [34] distinguishes between properties and characteristics. The properties describe the behavior of a product, they can not be determined directly by the developer, but only by means of the change of the specified characteristics. These characteristics can be determined directly by the developer. They capture the shape of a product, defined by the structure, the arrangement of the components as well as the shapes, dimensions, materials and surface parameters. With these structure describing characteristics and behavioral properties, Steinbach [27] characterizes his PSS development process. Integrating service development and modeling in the design of physical products, properties mirror the result dimension of services (and the whole PSS), whereas characteristics correspond to their potential and process dimension.
The approach presented by Morelli [21] shows processes for the PSS development based on the idea of 'blueprints' which uses various already successfully planned PSS. This is comparable to the procedure of using templates in software and product development. According to Hirz [13] product templates are predefined design model structures or geometry models, which can be augmented with additional functionalities. Product templates can be divided into structure, geometry and function templates, where geometry templates are subdivided into rigid and variable (component and assembly) templates.
From the point of view of system engineering Müller [22] presented the process oriented approach of 'layer-based PSS development' which is adapted from the V-Model® XT. It depicts the development of building blocks, which are deviated from the whole system, as well as the integration and validation of these building blocks in the final PSS. The approach combines the different perspectives 'life-cycle', 'architecture' and 'development and management' of PSS. The basic framework of the approach is presented as a 150% process, which has to be tailored for every new PSS development task, according to scope and requirements of the desired result.
The approaches presented in this section are summarized in Figure 2 with their main features. Further aspects of contributions on PSS Engineering, development process models, engineering process models

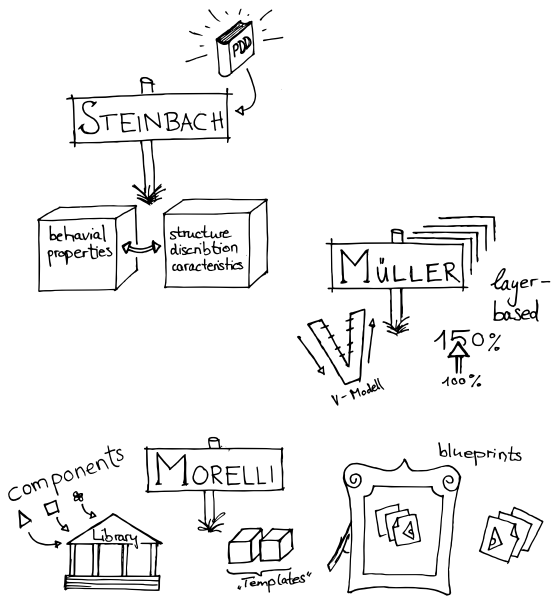


Figure 2. Design Processes for PSS

of PSS and adopted methods in the PSS Engineering literature are presented by Cavalieri and Pezzotta [7]. It is remarkable that modular product and service architectures as well as product configuration are not mentioned in this overview, since these are a major building block in variant design of physical products.

3.2 CAE for PSS

Today complex products are modeled in a CAE-environment. Their application is common and, at least in the everyday business, no task which is only carried out by specialists. CAE systems take over routine jobs and create additional capacities that can be used for further value-adding activities [32].

A computer-aided engineering environment is a toolbox for the development of domain-specific artifacts. This includes all the tools for all synthesis and analysis activities, as well as their information technology interfaces and data storage, which are needed for the engineering process [9].

From the perspective of product development, these systems are used to design the product (mechanical CAD, MCAD) and to derive the necessary production data in the sense of a technical drawing [13].

From the point of view of service development, there are only individual approaches documented (Service-CAD, SCAD). Sakao et al. [25] provide with their Service Explorer a computer aided service modeling tool which is based on a vendor-consumer system. They assume that a service is defined as an activity that a provider executes for a receiver to change something from an existing state to a new state that the receiver desires. Therefore, the main point in PSS is not the function of a product, but rather the state change of the receiver. In the presented system, first the requirements and the state of a buyer are modeled. Based on this, transformation rules are designed. This is realized with functional units of the service provider, similar to the feature-based modeling in MCAD. In figure 3, the main idea of the Service Explorer of Sakao is shown. The framework of the system is based on the idea of Roth's functional structures [23].

Hara et al. [12] point out that in CAD systems for physical products no modeling of the customer's use is possible. Gembarski et al. [9] mentioned that this statement is not sustainable, taking into account the possibilities of parametric and knowledge implementation in today's CAD systems, since the fulfillment of quantifiable requirements and the resulting benefits can very well be integrated into digital product models.

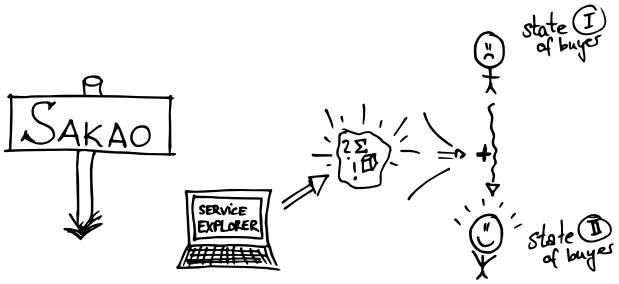


Figure 3. Service Explorer by Sakao

3.3 Knowledge-based modeling of PSS

In the product development of physical components, parametric, feature-based and even knowledge-based modeling are state of the art. All these techniques (shown in figure 4) are build on conventional CAD systems and have an inner connection [32].

In contrast to conventional models, parametric models have no fixed values but formulas and constraints in the models. Feature-based systems capture, besides the geometric data, additional informations (for example in terms of production characteristics). To be flexible and adaptable to the environment, the basic elements of these systems need to be parametric Thereby, feature-based systems can be understood as an extended parametric system [32]. Knowledge-based design assists to automate aspects of the design process as the system has the ability of reasoning and drawing conclusions [9].

Vanja [32] points out that, in the case of incomplete models, the advanced techniques quickly reach their limits, so that corresponding solutions cover only a very limited scope. This should be kept in mind while considering existing approaches.

Yang et al. [35] presented a life-cycle oriented approach based on the idea of knowledge-based assignment of service modules. In their considerations, a product already exists on which the service modules can be linked. Based on data monitored during the product use, the service modules are released. An example in this approach is the monitoring of a game console with regard to accelerations and mechanical shock. If a shock occurs, potentially damaged components of the console can be replaced directly, without an additional diagnostic step in customer service. Details on the necessary knowledge base for the evaluation of events or their design are not given, as well as information on possible reasoning mechanisms, which are typical for knowledge-based systems [35].

The service design catalog is an extension for the Service Explorer provided by Akasaka et al. [1]. The catalog is described as a support system for developers that provides service modules for functions that can be implemented. This is based on a merger of service parts for a PSS. According to the authors, they oriented themselves by the design catalog developed by Roth [23] as a knowledge base for design knowledge. However, the typical setup of classification part, main part and selection characteristics is not used.

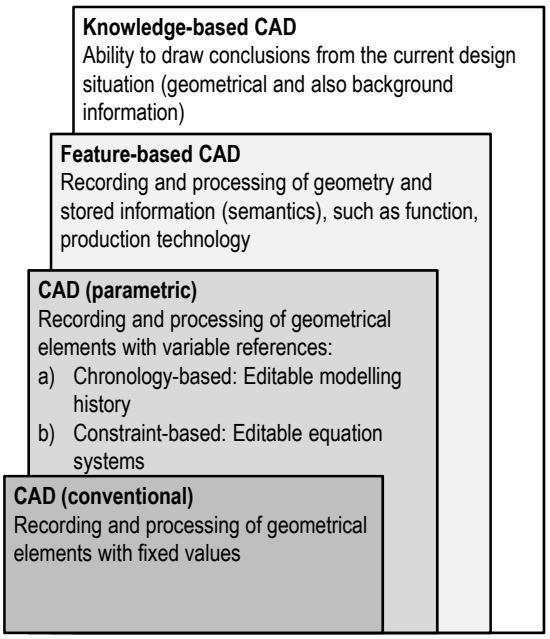


Figure 4. 3D modeling basics (VDI 2209) [32]

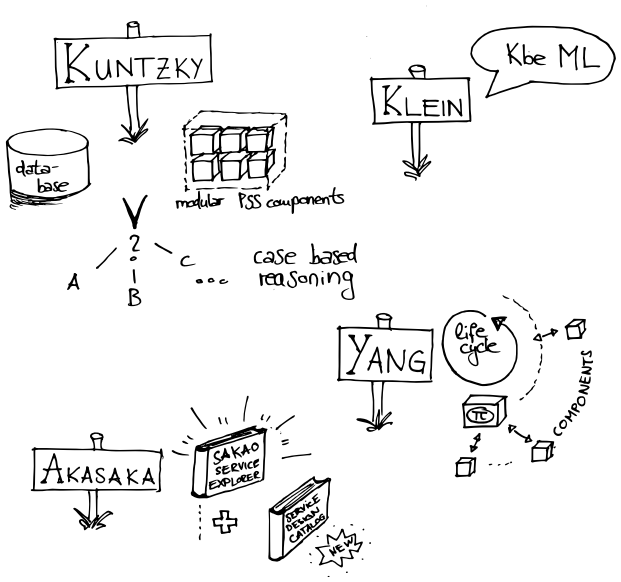


Figure 5. Knowledge-based engineering for PSS

Kuntzky [15] presents an approach for a knowledge-based development system for PSS based on case-based reasoning. For this purpose, she uses a modular design of the PSS components as well as the formulation of requirements and knowledge about the composition of a specific PSS. With these data, a configuration of PSS in the early stage of development is possible when the same or similar PSS and its requirements can be found and adapted to the case base. With this technique of knowledge-base systems, there is no need of a translation into a formal, explicit model.

Not a new approach for KBE modeling, but a modeling language (KbeML) has been specified by Klein [14]. In his point of view, the approach is a standardized representation for codified engineering knowledge. He describes KbeML as enabler for making development-related rules and algorithms accessible for different CAx systems, because it is based upon a formal machine-readable representation of knowledge. The general applicability, as well as the advantages over existing modeling languages like MML (MOKA Modeling Language invented by Brimble et al. [6]), has to be shown. Summing up this discussion, figure 5 shows knowledge-based engineering of PSS approaches.

3.4 Computer Aided Configuration of PSS

To fulfill the customer needs, the configuration of a PSS is an important part of efficient development. Therefore, the approaches of configuration of PSS discussed in literature are presented in the following section. Aurich et al. [3] focus on the possible product and service architectures for PSS. The approach uses combination matrices and is based on the idea of modularization.

The approach of Laurischkat [16] considers the configurability, only of service components of PSS. She specifies that a generation of PPS (which is equivalent to a configuration) can be made with five basic types of PSS. The service components are based on the criteria of value proposition, life-cycle phase, reference and allocation, legal liability, case distinction, remote support, degree of automation and accountability. With these criterias, service components can be connected to functions of PSS by use of configuration rules (if-then rules) or decision tables.

In contrast to the pure configuration of service components, Mannweiler [18] concentrates on predefined building blocks, which are mainly product components. He presents an approach for PSS based on customers requirements. In order to fulfill these requirements, the designer aggregates these building blocks and evaluates the degree of fulfillment.

Lubarski et al. [17] focus on the modularization of service components. They set up a framework by analyzing existing modularization methods. In the course of this framework, they discuss which method is localized in which phase of modularization (information capturing, decomposition, structuring, module creation, interface definition and testing) related to the structure level (logical, temporal, combined/ complex structure).

In the approach of integrated PSS development from Bochnig et al. [5, 4] a CAE tool is invented which is based on 16 modules including a configuration tool (module 6-9). This generates PSS variants by combining existing PSS modules. The CAE tool is designed to extend and link existing development tools from various disciplines (mechanics, electronics, software and service). It displays the interdependency among different elements and implements the service by symbols in the CAD environment. As of today, there is no documented modification in the physical product model through different services. The ideas of the approaches presented in this section are summarized in figure 6 with their main features.

In summary, it can be noted that methods existing in the literature and presented in this paper contain approaches for rule-based and case-based configuration. A model-based configuration has not yet been implemented. The reason is a lack of an integrated parametric PSS model that incorporates both product and service features.

Here again the statement of Vajna [32] is taken up that in CAD the models of conventional CAD, parametric CAD, feature-based CAD and knowledge-based CAD are build on each other and have an inner connection. With incomplete models, advanced techniques quickly reach their limits and can only be applied to a restricted area.

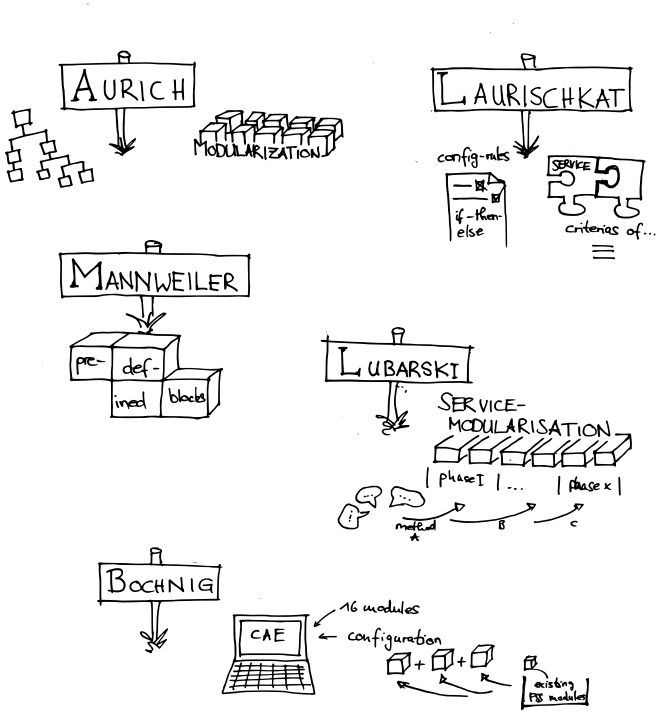


Figure 6. Computer aided configuration for PSS

This thesis fits very well with the result of the analysis of the existing approaches for design, computer aided engineering, knowledge-based engineering or computer aided configuration of PSS, with regard to the conceptuality of the approaches and their limitation to clearly defined examples. This awareness leads to the conclusion that there is a lack of a common data model for all artifacts of a PSS and their relationships.
One possibility is the development of a parametric model like the one that is already state of the art in available CAD systems for purely physical products. With such a model, the customization and variant design of PSS could be similar to that of physical products. In addition, computer-aided product optimization could be carried out [9].
This idea is further developed in the following chapter.

4 CONSTRAINT BASED APPROACH

In this section an introduction into the idea of a constraint based approach is given. Starting with the requirements on PSS models followed by showing the need and advantages of a parametric model. Subsequent an existing approach is presented and its advantages and disadvantages are shown.

4.1 Requirements on PSS Models

Requirements on PSS models depends on the requirements on the belonging PSS and the development process. The requirements for PSS, which are outlined by Ryynänen et al., can be served by the existing development tools in the classic product development. However, so far, a model is missing that PSS adequately maps for these tools [24].
According to Bochnig [4] a general PSS data model is needed for an efficient data exchange. Existing partial models of PSS form a very heterogeneous system with various data formats. The general model should, among other things, support the development process in each phase, show all components of the PSS, be able to combine elements to modules and allow links between partial models.
As shown in Chapter 3 and according to Gembarski [9], the literature-based approaches to PSS configuration are primarily rule-based or case-based systems, suggesting that parametric systems are difficult to set up for PSS and are therefore restricted to physical products.
However, a model-based configuration is not limited to physical models, as it was already showed in the 1990s by the concept of the XRAY expert system. This is a very promising concept for the configuration of physical and non-physical development artifacts based on requirements, dependencies and specification of tasks. XRAY was developed in PLAKON, an expert system core like today's software development environments. PLAKON provided necessary functions and classes for the creation of planning and configuration systems, including reasoning and conflict resolution mechanisms. The system has developed for prototypically X-ray analysis systems and fulfilled the following requirements: an interactive definition of the test task, an automatic selection and configuration of the hardware components, an automatic generation of a test plan, an automatic configuration of the software and an interactive simulation and test of the software [8, 9]. Special attention must be paid to the common configuration of hardware and software in this approach, which is essential for the efficient performance of the test tasks.
Meier [5] also points out that a parametric development of PSS can be made possible by the modeling of product and service in one data structure model. This model has to include the interactions between different parts and has the possibility to incorporate formula relationships between them .

4.2 Parametric model

Parameters are necessary to characterize the properties of a system. They can refer to geometry, kinematics, tensions, deformations, dynamics or other aspects. During the product development process, the developer defines numerous parameters "directly". This can be automated by the parametric modeling [32].
In CAD systems, relationships are established between directly defined parameters and system elements in order to reduce the number of independent parameters. The (mathematical and logical) constraints represent mathematical models. The parameters of the models usually have a hierarchical structure [32].
In figure 7 the product structure of a assembly design is shown, including the parameter-based relations.

The parameterization of geometry data in 3D-CAD leads to the separation of the management of geometry and its controlling parameters. The possibility of parameter-based control offers a wide field of application for problem-specific design applications. State of the art in CAD programs are additional functionalities, such as data interfaces, the integration of catalogue and knowledge functions, and the possibility of macro-based procedures. In product development, highly flexible development processes are possible with a development based on parametric product models [13].
These parametric models exist so far only for physical products. The complete and equitable implementation of services is a major challenge, but necessary if a parametric model is to be used for model-based configuration of PSS.
In this context, it appears meaningful to examine whether Steinbach's idea to transfer the approach of property driven development to PSS (as described in chapter 3.1) is promising [9].

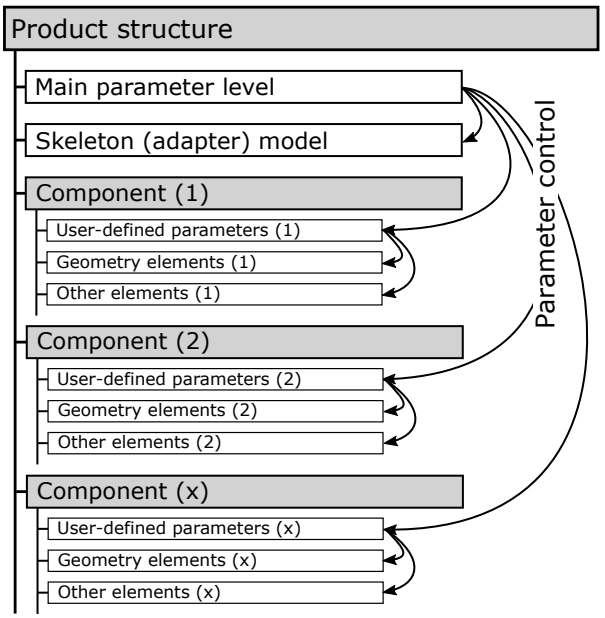


Figure 7. Parameter based relations in assembly design [13]

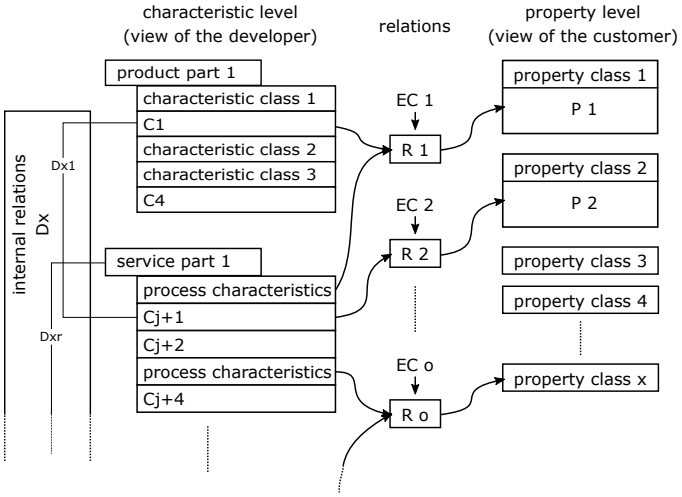


Figure 8. Concept of a PSS model based on Steinbach

4.3 Seinbachs approach

In his work, Steinbach [27] formulates the goal that a PSS must be described in such a way that both the requirements of the customer and the needs of the developer are respected. For this, it is necessary to create a structure for PSS, in which both products and services are consistently described on a common basis.
Based on the idea of Weber, the definitions of characteristics and properties are adapted to PSS. The characteristics are defined by the developer directly and define the PSS and its structure. Additionally, they contain on the service level potential and process dimension. The properties here are only indirectly influenced by the characteristics and represent the resulting dimension of the PSS [27].
Steinbach's concept can be used to represent PSS and to define and visualize the distinction between customer requirements (properties) and controllable parameters (characteristics), as well as their relations. An excerpt of this product model concept can be seen in Figure 8.
The customer requirements are notated on the property level and divided in different property classes. The product and service parts are notated on the characteristic level while product parts are subdivided in characteristic classes and services in process characteristics. The internal relationships of the PSS are notated and can exist between product parts and service parts, or between some of their characteristics [27].
Thus, an abstraction level can be found on which the entire system PSS is represented. In addition, relationships between different product and software elements are identified and referred to as internal relationships [27].
A configuration of this model could be done by adapting and changing the customer's wishes (properties), which would be passed on to the characteristics level via the relations, where they would change and influence the controllable parameters. Additionally the relations are influenced by external conditions (noted as EC). These are an important tool, that helps the developer to define restrictions, like profitability of the System or dimensions of transport capacities.
However, the implementation in a data model that is comparable to that of MCAD is missing. One reason for the lack of implementation could be that Steinbach had already published its approach more than 15 years ago and the CAD programs had not yet reached the standard of today. In addition, the application of the Steinbach approach was documented only in a very simple and theoretically constructed example. This leaves open the question whether the approach is also applicable in general and can be applied to examples with a higher complexity or other areas.
Nevertheless, the approach provides a good theoretical foundation for further development steps and researches with regard to the representation and modeling of PSS systems. It helps to capture the system of PSS and to sketch it in a first schematic design. The property and characteristic level provide a good description of the inhomogeneous PSS system and the relations between the parameters that can be influenced (characteristics) and the properties perceived by the customer. In order to configure this PSS and develop it further in a customer-specific manner and to use higher development techniques (up to knowledge-based development), a parametric model is required.
For the development of such a model, the Steinbach approach should be used as a starting point. This model and the necessary informations are already indicated in the approach by the block of internal relations. These relationships must be formulated and translated into a model.

5 CONLUSION AND FURTHER RESEARCH

In summary, the importance of research on the development of PSS is agreed in the literature. Nevertheless, it has not yet been possible to find a general approach. Since, as already described, the quality of the PSS depends on the quality of the development, further extensive and targeted research is needed in the development of PSS.
Because PSS represent solutions which fulfill the needs of customers, they must be in development configurable and adaptable to the respective customer.

5.1 Conclusion

Various approaches to develop PSS have been shown. There are approaches that look at the whole development process and approaches that look at specific kinds of development. These focus mainly on computer-based and knowledge-based development as well as a development for the computer-aided configuration.
All these approaches tend to be conceptual or concentrated on very specific examples. In this paper, it has been shown that the basic models (whether conventional or parametric) are skipped for the underlying models of these development procedures. In the literature of product development, there is the suggestion that the models (conventional, parametric, feature-based and knowledge-based) build on one another and incomplete models quickly push the advanced techniques like KBE and feature-based design to their limits. This is in line with the results of analysis of existing approaches in PSS development.
In this paper, an approach has been examined from the literature with the result that it provides a useful basis for describing a PSS. From this approach a further procedure could be derived to deduce a parametric model.

5.2 Further research

In the next steps, exemplary systems will be described with the discussed approach and parametric models will be created with the internal relationships resulting from the approach. Here, we will examine the extent to which the models can be derived directly from the formulas and the effects of different detailed formulas on the models. As CAD system, Autodesk Inventor will be used for further research and the prototypical implementation of examples. Gembarski et al. [10] showed in 2015 that in this CAD programm all common modeling techniques of knowledge-based design are available. The use of design rules is implemented in the iLogic concept and equations can directly be entered at dimensioning or in the parameter table. Additionally, it is possible to implement calculations due to the implementation by MS Excel.
Another point to be investigated is a possible change from the PSS models in the life cycle. Since an advantage of PSS is the configuration in the usage phase, the model would have to be adapted accordingly, because an existing product can not be parameterized and configured as desired.

ACKNOWLEDGEMENTS

This research was conducted in the scope of the research project SmartHybrid – Product Engineering (ID: 85003608) which is partly funded by the European Regional Development Fund (ERDF) and the State of Lower Saxony (Investitions- und Förderbank Niedersachsen NBank). We like to thank them for their support.

REFERENCES

[1] Fumiya Akasaka, Yutaro Nemoto, Koji Kimita, and Yoshiki Shimomura, 'Development of a knowledge-based design support system for product-service systems', **63**(4), 309–318.
[2] Alessandro Annarelli, Cinzia Battistella, and Fabio Nonino, 'Product service system: A conceptual framework from a systematic review', **139**, 1011–1032.
[3] J.C. Aurich, C. Fuchs, and C. Wagenknecht, 'Life cycle oriented design of technical product-service systems', **14**(17), 1480–1494.
[4] Holger Bochnig, Eckart Uhlmann, Hoi Nam Nguyn, and Rainer Stark, 'General data model for the IT support for the integrated planning and development of industrial product-service systems', in *Product-Service Integration for Sustainable Solutions*, ed., Horst Meier, 521–533, Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-30820-8_44.
[5] Holger Bochnig, Eckart Uhlmann, and Alexander Ziefle, 'Assistenzsystem IPSS-CAD als informationstechnische untersttzung der integrierten sach- und dienstleistungsentwicklung in der IPSS-entwurfsphase', in *Industrielle Produkt-Service Systeme*, eds., Horst Meier and Eckart Uhlmann, 95–115, Springer Berlin Heidelberg. DOI: 10.1007/978-3-662-48018-2_5.
[6] Richard Brimble and Florence Sellini, 'The MOKA modelling language', in *Knowledge Engineering and Knowledge Management Methods, Models, and Tools*, eds., Rose Dieng and Olivier Corby, volume 1937, 49–56, Springer Berlin Heidelberg. DOI: 10.1007/3-540-39967-4_4.
[7] Sergio Cavalieri and Giuditta Pezzotta, 'Productservice systems engineering: State of the art and research challenges', **63**(4), 278–288.
[8] Roman Cunis, Andreas Gnter, and Helmut Strecker. Das PLAKON-buch. DOI: 10.1007/978-3-662-06485-6.
[9] Paul Christoph Gembarski and Roland Lachmayer, 'Mass customization und product-service-systems: Vergleich der unternehmenstypen und der entwicklungsumgebungen', in *Smart Service Engineering*, eds., Oliver Thomas, Markus Nüttgens, and Michael Fellmann, 214–232, Springer Fachmedien Wiesbaden. DOI: 10.1007/978-3-658-16262-7_10.
[10] Paul Christoph Gembarski, Haibing Li, and Roland Lachmayer, 'KBE-modeling techniques in standard CAD-systems: Case studyautodesk inventor professional', in *Managing Complexity*, eds., Jocelyn Bellemare, Serge Carrier, Kjeld Nielsen, and Frank T. Piller, pp. 215–233. Springer International Publishing. DOI: 10.1007/978-3-319-29058-4_17.
[11] Marc Gräle, Thomas Thomas, Michael Fellmann, and Julian Krumeich, 'Vorgehensmodelle des product-service systems engineering: berblick, klassifikation und vergleich'.
[12] Tatsunori Hara, Tamio Arai, and Yoshiki Shimomura, 'A concept of service engineering: A modeling method and a tool for service design', pp. 13–18. IEEE.
[13] Mario Hirz, Wilhelm Dietrich, Anton Gfrerrer, and Johann Lang, *Integrated computer-aided design in automotive development*, Springer.
[14] Patrick Klein. Definition and development of KBE-systems.
[15] Katrin Kuntzky, *Systematische Entwicklung von Produkt-Service-Systemen*, Schriftenreihe des Instituts für Werkzeugmaschinen und Fertigungstechnik der TU Braunschweig, Vulkan-Verl. OCLC: 862841861.
[16] Katja Laurischkat, *Product-Service Systems: IT-gestützte Generierung und Modellierung von PSS-Dienstleistungsanteilen*, number 2012,3 in Schriftenreihe des Lehrstuhls für Produktionssysteme, Ruhr-Universität Bochum, Shaker. OCLC: 830664357.
[17] Aleksander Lubarski and Jens Poeppelbuss, *Methods for Service Modularization - a systematization framework*, number 277 in PACIS 2016 Proceedings. OCLC: 964654802.
[18] Carsten Mannweiler, *Konfiguration investiver Produkt-Service Systeme*, number 2014,1 in Produktionstechnische Berichte aus dem FBK, Lehrstuhl für Fertigungstechnik und Betriebsorganisation, Techn. Univ, als ms. gedr edn. OCLC: 894132943.
[19] H. Meier, R. Roy, and G. Seliger, 'Industrial product-service systemsIPS2', **59**(2), 607–627.
[20] O.K Mont, 'Clarifying the concept of productservice system', **10**(3), 237–245.
[21] Nicola Morelli, 'Developing new product service systems (PSS): methodologies and operational tools', **14**(17), 1495–1501.
[22] Patrick Müller and Rainer Stark, *Integrated engineering of products and services: [layer-based development methodology for product-service systems]*, Berichte aus dem Produktionstechnischen Zentrum Berlin, Fraunhofer Verl. OCLC: 931607716.
[23] Karlheinz Roth and Karlheinz Roth, *Konstruktionslehre*, number Karlheinz Roth ; Bd. 1 in Konstruieren mit Konstruktionskatalogen, Springer, 3. aufl., erw. und neu gestaltet edn. OCLC: 313839832.
[24] Tapani Ryynänen, Iris Karvonen, Kim Jannson, Heidi Korhonen, Matteo Cocco, Donatella Corti, and Reinhard Ahlens. Structuration and organization of requirements.
[25] Tomohiko Sakao, Yoshiki Shimomura, Erik Sundin, and Mica Comstock, 'Modeling design objects in CAD system for service/product en-

gineering', **41**(3), 197–213.
[26] Dieter Spath and Lutz Demuß, 'Entwicklung hybrider produkte gestaltung materieller und immaterieller leistungsbündel', in *Service Engineering*, eds., Hans-Jrg Bullinger and August-Wilhelm Scheer, 463–502, Springer-Verlag. DOI: 10.1007/3-540-29473-2_20.
[27] Michael Steinbach, *Systematische Gestaltung von Product-Service-Systems: integrierte Entwicklung von Product-Service-Systems auf Basis der Lehre von Merkmalen und Eigenschaften*, number 35 in Schriftenreihe Produktionstechnik, LKT. OCLC: 162223620.
[28] Flavius Sturm, Alexandra Bading, and Michael Schubert, *Investitionsgterhersteller auf dem Weg zum Lsungsanbieter: eine empirische Studie ; fit2solve*, IAT, Univ. OCLC: 213388428.
[29] Oliver Thomas, Philipp Walter, and Peter Loos, 'Konstruktion und anwendung einer entwicklungsmethodik für product-service systems', in *Hybride Wertschpfung*, eds., Oliver Thomas, Peter Loos, and Markus Nüttgens, 61–81, Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-11855-5_4.
[30] Arnold Tukker, 'Eight types of productservice system: eight ways to sustainability? experiences from SusProNet', **13**(4), 246–260.
[31] Arnold Tukker, 'Product services for a resource-efficient and circular economy a review', **97**, 76–91.
[32] Sandor Vajna, Christian Weber, and Peter Hehenberger. CAx für ingenieure: eine praxisbezogene einführung. OCLC: 301962310.
[33] Gokula Vijaykumar Annamalai Vasantha, Rajkumar Roy, Alan Lelah, and Daniel Brissaud, 'A review of productservice systems design methodologies', **23**(9), 635–659.
[34] Christian Weber, 'Modelling products and product development based on characteristics and properties', in *An Anthology of Theories and Models of Design*, eds., Amaresh Chakrabarti and Lucienne T. M. Blessing, 327–352, Springer London. DOI: 10.1007/978-1-4471-6338-1_16.
[35] Xiaoyu Yang, Philip Moore, Jun-Sheng Pu, and Chi-Biu Wong, 'A practical methodology for realizing product service systems for consumer products', **56**(1), 224–235.

Complexity of Configurators Relative to Integrations and Field of Application

Katrin Kristjansdottir[1] and **Sara Shafiee** and **Lars Hvam** and **Loris Battistello** and **Cipriano Forza**

Abstract. Configurators are applied widely to automate the specification processes at companies. The literature describes industrial application of configurators supporting both sales and engineering processes, where configurators supporting the the engineering processes are described more challenging. Moreover, configurators are commonly integrated to various IT systems within companies. Complexity of configurators is an important factor when it comes to performance, development and maintenance of the systems. Yet, a direct comparison of the complexity based on the different application and IT integrations is not addressed to great extent in the literature. Thus, this paper aims to analyse the relationship of complexity of the configurators, which is based on parameters (rules and attributes), in terms of first different applications of configurators (sales and engineering), and second integrations to other IT systems. The research method adopted in the paper is based on a survey followed with interviews where the unit of analysis is based on operating configurators within a company.

1 INTRODUCTION

In today's business environment customers are increasingly demanding high quality customised products, with short delivery time, and at competitive prices [1]. To respond to those increasing demands, mass customisation strategies have received increasing attention from both practitioners and researchers. Mass customisation refers to the ability to make customised products and services that fit all customers' needs through flexibility and integration at similar costs to mass-produced products [2]. Configurators are used to support design activities throughout the customisation process in which a set of components and connections are pre-defined and constraints are used to prevent infeasible configurations [3].

Configurators can be used to support different specification process at companies, which can include sales, design/engineering and/or production. Configurators can bring substantial benefits, such as shorter lead times for generating quotations, fewer errors, increased ability to meet customers' requirements regarding product functionality, use of fewer resources, optimised product designs, less routine work and improved on-time delivery [4–8].

Configurators used to support the engineering processes are considered more complex [1,9]. However, a direct comparison of configurators to support the different applications within the same company has not been conducted. Furthermore, in configuration projects there is usually the need of integration to IT systems, such as ERP, CAD, PLM and PIM systems. However, the literature does not address what influences it will have on the configurators complexity when integrations to other system are made.

In this paper the complexity of configurators is determined based on parameters, or number of rules and attributes, included in the configurators. By analysing the complexity in terms of application, configurators supporting sales and engineering processes, and in relation to different integrations, it will give more understanding of what factors influence the complexity of the configurators. Complexity of configurators is a relevant topic as it influences the performance of the system and affects the effort needed in terms of development and maintenance. Nevertheless, complexity can be both good and bad depending on whether it is value adding or not. This paper therefore aims to provide more understanding on factors influencing complexity of configurators by providing answers to the following research questions (RQs):

> ***RQ 1*:** What are the differences in terms of complexity between sales and engineering configurators?
>
> ***RQ 2*:** What are the differences in terms of complexity when configurators are integrated to other IT systems?

To answers to the RQs, a survey followed with interviews is conducted. The results presented in this paper are preliminary as this is an ongoing study. This includes analysis based on one company where the unit of analysis is based on operating configurators within the company.

The structure of the paper is as follows. Chapter 2 discusses the literature background for the study, and Chapter 3 explains the research method. Chapter 4 presents the results of the research, and Chapter 5 discusses the results in relation to the RQs and presents the conclusion.

[1] Management Engineering, Technical University of Denmark, email: katkr@dtu.dk

2 Literature Review

This section aims to provide the background for the study. Section 2.1 discusses configurators and integrated system, and provides definition of configurators complexity. Section 2.2 discusses the difference between configurators supporting sales and engineering processes.

2.1 Configurators and Integrated Systems

The underlying IT structure of a configurator consists of configuration knowledge representation and reasoning, conflict detection and explanation, and finally an user interface [10]. Configurators can be applied as standalone software, as well as data-integrative and application-integrative systems [11]. Data-integrative configurators can be used to avoid data redundancies and application-integrative configurators allow for communication across different applications (e.g. CAD drawings can be generated from the output of the configurator) [11]. In terms of data integration for configurators, common sources for master data can be found in Enterprise resource planning (ERP) systems that often define a production-relevant view of the material. This is required for the assembly process, product data management (PDM) and product lifecycle management (PLM) systems, which are used to maintain production relevant data. Finally, product information management (PIM) systems are used to maintain sales-relevant data [12]. Different configurators can be integrated in terms of, for example, sales and engineering configurators [13]. Finally, configurators can be integrated into suppliers systems to retrieve the required data from the configuration processes [14].

To measure the complexity of configurators, Brown et al. [15] categorize them into three major components; (1) execution complexity, (2) parameter complexity, and (3) memory complexity. Execution complexity covers the complexity involved in performing the configuration actions that make up the configuration procedure and the memory complexity refers to the number of parameters that system manager must remember. In this paper, the parameter complexity is considered the most important, as it measures the complexity of providing configuration data to the computer system during a configuration procedure [15]. Therefore, the article focuses on parameters complexity to determine the complexity of the configurators. The parameter complexity is determined based attributes and rules included in the configurators.

2.2 Sales and Engineering Configurators

Configurators are used to support the product configuration process, which consists of a set of activities that involve gathering information from customers and generating the required product specifications [13,16]. The product configuration process can be divided into sales and technical configuration processes [17]. The sales configuration process is concerned with identifying products that fulfil customers' needs and determining the main characteristics of the products [17]. The technical configuration process, on the other hand, is concerned with generating documentation for the product based on the input gathered during the sales phase [17]. In this article, the technical configurations are referred to as the configurators supporting the engineering processes. Another dimension of the configuration process is production configuration [18].

The challenges of configurators used to support the engineering companies are described in terms product characteristics, customer relations, and long time span of projects [19]. Further, the sales process in engineering companies can be categorized where a high-level design is made in the sales phase and the actual design processes does not start before the sale is confirmed. Thus, sales configurators in engineering companies are often modeled on high level of abstraction where the engineering configurators that are concerned with the actual design of the product have to include more detailed information [4]. This usually leads to higher complexity of the configurators supporting the engineering than the sales processes.

3 RESEARCH METHOOD

The chosen research method for this article is survey followed with interviews. As this is still ongoing study only one company is analyzed. However, by only including one company it was possible to get an in-depth knowledge about the configuration setup and compare the complexity of the configurators within the same settings. The unit of analysis is based on operational configurators at the company, where a configurator is defined as a system that has its own knowledge base or product model and user interface. The company uses commercial configuration software for all of their configurations. Meaning that the same modelling paradigms are used in the company for all the configurators, which is a requirement to compare of the complexity of the different configurators.

The case company introduced in the study has a world leading position in providing process plants and related equipment for industrial use. The company has utilized configurators since 1999 and has currently 159 operational configurators, which support the product specification processes both in sales and the engineering. The company therefore has an extensive experience from working with configurators.

To analyse the complexity of the configurators first a questioner was developed and reviewed several times by the research team in order to check consistency and understandability. Secondly, the questionnaire was emailed to the company and an interview was setup. Based on the first interview it was decided that the data gathering would be conducted in collaboration with one of the project manager from the configuration team for two days. The data was gathered from internals systems and evaluated by the project manager to check accuracy and consistency.

The data was then analyzed in Microsoft Excel in relation to the RQs. First, the configurators, were grouped according to processes they supported, or into sales, sales and engineering, engineering and few configurators where grouped under others. A limitation of the data is that the majority of the configurators are used to support the

engineering processes (75%), and sales and engineering processes (19%) while there are few configurators used to support only sales processes (3%) and finally configurators used to support other processes are (2%). Nevertheless, the results presented are thought to provide valuable insight into the parameters complexity of configurators, while further data gathering is planned to support the findings. Secondly, the data related to the configurators integrated IT systems was grouped. In cases where there is more than one integration to the configurators they were listed under combination of integrations, which included the following combinations: (1) CAD and ERP, (2) CAD, ERP and calculation systems, and finally (3) ERP and calculation system. This is required as the focus of the study is to analyze integrations to what IT systems results in the most complexity and therefore including combinations of integrations would give biased results.

4 RESULTS

In this chapter the main result from the survey are presented aligned with the two RQs introduce in the paper.

Section 4.1 elaborates on the complexity of the configurators used in the sales, both in sales and engineering processes and finally only in the engineering processes (RQ 1). Section 4.2 elaborates and the complexity of the configurators in relation to integrations to IT systems (RQ 2). The integrations include, ERP, CAD, calculation systems, integrations to other systems or combination of systems and finally few configurators that have no integrations. The results presented are based on data from 159 configurators that are used within on company as explained in Section 3.

4.1 Complexity in Relation to Engineering and Sales Configurators

This section provides the results in relation to the complexity based on sales and engineering configurators. Figure 1 shows the percentages of configurators used to support the (1) sales, (2) sales and engineering, (3) engineering, and finally (4) other activities.

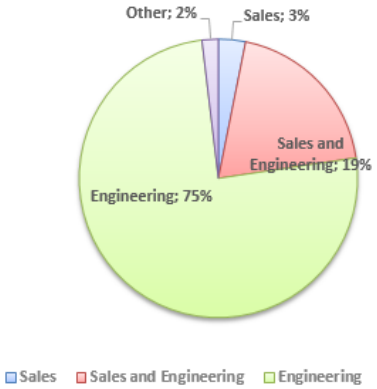
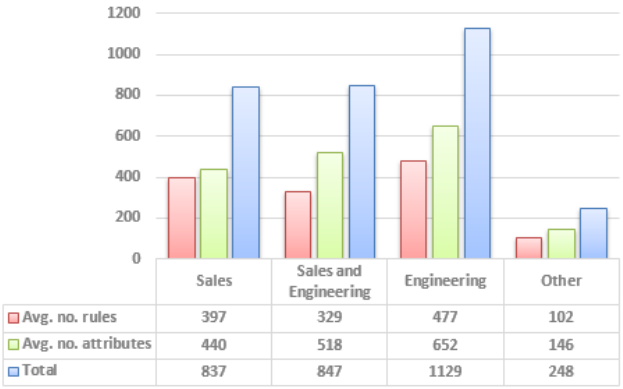


Figure 1. Percentages of configurators used to support different activates at the company.

As can be seen in Figure 1 only 5% of the total configurators support the sales processes, while 19% of the configurators are used to support both sales and engineering, 75% of the configurators are used to support only engineering and 2% support other activities.

The complexity of the configurators used for the different activities are shown in Figure 2 in terms of average numbers of rules and attributes and total where the numbers of rules and attributes are summarized.



	Sales	Sales and Engineering	Engineering	Other
Avg. no. rules	397	329	477	102
Avg. no. attributes	440	518	652	146
Total	837	847	1129	248

Figure 2. Complexity of the configurators used to support the different activities at the company.

Figure 2 shows that in terms of rules configurators used by engineering have on average 477, while sales have 397 and configurators used by sales and engineering have on average 329. In terms of attributes, configurators used by engineering have on average the most attributes or 652, while configurators used by sales and engineering have on average 518 and sales have 440. Finally, as previously defined, the complexity of the configurators is determined based on parameters or the sum of attributes and rules. Thus, configurators supporting only engineering activities have the highest total score of complexity or 1129 while if we look at the configurators only supporting sales or sales and engineering the total score is 837 and 847 respectively. Other configurators supporting simpler tasks at the company have the lowest rate of complexity or only 248.

4.2 Complexity of Configurators in Relation to Integrations

In the company used for this study, the application of the configurators was divided according the integrations. The integrations included the following IT systems (1) ERP, (2) CAD, (3) calculation systems, (4) combination of the above mentioned systems, and in few case (5) other systems. Only 4% of the configurators did not have any integration, while 70% of the configurators were integrated to one of the above mentioned systems and 26% were integrated to one or more of the systems. Figure 3 shows the percentages of integrations the different configurators have.

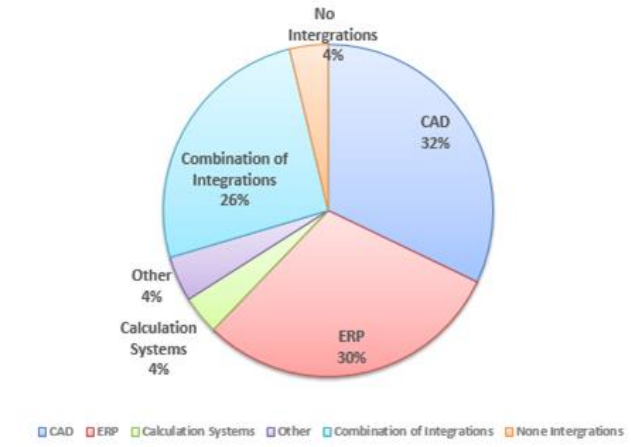
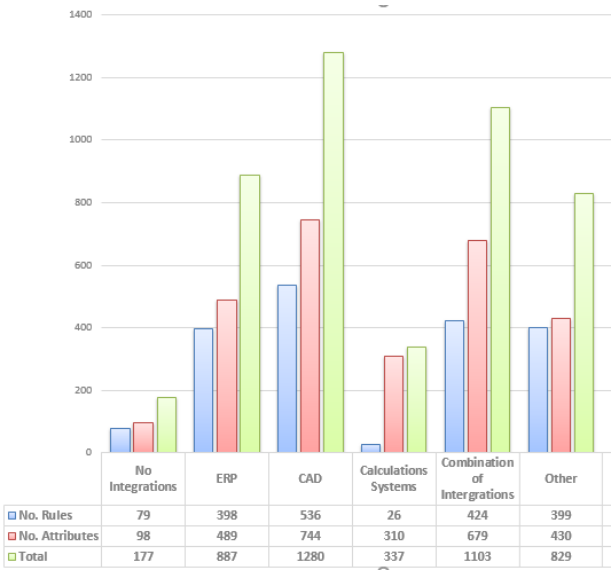


Figure 3. Percentages of integrations and combinations of integrations to different IT systems used at the company.

As can be seen in Figure 3 the majority of the configurators are intergraded to the CAD and the ERP system used at the company or 32% and 30% respectively while only 4% are integrated only to calculation systems or other IT systems used at the company. Finally, 26% of the configurators are integrated to more than one of the above mentioned IT systems.

The complexity of the configurators integrated to the different IT systems is shown in Figure 4 in terms of average numbers of rules, attributes and then the sum of the average rules and attributes.



	No Integrations	ERP	CAD	Calculations Systems	Combination of Intergrations	Other
No. Rules	79	398	536	26	424	399
No. Attributes	98	489	744	310	679	430
Total	177	887	1280	337	1103	829

Figure 4. The main characteristics of the configurators integrated to different IT systems at the company.

From Figure 4 it can be seen that in terms of both attributes and rules the configurators integrated to CAD system score the highest in terms of complexity. Configurators that have combinations of integrations, or more than one integration, have the second highest score. That can be explained by the fact that in most cases that also includes and integration to a CAD system. By looking into configurators that have integrations to calculation systems it can be seen that they have the fewest rules, may be due to the calculations being performed within another system. Finally, it can be seen that configurators with no integration have the lowest complexity factor.

5 DISSCUSSIONS AND CONCLUSIONS

This study provides insights into the complexity of the configurator where the complexity is analysed based on parameters, which consists of numbers of attributes and rules. The complexity is analysed first based on field of application (sales and engineering) and then based on integrations to different IT systems. The results provided in the present article aim to contribute to the field of configurators' complexity and the factors influencing them. This is an important topic not only for the research community but also for practitioners. The results show that a difference can be found in relation to the complexity by analysing the field of application and different kind of integrations.

The first research question in this study aims to identify if there is any relationship between the complexity of the configurators and the field of applications. Our analysis show that the configurators that are only aimed at supporting the engineering processes have the highest parameters complexity. However, there was only a slight difference between the complexity factor of the configurators only used to support sales and the configurators used to support both sales and engineering.

The second research question aims to analyse the relationship between integrations and complexity of the configurators. In the literature, it is discussed how configurators are integrated to different IT systems e.g., [11–14,18]. However, the literature does not explain to what extent the integrations to different IT system will influence the complexity level of the configurators. In this paper integration to CAD, ERP and calculation systems is analyzed. The result shows out of the above mention IT systems the complexity of the configurators integrated to CAD systems is the highest. This can be supported by the fact that in order to generate CAD files from the configurators, they have to be able to support the detail design including all the product dimensions, which will increase the complexity. Thus, configurators integrated to CAD systems can be defined as product design configurators, which support the engineering processes where the complexity can be anticipated to be higher even though not integrated to a CAD system. Configurators integrated to ERP systems scored as the second highest while configurators integrated to calculation systems scored the lowest out of those systems. When configurators are integrated to calculation a system the reason is usually that the calculations being too complex or specialized to handle within the configurator. This supports the fact that configurators integrated to calculations systems have very low number of rules and thereby they also have low parameters complexity.

The result presented in the paper is based on answers and interviews from one company. This is thought to provide a

valuable insight as by studying one company an in-depth knowledge about the configuration setup could be accessed. Furthermore, it allows comparison of the complexity as all the configurators are developed based on the same commercial configuration platform. More companies will be contacted in the future, to enable cross-functional comparison.

REFERENCES

[1] L. Hvam, J. Riis, N.H. Mortensen, Product customization, Springer, Berlin Heidelberg, 2008.

[2] B.J. Pine II, B. Victor, Boyton, 'Making mass customization work', *Harvard business review*, **71,** 109–119, (1993).

[3] A. Felfernig, G.E. Friedrich, D. Jannach, 'UML as domain specific language for the construction of knowledge-based configuration systems', International Journal of Software Engineering and Knowledge Engineering, **10**, 449–469, (2000).

[4] A. Haug, L. Hvam, N.H. Mortensen, 'The impact of product configurators on lead times in engineering-oriented companies', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **25**, 197–206, (2011).

[5] L. Hvam, A. Haug, N.H. Mortensen, C. Thuesen, 'Observed benefits from product configuration systems', *International Journal of Industrial Engineering-Theory Applications and Practice*, **20**¸ 329-338, (2013).

[6] A. Trentin, E. Perin, C. Forza, 'Product configurator impact on product quality', *International Journal of Production Economics*, **135**, 850–859 (2012).

[7] L.L. Zhang, 'Product configuration: a review of the state-of-the-art and future research', *International Journal of Production Research*, **52**, 6381-6398, (2014)

[8] A. Trentin, E. Perin and C. Forza, 'Overcoming the customization-responsiveness squeeze by using product configurators: Beyond anecdotal evidence', *Computers in Industry*, **62**, 260-268, (2011).

[9] S. Shafiee, L. Hvam, A. Haug, M. Dam, K. Kristjansdottir, 'The documentation of product configuration systems: A framework and an IT solution' *Advanced Engineering Informatics*, **32**, 163–175, (2017).

[10] A. Felfernig, L. Hotz, C. Bagley, J. Tiihonen, Knowledge-based configuration: From research to business cases, Morgan Kaufman, 2014.

[11] T. Blecker, N. Abdelkafi, G. Kreutler, G. Friedrich, 'Product configuration systems: state of the art, conceptualization and extensions, (2004).

[12] T. Krebs, 'Encoway', in: A. Felfernig, L. Hotz, C. Bagley, J. Tihonen (Eds.), Knowledge-Based Config. From Research to Bussiness Cases, Morgan Kaufman, pp. 271–279, 2014.

[13] C. Forza, F. Salvador, Product information management for mass customization, Palgrave Macmillan, New York, 2007.

[14] L. Ardissono, A. Felfernig, G. Friedrich, A. Goy, D. Jannach, G. Petrone, R. Schafer, M. Zanker, 'A Framework for the Development of Personalized, Distributed Web-Based Configuration Systems', *AI Magazine*, **24,** 93, (2003).

[15] A.B. Brown, A. Keller, J.L. Hellerstein, 'A Model of Configuration Complexity and its Application to a Change Management System' *Integrated Network Management, 2005. IM 2005. 2005 9th IFIP/IEEE International Symposium on*, **4**, 13-27, (2007).

[16] C. Forza, F. Salvador, 'Managing for variety in the order acquisition and fulfilment process: The contribution of product configuration systems', *International journal of production economics*, **76**, 87-98, (2002).

[17] C. Forza, F. Salvador, 'Application support to product variety management', *International Journal of Production Research*, **46**, 817-836, (2008)**.**

[18] L.L. Zhang, E. Vareilles, M. Aldanondo, 'Generic bill of functions, materials, and operations for SAP2 configuration', *International Journal of Production Research*, **51**, 465-478, (2013)..

[19] T. D. Petersen, "Product Configuration in ETO Companies," in Mass Customization Information Systems in Bussines, T. Blecker, Ed. Igi Global, 2007, ch. 3, pp. 59 – 76.