Proceedings of the Workshop on Configuration at ECAI 2012 (ConfWS'12)

August 27, 2012

Montpellier, France

Wolfgang Mayer and Patrick Albert, Editors

Foreword

In many cases competitiveness of modern products is defined by the degree of customization, i.e. the ability of a manufacturer to adapt a product according to customer requirements. Knowledge based configuration methods support the composition of complex systems from a set of adjustable components. However, there are two important prerequisites for a successful application of knowledge-based configuration in practice: (a) expressive knowledge representation languages, which are able to capture the complexity of various models of configurable products and (b) powerful reasoning methods which are capable of providing services such as solution search, optimization, diagnosis, etc. The Configuration Workshop aims to bring together industry representatives and researchers from various areas of AI to identify important configuration scenarios found in practice, exchange ideas and experiences and present original methods developed to solve configuration problems.

The workshop continues the series of successful Configuration Workshops started at the AAAI'96 Fall Symposium and continued on IJCAI, AAAI, and ECAI since 1999. During this time the focus of the events broadened from configuration approaches applied to traditional products such as cars, digital cameras, PC, telecommunication switches or railway interlock systems to configuration of software and services available on the Web. In parallel, research in the field of constraint programming, description logic, non-monotonic reasoning, fundamentals of configuration modeling and so forth pushed the limits of configuration systems even further.

The papers selected this year for presentation on the Configuration Workshop continue a recent trend in the research community and focus on modeling and solving of configuration problems.

Wolfgang Mayer and Patrick Albert
July 2012

Workshop Organization

Program Co-Chairs

Wolfgang Mayer	University of South Australia, Adelaide, Australia
Patrick Albert	IBM, France

Program Committee

Tomas Axling	Tacton Systems AB, Sweden
Claire Bagley	Oracle Corporation, USA
Conrad Drescher	University of Oxford, UK
Alexander Felfernig	Technische Universität Graz/Configworks, Austria
Gerhard Friedrich	AAU Klagenfurt, Austria
Albert Haag	SAP AG, Germany
Alois Haselboeck	Siemens AG, Austria
Lothar Hotz	Universität Hamburg, Germany
Klas Orsvarn	Tacton System AB, Sweden
Markus Stumptner	University of South Australia, Australia
Barry O'Sullivan	Cork Constraint Computation Centre, Ireland
Juha Tiihonen	Aalto University, Finland

Contents

Wolfgang Mayer and Patrick Albert, Editors.
Proceedings of the Workshop on Configuration at ECAI 2012 (ConfWS'12),
August 27, 2012, Montpellier, France.

Maintaining alternative values in constraint-based configuration

Caroline Becker and **Hélène Fargier** [1]

Abstract. Constraint programming techniques are widely used to model and solve interactive decision problems, an especially configuration problems. In this type of application, the configurable product is described by means of a set of constraint bearing on the configuration variables. The user then interactively solves the CSP by assigning (and possibly, relaxing) the configuration variables according to her preferences. The aim of the system is then to keep the domains of the other variables consistent with these choices. Since maintaining of the global inverse consistency is generally not tractable, the domains are instead filtered according to some level of local consistency, e.g. arc-consistency.

In the present work, we aim at offering a more convenient interaction by providing the user with possible alternative values for each of the already assigned variables - i.e. the values that could replace the current one without leading to the violation of some constraint. We thus present the new concept of *alternative domains* in a (possibly) partially assigned CSP. We propose a propagation algorithm that computes all the alternative domains in a single step. Its worst case complexity is comparable with the one of the naive algorithm that would run a full propagation for each variable, but its experimental efficiency is much better.

1 Introduction

The Constraint Satisfaction Problem (CSP) formalism offers a powerful framework for representing a great variety of problems, e.g. routing problems, resource allocation, frequency assignment, configuration problems, etc. The main task addressed by the algorithms is the determination of the consistency of the CSP and/or the search for an (optimal) solution, and this is a difficult task: determining whether a CSP is consistent is an NP-complete request. In the CSP community, the main research stream thus addresses this question, either directly (looking for efficient complete algorithms) or getting around (studying the polynomial subclasses or proposing incomplete algorithms).

But these algorithms do not help solving decision support problems that are interactive in essence. For such problems, the user herself is in charge of the choice of values for the variables and the role of the system is not to solve a CSP, but to help the user in this task. Constraint-based product configuration [14, 18, 12, 19, 20] is a typical example of such problems: a configurable product is defined by a finite set of components, options, or more generally by a set of attributes, the values of which have to be chosen by the user. These values must satisfy a finite set of configuration constraints that encode the feasibility of the product, the compatibility between components, their availability, etc.

Several extensions of the CSP paradigm have been proposed in order to handle the constraints-based definition of a catalog or a range of products, and more specifically the definition of configurable products. These extensions have been motivated by difficulties and characteristics that are specific to the modeling and the handling of catalogs of configurable products. Dynamic CSPs [13], for instance suit the problems where the existence of some optional variables depends on the value of another variable. Other extensions proposed by the CSP community include composite CSPs [17], interactive CSPs [10], hypothesis CSPs [1], generative constraint satisfaction [19, 7], etc.

In this article, we do not deal with such representation problems: we assume that the product range is specified by a classical CSP. Instead, our work focuses on the human-computer interaction. When configuring a product, the user specifies her requirements by interactively giving values to variables. Each time a new choice is made, the domains of the variables must be pruned so as to ensure that the values available for the further variables can lead to a feasible product (i.e., a product satisfying all the initial configuration constraints): the aim of the system is to keep the domains of the other variables consistent with these choices. Since the maintaining of the global inverse consistency is generally not tractable, the domains are rather filtered according to some level of local consistency, e.g. arc-consistency. In the present paper, we propose to make this interaction more user-friendly by showing not only (locally) consistent domains, but also what we call the alternative domains of the assigned variables, i.e. the values that could replace the one of the assigned variable without leading to the violation of some constraint.

The structure of the present article is as follows: the problematics of alternative domains is described in the next Section. Section 3 then develop the basis of our algorithm. Our first experimental results are shown in Section 4. Proofs are gathered in Appendix.

2 Background and Problematics

A CSP is classically defined by a triplet $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ where $\mathcal{X} = \{x_1, \ldots, x_m\}$ is a finite set of m variables, each x_i taking its values in a finite domain $D(x_i)$, and a finite set of constraints

[1] IRIT, University Toulouse III, France, email: {becker,fargier}@irit.fr

$\mathcal{C}$. We note $\mathcal{D} = \prod_{j=1}^{n} D(x_j)$. An assignment t of a set of variable $\mathcal{Y} \subseteq \mathcal{X}$ is an element of the cartesian product of the domains of these variables; for any $x_j \in \mathcal{Y}$ we denote by $t[x_j]$ the value assigned to x_j in t.

A constraint C in $\mathcal{C}$ involves a set $vars(C) \subseteq \mathcal{X}$ and can be viewed as a function from the set of assignments of $vars(C)$ to $\{\top, \bot\}$: $C(t) = \top$ iff t satisfies the constraint; for any x_j in $vars(C)$ and any v in its domain, we say that an assignment t of $vars(C)$ is a support of this value (more precisely, of (x_j, v) on C) iff $t[x_j] = v$ and t satisfies C.

An assignment t of $\mathcal{X}$ is a solution of the CSP iff it satisfies all the constraints. If such a solution exists, the CSP is said to be consistent, otherwise it is inconsistent.

Formally, a configurable product is represented as a CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ and the current choices of the user by a set of couples (x_i, v) where x_i is a variable in $\mathcal{X}$ and v the value assigned to this variable. Following [1], the problem can be represented by an Assumption-based CSP (A-CSP).

Definition 1 (A-CSP) *An* A-CSP *is a* 4*-uple* $(\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H})$ *where* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *is a CSP and* $\mathcal{H}$ *a finite set of constraints on variables of* $\mathcal{X}$*.*

In configuration, $\mathcal{H}$ represents the set of current user choices, i.e. assignments of the variables: *we suppose in the sequel of the paper that all the restrictions in* $\mathcal{H}$ *bear with different variables and restrict their domain to a unique value*[2]; we will denote by $h_i = (x_i \leftarrow v)$ the restriction from $\mathcal{H}$ on x_i, if it exists.

After each choice, the system filters the variables' domains, ideally leaving only the values compatible with current choices. Since such a computation is intractable in the general case, a weaker level of consistency is ensured in real applications, generally arc-consistency. Recall that a CSP is said to be arc consistent in the general sense (GAC) iff, for any variable $x_j \in \mathcal{X}$ and any value v in its domain, for any constraint C bearing on x_j, there exists an assignment t of the variables of C in their domains such that t is a support of (x_j, v). The role of an arc consistency algorithm is to remove from the domains the values that do not have any support so as to compute a CSP that is equivalent to the original one (i.e. having the same set of solution) and that is arc consistent; this problem is called the closure by arc consistency of the original one.

Other, more powerful, levels of local consistency can be ensured, e.g. Path Inverse Consistency [4], Singleton Arc Consistency [5], k-inverse consistency [8, 9]. In the following definitions, we do not make any assumption on the level of local consistency that is ensured. We simply consider that, after each choice, an algorithm is called that ensures some level l of local consistency - i.e. that computes the closure by l consistency of the original problem. We call the *current domain* of a variable its domain in this closure.

Definition 2 (Current domain of a variable) *Let l be a level of local consistency and* $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H})$ *an A-CSP. The* current domain *according to l of a variable x_i is its domain in the closure by l-consistency of* $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H})$.

We can now formally define the notion of alternative domain of an assigned variable as the current domain that it would have if the user would take this assignment back:

Definition 3 (Alternative domain) *Let l be a level of local consistency and* $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H})$ *an A-CSP. The alternative domain of a variable x_i according to l is its domain in the closure by l-consistency of the CSP*$(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H} \setminus \{h_i\})$*. We write it* $D^l_{alt}(x_i)$*.*

A value v is thus an alternative value for x_i either if it belongs to the current domain of x_i (it is in particular the case when x_i is assigned to v), or if (i) x_i is assigned another value than v and (ii) the single relaxation of this assignment would make v l-consistent. For instance, if x_i is the last assigned variable, all the values that were in the domain of x_i just before its assignment are alternative values.

Example 1 Consider the CSP $\mathcal{X} = \{x_1, x_2, x_3\}$, $\mathcal{D} = D_1 \times D_2 \times D_3 = \{1, 2, 3, 4\}^3$, $\mathcal{C} = \{Alldiff(x_1, x_2, x_3)\}$; initially, $\mathcal{H} = \emptyset$ and the current domains of the three variables are $D_C(x_1) = D_C(x_2) = D_C(x_3) = \{1, 2, 3, 4\}$. In this example, we suppose that that arc consistency is maintained.
Let theuser first assign value 1 to x_1. We get $\mathcal{H} = \{(x_1 = 1)\}$; then $D_C(x_1) = \{1\}$ and arc consistency removes value 1 from the current domains of x_2 and x_3: $D_C(x_2) = D_C(x_3) = \{2, 3, 4\}$. At this step, x_1 is the only assigned variable and has three alternative values, 2, 3 and 4.
Suppose that the user then assigns value 4 to x_2, i.e. $\mathcal{H} = \{(x_1 = 1), (x_2 = 4)\}$; arc consistency, removes 4 from the current domains of x_2 and x_3: $D_C(x_2) = D_C(x_3) = \{2, 3\}$ while $D_C(x_1) = \{1\}$ and $D_C(x_2) = \{4\}$. x_1 has only two alternative values left : 2 and 3; 4 is not alternative anymore since it does not belong to the closure by arc consistency of the CSP $< \mathcal{X} = \{x_1, x_2, x_3\}, \mathcal{D} = D_1 \times D_2 \times D_3 = \{1, 2, 3, 4\}^3, \mathcal{C} = \{Alldiff(x_1, x_2, x_3) \cup \{x_2 = 4\}\} >$. x_2 has also two alternative values, 2 and 3 (see Table 1).

	1	2	3	4
x_1	$\star$	$\diamond$	$\diamond$	$\times$
x_2	$\times$	$\diamond$	$\diamond$	$\star$
x_3	$\times$	$\diamond$	$\diamond$	$\times$

Table 1. Assigned ($\star$), forbidden ($\times$) and alternative ($\diamond$) values for the *A*-CSP $\mathcal{X} = \{x_1, x_2, x_3\}$, $\mathcal{D} = D_1 \times D_2 \times D_3 = \{1, 2, 3, 4\}^3$, $\mathcal{C} = \{Alldiff(x_1, x_2, x_3)\}$, $\mathcal{H} = \{(x_1 \leftarrow 1), (x_2 \leftarrow 4)\}$).

The notion of alternative domain is orthogonal to the notion of removal's explanation, such as proposed in PaLM [16]: explanations are a way to explain the pruning of the domains and aim at proposing a strategy of restoration of some value for an unassigned variable by the relaxation of a (minimal) subset of user's choices. On the contrary, the alternative domain of a variable provides a way to change the value of an assigned variable *without* any modification of the other user choices.

[2] Actually, the definitions and results could be set in a more general framework and capture any type of restriction; the meaning of alternative value when the restrictions in $\mathcal{H}$ are not unary is nevertheless questionable, hence our assumption.

The notion of alternative domain can be compared to the concept of fault tolerant solution [21]. A fault tolerant solution is actually a solution such as all the variables have a non-empty alternative domain: if one of the current value in the assignment is made unavailable for any reason, a solution can still be found by choosing a value from its alternative domain - this value is by definition compatible with the other choices. The notion has been generalized by Hebrard et al. [11] under the name "super-solutions". The main difference between the notion of fault tolerant solutions and the notion of alternative domains is that fault tolerant solutions deal with *complete* assignments while alternative domains suggests restoration values for *partial* assignments also. It should also be noticed that the two notions target different practical goals: when refereing to a super-solution, the one in looking for *some*, but not *all*, robust (and complete) solutions - there is indeed a potentially exponential number of fault tolerant solutions. When computing alternative domains, we are looking for *all* the alternative values, and this even during the search, when the assignments are *partial*.

3 Computing alternative domains

When n variables are assigned, a naive way of computing the alternative domains of these variables is to make $n+1$ copies of the CSP: a reference CSP P_0 (where all the n variables are assigned), and n CSP P_i where each P_i has exactly the same assignments than P_0, with the exception of the assignment of variable x_i. Each P_i is filtered by l-consistency. The alternative domain of variable x_i is obviously the domain of x_i in the arc consistent closure of P_i. This method does not require much space but does a lot of redundant computations. It will be the reference point from our method, which follows the opposite philosophy: memorizing information in order to avoid a duplicate work.

3.1 Removals and sufficient justifications

The main idea of our approach is to maintain, for each value removed by the filtering algorithm, a vector of boolean flags, one flag for each $h_i \in \mathcal{H}$. The flag on h_i must be true if and only if the single relaxation of the user's choice h_i will lead to have the value back in the domain of its variable. Let us formalize:

Definition 4 (Removal, invalid tuple)
Let $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H})$ be an A-CSP and P^l the closure of $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H})$ by some level of local consistency l.

A removal w.r.t. a level l of local consistency is a pair (x_j, v), $x_j \in \mathcal{X}, v \in D(x_j)$ such that v does not belong to the domain of x_j in P^l
We write $\mathcal{R}^l$ the set of removals of P w.r.t. l.

Let C a constraint in $\mathcal{C}$ and t an assignment of $vars(C)$ satisfying C. t is said to be invalid w.r.t. l iff there exists $x_j \in vars(C)$ such that $t[x_j]$ does not belong to domain of x_j in P^l; otherwise, it is said to be valid w.r.t. l.

To improve readability, a removal (x_j, v) will often be written $(x_j \neq v)$, and we will omit to mention level l to which the removal refers when not ambiguous.

Definition 5 (Sufficient Justification of a removal)
Let $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H})$ be an A-CSP, l a level of local consistency, and $\mathcal{R}^l$ the set of P's removal according to l.

An user choice $h_i \in \mathcal{H}$ is said to be an l-sufficient justification of a removal $(x_j \neq v) \in \mathcal{R}^l$ if and only if v belongs to the domain of x_j in the l-consistent closure of $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H} \setminus \{h_i\})$.

By extension, for any x_j in $\mathcal{X}$ and any v in $D(x_j)$, $h_i \in \mathcal{H}$ is said to be an l-sufficient justification of v for x_j if and only if v belongs to the domain of x_j in the l-consistent closure of $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H} \setminus \{h_i\})$.

For instance, if the propagation of the last assignment leads to the removal of the value v in the domain of x, this assignment is a sufficient justification of $x \neq v$. By extension, any h_i is a sufficient justification of a value that does belongs to the current domain of its variable.

Example 1 (cont') If we go back to example 1, once x_1 and x_2 are assigned, $\mathcal{H}$ contains two assumptions: $h_1 = (x_1 = 1)$, and $h_2 = (x_2 = 4)$.
All the values deleted from the domain of x_1 (resp. x_2), have h_1 (resp. h_2) as a (sole) sufficient justification.
Arc consistency has removed values 1 and 4 from the domains of x_2 and x_3. h_1 is a sufficent justification for the removals $(x_2 \neq 1)$ and $(x_3 \neq 1)$, and h_2 a sufficient justification of $(x_2 \neq 4)$ and $(x_3 \neq 4)$.
By convention, all the values that are still in the current domains of their variables receive both h_1 and $h2$ as a sufficient justifications.

Example 2 A removal may have several sufficient justifications, as shown by the following example. Consider the CSP $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$, $\mathcal{D} = D_1 \times D_2 \times D_3 \times D_4 = \{1, 2, 3\}^4$, $\mathcal{C} = \{x_1 \neq x_2, x_3 \neq x_2, x_4 \neq x_2)\}$. Value 2 for x_1 has two supports on x_2 : 1 and 3. Suppose that the user has assigned value 1 to x_3 (h_3) and value 3 to x_4 (h_4); in other terms, $\mathcal{H} = \{(x_3 = 1), (x_4 = 3)\}$. h_3 forbidds the first support of $x_1 = 2$ and h_4 forbids its second support ; value 2 is thus removed by arc consistency from the current domain of x_1: $D_C(x_1) = \{1, 3\}$ and this removal has two sufficient justifications: h_3 and h_4.

Of course, a value belongs v to the alternative domain of an assigned variable x_i iff h_i is a sufficient justification of the removal $(x_i \neq v)$:

Proposition 6 *Let $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H})$ be an A-CSP, l a level of local consistency.*

For any $x_i \in \mathcal{X}$, any $v \in D(x_i)$, v belongs to the alternative domain of x_i iff either v belongs to the domain of x_i in the closure by l consistency of $P = (\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H})$ or $(x_i \neq v) \in \mathcal{R}^l$ and h_i is a sufficient justification of $(x_i \neq v)$.

The notion of sufficient justification is extended to tuples as follows:

		supports of $(x = v)$			
removals	justification vector of each removal	t_1	t_2	t_3	t_4
$x_1 \neq v_1$	$\{h_1, h_2\}$	$\star$	$\star$		
$x_2 \neq v_2$	$\{h_1, h_3\}$	$\star$			$\star$
$x_3 \neq v_3$	$\{h_2, h_4\}$		$\star$	$\star$	$\star$
justification vector	$\{h_1, h_2, h_4\}$	$\{h_1\}$	$\{h_2\}$	$\{h_2, h_4\}$	$\emptyset$

Table 2. Computation of the vector of justifications of the removal $(x \neq v)$ on a given constraint C; the t_i are the supports of $x = v$. A $\star$ in cell $(t_i, x_j \neq v_j)$ means that t_i invalid when $x_j \neq v_j$

Definition 7 (Sufficient justification of a tuple)
Let $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H})$ be an A-CSP, l a level of local consistency, C a constraint in $\mathcal{C}$ and t an assignment of $vars(C)$ satisfying C.

An user choice $h_i \in \mathcal{H}$ is said to be an l- sufficient justification for t if and only if, for each $x_j \in vars(t)$, $t[x_j]$ belongs to the domain of x_j in the closure by l consistency of the CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H} \setminus \{h_i\})$.

Example 1 (cont') If we go back to example 1, once x_1 and x_2 have been assigned, tupple $(3, 2, 4)$ is not valid anymore and has one sufficient justification, h_1 (it is enough to relax $x_1 = 1$ to make this tupple valid again); remark that tupple $(4, 2, 1)$, that is also invalid, has no sufficient justification (the relaxation of the two choices is necessary to make it valid again).

Our algorithm is based on the fact that an assignment h_i is an l-sufficient justification for the tuple t if and only if, for each x_j involved by the tuple, either $t[x_j]$ is in the current domain of x_j or h_i is a sufficient justification of the removal $(x_j \neq t[x_j])$. Formally, let us call the conflict set of t the set of removals that make it invalid:

Definition 8 (Conflict set)
The conflict set of a tuple t w.r.t. some level of l consistency is the subset of $\mathcal{R}^l$ defined by: $CS(t) = \{(x_i \neq v) \in \mathcal{R}^l \text{ s. t. } t[x_i] = v\}$.

Of course, a tuple is invalid if and only if it has a non-empty conflict set.

Proposition 9 *h_i is an l-sufficient justification of a tuple t if and only it is an l-sufficient justification of each of the removals in its conflict set w.r.t. l.*

Finally, it can easily be shown that, when the level local consistency to maintain is generalized arc consistency:

Proposition 10 *h_i is a sufficient justification w.r.t. Arc consistency (GAC) for a removal $(x \neq v)$ iff, for each constraint C bearing on x, there exists a tuple t support of $(x = v)$ on C such that h_i is GAC-sufficient justification of t.*

Similar properties can be established for other levels of local consistency based on the notion of support, typically for k inverse consistency [8][3]

[3] A CSP is $(1, k)$ consistent iff, for each variable x and each value v in $D(x)$, for each set $\mathcal{V}$ of k additional variables, $x = v$ has a support on $\mathcal{V}$, i.e. there exists an assignment t of $\{x\} \cup \mathcal{V}$ such that for any $C \in \mathcal{C}$ with $vars(C) \subseteq \{x\} \cup \mathcal{V}$, t satisfies C

Proposition 11

$h_i \in \mathcal{H}$ is a $(1, k)$-sufficient justification of $(x \neq v) \in \mathcal{R}^{(1,k)}$ iff, for each set $\mathcal{V}$ of k variables there exists a support t of $x = v$ on $\mathcal{V}$ such as h_i is a $(1, k)$-sufficient justification of t.

3.2 An algorithm of maintenance of the alternative domains w.r.t. Arc Consistency

In our application, interactive configuration, the constraint to be taken into account are mostly table constraints and the level of consistency referred to is Generalized Arc Consistency. We thus propose to maintain the alternative domain upon the assignment of a variable using an extension of GAC4 [15]. Our algorithm propagates not only value removals, but also justifications: for each removal $(x_i \neq v)$, we maintain a vector $f_{(x_i \neq v)}$ of n boolean flags, one for each choice in $\mathcal{H}$, such that $f_{(x \neq v)}(h_i) =$ True if and only if h_i is a sufficient justification of $(x_i \neq v)$. According to Proposition 10, $f_{(x \neq v)}$ depends on the justifications of the tuples that support (x, v). Hence, we keep, for each tuple t, a bit vector f_t such as, for each h_i, $f_t[h_i]$ is true iff h_i is a sufficient justification of t. Intuitively (see Table 2 for an example), for the user choice h_i to be a sufficient justification for a removal $(x \neq v)$ provoked by constraint C, it is needed that the relaxation of h_i makes at least one support t of $(x = v)$ on C valid again, i.e. that all the elements in the conflict set of t have h_i as a sufficient justification (this is the meaning of Proposition 9). In other words, f_t is the intersection of the $f_{(x_j \neq w)}$ flags of all the removals $(x_j \neq w)$ in the conflict set of t. Formally:

Proposition 12

$$f_{(x \neq v)} = \bigwedge_{C | x \in vars(C)} \Big(\bigvee_{t \in Support(x,v,C)} \big(\bigwedge_{r \in CS(t)} f_r \big)\Big)$$

where $Support(x, v, C)$ is the set of assignments of $vars(C)$ that support (x, v).

We propose here a GAC4 like algorithm, the initialization and main propagation of which are depicted by algorithms 1 and 2. We use the following notations:

- $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is the original CSP, that is supposed arc consistent;
- for any constraint $C \in \mathcal{C}$, $Table(c)$ is the set of assignments of $vars(c)$ that satisfy it. We moreover the tuples involved in the tables are valid (i.e. $Table(c)$ is a subset of the cartesian product of the domains of the variables its bears on.

- $D_c(x_i)$ is the current domain of x_i
- $S_{x_i,v,C}$ is the set of supports of (x_i, v) on C and $Cpt(x_i, v, C)$ is the number of supports of (x_i, v) on C.
- for any tuple t, f_t is its vector of justifications; for any removal $(x_i \neq v)$ $f_{(x_i \neq v)}$ is its vector of justifications; for any removal $(x_i \neq v)$ and any constraint C bearing on x_i, $f_{(x_i \neq v, C)}$ is the vector of justification of $(x_i \neq v)$ on C.

The difference with GAC4 is that a removal $(x \neq v)$ must be propagated non only when it is created, but for each change in its vector of justifications. Since the updating of the vectors of justification is monotonic (a h_i might go from being sufficient to not, but not the other way around), the algorithm terminates. More precisely, instead of entering just once in Q, each removal can enter in the queue n times at most (n being the number of h_i in $\mathcal{H}$), i.e.as much as the number of possible changes in a vector of justifications. The worst case complexity is thus bounded by $O(ned^k)$ with e the number of constraints, m the number of variables, n the maximal number of assumptions (typically, $n = m$), d the maximum size of the domains and k the maximum arity of constraints. It is thus the same complexity as the GAC-4 based naive method: $n.O(e.d^k)$. With the important difference that in the naive method, GAC-4 is called exactly n times while n is a worst case bound for justification-based algorithm.

Concerning space complexity, GAC4 memorizes the support $S_{i,v,C}$ for each x_i , each value v in its domain and each constraint C bearing on x_i; Let say that this structure is in $O(T)$ (T is actually proportional to the space taken by valid tuples in constraint tables). Our algorithm also maintains, for each tuple t, a vector of n flags, meaning a $O(T.n)$ space. For each removal and each constraint bearing on the variable of the removal, we also keep a vector of n boolean flags. Since the number of removals is bounded by the number of variable/value pairs (x_i, v) in the problem, the algorithm involves in the worst case as many boolean vectors as the number of $S_{i,v,C}$ sets used by GAC4; Hence a global a spatial consumption bounded by $O(n.T)$.

```
Procedure Initialize(($\mathcal{X}, \mathcal{D}, \mathcal{C}$):CSP; $n$: integer)
/* ($\mathcal{X}, \mathcal{D}, \mathcal{C}$) is the original CSP assumed to be arc consistent */
/* All the tuples are supposed to be valid */
/* $n$ is the maximal number of assumptions to be considered */
 begin
   foreach $C \in \mathcal{C}$ do
      foreach $x_i \in vars(C)$, $v \in D(x_i)$ do
         $Cpt(x_i, v, C) := 0$;
         $S_{i,v,C} = \emptyset$
      end
      foreach $t \in Table(C)$ do
         $f_t = \text{True}^n$;
         $valid(t) = \text{True}$;
         $CS(t) = \text{False}^n$;
         foreach $x_i \in vars(C)$ do
            $Cpt(x_i, t[x_i], C) + +$;
            Add $t$ to $S_{i,t[x_i],C}$
         end
      end
   end
end
```

Algorithm 1: Initialization

```
Procedure Propagate( $(x_k, w)$: assumption; $(\mathcal{X}, \mathcal{D}, \mathcal{C})$: the
initial CSP; $\mathcal{H}$: the past assumptions; $D_c$: the current
domains);
Add $(x_k, w)$ to $\mathcal{H}$;
$Q := \emptyset$;
/* The removal of the other values in the current domain of $x_k$ is
due to $h_k$ */
foreach $u \neq w \in D_c(x_k)$ do
   $f_{(x_k \neq u)} \leftarrow \text{False}^m$;
   $f_{(x_k \neq u)}[h_k] \leftarrow \text{True}$;
end
Add $(x_k \neq u)$ to $Q$;
while $Q \neq \emptyset$ do
   Choose and remove a $(x_i \neq v)$ from $Q$;
   if $v \in D_c(x_i)$ then
      Remove $v$ from $D_c(x_i)$;
   end
   foreach $C$ s.t. $x_i \in vars(C)$ and each tuple $t$ in
   $S_{i,v,C}$ do
      $Mem \leftarrow f_t$;
      $f_t \leftarrow f_t \wedge f_{(x_i \neq v)}$;
      if valid(t) then
         foreach $x_j \in vars(t)$ s.t. $j \neq i$ do
            $Cpt(x_j, t[x_j], C)^{--}$;
            if $Cpt(x_j, t[x_j], C) == 0$ then
               $f_{(x_j \neq t[x_j]),C} \leftarrow \text{False}^m$ /* init; will be
               computed later */ ;
               Add $(x_j \neq t[x_j])$ to $Q$;
               if $t[x_j] \in D_c(x_j)$ then
                  $f_{(x_j \neq t[x_j])} \leftarrow \text{True}^m$ /* init */ ;
               end
            end
         end
         $valid(t) = false$;
      end
      if $Mem! = f_t$ /* A justif. of t is not sufficient
      anymore */ then
         foreach $x_j \in vars(t)$ s.t. $j \neq i$ do
            $mem' = f_{(x_j \neq t[x_j])}$;
            $f_{(x_j \neq t[x_j]),C} = f_{(x_j \neq t[x_j]),C} \vee f_t$;
            $f_{x_j \neq t[x_j]} = f_{x_j \neq t[x_j]} \wedge f_{x_j \neq t[x_j],C}$;
            if $mem' \neq f_{(x_j \neq t[x_j])}$ then
               Add $(x_j \neq t[x_j])$ to $Q$;
            end
         end
      end
   end
end
foreach $h_i \in \mathcal{H}$ do
   $D^{alt}(x_i) = \emptyset$; foreach $v \in D_{x_i}$ do
      if $f_(x_i \neq v)[h_i]$ then
         Add $v$ to $D^{alt}(x_i)$
      end
   end
end
```

Algorithm 2: Propagation of decision $h_k = (x_k \leftarrow w)$

4 First experimental results

We have tested this algorithm on an industrial problem of configuration. It involves 32 variables of domain of size 2 to 10, 35 binary constraints. The product to configure is a blowing machine, which blows bottles for different matters. The benchark can be found at url `ftp://ftp.irit.fr/pub/IRIT/ADRIA/PapersFargier/Config/souffleuse.xml`.

The protocol simulates 1000 sessions of configurations as follows. First, a sample of 1000 consistent complete assignments is randomly fired. For each of then, the corresponding session is simulated by assigning the variables following a random (uniform) order. After each assignment, we measure the cpu time needed to make the current problem arc-consistent and to compute the alternative domains of all the already assigned variables are computed. The whole protocol is applied by both the justification-based algorithm described in the previous Section and the naive method (that works on as may copies of the original CSP as the number of user choices in $\mathcal{H}$, as decribed in introduction of Section 3) ; for the shake of rigor, the two algorithms play on the same assignments and the same assignment orders.

Figure 1 presents the result of these experiments. On the x-axis is the number of the assignment in the sequence; the y-axis is logarithmic and indicates the mean cpu time need for the naive method (plain line, with rounds) and for the justification-based algorithm dotted line, with squares.

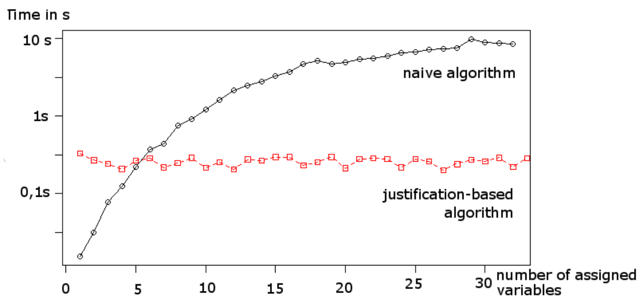


Figure 1. Computation time required by both the naive method and the justification-based algorithm (logarithmic scale).

The results are quite good: our algorithm is faster as soon as more than 5 variables are assigned, i.e.when more than 5 alternative domains are to be computed. As expected, the time required by the naive algorithm grows linearly with the number of variables, while our algorithm has stable computation time. These first results have obviously to be confirmed by more experiments on bigger configuration problems.

5 Conclusion

In this work, we have coined the new concept of the alternative domain of a variable with respect to a local consistency level and proposed an extension of GAC4 algorithm as a way to compute the alternative domains when maintaining General Arc Consistency on problems involving table constraints.

Contrarily to the naive method that applies the propagation algorithm as many times as the number of alternative domains to be computed, our approach keeps limited justifications of the removals. Tested on two industrial benchmarks, this method quickly outperforms the naive method.

The main limitation of our method is obviously its space consumption; the extra space consumption depends directly of the number of variables for which we want to compute the alternative domain. This being said, it should be kept in mind that for practical purposes the system is not asked to display all the alternative domains; the human user has with a limited mental capacity and it is not obvious that she can or even wants to see a lot of alternative domains at a glance. In a configuration application for instance, the user looks at only a small number of variable simultaneously, typically the ones involved in the subcomponent currently being configured.

The concepts we have coined are close to the notion of value restoration. In the current work, we focused on the computation of alternative domains; an alternative value is a forbidden value that can be restored by the sole relaxation of the assignment of its variable. But more generally any value having at least one sufficient justification can be restored by the relaxation of only one assignment. For each value in the domain of an assigned variable, the user knows whether she can change her choice to this value without modifying the other choices - this is the notion alternative domain. But the user also knows something about the values that have been filtered from the domains of the *unassigned* variables: the justification vector of such a value provides her with the set of , previous choices (on *other* variables) she could relax in order to make the value available again. Hence the potential use of the algorithm proposed by this paper to provide the user with alternative values in a wider sense, and more generally to support the task of interactive relaxation by providing easily restorable values.

This work has a huge potential for developments and perspectives. Firstly, our algorithm obviously needs to be improved, for instance with a lazy implementation, and our experiments must be completed. Secondly, we should think about the extension of the maintenance of alternative domain in CSP with general constraints, and not just in table constraints ; such an algorithm is not too difficult to conceive for CSPs involving binary constraints only, but the task seems much more tricky for general constraints. Finally, we should be able to consider the whole interaction; for the moment, we only considered the assignment of a value to a variable: we need to study the relaxation of choices also. This adaptation might mean an hybridizing with the maintenance algorithms of propagation/depropagation in dynamic CSP[2, 3, 6].

ACKNOWLEDGEMENTS

This work is partially funded by the ANR project BR4CP (ANR-11-BS02-008)

REFERENCES

[1] Jérôme Amilhastre, Hélène Fargier, and Pierre Marquis, 'Consistency restoration and explanations in dynamic csps application to configuration', *Artificial Intelligence*, **135**(1-2), 199–234, (2002).

[2] Christian Bessière, 'Arc-consistency for non-binary dynamic csps', in *Proceedings of ECAI'92*, pp. 23–27, (1992).

[3] Romuald Debruyne, 'Arc-consistency in dynamic csps is no more prohibitive', in *Proceedings of ICTAI'96*, pp. 299–307, (1996).

[4] Romuald Debruyne, ‘A property of path inverse consistency leading to an optimal pic algorithm’, in *Proceedings of ECAI’2000*, pp. 88–92, (2000).

[5] Romuald Debruyne and Christian Bessière, ‘Some practicable filtering techniques for the constraint satisfaction problem’, in *Proceedings of IJCAI’97*, pp. 412–417, (1997).

[6] Romuald Debruyne, Gérard Ferrand, Narendra Jussien, Willy Lesaint, Samir Ouis, and Alexandre Tessier, ‘Correctness of constraint retraction algorithms’, in *Proceedings of FLAIRS’03*, pp. 172–176, (2003).

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

[8] Eugene C. Freuder, ‘A sufficient condition for backtrack-bounded search’, *Journal of the ACM*, **32**(4), 755–761, (1985).

[9] Eugene C. Freuder and Charles D. Elfe, ‘Neighborhood inverse consistency preprocessing’, in *Proceedings of AAAI’96*, pp. 202–208, (1996).

[10] Ester Gelle and Rainer Weigel, ‘Interactive configuration using constraint satisfaction techniques’, in *Proceedings of PACT-96*, pp. 37–44, (1996).

[11] Emmanuel Hebrard, Brahim Hnich, and Toby Walsh, ‘Super solutions in constraint programming’, in *Proceedings of CPAIO’04*, pp. 157–172, (2004).

[12] Daniel Mailharro, ‘A classification and constraint-based framework for configuration’, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **12**, 383–397, (September 1998).

[13] Sanjay Mittal and B. Falkenhainer, ‘Dynamic constraint satisfaction problems’, in *Proceedings of AAAI’90*, pp. 25–32, (1990).

[14] Sanjay Mittal and Felix Frayman, ‘Towards a generic model of configuraton tasks’, in *Proceedings of the IJCAI’89*, pp. 1395–1401, (1989).

[15] Roger Mohr and Gérald Masini, ‘Good old discrete relaxation’, in *Proceedings of the ECAI’88*, pp. 651–656, (1988).

[16] Samir Ouis, Narendra Jussien, and Olivier Lhomme, ‘Explications conviviales pour la programmation par contraintes’, in *Actes de JFPLC*, pp. 105–118, (2002).

[17] Daniel Sabin and Eugene C. Freuder, ‘Configuration as composite constraint satisfaction’, in *AI and Manufacturing Research Planning Workshop*, pp. 153–161, (1996).

[18] Daniel Sabin and Rainer Weigel, ‘Product configuration frameworks — a survey’, *IEEE Intelligent Systems*, **13**(4), 42–49, (1998).

[19] Markus Stumptner, Gerhard E. Friedrich, and Alois Haselböck, ‘Generative constraint-based configuration of large technical systems’, *AI EDAM*, **12**(04), 307–320, (1998).

[20] Junker Ulrich, ‘Configuration’, in *Handbook of Constraint Programming*, 837–874, Elsevier Science, (2006).

[21] Rainer Weigel and Christian Bliek, ‘On reformulation of constraint satisfaction problems’, in *Proceedings of ECAI’98*, pp. 254–258, (1998).

A Proofs

[Proof of Proposition 9]
Of course, the proposition holds when t is valid (it has an empty conflict set). Let us examine the case of an invalid tuple.

$\boxed{\Rightarrow}$ Let h_i be l-sufficient justification of an invalid tuple t and suppose that there exists a removal $(x \neq v)$ in the conflict set of t such that h_i is not a sufficient justification of $(x \neq v)$.
We write P_i^l the l-consistent closure of $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H} \setminus \{h_i\})$. Since h_i is an l-sufficient justification of t, by definition, t is valid in P_i^l. Since h_i is also not a sufficient justification of $(x \neq v)$, v is not in the domain of x in P_i^l; t is thus invalid in P_i^l, which is a contradiction.

$\boxed{\Leftarrow}$ Reciprocally, let h_i be an l-sufficient justification of all the removals in the conflict set of t. For each of these $(x_j \neq v_j)$, v_j is by definition in the domain of x_j in P_i^l. Thus, t is a valid tuple in P_i^l - by definition of the notion of justification, h_i is thus an l-sufficient justification of t. □

[Proof of Proposition 10]
$\boxed{\Rightarrow}$ Let h_i be a GAC sufficient justification of a removal $(x \neq v)$. Suppose that there exits a constraint C bearing on x such that none of the supports of $x = v$ on C admits h_i as a sufficient justification. This means that these tuples are not valid in the arc consistent closure of $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H} \setminus \{h_i\})$ (denoted P_i^{GAC}). Thus v has no support on C in P_i^{GAC}: it does not belongs to the domain of x in P_i^{GAC}; h_i is thus not a sufficient justification of $(x \neq v)$.

$\boxed{\Leftarrow}$ Reciprocally, consider an assumption h_i and suppose that $\forall C$ bearing x, $\exists t$ support of $x = v$ such that h_i is a GAC sufficient justification of t . This means that, for any constraint bearing on x there exists a support t of $x = v$ valid in P_i^{GAC}; v thus belongs to the domain of x in P_i^{GAC} - by definition, this meant that h_i is a GAC-sufficient justification of $(x \neq v)$. □

[Proof of Proposition 11]
$\forall(x_1, ..., x_k), \exists(v_1, ..., v_k)$ a support of $x = v$ such that h_i is a $(1, k)$-sufficient justification of $(v_1, ..., v_k)$

$\Leftrightarrow \forall(x_1, ..., x_k), \exists(v_1, ..., v_k)$ support of $x = v$ such that any of the v_j belongs to the domain of its variable in the closure by $(1, k)$-consistency of $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H} \setminus \{h_i\})$

$\Leftrightarrow$ v belongs to the domain of x the closure by $(1, k)$-consistency of $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H} \setminus \{h_i\})$ (definition of the $1, k$ consistency)
$\Leftrightarrow$ h_i is a justification $(1, k)$-sufficient of $x \neq v$. □

[Proof of Proposition 12]
According to Proposition 10, when GAC is ensured

$$f_{(x \neq v)}[h_i] = \bigwedge_{C, x \in vars(C)} \bigvee_{t \in Support(x,v,C)} f_t[h_i]$$

For any h_i, any $x \in \mathcal{X}$ and any $v \in D_x$. I.e.:

$$f_{(x \neq v)} = \bigwedge_{C, x = vars(C)} \bigvee_{t \in Support(x,v,C)} f_t$$

Yet, according to Proposition 9, the GAC-sufficient justifications set of a tuple is the intersection of GAC-sufficient justifications of the removals from its conflict set: $f_t = \bigwedge_{r \in CS(t)} f_r$. Hence the result. □

K-Model – Structured Design of Configuration Models

Dr. Axel Brinkop[1] and **Dr. Thorsten Krebs**[2] and **Hartmut Schlee**[3]

Abstract. The purpose of this paper is to introduce the novel knowledge acquisition methodology K-Model. We describe the methodology itself and how it was applied within a project for creating a prototype configuration application at J. Schmalz GmbH. K-Model is supporting both the formalism of designing configuration models on a conceptual level as well as the method to actually implement these models. Based on the experience that configuration knowledge is tacit and distributed within the heads of several product experts', the methodology is focusing on cross-department communication about future goals of the configuration application. The visualization facilities of standard mind maps help them to achieve a common agreement and to focus on the product domain rather than on knowledge representation formalisms. The methodology was successfully used in the project to set up a configuration prototype for complex products in the area of vacuum technology.

1 MOTIVATION

A major challenge in realizing knowledge-based configuration systems is the acquisition and formalization of configuration knowledge. But knowledge acquisition is notoriously a very expensive process. Actually, most of the complexity of solving a configuration problem is said to lie in representing the domain knowledge [2].

One of the main reasons for the complexity of knowledge acquisition is that two types of expertise are required: knowledge about the product domain and dealing with the representation language that is used for modeling the product domain. But very few persons are both domain expert and knowledge modeling expert. Thus, in practice the modeling task is carried out by one of the two engineers, probably being assisted by the other one.

In this paper we introduce the knowledge acquisition methodology K-Model. This methodology helps the knowledge engineer to focus on the product domain rather than on knowledge representation formalisms. K-Model consists of formalism for designing the contents of a configuration model and a method for acquiring configuration knowledge and actually creating the contents. The formalism describes the types of knowledge required for creating a configuration system in a way that is well-founded on semantics but at the same time easily understandable for domain experts like product managers or sales engineers. We use mind map structures to visualize the relevant types of content, i.e. classification data, sales questions and the sales bill of materials together with their interdependencies. We further use MS Excel to define data about available components according to the definition of classification data as well as tabular dependencies, i.e. variant tables. The method describes a process consisting of workshops, reviews and "offline" refinement steps in which the relevant configuration knowledge for the product domain is acquired and actually implemented within a configuration model.

J. Schmalz GmbH is a family-run company situated in Glatten, Germany. Schmalz is a leading global supplier of vacuum technology in the fields of automation, handling and clamping technology with an export quota of 50%, 15 subsidiaries abroad, and sales partners within 40 countries all over the world. When it comes to automated production processes, Schmalz offers a wide range of individual vacuum components and related services. Different vacuum systems can be operated in different environments, e.g. vacuum gripper systems are ready-to-connect modular systems for usage in robotic applications, vacuum handling systems are operated manually and ease the handling of work pieces and vacuum clamping systems offer short set-up times for CNC machining centers.

Schmalz is a very innovative company with permanent readiness to implement and accept changes. A current change of the company is driven by investing in a quote generation process including configuration of vacuum products. The goal of this change is to ease generating technically correct solutions for complex configuration problems together with high quality quote documents. The K-Model methodology was used to set up a prototype quote generation application for complex configurable products from the families of vacuum handling systems and vacuum clamping systems.

The remainder of this paper is organized as follows. In Chapter 2 we describe the knowledge acquisition methodology K-Model, i.e. both the formalism and the method, in more detail and give mind map representation examples. Chapter 3 describes the application of K-Model within a real-life customer project, i.e. both applying the formalism and method of K-Model for acquiring a configuration model as well as implementing the acquired contents. Chapter 4 concludes this paper with the major findings and in Chapter 5 we present related work.

[1] Brinkop Consulting, Oberschlettenbach, Germany, email: Brinkop@brinkop-consulting.com
[2] encoway GmbH, Bremen, Germany, email: Krebs@encoway.de
[3] J. Schmalz GmbH, Glatten, Germany, email: Hartmut.Schlee@schmalz.de

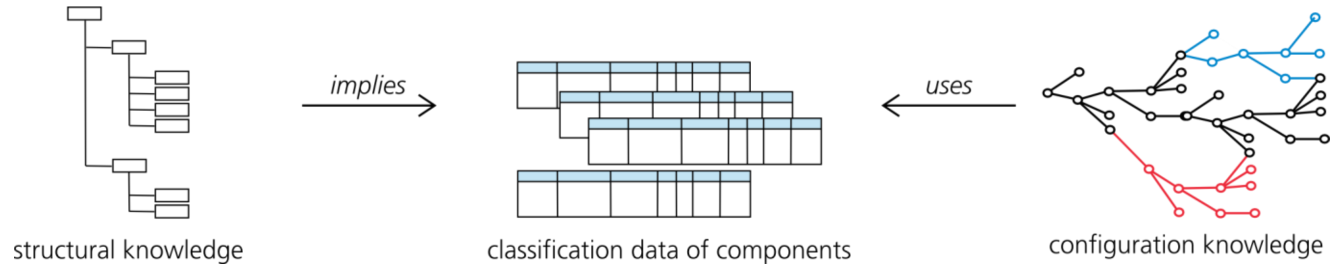


Figure 1: The relations between structural knowledge, classification data and configuration knowledge.

2 K-MODEL

K-Model is a methodology developed by Brinkop Consulting supporting both the formalism of designing configuration models as well as the method to actually develop these models ("K" = "Konfiguration", German for configuration). It is not designed for any specific software but it is based on the approach to separately represent structural knowledge, configuration knowledge and available components.

The structural knowledge is a conceptual-level representation of the internal structure of the product to be configured; i.e. the product itself together with the parts from which it is assembled. The options for each part are defined in the available components themselves. Structural knowledge and available components are strongly related, though. The structural knowledge is expressed as a hierarchy of classes, each class defined by a set of attributes. Inheritance of attributes is assumed. Every available component is an instance of a class with given attribute values. The configuration knowledge represents knowledge about dependencies and methods to determine components. Figure 1 illustrates the corresponding relations. The result of the configuration process is a sales bill of material consisting of well-defined instances from these classes. In short, the structural knowledge defines the classes; the available components are defining the instances.

K-Model assumes that the configuration model and the underlying configuration engine are separated. There is no need (and no possibility) to express specific solution strategies. It is assumed that the configuration engine can interpret the dependencies specified. No specific configuration software is targeted; several commercial configuration engines can handle configuration problems designed with the K-Model methodology.

K-Model is evolved by Brinkop Consulting in a multitude of projects. It was learned that configuration knowledge is distributed on several persons, each focusing on a different perspective of the configuration task. The challenge is not to acquire the configuration knowledge but to achieve a shared commitment of the way how to solve the configuration task at hand. Therefore K-Model concentrates on cross-department communication. The methodology addresses product experts with no specific IT skills. The formalism allows informal descriptions of configuration details as well as formal specifications. The description of the configuration model is based on a mind map with special keywords and structure. The tool of choice is Freeplane[4], which is open source and easy to use.

Experience shows that methodology is very well suited for workshops from several departments such as product management, research & development, and sales. By applying the methodology to a known domain, the participants are learning the formalism very easily. In early phases the discussion is focusing on domain specific configuration problems. There is no need for deep IT background; the content of the mind map is understood by anybody easily. It is a good basis to discuss alternative ways for solving the configuration problem and to achieve a shared commitment.

2.1 The Formalism

The formalism distinguishes between the items (i.e. classification data of available components), the questions (i.e. sales-relevant configuration questions) and the resulting sales bill of material (i.e. proposal items).

2.1.1 *The Items*

The tag ITEMS introduces the class hierarchy of available components. Below the tag ATTRIBUTES introducing the classifying attributes with their data type, possible values and translations and are listed. The optional tag SUBTYPES marks the classes of the next hierarchy level. The attributes are inherited along the hierarchy, i.e. all attributes of higher levels are known as well. Structural sub-components might be defined using the tag HAS-PARTS.

The data about available components is defined in so-called selection lists in MS Excel format. The structure of the selection lists must be consistent with the structure defined herein.

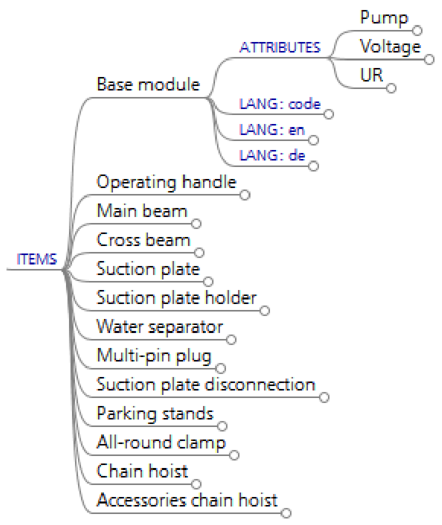


Figure 2: The ITEMS.

[4] http://freeplane.sourceforge.net/

2.1.2 *The Catalog*

The catalog is the starting point for the user in the quote generation process. A user can do both, select completely defined (standard) products or configure an individual product that consists of a set of items. Both types of products can be included in a quote. The catalog is structured by categories; each category contains either item classes or other categories. An item may be assigned to several categories; i.e. the assignment must not be unique. The user can find such an item on several paths.

The tags DISPLAY and SEARCH are used to define the attributes to be shown or searched respectively.

2.1.3 *The Questions*

Variables are specifying the object to be configured. They are organized in classes below the tag QUESTIONS. In fact, variables are grouped in classes defining the user interaction. For easy handling variables of a class might be organized additionally in topics. This organization results in a three level hierarchy "class-topic-attribute" which can be found again in the formal names of variables. The use of just three levels is a simplification which was not perceived as a restriction in past projects.

K-Model assumes that there is no additional specification for the user interface; the variables are presented to the user "as they are". Input variables are tagged as EDIT, SELECT, CHECKBOX etc. and output variables as OUTPUT or HIDDEN. The organization in classes and topics is assumed to be used for organizing the questions, for instance in tabs.

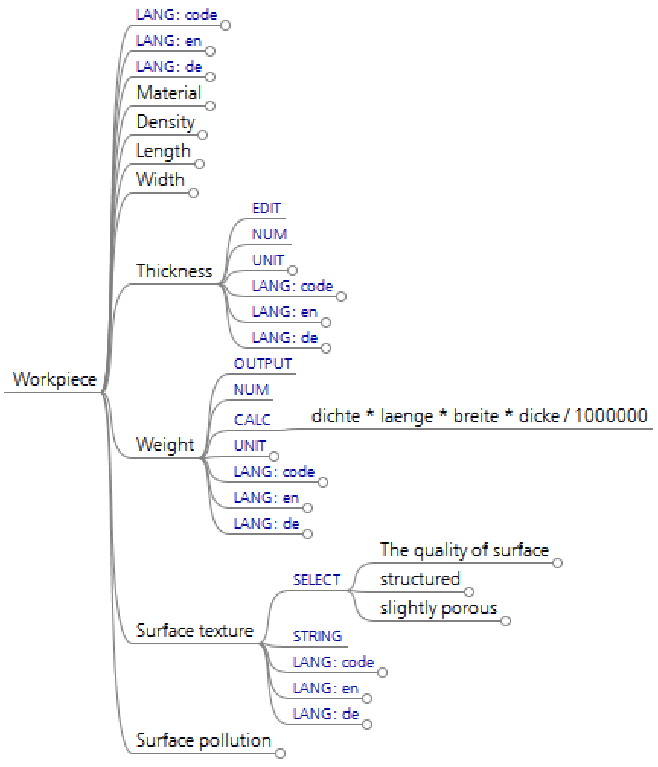


Figure 3: The QUESTIONS.

Tags for language specific translations of the variables are included as well (LANG: en, LANG: de, etc.).

2.1.4 *The Sales Bill of Material*

The result of the configuration process is a sales bill of material tagged as BOM. The bill of material can be structured to any level desired, the "leaves" are instances of the classes below ITEMS. Hereafter the "leaves" are called in short "positions".

Every position is defined by a query. A query to select a position consists of the specification of the class to be searched and conditions to be met by the attributes of the position. It is required that the assignment is unique.

To express relations like "select the drive with the lowest power which is higher as required by x, queries can be specified using the combination of the tags ORDER-BY with FIRST or LAST with the same meaning as in SQL.

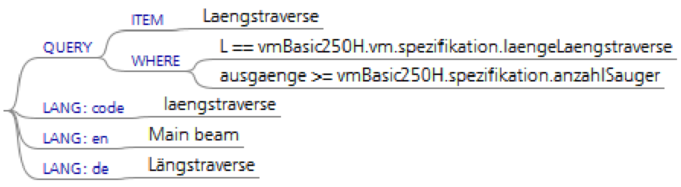

QUERY
ITEM
Laengstraverse
WHERE
L == vmBasic250H.vm.spezifikation.laengeLaengstraverse
ausgaenge >= vmBasic250H.spezifikation.anzahlSauger
LANG: code
laengstraverse
LANG: en
Main beam
LANG: de
Längstraverse

Figure 4: QUERY in the BOM.

2.2 The Method

The method describes the steps that are necessary to set up a configuration model using the formalism presented in the previous section. The following sections each describe a step of the process that are carried out during workshops or reviews, according to figure 5.

2.2.1 *Capture variables*

The model design process starts with a kick-off workshop to define the scope of the model and to get an idea about the configuration problem. In a kind of brainstorming the relevant characteristics are collected and captured in the mind map. These characteristics are called variables in the following. The objective is not to describe the configuration problem formally but to gain the key variables for the problem.
After the first phase the variables are discussed more in detail, e.g. whether a variable is an input or an output. In case of an input, does the value come from a fixed set of values or is a user free to enter any value. In case of an output it is discussed how the value can be computed.

2.2.2 *Organize variables*

Variables describing the same object should be placed as attributes of the same class. K-Model has the concept of a "topic" to organize attributes of a class in another level. This allows easily handling classes with a large number of attributes.
As already stated, it is assumed that the organization of the variables directly influences the user interface. Variables belonging to the same topic are represented to the user at the same time: e.g. classes in tabs and topics as groups of decisions.

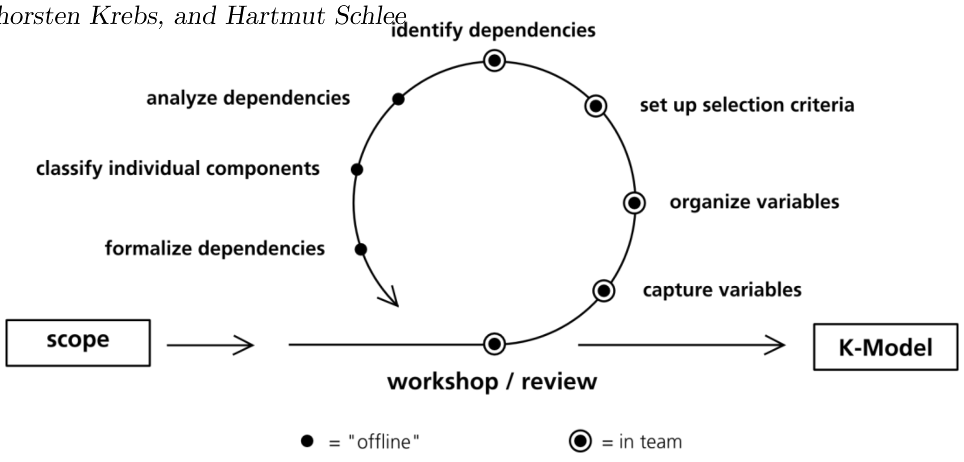


Figure 5: The cycle of workshops and reviews for designing configuration models.

2.2.3 *Set up component selection criteria*

Available components are organized in classes according to the defined ITEMS. Individual components as parts of the configuration solution are selected by a set of specific criteria. These criteria make up the characteristic attributes of the components' class.

For every component class these characteristic attributes have to be listed and their domains specified. Especially for discrete value domains, every possible value has to be specified.

2.2.4 *Identify dependencies*

It is the dependencies between variables that turn a configuration problem into a hard problem. There are several ways in which variables can influence one another. The calculation of the following variables' properties may be based on other variables:

- Value
- Default value
- Existence condition
- Selection of component type (i.e. the type of class)

It is assumed that the configuration engine selected for the implementation exposes default values to the user and does not assign defaults directly to the variable.

2.2.5 *Analyze dependencies*

After the informal definition dependencies are analyzed. As already stated, the variables ("questions") are describing classes of user interaction. Variables which have a strong relationship should be placed in the same class. This reduces complexity for the model as well as complexity of user interaction.

A good tool for analysis is a dependency matrix containing the configuration variables as header for rows and columns. A field (x, y) contains a cross when variable x influences variable y. The distribution of the crosses visualizes the dependencies.

2.2.6 *Classify available components*

Available components are organized in classes with characteristic attributes; every individual component is classified by assigning values to its attributes. Available components are selected by their attribute values; i.e. they represent the providing "function" within the attributes.

In case of automatic selection, the components must have mutually exclusive sets of attribute values. Each query should have exactly one hit. This requirement can be relaxed when there is a scenario of interactive selection by the user. In that case there might be multiple hits of a query, but user must have the possibility to distinguish between the components. Ideally, there is either a text or a picture describing the components.

2.2.7 *Formalize dependencies*

Finally the specifications of the variables and the dependencies are written down formally using formulas, algorithms and query statements. This step is required for ensuring that the model can be implemented with the configuration software.

It is important to keep the informal description as well for documentation purposes and to control the formalization and to keep the ability for an easy discussion.

3 APPLICATION OF K-MODEL IN THE CASE OF SCHMALZ

This section describes how K-Model was applied at J. Schmalz GmbH to acquire the relevant configuration knowledge and for realizing a prototype configuration application.

3.1 Procedure in the Workshops

According to the K-Model method the relevant configuration knowledge for realizing a configuration application was acquired during a kick-off workshop and follow-up review workshops.

The configuration team was set up from product manager, sales manager, product data management and K-Model expert. The knowledge acquisition process is driven by the K-Model expert and supported by all other team members. During all workshops the K-Model expert takes notes visible for every participant using the K-Model mind map.

The kick-off workshop started with specifying the scope of the configuration model. Two distinct product families were chosen for realizing the prototype application in order for being able to assess the results independent from a single product domain. After this the K-Model methodology was applied by capturing variables, organizing variables, setting up components' selection criteria, and identifying dependencies. This work is done in a kind of "brainstorming" style with documenting every statement informally in the K-Model mind map.

After the kick-off workshop the mind map was refined by adding formalized definitions according to the informal notes that were taken within the workshop. The individual components are classified and the specifications and the dependencies are formalized. This is typically done "offline"; the K-Model expert is extending the mind maps and dependencies accordingly and the product managers or other persons at the customer's site define the available components within MS Excel sheets.

The resulting mind map and data in Excel sheets were reviewed in some follow-up review workshops by the same team. Just a few cycles of the design process were required to extend the mind map and MS Excel documents for reaching a level that satisfies all participants. After that the model was released for realization.

The main point of discussion in the workshops at J. Schmalz GmbH was about the targeted user group. Should the product configurator be designed for the product novice with only little knowledge about Schmalz' products and enrich the application with product details or should it rather address the expert and thus focus on few decisions without explicit marketing information? At the end it was decided to assist both of them. The system should guide the novice and should not stop the expert from realizing the configuration he has in mind.

3.2 Realization at encoway

encoway received the mind maps and Excel sheets that are the result of applying the K-Model methodology. The documents contain a formal description of the product structure and dependencies together with the available component for two product families. Our modeling experts were directly able to use this structured information for modeling the products within encoway's modeling environment K-Build.

K-Build is web-based application for formalizing configuration knowledge consisting of structure-based modeling facilities, i.e. concepts together with their attributes arranged in taxonomy and partonomy, as well as constraint definitions. This modeling tool contains a test environment which uses the inference engine engcon for interpreting the configuration knowledge. For detailed information about structure-based configuration and engcon the interested reader is referred to [4] and [3], respectively.

The structure within a K-Model mind map can be mapped directly to concepts representing separate branches within the taxonomy: one each represents the sales questions, the classification data and the sales bill of materials:

- A group of sales questions is mapped to a single concept; the questions themselves are mapped to attributes of that concept.
- The classification data is mapped to concepts and the available components defined in MS Excel sheets are imported into lower levels of the specialization hierarchy; i.e. as specializations of those concepts.
- The sales bill of materials (also called bom) can be structured into groups. Each group is mapped to a concept. The configuration solution consists of instances of the available components which are modeled as parts of the bom group structure.

The dependencies within a K-Model mind map can be mapped to so-called rules, each being equipped with a condition and possibly multiple constraints. A condition describes a situation of the configuration solution that must be given for the constraints to be evaluated. engcon offers a wide variety of pre-defined constraints that restrict a given set of concept attributes, including formulae and tables. Simple dependencies (such as greater, less, equals, and so on) and formulae can easily be created using K-Build. Tabular dependencies from K-Model can also be mapped to K-Build's Excel representation with little effort.

The configuration application for Schmalz was set up in two distinct steps. In a first step we created a proof-of-concept for which the least effort should be used. This proof-of-concept was the configuration model running in K-Build's test environment K-Test. In a second step we realized the configuration application full-scale: with stable data exchange interfaces and full graphical user interface. Hence, the data about available components was received in two different ways within the respective steps.

1. In the proof-of-concept step the product data was transferred from the K-Model Excel format to the K-Build Excel format.
2. In the full-scale step the configuration application was set up using encoway's standard architecture. The product data contained in the Excel sheets was converted into encat, which is encoway's standard format for realizing media-neutral master data exchange, based on a well-defined xml structure.

The K-Model Excel sheets containing product data can be transformed into a corresponding K-Build Excel sheet with little manual effort. This way, the available components are imported into the configuration model as specializations of concepts that stem from the classification data. This first step was carried out for testing purposes.

The encat xml document containing product data was imported into the so-called catalogue. encat documents also contain all relevant translations and pricing information, which is relevant for the application user interface and for quote generation, i.e. during run-time, not for creating the configuration model during build-time. Instead, encoway configuration models are typically language-neutral and do not contain the available components or pricing information. The catalogue is a single place for all this information. Technically, it is a database that comes with an advanced API for querying the different types of data during run-time.

While product information, including the translations and pricing information change over time, the physics, on which the product configuration is based, typically stays stable. The physics is represented within the configuration model while the actual components are not. The major benefit of using encat as stable data exchange interface is thus that the configuration model need not be changed when importing new product data.

For realizing the Schmalz configuration application we use encoway's quoting process-supporting tool QuoteAssistant. The QuoteAssistant is a web-based application for browsing catalogue content, configuring products, creating quoting structures together with pricing and generating high quality quote documents; all in one place. The QuoteAssistant contains a standard user interface design for displaying concepts and their attributes within a tab structure using a widget collection containing checkboxes, select boxes or text input fields. This means that, when treating all concepts that are modeled as parts of the K-Model questions as tabs, the placement of sales questions is determined by their attributes and no extra definition for user interface is required.

The user is free in structuring configurable products and available components from the catalogue within folders of a quoting structure. The result of the quoting process is such a structure together with pricing and conditions. This quote result can be exported to a MS Word or PDF documents via the tool K-Document. This tool allows using pre-defined MS Word templates and enriching them with the configuration results, content from a CRM system (such as address data) and from the catalogue (product information or images) automatically during run-time.

4 CONCLUSION

In this paper we have shown how the knowledge acquisition methodology K-Model helps a knowledge engineer to focus on the product domain rather than on knowledge representation formalisms while creating a configuration model. The visualization facilities of standard mind maps ease the creation of configuration models for product managers and sales personnel who are typically not experts in the area of knowledge representation. Especially within workshops where persons with different backgrounds together acquire the relevant knowledge for a configuration applications this informal mind map representation is a valuable tool.

The results which are produced by the analysis steps described in Section 2.2 may seem rather tentative at first sight. However, the results remain stable once the process of designing a configuration model has gone through a small number of design cycles (see also Figure 5). K-Model was already used to analyze and design configuration models for roughly 20 domains, mostly of very different nature and size. The largest domains consist of up to 2000 variables that are relevant for product configuration within this domain. We thus see this as a significant number of cases to call K-Model a success for supporting the process of analyzing a configuration domain and designing a respective configuration model. For encoway, however, the Schmalz configurator is the first application of a K-Model. But nonetheless, the input in form of well-designed mind maps and Excel sheets significantly improved setting up a configuration model from scratch.

An extension of K-Model that is currently under development is modularizing the mind map in multiple sub-maps. With this approach it is possible to describe smaller parts of a configuration that can be reused (multiple times) within larger configuration contexts. The modularization also enhances keeping an overview of large configuration domains.

For J. Schmalz GmbH, K-Model was applied while creating a working prototype configuration application. It took just a few workshops with product managers and sales personnel to set up the K-Model mind maps and MS Excel. This input data was of high quality and could be directly used by encoway modeling experts for creating a configuration model of the product domain.

Schmalz is now able to fully benefit from the configuration application that was set-up using the K-Model knowledge acquisition methodology. Applying K-Model within this project was successful in that all relevant persons – including product managers, sales personnel and technicians – were able to focus on the specific characteristics of the desired configuration application without extra effort for learning representation facilities. The methodology significantly increased the efficiency of cross-department communication and reduced the time-to-prototype during realization.

5 RELATED WORK

Because knowledge acquisition in the environment of knowledge-based configuration systems is notoriously a very expensive process, there is other work concentrating on this task. Support for knowledge acquisition tasks ranges from propose-and-revise techniques that help users in deciding on correctness to graphical representation in form of UML class diagrams or mind maps.

The work described in [7] explicitly targets to support the task of knowledge acquisition for configuration knowledge bases with a propose-and-revise strategy. It is implemented in the knowledge acquisition tool EXPECT, which uses LOOM, a knowledge representation system based on description logics. The focus of this work is on correctness of the underlying knowledge and does not take graphical representation into account.

In [6] a UML representation for configuration knowledge bases is introduced for the purpose of enhancing sharing, distribution and cooperation within the use configuration knowledge. UML stereotypes are defined to represent the specifics of configuration such as concepts, attributes, taxonomy and partonomy. Constraints are defined using OCL. In [8] the authors bring the idea one step further by introducing a set of rules for transforming UML models into configuration knowledge based on description logics such as OIL or DAML+OIL. This work explicitly aims at supporting the knowledge acquisition bottleneck with graphical representation as a frontend and can thus be seen similar to the K-Model approach, although K-Model prefers mind maps over UML diagrams.

The authors of [5] also use mind map structures to support knowledge engineers. However, their work focuses on formalizing, sharing and reusing experiences of past projects in order to help avoiding mistakes that these projects have already encountered. Their work differs from ours in the sense that they use mind maps to capture and represent project experience while we use mind maps to capture and represent configuration knowledge.

The methodology K-Model is novel in the way that is explicitly targets to support non-experts during the acquisition of configuration knowledge by using mind maps as a graphical frontend. Furthermore, the K-Model explicitly distinguishes master data and product structure, configuration decisions and the configuration solution. It defines the syntax and semantics of usable mind map structures as well as the modeling process, i.e. how to use the mind maps in workshop situations together with non-experts such as product managers or sales personnel.

REFERENCES

[1] A. Brinkop. *Variantenkonstruktion durch Auswertung der Abhängigkeiten zwischen den Konstruktionsbauteilen*, Dissertationen zur Künstlichen Intelligenz, Band 204, Infix, St. Augustin, 1999.

[2] D. Sabin and R. Weigel. Product Configuration Frameworks – a Survey. *IEEE Intelligent Systems*, 13(4):42–49, 1998.

[3] O. Hollmann et al. EngCon: A Flexible Domain-independent Configuration Engine. In: *Proceedings of Configuration (ECAI 2000-Workshop)*:94–96, 2000.

[4] C. Ranze et al. A Structure-based Configuration Tool: Drive Solution Designer (DSD), In: *Proceedings of AAAI02 / IAAI02*: 845–852, 2002.

[5] C.-S. Chen and Y.-C. Lin. Enhancing Knowledge Management for Engineers Using Mind Mapping in Construction, *New Research on Knowledge Management Technology*, Dr. Huei Tse Hou (Ed.), ISBN: 978-953-51-0074-4, 2012.

[6] A. Felfernig et al. Configuration Knowledge Representation Using UML/OCL, *Lecture Notes in Computer Science*, Volume 2460, 91-108, 2002.

[7] S. Ramachandran, Y. Gil. Knowledge Acquisition for Configuration Tasks: The EXPECT Approach. In: *Proceedings of Configuration (AAAI 1999-Workshop)*:29–34, 1999.

[8] A. Felfernig at al. UML As Knowledge Acquisition Frontend for Semantic Web Configuration Knowledge Bases, In: *RuleML 2002 – Proceedings of the International Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, Michael Schröder and Gerd Wagner (Eds.), Volume 60, 2002.

Towards a Formalism of Configuration Properties Propagation

David Fabian[1] and **Radek Mařík**[2] and **Tomáš Oberhuber**[3]

Abstract. Software configuration often studies two issues: firstly, how to merge various software components together to create a program with a fixed structure that fits the requirements, and secondly, how to effectively set up the remaining (usually installation specific) configuration options when deploying the program. Nowadays, the user demands a simple and well arranged way to set up these options, possibly through a graphical user interface (GUI). There are various tools designed to assist the user with these tasks. In this paper, a general multi-platform configuration tool Freeconf is introduced. Our technique to simplify a GUI, which has been incorporated into Freeconf, is described. This technique is based on a set of properties that allow splitting the universe of configuration options into several categories with a clear semantics and rules that control the dynamics of options distribution to these categories in response to the user's actions. The rules are currently only implemented in the source code of Freeconf as a proof-of-concept without any formal proof of soundness or completeness. Results from the domain of Rule-Based Constraint Programming have been applied in the paper to develop a formal description of the rules.

1 Introduction

While working with a software application, the user usually needs to adjust a working environment to her needs. Nowadays, almost every application lets the user to perform some configuration. The average user often does not understand the background of the program and expects a nice graphical user interface (GUI) to assist her. However, there are many applications (especially in the GNU/Linux environment) that do not have any GUI whatsoever and the only way how to configure them is through configuration text files. A serious problem of these files is that their syntax differs greatly, so the user must learn it first from the documentation. It is also necessary for the user to deeply understand the meaning of various configuration options (configuration keys), their dependencies, and their possible values.

1.1 Configuration Tools

There exist tools that address the above mentioned difficulties. Some are focused on a given domain (or even at one application, Linux kernel is popular) like SmartFrog [2] and LCFG [1, 3] which are designed to administer the installation of large scaled networks of UNIX systems, or MenuConfig [16, ch. 7] which is a primary tool for Linux kernel configuration. Then there exist general tools like LinuxConf [10] and Freeconf [9]. Freeconf is a unique tool as it offers a multi-platform and a multi-desktop configuration of applications of any kind.

1.2 Configuration Properties

Many automatic configuration tools suffer from the overwhelming complexity of the user interface they generate which is a severe problem for the user. One of the possibilities of solving this issue consists in breaking the uniform universe of keys to several categories and providing the categories with an exact semantics. Then, only the keys from a category which is the most interesting to the user at a given moment can be displayed. The solution presented here is based on a set of properties of keys, in other words, on a set of labels that are assigned to every key and determine its membership to a category. In Table 1, there is an example of a set of possible properties for keys categorization. The last property *undefined* represents a set of keys that do not have their value set and therefor could cause problems in the output of the configuration. In other words, this property allows us to describe a form of inconsistency with the instant state of the configuration.

property	meaning
mandatory	The key is important to the configured application and must be filled in.
meaningful	The key has sense in the present settings and its existence is not ruled out by any dependency.
undefined	The key finds itself in an invalid state such as that it has no value set or the value is in conflict with dependencies.

Table 1. Properties used for configuration keys categorization.

Having each key as a feature, this approach resembles feature modeling [7] with extra-functional attributes. The semantics for operators and dependencies, however, is different.

The distribution of keys into categories does not have to be static, some keys can change their roles during the configuration process in response to an outer activity (a dependency event, user input). A mechanism is needed to control the development of the categories. For this, Freeconf uses rules to control the propagation of properties. The current set of rules in Freeconf has been constructed by hand and tested only empirically; it has not been proved, whether the rules are sound or complete. Techniques of the rule-based constraint programming can provide a proof; however, one must first give a formal description to the rules.

[1] Dept. of Mathematics, Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University in Prague, (`fabiadav@fjfi.cvut.cz`).
[2] Dept. of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague.
[3] Dept. of Mathematics, Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University in Prague.

The rest of this paper is divided as follows. In Section 2, a brief introduction to Rule-Based constraint programming is presented. Section 3 describes the structure of Freeconf and a set of properties used in the tool. Section 4 introduces the way Freeconf handles propagation of properties. Finally, Section 5 presents a formalized set of rules, and Section 6 concludes.

2 Rule-based Constraint Programming

Rule-Based constraint programming is a special case of a more general constraint programming studied in [4, 8, 12, 13]. Constraint programming is an alternative approach to programming in which a model of a problem is declarative, and then it is solved by general or domain-specific methods. The model is formed by a set of constraints (requirements) on variables so that acceptable variable assignments correspond to solutions to the problem [4]. Following [8], let us assume a sequence of variables

$$X = x_1, \dots, x_n$$

with respective domains

$$D_1, \dots, D_n,$$

so each x_i takes its value from the set D_i.

Definition A constraint C on X is a pair $\langle C_R, X\rangle$ where C_R is an n-ary relation over the domains D_i, i.e., $C_R \subseteq D_1 \times \dots \times D_n$, of solutions to the constraint.

Definition A constraint satisfaction problem (CSP) is a triple $\langle \mathcal{C}, X, D\rangle$ where $X = x_1, \dots, x_n$ is a finite sequence of variables with respective domains $D = D_1, \dots, D_n$, and $\mathcal{C}$ is a finite set of constraints, each on a sub-sequence of X.

Definition A solution to the CSP $\langle \mathcal{C}, X, D\rangle$ is an element $d \in D_1 \times \dots \times D_n$ such that for each constraint $C \in \mathcal{C}$ on a sequence of variables Y it holds $d\,[Y] \in C$ where $d\,[Y]$ stands for a projection of d to $Y = x_{i(1)}, \dots, x_{i(l)}$, i.e., $d\,[Y] = d_{i(1)}, \dots, d_{i(l)}$. $Sol\,(\langle \mathcal{C}, X, D\rangle)$ will denote the set of all solutions to the CSP $\langle \mathcal{C}, X, D\rangle$.

There exist general algorithms for solving a CSP [8, 14] and even entire frameworks, such as Skyblue [15], ECLiPSe [5], and Minion [11]..

The core concept of Rule-Based programming is a rule. Rules are condition-action pairs where the condition part is used to determine whether the rule is applicable and the action part defines the action to be taken [6]. A formal definition of a rule taken from [8] follows.

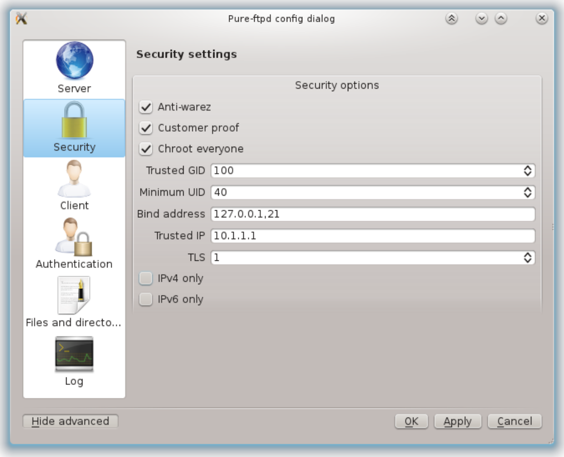
Definition Assume that $\mathcal{A}$ and $\mathcal{B}$ are sets of constraints such that the constraints in $\mathcal{A}$ and $\mathcal{B}$ are on the variables X with domains D. The expression $\mathcal{B} \leftarrow \mathcal{A}$ is a constraint propagation rule. $\mathcal{A}$ is called the condition and $\mathcal{B}$ the body of the rule. Rules act as functions on CSPs. The application of a rule to a CSP with the variables X is given by

$$(\mathcal{B} \leftarrow \mathcal{A})(\langle \mathcal{C}, X, D\rangle) := \begin{cases} \langle \mathcal{C} \cup \mathcal{B}, X, D\rangle & \text{if} \mathcal{A} \subseteq \mathcal{C}, \\ \langle \mathcal{C}, X, D\rangle & \text{otherwise.} \end{cases}$$

The rule $\mathcal{B} \leftarrow \mathcal{A}$ is correct if $Sol(\langle \mathcal{A}, X, D\rangle) \subseteq Sol(\langle \mathcal{B}, X, D\rangle)$.

3 Freeconf

To help the user to overcome difficulties of software configuration, the project Freeconf has been established. The purpose of this project is to unify the existing configuration process and to assist the user by automatically creating the configuration dialog window similar to one in Fig.1. The configuration process must be smooth from the point-of-view of the user: the dialog must fit nicely into the desktop environment, configuration options must be presented in the logical order, and only what is crucial to fill in should be shown. To better understand how Freeconf addresses these tasks, a short description of its inner structure is presented.



Security settings

Security options

[✓] Anti-warez
[✓] Customer proof
[✓] Chroot everyone
Trusted GID 100
Minimum UID 40
Bind address 127.0.0.1,21
Trusted IP 10.1.1.1
TLS 1
[ ] IPv4 only
[ ] IPv6 only

Figure 1. Example of Freeconf generated configuration dialog.

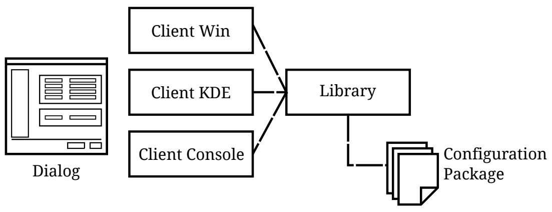


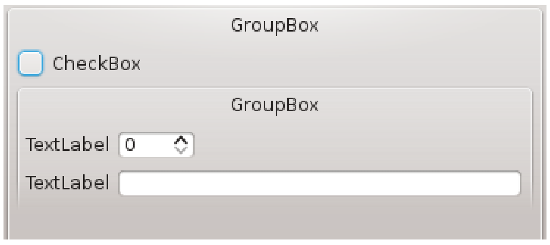
Figure 2. Components of the Freeconf project.

3.1 Structure of Freeconf

The project consists of several components as shown in Fig.2. The most important one is the Freeconf library which contains the entire functionality. The library is shared by graphical client applications (clients), the sole purpose of which is to present a configuration dialog to the user. One can have many different clients, each for one desktop environment. The clients are supposed to be very small (less than a 1000 lines of code), so it should be very simple to create another one. To be able to configure an existing application, the library needs to be provided with an appropriate configuration package. The package is a collection of XML files that semantically describe the configuration file the application understands (native configuration file). It contains a list of all keys, their default values, properties and dependencies, and a rough description of what the resulting dialog should look like. Keys can be organized into configuration sections.

The number of keys can vary depending on the configured application. Usually, configuration packages have tens to hundreds of keys, but some configurations utilize up to thousands of keys like in the case of the Linux kernel.

When a package is loaded, the library constructs three tree structures — a *template tree* for storing key properties, a *configuration tree* for storing values and handling dependencies, and a *GUI tree* for dialog modeling. The trees are interconnected, and one can freely traverse from one node of one tree to the matching node of another tree. Leave nodes correspond to keys and their properties, the non-leave nodes represent configuration sections (there is always a root-section present). When a client needs data for dialog construction, it connects to the library through the client-library interface and obtains various properties for each node. Fig.3 shows the situation. The interface forms a tree which is placed between the GUI tree in the library and the hierarchy of GUI elements (group-boxes, line edits, check-boxes, etc.). The client organizes the GUI elements to another tree that is very closely related to the actual look of the dialog.



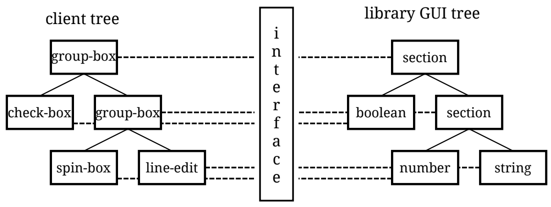


Figure 3. Tree data structures and the client-library interface. The top figure shows how the client tree is transformed into a dialog form.

3.2 Properties in Freeconf

Since the beginning of the project, every client could use the basic set of properties for each key. These basic properties are presented in Table 2.

property	meaning
name	Name of the key.
label	Label for the key.
help	Tooltip help text.
value	Value of the key or default value if no value exists.
type	Type of the key. It can be: boolean, number, string, stringlist, or section.

Table 2. Basic properties of configuration keys.

Every key type adds additional properties, e.g., a *number* can have a *minimum*, a *maximum*, and a *step* by which the value increments and decrements. String keys usually have a regular expressions associated with them constraining their value.

While the basic set of properties would generally suffice to construct a dialog, the dialog would look overfilled and confusing to the user. That is why another set of properties was added to Freeconf which would enable splitting keys into different categories. Thus, only keys from a specific category can be shown to the user. The current set of properties which extends those presented in Table 1 is summarized in Table 3.

property	meaning
static mandatory	If it is true, the key is mandatory and must be always shown.
static active	If it is false, the key is not visible to the client.
dynamic mandatory	This property can only be set from a dependency handler. If it is true, the key is mandatory and must be shown. This property has no meaning when the static mandatory property is set to false.
dynamic active	This property can only be set from a dependency handler. If it is false, the key does not have sense in the current settings. This property has no meaning when the static active property is set to false.
inconsistent	The key does not have neither a value, nor a default value set and is dynamically mandatory and dynamically active.
empty	The property is only applicable to section nodes. It states whether the section is empty, i.e., all its children are hidden.

Table 3. Additional properties used for keys categorization.

All *static* properties are stored in the package as a part of a template describing the native configuration file, while the *dynamic* ones are a part of a file describing dependencies.

Mandatory property states, whether the key is important or not. Important keys should be visible in the dialog while non-mandatory keys can be hidden, so the dialog becomes less confusing. The *static* version of this property is used for packages with no dependencies or for keys unaffected by any dependency. The *dynamic* version can, as in the current version of Freeconf, override the static state only when the static mandatory property is not false (that means static non-mandatory keys are definite).

Active property has two different semantics. In its *static* version, it is used to prevent the library from announcing the key to the client. In other words, if the property is set to false, the key is virtually commented out. It is easier to disable the key that way than to delete it from the entire package which is non-trivial. The *dynamic* version of this property serves the purpose of ruling out situations that do not have sense (e.g., when the user checks the "no sound" option, setting the "volume" option becomes nonsensical and this option should be left out from the dialog or at least disabled).

Inconsistency is a special situation when the key does not have any value set, but it is important to the configured program. This can happen, especially when the configuration is run for the first time, and there exist keys which do not have default value set by the creator of the package. When this situation occurs, the user has to be told so and must be able to solve the problem with minimum effort. It follows from the above mentioned description of the properties there are situations where inconsistency is acceptable and the user needs not to be alerted (e.g., when the key is statically inactive). In fact, there exists exactly one combination of the properties which needs some user assistance (i.e., the client must be informed) — the key is dynamically mandatory, dynamically active, and inconsistent at the same time. In other situations, such as when the key is only dynamically active but not mandatory, the key is simply left out from the native output (so the native output will always contain only keys with a defined value).

Emptiness is an important property which naturally arose from the need of hiding non-mandatory keys. The user cannot be distracted by optional keys, so those must be hidden. If there is a section containing only hidden keys, there is no point of displaying it. The empty property can help the client with hiding of unnecessary GUI elements.

4 Properties Propagation

Freeconf maintains properties in connected tree structures as described in Section 3. The Freeconf library must be able to inform the clients about the state of each property in every node the client–library interface announces. For leave nodes, this becomes trivial. For non-leave nodes, however, the state of a property must reflect what is happening in all of node's direct successors.

4.1 Propagation Mechanism

To keep record of the number of properties in children nodes, every section has a set of counters, each bound to a specific property. Counters hold how many times the matching property occurs in the successors. For example, for the inconsistent property the "inconsistent count" counter exists in each section and if it is, for instance, set to two, then there are exactly two children nodes that are inconsistent.

If a counter reaches zero, a message about the change of a property is sent to the client from the affected section. The section must also inform its parent (i.e., another section) about the change, so the appropriate counter of the parent can be adjusted. Similarly, a message must be sent whenever a nullified counter is incremented.

The entire propagation schema can be seen in Fig.4.

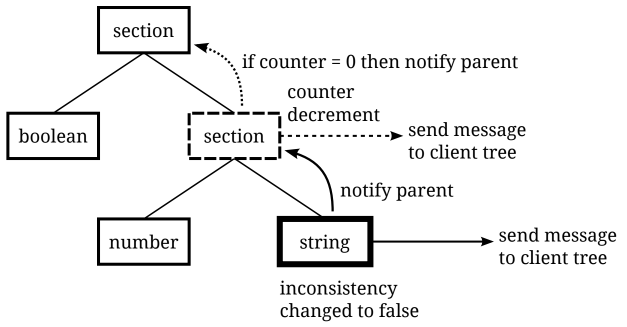

section
if counter = 0 then notify parent
counter decrement
boolean
section
send message to client tree
notify parent
number
string
send message to client tree
inconsistency changed to false

Figure 4. Propagation of properties in Freeconf. Inconsistency of the bold node has been changed. The information is propagated into the parent node (dashed). The property can be further propagated. Every change is sent to the client tree also.

When the state of a property (inconsistency in this case) has been changed, the node notifies its parent about the change. The parent section increments or decrements the matching counter and checks, whether the counter is zero or not. If it is zero, the notification is propagated further up the tree. This leads to the expected behavior in the client since every path leading to an inconsistent key is marked, so the client can render it appropriately. The top-level section (a configuration tab in fact) also knows about the overall state of all keys underneath and it can, for instance, forbid creating the native output until all inconsistencies have been resolved. This method requires a protection mechanism against resending the same message. An obvious solution would be to remember the last state of each property for every node and inform the parent only if the state changes.

For this algorithm to be valid, all counter must be set to the correct value at start time. This is called the initialization phase. All counters are set to zero, and the tree is traversed by depth-first search. Every leave node is evaluated and the existing propagation framework is used to initialize all counter values.

It is also possible to emit a global change, for instance, when the user overrides the mandatory property and enforces showing all keys which are dynamically active. In such a case, all leave nodes are asked to reevaluate their states similarly to the initialization phase. In fact, the initialization phase is a form of a global change.

5 Rules

The above mentioned algorithm was implemented in an ad hoc manner. All property evaluation procedures were tailored to the semantics described in Section 3. The result is a set of rules implemented as condition statements in the source code.

The goal of this section is to bring a formal description of the resulting rules based on definitions from Section 2.

5.1 Formal Description

Let $K = \{k_1, \ldots, k_n\}$ be a set of indices for keys and $S = \{s_1, \ldots, s_l\}$ a set of indices for sections. Let $parent : K \cup S \to S \cup \{\emptyset\}$ be a mapping returning for each key or section its parent. The symbol of an empty set is returned for the top-level section. All properties of keys will be modeled as Boolean variables. For example, $dynact_x$ will denote a dynamic active property of a key with an index $x \in K$. Together with the properties from Table 3, variables $defvalset_x$ and $valset_x$ will be used to describe the states where a default value and a value have been set to the key, respectively.

Section counters will be modeled as non-negative integer variables. As an example, $inconsistcount_y$ represents the state of an inconsistent counter in a section with an index $y \in S$. If the index is $\emptyset$, no action is performed.

Moreover, there is a Boolean variable called $showallact$ which enables showing even non-mandatory properties (i.e., showing all active keys regardless of the state of the mandatory property). The list of all rules currently used in Freeconf follows.

In the initialization phase, $dynact_x$ and $dynman_x$ are set according to the static version of the properties and $inconsistent_x$ is evaluated for the first time.

$$\begin{aligned} dynact_x &\leftarrow staticact_x \ \forall x \in K \\ dynman_x &\leftarrow staticman_x \ \forall x \in K \\ inconsistent_x &\leftarrow (\neg defvalset_x \wedge \neg valset_x) \wedge \\ &\quad \wedge dynman_x \wedge dynact_x \ \forall x \in K \end{aligned}$$

When the $valset_x$ variable changes its value, these rules are applied to update inconsistency.

$$\begin{aligned} inconsistent_x &\leftarrow (\neg defvalset_x \wedge \neg valset_x) \wedge \\ &\quad \wedge \neg inconsistent_x \wedge dynman_x \wedge \\ &\quad \wedge dynact_x \ \forall x \in K \end{aligned}$$

$$\begin{aligned} \neg inconsistent_x &\leftarrow \neg(\neg defvalset_x \wedge \neg valset_x) \wedge \\ &\quad \wedge inconsistent_x \ \forall x \in K \end{aligned}$$

Whenever either $dynman_x$ or $dynact_x$ variable changes its value, the inconsistent state of the node must be reevaluated and the parent's counter is adjusted accordingly.

$$
\begin{aligned}
&inc(inconsistcount_{parent(x)}) \leftarrow \\
&\leftarrow dynman_x \wedge inconsistent_x \wedge dynact_x \; \forall x \in K \\
\\
&dec(inconsistcount_{parent(x)}) \leftarrow \\
&\leftarrow (dynman_x \wedge inconsistent_x \wedge \neg dynact_x) \vee \\
&\vee (dynact_x \wedge inconsistent_x \wedge \\
&\wedge (\neg dynman_x \vee \neg inconsistent_x)) \; \forall x \in K
\end{aligned} \tag{1}
$$

If the $inconsistcount_y$ alters, the section must test if it is not necessary to propagate the information further.

$$
\begin{aligned}
&dec(inconsistcount_{parent(y)}) \wedge \neg inconsistent_y \leftarrow \\
&\leftarrow (inconsistcount_y = 0) \wedge inconsistent_y \; \forall y \in S \\
\\
&inc(inconsistcount_{parent(y)}) \wedge inconsistent_y \leftarrow \\
&\leftarrow \neg(inconsistcount_y = 0) \wedge \neg inconsistent_y \; \forall y \in S
\end{aligned}
$$

The $mandatoryshown_y$ and $activeshown_y$ counters change when a dependency alters $dynman_x$ and $dynact_x$, respectively. It must be also tested whether the static equivalents to the respective properties have not been set to false.

$$
\begin{aligned}
\neg dynman_x \leftarrow& \neg staticman_x \wedge \\
&\wedge dynman_x \; \forall x \in K \\
\neg dynact_x \leftarrow& \neg staticact_x \wedge \\
&\wedge dynact_x \; \forall x \in K \\
inc(mandatoryshown_{parent(x)}) \leftarrow& dynman_x \; \forall x \in K \\
dec(mandatoryshown_{parent(x)}) \leftarrow& \neg dynman_x \; \forall x \in K \\
inc(activeshown_{parent(x)}) \leftarrow& dynact_x \; \forall x \in K \\
dec(activeshown_{parent(x)}) \leftarrow& \neg dynact_x \; \forall x \in K
\end{aligned}
$$

The $empty_y$ variable must be reevaluated for each section every time any of its counters (except $inconsistcount_y$) changes.

$$
\begin{aligned}
&\neg empty_y \wedge dec(sectionshown_{parent(y)}) \leftarrow \\
&\leftarrow empty_y \wedge \neg(sectionshown_y = 0) \vee \\
&\vee (showallact \wedge \neg(activeshown_y = 0)) \vee \\
&\vee (\neg showallact \wedge \neg(activeshown_y = 0) \wedge \\
&\wedge \neg(mandatoryshown_y = 0)) \; \forall y \in S \\
\\
&empty_y \wedge inc(sectionshown_{parent(y)}) \leftarrow \\
&\leftarrow \neg empty_y \wedge ((mandatoryshown_y = 0) \wedge \\
&\wedge (sectionshown_y = 0)) \vee ((activeshown_y = 0) \wedge \\
&\wedge (sectionshown_y = 0)) \; \forall y \in S
\end{aligned} \tag{2}
$$

5.2 Weaknesses of Freeconf Design

It can be easily seen that some of the rules are not optimal. For instance, the second rule in 1 could be shortened by leaving out the last occurrence of $inconsistent_x$. In 2, the rules should be mutually exclusive, but it is non-trivial showing the head formulas really behave that way.

Clearly, a problem of the current implementation is the lack of formal description. All condition statements are scattered across the source code, and it is very complicated maintaining them even though the number of the properties is very small. The design is also not very robust since a small change in any of the conditions will render the system non-functioning. This actually happened — one conjunction was overwritten by mistake by a disjunction, and the client started behaving strangely. It was obvious there was a mistake in a condition, but it was difficult to find it.

6 Conclusion

This paper introduces Freeconf, a multi-platform configuration tool, and a technique which reduces the problem of very complex graphical user interfaces that are often generated by automatic configuration tools. The technique is based on splitting configuration options into categories using properties and forming a set of rules that control the dynamics of the evolution of the categories. A set of rules has been proposed to be used in Freeconf to simpify its graphical output. The rules have been implemented in the source code as a proof-of-concept, and it has been empirically verified that the rules work. In this paper, a formal description of the rules has been presented based on the theory of Rule-Based programming. The proof of soundness and completeness of the rules is subject of future work.

7 Acknowledgments

This work was partially supported by the project of the Student Grant Agency of the Czech Technical University in Prague No. SGS11/161/OHK4/3T/14, 2011-13 and Research Direction Project of the Ministry of Education of the Czech Republic No. MSM6840770010.

REFERENCES

[1] Paul Anderson, *LCFG: A Practical Tool for System Configuration*, The USENIX Association, 2008.

[2] Paul Anderson, Patrick Goldsack, and Jim Paterson, 'SmartFrog Meets LCFG: Autonomous Reconfiguration with Central Policy Control', in *Proceedings of the 17th USENIX conference on System administration*, LISA '03, pp. 213–222, Berkeley, CA, USA, (2003). USENIX Association.

[3] Paul Anderson, Alastair Scobie, and Division Of, 'LCFG - the Next Generation', in *UKUUG Winter Conference. UKUUG*, (2002).

[4] K.R. Apt, *Principles of Constraint Programming*, Cambridge University Press, 2003.

[5] K.R. Apt and M. Wallace, *Constraint Logic Programming Using ECLiPSe*, Cambridge University Press, 2007.

[6] Krzysztof R. Apt and Eric Monfroy, 'Constraint Programming viewed as Rule-based Programming', *CoRR*, **cs.AI/0003076**, (2000).

[7] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés, 'Automated reasoning on feature models', in *Proceedings of the 17th international conference on Advanced Information Systems Engineering*, CAiSE'05, pp. 491–503, Berlin, Heidelberg, (2005). Springer-Verlag.

[8] Sebastian Brand, *Rule-Based Constraint Propagation Theory and Applications*, Ph.D. dissertation, 2004.

[9] David Fabian, *System for Simplified Generating of Configurations*, Master thesis, Faculty of Nuclear Sciences and Physical Engineering, Prague, 2011. in Czech.

[10] Jacques Gélinas. Linuxconf homepage, 2005. http://www.solucorp.qc.ca/linuxconf/.

[11] Ian P. Gent, Chris Jefferson, and Ian Miguel, 'Minion: A fast scalable constraint solver', in *Proceedings of ECAI 2006, Riva del Garda*, pp. 98–102. IOS Press, (2006).

[12] Michael Gleicher, 'Practical issues in graphical constraints', in *Principles and Practice of Constraint Programming*, pp. 407–426. MIT Press, (1995).

[13] K. Marriott and P.J. Stuckey, *Programming With Constraints: An Introduction*, Mit Press, 1998.
[14] Nico Roos, Yongping Ran, and H. Jaap van den Herik, 'Combining Local Search and Constraint Propagation to Find a Minimal Change Solution for a Dynamic CSP', in *Proceedings of the 9th International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, AIMSA '00, pp. 272–282, London, UK, UK, (2000). Springer-Verlag.
[15] V. Saraswat and P. Van Hentenryck, *Principles and Practice of Constraint Programming: The Newport Papers*, chapter The SkyBlue Constraint Solver and Its Applications, 385–405, Mit Press, 1995.
[16] Sven Vermeulen, 'Linux sea'. http://swift.siphos.be/linux_sea, 2012.

Testing Object-Oriented Configurators With ASP [1]

Andreas A. Falkner and **Gottfried Schenner** [2], **Gerhard Friedrich** and **Anna Ryabokon** [3]

Abstract. Testing is an important aspect of every software project. For configurator applications it is equally important but often neglected. This paper shows how to support testing object-oriented and constraint-based configurators by automatically generating positive and negative test cases using answer set programming (ASP). The object-model of the configurator is mapped to ASP code; the constraints to be tested are coded redundantly in ASP. Based on that, the ASP solver generates appropriate test cases, which are then used for unit testing in the object-oriented configurator. There are different strategies to improve this basic process, e.g. reduction of the number of test cases with symmetry breaking.

1 Introduction

Testing is an important but often neglected aspect of every software development project. Especially for object-oriented (OO) languages, unit testing with a testing framework like JUnit [3] is well established and an integral part of development methods like Extreme Programming [4]. Unit testing frameworks are also gaining acceptance outside of object-oriented programming [9].

A configurator is a software system that enables the user to configure complex systems or services using predefined components. In a constraint-based configurator, constraints describe the conditions which the configured system must satisfy. In order to test the correctness of each individual constraint, the tester must provide positive and negative test cases for it. A positive (negative) test case is a partial configuration where the constraint is satisfied (violated). Obviously, the test cases cannot be created by the solver of the configurator because one cannot use the possible faulty constraint to generate the test case. Therefore, the test cases currently must be created manually.

There are different testing strategies such as black-box and white box testing ([16]. In black-box testing, the internal structure of the test object must not be known to the tester and the tests are devised according to the specification of the software system. In white-box testing, the internal structure is known and the tester designs the tests to achieve a high test coverage. In practise both strategies should be used because they tend to find different kind of errors.

The basic idea of this paper is to semi-automatically generate test cases for object-oriented configurators by first translating the configurator's knowledge base (without the constraints to be tested) to an answer set programming (ASP) program. The constraints to be tested are then coded manually in ASP. Implementing the same constraint both in Java and ASP achieves the necessary diversity to detect conceptional errors (similar to N-Version programming [1]). The ASP solver runs this program and generates positive and negative test cases which are translated back into test cases for the object-oriented configurator.

The following section defines necessary features of the configurator and provides a brief introduction to the ASP systems. In Section 3 we describe the approach in more details presenting the OO-ASP mapping and examples for a small application. We show different ways to reduce the number of generated test cases to a reasonable size in Section 4 and conclude in Section 5.

2 Context

For this work, we used a configurator based on Generative Constraint Satisfaction (GCSP) which is a combination of object-oriented and constraint-based technologies. In general however, any system that complies to the definition of the following subsection can be used. The current target system is the Potassco ASP suite[4] [12] - it could easily be replaced by another ASP system.

2.1 Object-oriented constraint-based configurator

The results of this paper can be applied to any existing configurator framework which complies to the following definitions.

Definition 1 (Knowledge Base, KB) *The knowledge base of an object-oriented and constraint-based configurator comprises an object model and a set of constraints.*

The KB specifies the relevant domain knowledge in a declarative way. The solver comprises a general constraint solver which reasons over that knowledge, e.g. checks consistency, searches solutions (i.e. valid configurations), etc.

Definition 2 (Object Model) *An object model contains classes, their inheritance hierarchy, attributes (Boolean, enumeration, integer), and associations (bidirectional).*

The object model describes the structure of the possible configurations, including the multiplicities (cardinalities) of the parts. It can be specified by an UML class diagram [17].

Definition 3 (Configuration) *A configuration is an instantiation of the object model.*

Without loss of generality, only instances of leaf classes (classes without subclasses) are allowed in a configuration. For the course of this paper, it is assumed that the configurator maintains one current configuration. In an interactive configurator, the user would manipulate the current configuration by adding/deleting objects and setting

[1] This work has been developed within the scope of the project RECONCILE (reconciling legacy instances with changed ontologies) and was funded by FFG FIT-IT (grant number 825071).

[2] Siemens AG Österreich, Vienna, Austria firstname.{middleinitial.}lastname@siemens.com

[3] Universität Klagenfurt, Austria firstname.lastname@aau.at

[4] http://potassco.sourceforge.net

attributes and associations until a valid configuration is found. Alternatively the constraint solver can be used to extend a configuration to a valid configuration. Constraints are used to describe the valid configurations of the configurator.

Definition 4 (Constraint) *A constraint is a condition which every valid configuration must satisfy.*

This is a very general definition of the concept constraint. To make our approach broadly applicable, no special constraint techniques like domain-filtering, constraint propagation, etc. are required. A constraint can be thought of as an invariant constraint in UML/OCL. In its simplest form, constraints are Boolean methods of an object-oriented language defined over the current configuration. From a knowledge engineering view, constraints should correspond to some requirements that the product to configure must satisfy. The scope of a constraint can range from simple expressions like 'wheel1.size = wheel2.size' to 'The light-system of this vehicle is configured correctly' (represented by some complex code accessing sub-parts and their properties).

2.2 Answer Set Programming

Answer set programming is an approach to declarative problem solving which has its roots in logic programming and deductive databases. This is a decidable fragment of first-order logic extended with default negation, aggregation and weight constraints. ASP allows modeling of a variety of search and optimization problems in a declarative way [13, 7, 5] using model-based problem specification methodology. Efficient ASP solvers allow fast identification of solutions that correspond to answer sets of a program. Recent examples include areas such as molecular biology, decision support and planning. The DLV system [15] was used to plan shifts at Gioia-Tauro Seaport which reduced the time required to define working teams' assignments from hours to just a few minutes. A Potassco [12] program is able to detect inconsistencies in large biological networks.

Since configuration problems are a type of combinatorial (optimization) problems, ASP was used by Soininen et al. [18] in their approach which was one of the earliest industrial applications of ASP. This first approach to the configuration problem was extended by Friedrich et al. [10] to both configuration and reconfiguration cases. Recently, Gebser et al. [11] have suggested a novel ASP based modeling approach to configuration support of a Linux package management system.

This work uses the following language constructs of Potassco (similar constructs are available in DLV):

- constant: lower-case string or number
- variable: upper-case string or _
- predicate: $predicatename(A_1, \ldots, A_n)$ with each A_i being a constant or variable
- condition: $P : C$ (with P and C being predicates) generating a set of ground instances for P corresponding to the existence of ground instances of C
- (counting) aggregate: $L\{A_1, \ldots, A_n\}U$ (with L being a lower bound, U an upper bound, and each A_i a predicate possibly generated by a condition) stating that the number of ground instances A_i shall be within the bounds
- fact: A_0. with A_0 being a predicate
- rule: A_0:-$L_1, \ldots, L_n$. with A_0 and L_i being predicates or aggregates, L_i possibly negated
- constraint: :-$L_1, \ldots, L_n$. with L_i being predicates or aggregates, possibly negated

3 Test case generation

Figure 1 shows the main use-case of our approach. To generate test cases for a specific constraint, one identifies the fragment of the object model relevant for the constraint. Using a generic OO-ASP mapping, described in the next section, this fragment is translated into an ASP program capable of enumerating all (up to a given upper bound) instantiations of the object model i.e. all possible configurations.

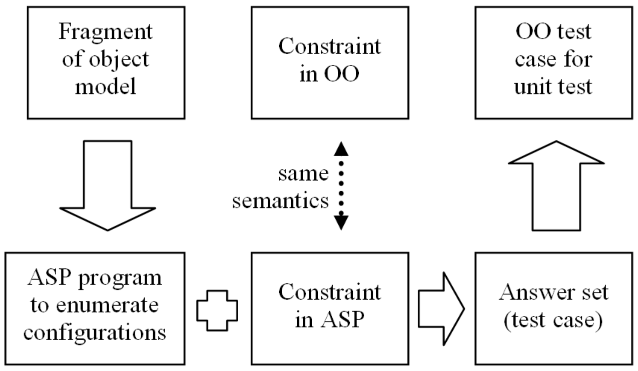


Figure 1. Generation of test cases

Since the main purpose of our approach is to detect conceptual errors, the tester has to reimplement the constraint to be tested in ASP, based on the requirements describing the constraint. Although possible, one cannot automatically translate the constraint from the OO configurator to ASP because an automatic translation would also translate the errors in the constraint. For the same reason the tester should be unaware of the implementation of the constraint in the object-oriented configurator. This process implements a black-box testing strategy like in traditional software engineering.

The generated ASP code and the ASP definitions for the constraint are used to compute answer sets that represent positive and negative test cases. These answer sets are then translated back into an object-oriented configuration and used in unit tests for the constraint.

3.1 OO-ASP Mapping

To illustrate the approach, a simple example domain for configuring bicycles (Figure 2) is used. A bicycle has a frame, two wheels and optional lights. A possible configuration can contain multiple bikes of different types, wheel sizes, etc. A valid configuration consists of a collection of correctly configured bicycles as defined by the allowed domains of the attributes (e.g. type), the given cardinalities of the associations (e.g. 0..1 for the lights), and two explicit constraints:

- constraintWheelsize disallows wheels of different sizes.
- constraintLights is complexer and requires that city bikes have lights, that racing bikes do not have lights, that mountain bikes may only have battery lights, and that the Boolean attribute hasLights must correspond to the existence of a Lights instance.

The object model is mapped to ASP according to the following schema:

- Every class C is mapped to two unary predicates <aspnameC>(X) and <aspnameC>Domain(X). The domain predicates are needed to describe the possible instances of a

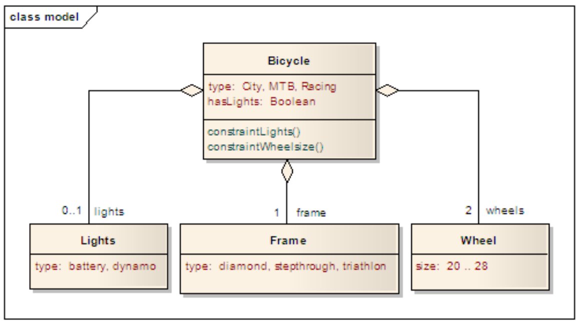


Figure 2. UML-diagram for bikeshop example

class - similar to variables to be activated in conditional constraint satisfaction problems. The maximal number of instances is defined manually via predicate <aspnameC>MaxInstances(X). Instances are identified by integers values.

- Every attribute ATTR of class C is mapped to a binary predicate <aspnameATTR>(X,Y) where X is an integer representing an instance of class X and Y is a possible value of attribute ATTR.
- Every association ASSOC between class C1 and C2 is mapped to a binary predicate <aspnameASSOC>(X,Y), where X and Y are integers representing instances of class C1 and C2.

The mapping is controlled by an XML file. It can be used to ignore irrelevant information, e.g. the attribute type of the frame. The following excerpt shows those parts of the mapping which are needed for constraintWheelsize.

```
<classmapping>
    <javaname>bikeshop.kb.Bicycle</javaname>
    <aspname>bicycle</aspname>
    ...
    <assocmapping>
        <javaname>wheels</javaname>
        <javaotherclass>bikeshop.kb.Wheel
        </javaotherclass>
        <aspname>bicycle2wheel</aspname>
    </assocmapping>
</classmapping>
<classmapping>
    <javaname>bikeshop.kb.Wheel</javaname>
    <aspname>wheel</aspname>
    <attrmapping>
        <javaname>size</javaname>
        <aspname>wheelSize</aspname>
    </attrmapping>
</classmapping>
```

By this mapping the Java class Bicycle is mapped to the unary predicate bicycle, class Wheel to predicate wheel, its attribute size to the binary predicate wheelSize, and the association between Bicycle and Wheel to the binary predicate bicycle2wheel.

Example of generated facts (for the listed part of the mapping):

```
bicycleMaxInstances(1).
bicycleDomain(1).
wheelMaxInstances(3).
wheelDomain(201).
wheelDomain(202).
wheelDomain(203).
```

Examples of user-defined maximum of instances and of facts generated by the solver as part of a test case like in Figure 5:

```
bicycle(1).
bicycle2wheel(1,201).
bicycle2wheel(1,202).
wheel(201).
wheelSize(201,24).
wheel(202).
wheelSize(202,25).
```

In order to be able to enumerate every possible configuration, the following additional ASP code is generated:

1. For every class C, the instances up to the given maximal number are generated by:

   ```
   0{<aspnameC>(X):<aspnameC>Domain(X)}MAX :-
     <aspnameC>MaxInstances(MAX).
   ```

   Example:

   ```
   0{bicycle(X):bicycleDomain(X)}MAX :-
     bicycleMaxInstances(MAX).
   ```

2. For every attribute ATTR of class C and possible values V1..Vn, one rule is needed to ensure exactly one value:

   ```
   1{<aspnameATTR>(X,V1),...,
     <aspnameATTR>(X,Vn)}1 :- <aspnameC>(X).
   ```

   Example:

   ```
   1{wheelSize(X,20),...,wheelSize(X,28)}1 :-
                                    wheel(X).
   ```

3. For every association ASSOC between C1 and C2 and cardinality restrictions L..U, a rule is generated for the lower bound:

   ```
   <L>{<aspnameASSOC>(X,Y):
       <aspnameC2>Domain(Y)} :- <aspnameC1>(X).
   ```

   The upper bound of the association is checked with a constraint:

   ```
   :- <aspnameC1>(X), U+1{<aspnameASSOC>(X,Y):
                        <aspnameC2>Domain(Y)}.
   ```

   Example (upper bound = lower bound = 2):

   ```
   2{bicycle2wheel(X,Y):wheelDomain(Y)} :-
                                 bicycle(X).
   :- bicycle(X),
      3 {bicycle2wheel(X,Y):wheelDomain(Y)}.
   ```

4. Especially for big domains, some basic symmetry breaking constraints are required to avoid explosion of the number of generated test cases. Since the instances of a class are interchangeable we disallow usage of instances with a higher ID unless all instances with a lower ID are used as well:

   ```
   :- <aspnameC>Domain(X), <aspnameC>Domain(Y),
      X<Y, <aspnameC>(Y), not <aspnameC>(X).
   ```

Example:

```
:- wheelDomain(W1), wheelDomain(W2),
   W1<W2, wheel(W2), not wheel(W1).
```

With this mapping it is possible to enumerate all configurations up to the given upper bound of the number of instances (preferring instances with a lower ID). The mapping is also used to translate an answer set back into a configuration of the object-oriented configurator. E.g. for the term bicycle(1) an instance of class Bicycle is created, for bicycle2wheel(1,201) the objects for bicycle with id 1 and the wheel with id 201 are associated, etc.

If the generated program does not have an answer set (unsatisfiable) then the object model itself is inconsistent. The UML class diagram in Figure 3 shows an example of an inconsistent object model.

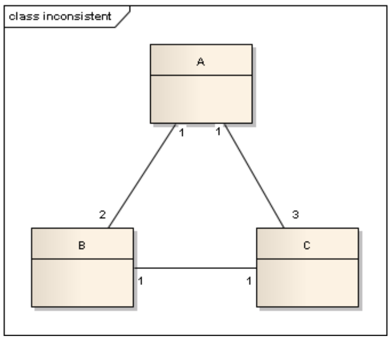


Figure 3. UML-diagram for inconsistent model

For every instance of class A, two instances of B and three instances of C must exist. Since there is a 1-1 association between B and C this class diagram is inconsistent. In this case the testing system reproduces the functionality of an earlier method [8] that uses integer programming for automatic detection of inconsistencies in UML class diagrams.

3.2 Test cases for constraints

To test a constraint, the tester needs to implement the constraint in ASP using the predicates of the generic mapping. With the generated program code of the preceding section and the manually written constraint, an ASP solver can find answer sets which satisfy the constraint or violate the constraint (counterexamples). By that, we get a set of test cases (represented as partial configurations) for each constraint. This approach is similar to the one supported by Alloy ([14]).

To avoid making the same conceptional errors as the implementer of the OO constraint, the tester should be unaware of the OO constraint code when writing the constraint. The implementer of the ASP constraint is only given a verbal description of the constraint or the requirement that should be checked by the constraint.

As an example, take the constraint that the wheels of a bicycle must have the same size (constraintWheelsize in Section 3.1). Following the convention that ASP constraints specify what is *not* a valid configuration, the tester expresses this with the following ASP code:

```
constraintWheelsize :-
  bicycle(X),
  bicycle2wheel(X,W1),
  bicycle2wheel(X,W2),
  W1!=W2,
  wheelSize(W1,S1),
  wheelSize(W2,S2),
  S1!=S2.
% find positive test case
:- testpositive, constraintWheelsize.
% find negative test case (counterexample)
:- testnegative, not constraintWheelsize.
```

The two atoms *testpositive* and *testnegative* control whether the solver finds positive or negative test cases for the tested constraint. In a positive test case the constraint is satisfied, in a negative one it is violated.

Figure 4 shows a positive test case found by the ASP solver running the program for constraintWheelsize. In this automatically generated graphical representation, rectangles represent instances, ellipses represent values, and the edges are labeled by the predicates between the nodes.

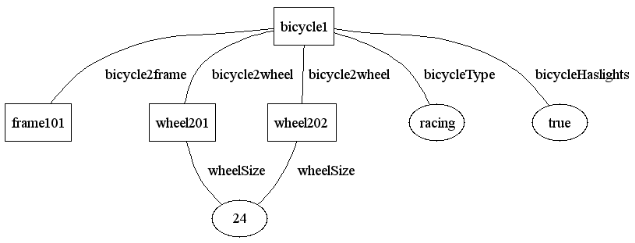


Figure 4. Positive test case

Running the same program with the fact *testnegative* produces the negative test case in Figure 5, i.e. a counterexample for the constraint.

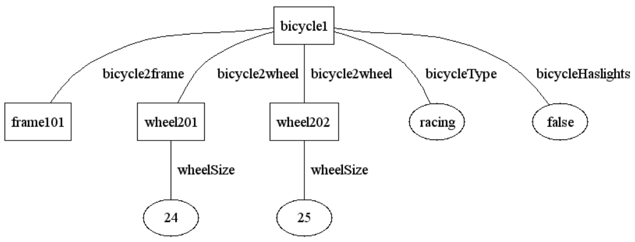


Figure 5. Negative test case

Each answer set represents one test case and can be translated into a partial configuration for the object-oriented configurator. All positive and negative test cases can be used for unit testing the constraint.

Note that the generated partial configurations for positive test cases might violate other constraints of the domain. For instance, constraintLights requires that the attribute bicycleHaslights is true, iff the bicycle has lights. This constraint is violated in the positive test case of Figure 4.

3.3 Unit testing

The whole process of test case generation can easily be integrated into unit testing. The test cases are added to the unit test suite of the configurator and used for regression testing. The following code sequence shows the unit test for constraintWheelsize which first runs all positive test cases and then all negative ones. The function $generateTestcases$ executes the ASP solver as described in the preceding section and returns a list of answer sets. The function $createConfigurationFor$ creates the partial configuration for an answer set which can be accessed by getter methods such as $getBicyles$.

```
public void testConstraintWheelsize() {
  List<Set<String>> tcs;
  tcs = generateTestcases("testpositive");
  for (Set<String> answerSet : tcs) {
    createConfigurationFor(answerSet);
    Bicycle bike = getBicycles().get(0);
    IConstraint c =
    bike.getConstraint(CONSTRAINTWHEELSIZE);
    assertEquals(Boolean.TRUE,c.getVal());
  }
  tcs = generateTestcases("testnegative");
  for (Set<String> answerSet : tcs) {
    createConfigurationFor(answerSet);
    Bicycle bike = getBicycles().get(0);
    IConstraint c =
    bike.getConstraint(CONSTRAINTWHEELSIZE);
    assertEquals(Boolean.FALSE,c.getVal());
  }
}
```

If an assert fails (i.e. a test case reports a discrepancy) then the reason for it has still to be found. For example, consider the following faulty Java implementation of the constraint. Since it returns true for the counterexample, we know there is a discrepancy between the ASP and the OO implementation of the constraint. Looking at the Java code below it is easy to identify the error. Due to a typing error, w2 is never referenced.

```
// in class Bicycle
public boolean constraintWheelsize() {
  List wheels = getWheels();
  if (wheels.size()!=2) { return false; }
  Wheel w1 = wheels().get(0);
  Wheel w2 = wheels().get(1);
  return w1.getSize()==w1.getSize();
}
```

In many cases, comparing the two implementations (i.e. static analysis) is sufficient for identifying an error. If two constraint implementations use different parts of the model this is an indication of an error. For instance, if one constraint depends on an attribute value and the other does not then there is a high chance that the first is more specific than the other.

Note that an OO implementation of the constraint in the configurator is not needed to generate test cases with our approach. Therefore, this method can also be used for the test-first approach of Test Driven Development [2].

4 Improving the test cases

Uninformed test case generation as described in the last section creates many possible configurations. Usually, this leads to a good test coverage. However, the number of test cases gets too large for practical use, especially for large-scale configuration.

Therefore, a method is needed to choose test cases which are likely to detect errors in the implementation. To generate test cases with specific properties, the tester can add statements describing those properties to the ASP implementation. For instance, adding

```
1 { wheelSize(X,Y):Y=24..25 } 1 :- wheel(X).
```

will only generate test cases where the wheelSize is 24 or 25. Specifying all relevant test cases manually this way is a tedious task. The alternative is to use advanced filtering techniques like symmetry breaking.

4.1 Symmetry breaking

For black-box testing in software engineering, techniques such as equivalence partitioning and boundary value analysis have been developed to reduce the number of test cases [16]. These techniques define equivalence classes for the input data and test only one value from every equivalence class.

For constraint-based systems a similar effect can be achieved by defining equivalence classes over the possible configurations by using symmetry breaking techniques. For instance, in the positive test cases for constraintWheelsize, the actual value of the wheel size is irrelevant as long as the values are all the same (assuming a reasonable implementation). For negative test cases, at least two different values are needed, but it does not matter which values are actually chosen.

Detection of the equivalence classes for an ASP program is done by reducing it to the colored graph automorphism problem [6]. In this case, the grounded program is represented as a colored graph. The symmetry breaking tool is searching for such transformations of the graph (permutation) that map vertices of it to vertices of the same color. The coloring schema in [6] allows to identify permutations of graph vertices corresponding to equivalent grounded atoms, e.g. wheels of different sizes, in a program. The permutations are used by the preprocessor SBASS[5] [6] to generate symmetry breaking constraints that introduce a lexicographic order on elements of a solution space. The symmetry breaking constraints are added to the grounded program and the result is forwarded to the solver.

Roughly speaking, in the case of wheel sizes, constraints will require to use wheels of size 20 first, since 20 is lexicographically the smallest value. Only if it is impossible to find a configuration with wheels of the size 20, the solver will try the size 21 and so on.

Inclusion of the preprocessing step in the testing tool chain reduces the number of possible configurations for the bicycle example (without coding the two constraints in ASP) from 1459 to 129. For the test case generation example for constraintWheelsize as described in Section 3.2, execution of SBASS reduces the number of positive test cases from 163 to 13. Although the number of test cases can be reduced drastically by symmetry breaking, one still has to ensure that the coverage of the created test cases is enough to find potential errors.

For instance, consider the case where the knowledge base is modified by allowing bicycles to have more than two wheels (i.e. tricycles, etc). The following faulty constraint implementation works, if

[5] http://potassco.sourceforge.net/labs.html

the differences in wheel sizes always occur in the first two wheels. If symmetry breaking creates only such test configurations, one can no longer detect the error in the constraint implementation.

```
// faulty implementation, but works
// if the "first" 2 wheels are of same size
public boolean constraintWheelsize() {
  List<Wheel> wheels = getWheels();
  for(int i = 0 ; i<wheels.size() ; i++) {
    for(int j=i+1 ; j<wheels.size() ; j++) {
      if (wheels.get(i).getSize().equals(
          wheels.get(j).getSize())) {
        return true;
      }
    }
  }
  return false;
}
```

4.2 White box testing

The kind of errors in the implementation often depend on the programming language used. A knowledge engineer using a specific Java framework will make different errors than a knowledge engineer using ASP. Therefore, the generation of test cases cannot be fully automated without additional information about likely errors.

By looking at the code (white box testing) an experienced developer can identify suspicious parts of the code which should be tested. From that, she can derive which properties a test case must have and can create such test cases manually. For automated test case generation, it is possible to generate test cases with specific properties by adding additional constraints to the ASP program similar to the example at the beginning of Section 4.

4.3 Maintaining test configurations

By combining all the constraints of the domain, the described approach can also be used to generate complete and valid test configurations for the object-oriented configurator. The limiting factor here is that the performance of the generic mapping and the fact that all constraints of the configurator must be reimplemented in ASP.

Complete test configurations are often used for integration testing, system tests, etc. A common problem is how to maintain the consistency of the test configurations in case of knowledge base evolution. Whenever the requirement of a constraint changes, one needs to reconcile those changes with existing legacy test configurations.

Since many tests may depend on the existing test configuration, the changes in the legacy test configuration should be minimal. An ASP method for finding reconfigurations with minimal costs is suggested in [10].

5 Conclusions

We described how to map the object model and configurations of an object-oriented and constraint-based configurator to and from ASP. One application of this mapping is the generation of test cases for the OO configurator. Since the mapping is symmetric it could also be used to generate test cases for an ASP-based configurator.

The generation of test cases so far has been tried for toy examples like the bikeshop domain and some small fragments of real world domains. For the future we plan to evaluate translation of existing knowledge bases of our real-world configurators (>100 classes) into ASP. We expect that we have to refine the techniques of Section 4 in order to get sufficient performance.

The current approach cannot be used for test cases containing a lot of components. For instance, the bikeshop domain already uses more than 1GB of memory if the domain size is set to more than 50. Fortunately, test cases for single constraints usually do not involve hundreds of components.

REFERENCES

[1] Algirdas A. Avizienis, *Software Fault Tolerance*, volume 2, chapter "The Methodology of N-Version Programming", 22–45, John Wiley & Sons, 1995.

[2] Beck, *Test Driven Development: By Example*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[3] Kent Beck. JUnit, 2010.

[4] Kent Beck and Cynthia Andres, *Extreme Programming Explained: Embrace Change (2nd Edition)*, Addison-Wesley Professional, 2004.

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

[6] Christian Drescher, Oana Tifrea, and Toby Walsh, 'Symmetry-breaking answer set solving', *AI Commun.*, **24**(2), 177–194, (2011).

[7] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner, 'Answer set programming: A primer', in *Reasoning Web*, pp. 40–110, (2009).

[8] Andreas Falkner, Ingo Feinerer, Gernot Salzer, and Gottfried Schenner, 'Solving practical configuration problems using UML', in *Proceedings of ECAI 2008 Workshop on Configuration Systems*, pp. 1–6, (2008).

[9] Onofrio Febbraro, Nicola Leone, Kristian Reale, and Francesco Ricca, 'Unit testing in aspide', *CoRR*, (2011).

[10] Gerhard Friedrich, Anna Ryabokon, Andreas A. Falkner, Alois Haselböck, Gottfried Schenner, and Herwig Schreiner, '(Re)configuration based on model generation', in *LoCoCo*, eds., Conrad Drescher, Ins Lynce, and Ralf Treinen, volume 65 of *EPTCS*, pp. 26–35, (2011).

[11] Martin Gebser, Roland Kaminski, and Torsten Schaub, 'aspcud: A linux package configuration tool based on answer set programming', in *LoCoCo*, pp. 12–25, (2011).

[12] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub, 'Conflict-driven answer set solving: From theory to practice', *Artif. Intell.*, **187**, 52–89, (2012).

[13] Michael Gelfond and Vladimir Lifschitz, 'The stable model semantics for logic programming', in *5th International Conference and Symposium on Logic Programming*, pp. 1070–1080, (1988).

[14] Daniel Jackson, 'Alloy: A logical modelling language', in *ZB*, p. 1, (2003).

[15] Nikola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello, 'The DLV system for knowledge representation and reasoning', *ACM Transactions on Computational Logic (TOCL)*, **7**(3), 499–562, (2006).

[16] Glenford J. Myers and Corey Sandler, *The Art of Software Testing*, John Wiley & Sons, 2004.

[17] James Rumbaugh, Ivar Jacobson, and Grady Booch, 'The unified modeling language reference manual', in *The unified modeling language reference manual*, (2005).

[18] Timo Soininen, Ilkka Niemelä, Juha Tiihonen, and Reijo Sulonen, 'Representing configuration knowledge with weight constraint rules', in *1st International Workshop on Answer Set Programming: Towards Efficient and Scalable Knowledge*, pp. 195–201, (2001).

Towards Hybrid Techniques for Efficient Declarative Configuration

Ingo Feinerer[1]

Abstract. During the last decades configuration has been extensively employed in a wide range of application domains, implemented by a multitude of techniques like logics, procedural, object-oriented, or resourced-driven approaches. Especially declarative methods provide the foundation for precise and well-understood semantics for reasoning tasks and allow for a succinct representation of the underlying knowledge base. However, a drawback in using such powerful declarative techniques lies in their computational complexity. In this paper we present a simple declarative framework for configuration in Prolog in order to show the advantages of logic-based techniques but also to identify some challenges for such formalisms. We argue for hybrid systems which combine and utilize efficient techniques from different configuration methodologies under a unified declarative interface.

1 INTRODUCTION

In the history of configuration research multiple high-level approaches for the design and solution of configuration tasks have been presented. Declarative systems have had strong proponents due to their expressive semantics based on well investigated and understood logics. Unfortunately the expressive power limits the applicability due to high computational costs, observable for a broad class of implementations [3]. Related topics like reconfiguration unveil further interesting but often computational hard topics [6, 2] for declarative formalisms. In this paper we identify some typical challenges for declarative frameworks and propose a combined hybrid approach which utilizes techniques from other fields, like integer linear programming (ILP), to overcome these. Section 2 presents a simple declarative framework in order to demonstrate the benefits but also the challenges (Section 3) of such a system. Section 4 shows three strategies to deal with the discussed challenges.

2 DECLARATIVE FORMULATION

Object-oriented modeling languages provide a natural formalism for the design of configuration systems. Historically entity-relationship diagrams had a strong toehold but during the recent decade class diagrams in the Unified Modeling Language [8] (UML) have seen a substantial growth as a domain-specific language for configuration. [5, 1] Consequently we use UML class diagrams as our starting point and propose a declarative formulation translating its key features into a declarative framework. We chose Prolog for this task as it has an established and well-known semantics and provides several efficient implementations. The main ideas can be easily transfered to other declarative formalisms (e.g. logics or answer-set programming) as well. The following framework was implemented and tested with SWI-Prolog but should work equally well in any other dialect with minor modifications.

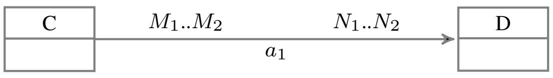


Figure 1: Specification with two classes, an association with multiplicities, and the OCL constraint `context C inv: C.allInstances()->size() >= L`.

Figure 1 depicts a minimal UML class diagrams as typically used for configuration purposes. It shows a specification with two classes C and D and one association a_1 relating them. The multiplicities restrict the number of valid links; each object of class C must be connected with at least N_1 and with at most N_2 objects of class D. Analogously, each object of class D must have links to $M_1..M_2$ objects of class C. The small arrow at the end of the association shows the direction (and not a hierarchy). The constraint in the Object Constraint Language [9] (OCL) for class C states that C must be instantiated with at least L objects to obtain a valid configuration.

The concepts of a class and of an association can be directly translated to

```
class(CN-L) :- atom(CN), L >= 0.

assoc(CNs, AN-(C,I)>>(D,J)) :-
  atom(AN), member(C, CNs), member(D, CNs),
  mult(I), mult(J).
```

where each class has a name (`CN`) and a lower bound (`L`). Each association has a name (`AN`) and includes information on the related classes (`C`, `D`) and the multiplicities (`I`, `J`). Further the class names are checked for validity against a predefined list of names (`CNs`).

Now we can define a specification as a tuple `(Cs, As)` of classes `Cs` and associations `As`

```
spec((Cs, As)) :-
  maplist(class, Cs), pairs_keys(Cs, CNs),
  maplist(assoc(CNs), As).
```

where `maplist()` applies a predicate to all arguments of a list and `pairs_keys()` provides access to the subterms of Pair-Key structures.

E.g., instantiating the multiplicities of a_1 in Figure 1 with $M_1 = 1$, $M_2 = 2$, $N_1 = 3$, and $N_2 = 4$ and enforcing a lower bound L of 1 for the number of objects of class C this yields

```
:- spec(([c-1,d-0], [a1-(c,1..2)>>(d,3..4)]))
```

[1] Vienna University of Technology, Austria, email: Ingo.Feinerer@tuwien.ac.at

A configuration can be modeled as a tuple `(Os, Ls)` consisting of objects `Os` and of links `Ls`. Each object is identified by its name `O` and its corresponding class `C`. Each link has a name `L` and stores information on the corresponding association `A` and on the objects `[O,P]` it consists of.

```
is_object(CNs, O-C) :-
  atom(O), member(C, CNs).

is_link(As, Os, L-(A,[O,P])) :-
  atom(L), member(A-(C,_)>>(D,_), As),
  member(O-C, Os), member(P-D, Os).
```

We call a configuration an *instance* of a specification if all the objects and links have a corresponding class and association, respectively.

```
instance((Os, Ls), (Cs, As)) :-
  pairs_keys(Cs, CNs),
  maplist(is_object(CNs), Os),
  maplist(is_link(As, Os), Ls).
```

For each object we check its class and for each link we identify a matching association with compatible participating classes.

In order to handle multiplicities as defined by the UML standard, we define a predicate `gamma()` which counts for each association and a given object the number of different partner objects induced by the links of a configuration [4]

```
gamma(I, P, A, N, Ls) :-
  reduct(I, Os, P),
  findall(Os, member(_-(A,Os), Ls), Bag),
  list_to_set(Bag, Set), length(Set, N).
```

where `I` denotes the "position" (index) of the partner objects to be counted for within the given binary or multiary association `A`, the list `P` contains the single object to be fixed (in general this could be any partial link), and `Ls` is a list of links to be considered. The result (count) is unified with `N`. The `reduct()` predicate generates a list of objects (to be used as partial links) such that `P` is the projection on all but the `I`-th component, and `findall()` collects all such objects forming a link in `Ls`. For example

```
:- gamma(2, [c1], a1, 3,
         [l1-(a1, [c1, d1]), l2-(a1, [c1, d2]),
          l3-(a1, [c1, d3])]).
```

fixes a single object c_1 (of class C) and counts how many different objects (of class D corresponding to index 2) can be reached over association a_1 when considering the provided links: three.

Now we have all parts to define the notion of a valid configuration. We say a configuration *satisfies* a specification if it is an instance and both the lower bounds for each class and the multiplicities of each association are respected.

```
satisfies((Os, Ls), (Cs, As)) :-
  forall(member(C-LB, Cs),
         (findall(ON, member(ON-C, Os), ONs),
          length(ONs, N), N >= LB)),
  forall((member(O-C, Os),
    (member(AN-(C,_)>>(_,L..U),As),I=2
    ;member(AN-(_,L..U)>>(C,_),As),I=1)),
         (gamma(I, [O], AN, N, Ls),
          between(L, U, N))).
```

The predicate `satisfies()` checks whether `(Os, Ls)` is a valid configuration for the specification `(Cs, As)`. The first `forall()` checks the number of objects for each class against the corresponding lower bound whereas the second `forall()` checks that each object `O` of the configuration is linked to the right number of partner objects as constrained by all participating associations and their multiplicities. E.g.

```
:- satisfies(([c1-c, d1-d, d2-d, d3-d],
    [l1-(a1,[c1,d1]),l2-(a1,[c1,d2]),
     l1-(a1,[c1,d3])]),
   ([c-1,d-0], [a1-(c,1..2)>>(d,3..4)])).
```

Although simplistic and minimal this framework allow us to model reasonable configurations and will serve the purpose of presenting the advantages but also challenges for such a declarative framework. It can be easily extended to cover multiple aspects which are necessary for a more complete handling of configurations in real-world applications (and in fact many features are already implemented in a prototype).

A central advantage we observe in the design phase is the clear and straightforward formulation of the underlying terminology. Specifications, configurations, and the notions of instance, validity and satisfiability can be defined with just a few lines of code in parallel to the formal definitions of the underlying concepts. Prototyping is rapid and succinct; the visualization of graph structures and configurations is straightforward. For the computation of configurations one of the main advantages of this declarative formulation is the semi-automatic search for solutions. Prolog provides efficient algorithms to explore and backtrack within the search space; further it has optimizations for tail recursion. E.g., with trivial strategies for object and link creation

```
:- gen_objects([c-2, d-3],
  [c_0-c, c_1-c, d_0-d, d_1-d, d_2-d]).

:- gen_links([a1-1, a2-2],
  [a1_0-(a1,_), a2_0-(a2,_), a2_1-(a2,_)]).
```

we can generate configurations on the fly (we implement the predicate `gen_instance()` for this purpose) and check if they satisfy a given specification

```
gen_model(C, S) :-
  gen_instance(C, S), satisfies(C, S).
```

For a small number of classes and associations this strategy works very well, is intuitive, and allow us to explore the whole search space without extra programming.

3 CHALLENGES

For a declarative framework as presented in the previous section challenges typically arise when the search space is large and/or backtracking is expensive. The former is not a drawback of declarative formulations per se; other formalisms which need to deal with problem instances with a large solution space will suffer from the same performance penalties. However, when backtracking is expensive, i.e., when it takes some computational effort to find out that the current unification state will never lead to a valid configuration, there is often room for improvement. First of all, it depends whether the computational complexity is inherent to the problem or just induced by the use of expressive logics or other declarative formalisms. Second, there exists a plethora of methods and algorithms in computer science which are tailored to specific problem classes.

3.1 Equations over association chains

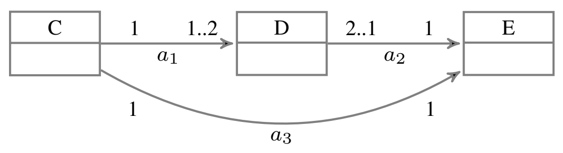
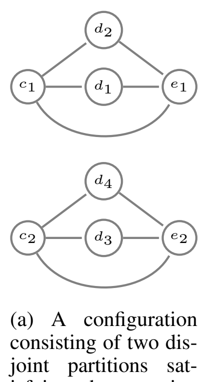


Figure 2: Specification with three classes and the OCL constraint `context C inv: C.allInstances()->size() >= 2.`

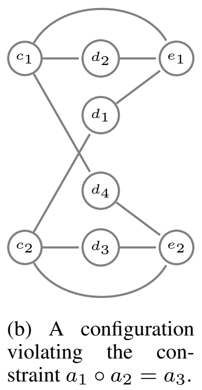
Consider the specification depicted in Figure 2. There are three classes C, D, and E, where C has a lower bound of two on the number of instantiated objects. Each object of class C should be connected via association a_1 to one or two objects of class D which in turn are uniquely associated with an object of class E via a_2. Finally association a_3 enforces a one-to-one relationship between objects of class C and objects of class E.

This poses no severe problem for our declarative implementation yet. We can find a valid configuration by stepwise incrementing the number of instantiated objects and the number of desired links and the `satisfies()` predicate will filter out all invalid combinations. However, the situation gets interesting if we add some constraints; the specification in Figure 2 probably tells not the full story of the intended semantics. We might want to ensure that each object of class E which can be reached from an object of class C over intermediary objects of class D via associations a_1 and a_2 is in fact the same as linked by a_3. This models the concept of a modular compound unit which consists of a set of interconnected subcomponents.

We can formalize such constraints by introducing equations on association chains. They restrict the links of valid configurations by imposing limits on valid objects involved. Association chains can be modeled by classical tuple composition, similar to a join in database theory. E.g., for Figure 2 we would add the equation $a_1 \circ a_2 = a_3$ to achieve the semantics mentioned before.



(a) A configuration consisting of two disjoint partitions satisfying the equation $a_1 \circ a_2 = a_3$.



(b) A configuration violating the constraint $a_1 \circ a_2 = a_3$.

Figure 3: Two satisfying configurations for the specification in Figure 2, but only the left one adheres to the constraint $a_1 \circ a_2 = a_3$.

Figures 3a and 3b both show valid configurations for the specification 2 before adding the new constraint on the association chain. Once we enforce the constraint $a_1 \circ a_2 = a_3$ only Figure 3a remains a satisfying instance; Figure 3b is not valid anymore as the configuration violates the equation since there is a connection from c_1 over d_4 to e_2 which is not reachable via a link of association a_3.

This subtle change makes a significant difference in runtime and backtrack behavior for our declarative formulation. Although the number of objects and links to be considered is still rather small (8 and 10, respectively) the number of combinations to be explored before the equation can be checked is exponential. Backtracking is therefore very expensive as basically a full configuration needs to be built before the equation can be checked. We see a major increase in the runtime (and stopped measuring after 3600 seconds).

3.2 Partial configurations

We would like to use our framework not only for configuration but also for reconfiguration in order to repair existing configurations. A main challenge is a fast way to identify parts of an input configuration which cannot be taken as a subcomponent of the overall solution. This is important to avoid late backtracking where a lot of computation time has been used for building a variety of alternations which can never lead to a valid configuration. Clearly, for arbitrary complex input configurations this problem is NP-hard, however for many basic tasks it is not (e.g., checking whether some links will violate certain sets of constraints later on).

3.3 Cost functions

Another limiting factor of logic-based formalisms is finite domain reasoning, especially notable when working with numbers in an integer domain. Configuration tasks often need ways to express how expensive certain components or connections are. A natural implementation is to use cost functions which assign costs or other integer numbers to individual objects in a configuration. The aim is now to minimize an objective function which takes into account all components and their corresponding costs.

4 TOWARDS A HYBRID APPROACH

In the previous section we saw some typical challenges for declarative frameworks. These are by far not the only ones but give a sample of relevant problems of various kinds. Fortunately, there has been tremendous progress in configuration research and artificial intelligence in general which triggered specialized algorithms and strategies for specific problems. In order to tackle the outlined challenges we propose to use specialized algorithms from other formalisms and to include them in a unified declarative framework. Such a hybrid approach, taking the best from multiple worlds and combining them in a consolidated interface, allows us to attack the previously described challenges towards efficient declarative configuration.

4.1 Integer linear programming

We start out with the observation that backtracking can be very expensive if the domain of variables is not restricted or bounded to a specific range. Ideally we would like to compute the exact number of needed components for a configuration and then our declarative framework can concentrate on finding appropriate links to connect the parts. Clearly, this approach cannot always work as some special constraints on the interconnection of components may enforce additional objects which are not required by pure associations (and their multiplicities) in the underlying UML class diagram.

For the computation of the needed number of objects for each class we use a highly efficient technique based on integer linear programming. [7, 4] The idea is to translate a UML class diagram to a system of inequalities. Its solution indicates whether an instantiation into a valid configuration is possible at all (satisfiability problem) but more interestingly in our context gives also the number of objects for each component. E.g., for the specification in Figure 1 with association a_1 and the lower bound on C we obtain

$$M_1 \cdot |D| \leq N_2 \cdot |C| \qquad |C| \geq M_1 \qquad |C| \geq L$$
$$N_1 \cdot |C| \leq M_2 \cdot |D| \qquad |D| \geq N_1$$

expressing constraints enforced by the multiplicities (left), constraints on the minimal number of objects due to the association semantics (middle), and the lower bound constraint on C (right). This translation is performed for all classes and associations involved in a specification forming an integer linear program. The objective function is typically just the minimum over all classes involved.

SWI-Prolog provides support for integer linear programming via its extension library "simplex" shipped with the standard installation. We added a set of DCG (definite clause grammar) rules which generate ILP constraints out of each association:

```
assoc_constraint(
    _AN-(C,M1..M2)>>(D,N1..N2)) -->
  constraint([M1*D, -N2*C] =< 0),
  constraint([N1*C, -M2*D] =< 0).
```

Further constraints like lower bounds or the objective function are added separately forming the whole ILP program.

The native integration of ILP into our declarative framework allows us to rule out a broad range of possibilities which do not need to be explored any more. This has a significant effect on backtracking as it cuts the search space into more fine grained areas. With this technique we can scale our framework to a greater number of objects and links when dealing with the challenges "equations on association chains" and "partial configurations". Costs or weights are an integral part of ILP and can thus easily be handled with this approach; this addresses several aspects of the challenge "cost functions".

4.2 Procedural link generation

So far we have only identified one possible way to efficiently compute the number of necessary components. Still, it might take a long time to find valid links between these objects (as motivated in the challenge on association chains). Therefore it seems a promising approach to optimize the way links are generated. One strategy we found useful for certain classes of configurations is to "balance" links between objects as far as possible. This can be seen as a procedural implementation to form a uniform distribution among the links between the participating classes of a given association:

```
seq(J, M, N, [X, Y]) :-
  J >= 0, M > 0, N > 0,
  X is J mod M,
  lcm(M, N, LCM),
  Y is (J + floor(J/LCM)) mod N.
```

The idea of the `seq()` predicate is to generate a sequence (with `J` as the index into it) of tuples `[X,Y]` for `M` objects of the first class and `N` objects of the second class (for a binary association) which can be used as links and form a uniform distribution. E.g., for $J = 0, \ldots, 3$ and $M = 2$ and $N = 6$ we obtain the tuples $(0, 0)$, $(1, 1)$, $(0, 2)$, and $(1, 3)$. Consequently we would generate a link from object 0 of class C to object 0 of class D, from object 1 of class C to object 1 of class D, and so on.

Such a strategy is especially suited for finding an initial linking for a whole configuration. This can be both the basis for further investigation regarding the challenge "equations on association chains" or provide initial solutions for a reconfiguration problem as necessary for the challenge "partial configurations".

4.3 Generate and test

Our final suggestion towards hybrid systems is a meta-strategy. Both previous techniques also fall into this category as special cases. Declarative systems have a strong standing with their efficient and native backtracking as it is at the heart of their operation. This property is extremely useful for configuration as exploring the solution space is one of the fundamental aspects in a configuration task. The main advantage is that almost arbitrary constraints can be added with minimal changes to the declarative implementation. This allows us fast prototyping and complex constraint checking. We therefore argue in favor of such systems and the simple but effective strategy of "generate and test". The generation step is either done by more sophisticated algorithms implemented by the user in Prolog (as shown in the previous subsections) or by calls to external tools which are specialized on a specific tasks. The declarative framework can then take these parts and combine them into a configuration and test it for validity.

5 CONCLUSION

Declarative formalisms provide a natural environment for the implementation of configuration systems as it is easy to write succinct and precise programs with integrated support to explore the solution space with backtracking. We motivated this by a simple declarative framework but showed challenges arising from their use. We argue for hybrid systems which combine specialized techniques from other fields into a unified declarative interface.

REFERENCES

[1] Andreas Falkner, Ingo Feinerer, Gernot Salzer, and Gottfried Schenner, 'Computing product configurations via UML and integer linear programming', *Int. Journal of Mass Customisation*, **3**(4), 351–367, (2010).

[2] Andreas Falkner, Gerhard Friedrich, Alois Haselböck, Anna Ryabokon, Gottfried Schenner, and Herwig Schreiner, '(Re)configuration using answer set programming', in *IJCAI 2011 Configuration Workshop*, (2011).

[3] Andreas Falkner, Alois Haselböck, Gottfried Schenner, and Herwig Schreiner, 'Modeling and solving technical product configuration problems', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **25**, 115–129, (2011).

[4] Ingo Feinerer and Gernot Salzer, 'Consistency and minimality of UML class specifications with multiplicities and uniqueness constraints', in *Proceedings of the 1st IEEE/IFIP International Symposium on Theoretical Aspects of Software Engineering, June 6–8, 2007, Shanghai, China*, pp. 411–420. IEEE Computer Society Press, (2007).

[5] Alexander Felfernig, Gerhard Friedrich, and Dietmar Jannach, 'UML as domain specific language for the construction of knowledge-based configuration systems', *International Journal of Software Engineering and Knowledge Engineering*, **10**(4), 449–469, (2000).

[6] Gerhard Friedrich, Anna Ryabokon, Andreas A. Falkner, Alois Haselböck, Gottfried Schenner, and Herwig Schreiner, '(Re)configuration based on model generation', in *2nd LoCoCo Workshop*, volume 65 of *EPTCS*, pp. 26–35, (2011).

[7] Maurizio Lenzerini and Paolo Nobili, 'On the satisfiability of dependency constraints in entity-relationship schemata', *Information Systems*, **15**(4), 453–461, (1990).

[8] Object Management Group, *Unified Modeling Language 2.4.1*, 2011.

[9] Object Management Group, *Object Constraint Language 2.3.1*, 2012.

Unifying Software and Product Configuration: A Research Roadmap

Arnaud Hubaux[1] and **Dietmar Jannach**[2] and **Conrad Drescher**[3] and **Leonardo Murta**[4] and **Tomi Männistö**[5] and **Krzysztof Czarnecki**[6] and **Patrick Heymans**[7] and **Tien Nguyen**[8] and **Markus Zanker**[9]

Abstract.
For more than 30 years, knowledge-based product configuration systems have been successfully applied in many industrial domains. Correspondingly, a large number of advanced techniques and algorithms have been developed in academia and industry to support different aspects of configuration reasoning. While traditional research in the field focused on the configuration of physical artefacts, recognition of the business value of customizable software products led to the emergence of software product line engineering. Despite the significant overlap in research interests, the two fields mainly evolved in isolation. Only limited attempts were made at combining the approaches developed in the different fields. In this paper, we first aim to give an overview of commonalities and differences between software product line engineering and product configuration. We then identify opportunities for cross-fertilization between these fields and finally develop a research agenda to combine their respective techniques. Ultimately, this should lead to a unified configuration approach.

1 Introduction

Customizable products are an integral part of most B2B and B2C markets. Mass-customization strategies have been applied to tangible products (e.g., cars and mobile phones) as well as intangible products like software (e.g., operating systems and ERPs) and services (e.g., insurance). To this end, companies use software *configurators* that provide automated support to tailor products to the requirements of specific customers or market segments.

[1] PReCISE Research Center, University of Namur, Belgium, ahu@info.fundp.ac.be
[2] Department of Computer Science, TU Dortmund, Germany, dietmar.jannach@tu-dortmund.de
[3] Computing Laboratory, University of Oxford, UK, Conrad.Drescher@comlab.ox.ac.uk
[4] Fluminense Federal University, Niterói, Brazil, leomurta@ic.uff.br
[5] Aalto University School of Science, Finland, Tomi.Mannisto@aalto.fi
[6] Generative Software Development Lab, University of Waterloo, Canada, kczarnec@gsd.uwaterloo.ca
[7] PReCISE Research Center, University of Namur, Belgium, phe@info.fundp.ac.be
[8] Electrical and Computer Engineering Department, Iowa State University, USA, tien@iastate.edu
[9] Alpen-Adria-Universität Klagenfurt, Austria, markus.zanker@aau.at

Compared to the long history of computer-supported configuration of *products*, research on the configuration of parametrizable *software* is rather new. *Product configuration* (PC) is the umbrella activity of assembling and customizing physical artefacts (e.g. technical equipment, cars or muesli) or services. Historically, PC has been a subfield of *artificial intelligence* (AI), focusing on knowledge representation and reasoning techniques to support configuration. Mostly independent of PC, the field of *software product line engineering* (SPLE) emerged in the *software engineering* community. SPLE deals with the design and implementation of software components that can be adapted and parametrized according to customer requirements and business or technical constraints [47]. As in PC approaches, the goal is to save costs by assembling individualized systems from reusable components [41]. Typical application domains for SPLE include embedded systems, device drivers, and operating systems.

Interestingly, research in these two fields has been carried out so far mostly independently. Except in rare cases (e.g. [28, 5]), researchers in both fields are often unaware of approaches that have been developed in the other community. Further, even though a lot of PC work has focused on configuring technical equipment, such equipment increasingly contains software. At the same time, SPLE increasingly targets software-intensive systems that also include computing and other types of equipment. Based on these observations, our hypothesis is that both the PC and SPLE communities have produced results that are applicable in the other domain.

The remainder of this paper explores further the opportunity for cross-fertilization (Section 2) and proposes a research roadmap (Section 3) to systematically compare the two domains and foster efforts towards unifying both fields. In particular, we hope to find innovative approaches to questions that are largely open in one or the other community such as the reconfiguration of deployed systems, better interactive configuration support (e.g., in case of unsatisfiable requirements), methods for full lifecycle support and the evolution of models and knowledge bases. For this last question, we will also explore how techniques from software configuration management (CM) can be integrated.

2 Motivation

Questions of knowledge acquisition, knowledge representation as well as different types of reasoning support have been investigated for many years in PC and SPLE. We highlight some key results in both fields to show their commonalities and differences. These preliminary observations motivate our endeavour to study, compare, and eventually unify research on configuration. We split the introduction of our motivations along five dimensions. For each dimension, we also formulate the research questions whose answers should expose opportunities for cross-fertilization.

2.1 Knowledge acquisition and modelling

Research on PC has used a wide range of knowledge modeling approaches (based, e.g., on UML [22] or description logic [40]), involving different types of logics and constraints. While a few SPLE approaches also used UML to capture aspects of configuration knowledge (e.g. [60, 25]), most results build upon the seminal work on *feature-oriented domain analysis* (FODA) initiated in 1990 by Kang et al. [37], which today is converging with decision models [17]. The cornerstone of FODA are *feature models* (FMs), a graphical notation to capture and express the commonalities and variabilities of a product family. FMs are menu-like hierarchies of mandatory, optional, and alternative features, with cross-hierarchy relationships to express dependencies and incompatibilities. This initial FM notation has been gradually extended to support, for example, multiple instances [18, 44] or the configuration process [29].

Compared to configuration modelling ontologies used in PC (e.g., [22] or [53]), the expressiveness of FMs (even extended ones) appears too limited compared to more complex PC ontologies. Examples of advanced PC problems include connecting components via ports (i.e., inferring complex topologies), finding optimal or at least good configurations, integrating iteratively new components, and distributing knowledge over different agents or business entities [32]. Some work exists in SPLE on component connection and integration (e.g. [5]) and optimization (e.g. [58]). This motivated the creation of more expressive languages (e.g., [8]). PC appears to offer a richer body of work in this area, though.

Opportunities for cross-fertilization: Some authors have already acknowledged the bond between configuration in PC and in SPLE through feature-based configuration. Günter et al. [27] recognize concept hierarchies (similar to FMs) as a fundamental concept in their survey of knowledge-based configuration methods. According to Junker's classification of known configuration problems [34], feature-based configuration falls in the option selection or shopping list problems. To systematically identify such synergies, our research agenda should answer the following questions:

RQ1 *What classes of configuration problems exist?*
RQ2 *How are these problems modelled?*

2.2 Automated reasoning

Generally, with respect to modeling and knowledge representation, the AI-rooted PC community is usually interested in "executable" models that can be directly translated into a representation processable by a reasoning engine. The formal basis of most knowledge modelling languages lays the foundation for advanced configuration reasoning techniques (e.g., checking for consistency of configurations, completing partial configurations, or supporting interactive configuration processes). In contrast, the SPLE community only started recently to develop a formal foundation of FMs (e.g. [9, 49]) and their analyses (e.g. [5, 43, 33, 59]). Based on precise formal problem characterizations, additional automations for SPLE become feasible. An example is the automated analysis of FMs; see [10] for an overview. Furthermore, Benavides et al. [11] propose to translate FMs into a Constraint Satisfaction Problem (CSP) and apply Reiter's model-based diagnosis (MBD) approach to detect problems in the models. Xiong et al. [59] combine MBD and Satisfiability Modulo Theory (SMT) solvers to generate range fixes in software configuration. Mendonca et al. [43] report on experiments with a SAT-encoding of FMs. Finally, Bagheri et al. [7], support hard and soft requirements in the configuration process.

The SPLE community sometimes reinvents techniques which have been developed previously in PC. Encoding configuration problems in some logic or as CSPs has a long history in the AI community [34]. The PC community was also the first to apply SAT solvers to configuration problems [51]. New CSP representations such as Dynamic, Composite or Generative CSPs [45, 48, 54, 32] as well as logics [26, 4] were partially inspired by the challenges observed in PC. This latter pool of techniques addresses the problem of conditionally including multiple instances of a certain component type. The PC community was also first to use MBD for configuration, e.g., for detecting problems in configuration knowledge bases [23]. Regarding soft constraints and preferences, there is abundant literature in constraint programming (e.g. [36, 46]). Finally, binary decision diagrams (BDDs) have also been used for building fast interactive configurators (trading time vs space from a complexity point of view) [3]. That latter approach has been explored in SPLE as well (e.g. [42]), but it turned intractable on large FMs.

Opportunities for cross-fertilization: In contrast to physical components, software components are represented completely as computer artifacts. While physical components need to be specified explicitly in the computer to check cross-component compatibility, software configuration can analyze variability models and the actual configurable artifacts at the same time. This opens up new possibilities for configuration, where the compatibility of components can be checked on the fly during configuration without going back to the design phase and modifying configuration knowledge. To identify overlaps and differences between SPLE and PC, we include the questions:

RQ3 *What automated tasks are supported (e.g., completion, repair, and optimization of configurations)?*
RQ4 *How are these automated tasks implemented?*

2.3 Complexity

The computational complexity is an indicator of the amount of resources needed to solve a given problem. In PC, reasoning is usually achieved by encoding the problem in formalisms such as CSP, SAT, answer set programming or description logics, all of which are being supported by mature reasoners. Both SAT and CSP are well-known to be NP complete [15, 38]. Some extensions of CSPs (dynamic, composite) polynomially reduce to classical CSP [56] whereas the decidability of Generative CSP has yet to be established. Ground answer set programs are NP complete (Σ_2^p complete in the case of optimization); if programs contain uninstantiated variables we obtain NEXPTIME completeness [50, 19]. Description logics are typically decidable fragments of first order logic [6]; the DLs used in PC range from polynomial over PSPACE complete to undecidable [35, 40, 12]. Let us emphasize that the aforementioned complexity results are only upper bounds: Precise complexity results for classes of configuration problems are still too rare (e.g., [52]).

The same symptom can be observed in SPLE where only a tiny fraction of the papers study complexity aspects (e.g. [49]). While some experimental results exist, e.g., [43], theoretical results are largely missing.

Opportunities for cross-fertilization. Although to a different extent, both PC and SPLE do not fully cover questions related to the complexity of the automated tasks they support for different classes of configuration problems. To follow up on **RQ1** and **RQ3**, we propose these new questions that study their complexities:

RQ5 *What is the complexity of automated tasks for relevant classes of configuration problems?*
RQ6 *What reasoning frameworks can be used to build scalable tools for each class of configuration problems?*

2.4 Life cycle coverage

SPLE suffers from a certain lack of homogeneity across the modelling artefacts used throughout the engineering life cycle. From requirements engineering down to code generation, a myriad of alternative techniques exist which are used and combined differently depending on the application domain and project context. Therefore, there is no standard view on how they should be integrated. As for PC, configuration tasks can range from bill-of-material configuration over cement factory design to t-shirt customization. These tasks call for very different methods and techniques whose applications have been insufficiently studied. Additionally, the creation of feedback loops from productive use back to variability design decisions is rather explored in the more business- or management-centric literature without transfer to PC.

Configurator engineering is a more mature discipline in PC than in SPLE, aiming at the co-design of the configurator and the configurable artefact. According to Hvam et al. [30], the creation, implementation and operation of a configurator is a seven-phase procedure. The first phase identifies the product specification process, used for analyzing customer needs, creating a customized product, and prescribing other related activities, such as purchasing, delivery, servicing, and recycling. The specification process also defines the configuration system that supports the activities composing it. The second phase deals with the definition of the product portfolio. Phases three to six deal with the modelling and implementation of the configuration system. The seventh phase focuses on maintenance.

Finally, the commercial side of configuration is also important. PC has to deal typically with sales, consumer goods, and engineers, wheras SPLE is more geared towards software engineers and other technical experts. Although stakeholder profiles vary from one case to the other, some configuration tasks overlap and techniques could be shared.

Opportunities for cross-fertilization. To better pinpoint overlaps, we split the problem into three questions:

RQ7 *What are the configuration tasks?*
RQ8 *How is a configurable product engineered?*
RQ9 *How is a configurator engineered?*

2.5 Knowledge evolution

The main concern of the software configuration management (CM) discipline is controlling and tracking the evolution of products in response to changes. To do so, it introduces the concept of *versions* that represent instances of products and its parts over time. In CM, there are two main types of versions [14]: *revisions* and *variants*. Revisions are versions that supersede other versions due to bug fixes or addition of new functionalities. Variants are versions intended to coexist through time to satisfy different user or platform needs. Variants are well known in the PC and SPLE fields. However, the management of revisions and the interaction of both is still a weakness of PC and SPLE, especially when the product and its parts evolve frequently. The need to deploy change management techniques in SPLE is recognized [13, 47], and some researchers have started working in this direction (e.g. [2, 57]). Although promising, these results are still incomplete, and need to be extended and consolidated.

Problems related to the evolution of the configuration knowledge and system (e.g. knowledge base, database, and product instances) are also known in PC [21]. PC research has addressed some of these evolution-related aspects in the context of *reconfiguration* problems (e.g. [55, 24]), which consist in changing an already existing or deployed configuration to accommodate new or changed customer requirements or constraints in the knowledge base. Männistö et al. [39] discuss the issue of the evolution of configuration knowledge and instances, proposing a framework to address it. The key idea is to accept the independence of these evolutions, capture the evolution in the models, and then do reconfiguration. Problems similar to reconfiguration have been addressed in the software domain: given a component-based software installation (e.g. Linux, Eclipse), the component dependencies and an (un-)install request, compute a best new installation [1].

Another practical challenge both in PC and SPLE is the constantly growing number of components that can be part of a configuration, be they semi-conductors, switches, or software plug-ins. The corresponding knowledge bases soon become hard to manage because they describe how older components have to be replaced by newer ones or which component versions are compatible.

In CM, some have tried to provide a unified model for software CM and product data management (e.g. [20, 16]). Those models stop at the conceptual level without providing operational solutions, however.

Opportunities for cross-fertilization. Since the 1970s, research in CM focused on change management. Mature solutions are now available and could be applicable to PC. Furthermore, researchers in SPLE and PC could join forces to develop scalable techniques to address the explosion of the number of components and their evolution. Those results could then be contributed back to CM. The resulting questions are:

RQ10 *How can CM techniques be applied to PC and SPLE?*

RQ11 *How to scale up with growing revision knowledge?*

3 Research Roadmap

The first step of our project was the composition of an heterogeneous panel of experts from the different communities and the creation of a knowledge exchange portal. Once populated with our initial results, our intention is to open this shared portal to a wider community and invite collaborations. Our research roadmap has five phases.

1. **Literature survey.** The first phase focuses on a literature survey that should answer **RQ1-9**. The part of the survey related to **RQ1-4** is already underway. The body of knowledge gathered during this initial phase will be the foundation upon which the panel will start its unification endeavour.
2. **Domain understanding and classification.** The second phase paves the road toward a unified theory of configuration. Its objective is to analyze the material collected in the first phase, to classify the application domains, study their differences and commonalities, and to identify the possible bridges between these domains.
3. **Unified theory definition.** The foundation for this theory will be a mathematical model of the classes of configuration problems and their properties. These classes of configuration problems will be characterized by the expressiveness needed to solve the problems. Analogous to the classification of description logics [6], this will enable the study of the theoretical complexity of the associated computational problems. Problem frames [31] could be used as the macro-structure for this classification.
4. **Unified theory operationalization.** Upon this definition, we will build two layers: modelling languages and reasoning techniques. In the modelling layer, we will first sort existing languages according to the configuration problems they address. We will then work toward their unification, possibly by mapping into a common core language. The reasoning layer will gather the configuration tasks (e.g., consistency check, repair, and completion) identified in the previous phases. For each task and a class of configuration problems captured by a language, we will study alternative reasoning techniques and assess their applicability (see next phase). Finally, we will investigate how version management techniques from CM can be adapted to support configuration evolution. Based on these elements, we will answer **RQ6, 10** and **11**.
5. **Unified theory validation.** Configuration can be applied in a myriad of application domains to support many different usage scenarios. Each situation will need its own benchmarks and analyses. The objective of the panel will be to validate the results of the theory operationalization on a few industry-size models. In particular, it will seek to better understand the respective merits and limitations of the various reasoning techniques. Benchmarks will be needed to assess their applicability and tractability on various classes of problems. This pilot validation will be a first step in setting up a framework in which parallel efforts of the community could commence.

This roadmap is not intended to be executed in a waterfall fashion: we expect several iterations through phases 2-5.

4 Conclusion

Configuration, both of software and other types of products, continues to be a timely business strategy as customers consistently strive for affordable tailor-made products. Yet, research in product and software configuration progresses on different and rarely intersecting paths. This paper (1) motivated the need for bridging the current gap between these two domains, and (2) presented a roadmap to build such a bridge and set cross-fertilization in motion.

Our initial observations show that the contribution of the research in product configuration to software product configuration is rather glaring. The inverse is, however, less obvious. As possible contributions, we see methodological aspects as well as modelling techniques. Once laid upon more formal foundations, some of these models could further improve product configuration.

Finally, the *evolution* problem is still under-explored in both software and product configuration. They both have a lot to gain from the techniques promoted in configuration management. Conversely, the formal treatment of configuration problems and automated reasoning could enhance existing work in configuration management.

REFERENCES

[1] P. Abate, R. Di Cosmo, R. Treinen, and S. Zacchiroli, 'Dependency solving: a separate concern in component evolution management', *Systems and Software*, (2012). To appear.

[2] M. Anastasopoulos, D. Muthig, T. H. B. de Oliveira, E. S. Almeida, and S. R. de Lemos Meira, 'Evolving a software product line reuse infrastructure: A configuration management solution', in *Proc. VaMoS'09*, Sevilla, (2009).

[3] H. R. Andersen, T. Hadzic, and D. Pisinger, 'Interactive cost configuration over decision diagrams', *Artificial Intelligence Research (JAIR)*, **37**, 99–139, (2010).

[4] M. Aschinger, C. Drescher, and H. Vollmer, 'LoCo - A Logic for Configuration Problems', in *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI), Montpellier, France*, (2012).

[5] T. Asikainen, T. Soininen, and T. Männistö, 'A Koala-based approach for modelling and deploying configurable software product families', in *Porc PLE'03*, Springer LNCS 3014, pp. 225–249, (2003).

[6] *The Description Logic Handbook: Theory, Implementation, and Applications*, eds., F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Cambridge University Press, 2003.

[7] E. Bagheri, T. Di Noia, A. Ragone, and D. Gasevic, 'Configuring software product line feature models based on stakeholders' soft and hard requirements', in *SPLC'10*, pp. 16–31, (2010).

[8] K. Bak, K. Czarnecki, and Andrzej W., 'Feature and metamodels in clafer: Mixed, specialized, and coupled', in *Proc. SLE'10*, pp. 102–122, Eindhoven, The Netherlands, (2010). Springer-Verlag.

[9] D. Batory, 'Feature models, grammars, and propositional formulas', in *SPLC'05*, pp. 7–20, Rennes, France, (2005).

[10] D. Benavides, S. Segura, and A. Ruiz-Cortes, 'Automated analysis of feature models 20 years later: a literature review', *Information Systems*, **35**(6), 615 – 636, (2010).

[11] D. Benavides, P. Trinidad, and Ruiz-Cortez A., 'Automated reasoning on feature models', in *Proc. CAiSE'05*, pp. 491–503, Porto, Portugal, (2005). Springer.

[12] M. Buchheit, R. Klein, and W. Nutt, 'Constructive Problem Solving: A Model Construction Approach towards Configuration', Technical Report TM-95-01, DFKI, (1995).

[13] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2001.

[14] R. Conradi and B. Westfechtel, 'Version models for software configuration management', *ACM Computing Surveys*, **30**, 232–282, (June 1998).

[15] S. A. Cook, 'The complexity of theorem-proving procedures', in *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, (1971).

[16] I. Crnkovic, U. Asklund, and A. P. Dahlqvist, *Implementing and integrating product data management and software configuration management*, Artech House Publishers, 2003.

[17] K. Czarnecki, P. Grunbacher, R. Rabiser, K. Schmid, and A. Wasowski, 'Cool features and tough decisions: Two decades of variability modeling', in *Proc. VaMoS'12*, Leipzig, Germany, (2012). ACM Press.

[18] K. Czarnecki, S. Helsen, and U. W. Eisenecker, 'Staged configuration through specialization and multi-level configuration of feature models', *Software Process: Improvement and Practice*, **10**(2), 143–169, (2005).

[19] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov, 'Complexity and expressive power of logic programming', *ACM Computing Surveys*, **33**(3), 374–425, (2001).

[20] J. Estublier, J.-M. Favre, and P. Morat, 'Toward scm / pdm integration?', in *Proc. SCM'98*, pp. 75–94, London, UK, UK, (1998). Springer-Verlag.

[21] A. Falkner and A. Haselböck, 'Challenges of knowledge evolution in practice', in *ECAI 2010 IKBET Wks*, pp. 1–5, (2010).

[22] A. Felfernig, G. Friedrich, and D. Jannach, 'UML as domain specific language for the construction of knowledge-based configuration systems', in *Proc. SEKE 99*, pp. 337–345, Kaiserslautern, Germany, (1999).

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

[24] G. Friedrich, A. Ryabokon, A. Falkner, A. Haselböck, G. Schenner, and H. Schreiner, '(Re)configuration based on model generation', in *Proc. LoCoCo Workshop*, EPTCS, pp. 26–35, (2011).

[25] H. Gomaa and M. Eonsuk Shin, 'Multiple-view modelling and meta-modelling of software product lines', *IET Software*, **2**(2), 94–122, (2008).

[26] G. Gottlob, G. Greco, and T. Mancini, 'Conditional Constraint Satisfaction: Logical Foundations and Complexity', in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI), Hyderabad, India*, (2007).

[27] A. Günter and C. Kühn, 'Knowledge-based configuration: Survey and future directions', in *Proc. XPS'99*, pp. 47–66, London, (1999).

[28] L. Hotz, K. Wolter, and T. Krebs, *Configuration in Industrial Product Families: The ConIPF Methodology*, IOS Press, Inc., 2006.

[29] A. Hubaux, A. Classen, and P. Heymans, 'Formal modelling of feature configuration workflow', in *Proc. SPLC'09*, pp. 221–230, San Francisco, (2009).

[30] L. Hvam, N. Henrik Mortensen, and J. Riis, *Product Customization*, Springer-Verlag Berlin Heidelberg, 2008.

[31] M. Jackson, *Problem frames: analyzing and structuring software development problems*, Addison-Wesley, 2001.

[32] D. Jannach and M. Zanker, 'Modeling and solving distributed configuration problems: A CSP-based approach', *IEEE TKDE*, http://doi.ieeecomputersociety.org/10.1109/TKDE.2011.236.

[33] M. Janota, 'Do SAT solvers make good configurators?', in *Proc. ASPL'08*, pp. 191–195, Limerick, Ireland, (2008).

[34] U. Junker, *Handbook of Constraint Programming*, chapter Configuration, 837–873, Elsevier, 2006.

[35] U. Junker and D. Mailharro, 'The logic of ILOG (J)Configurator: Combining constraint programming with a description logic', in *Proc. IJCAI-03 Configuration Workshop*, (2003).

[36] U. Junker and D. Mailharro, 'Preference programming: Advanced problem solving for configuration', *AIEDAM*, **17**(1), 13–29, (2003).

[37] K. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, 'Feature-Oriented Domain Analysis (FODA) Feasibility Study', Technical report, SEI, CMU, (1990).

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

[39] T. Männistö, *A Conceptual Modelling Approach to Product Families and Their Evolution*, Ph.D. dissertation, Helsinki University of Technology, Finland, 2000.

[40] D. L. McGuinness and J. R. Wright, 'Conceptual modelling for configuration: A description logic-based approach', *Artif. Intell. Eng. Des. Anal. Manuf.*, **12**(4), 333–344, (1998).

[41] M. D. McIlroy, 'Mass produced software components', in *Proc. Software Engineering Concepts and Techniques*, pp. 138–150. NATO Science Committee, (1968).

[42] M. Mendonça, *Efficient Reasoning Techniques for Large Scale Feature Models*, Ph.D. dissertation, U Waterloo, 2009.

[43] M. Mendonça, A. Wasowski, and K. Czarnecki, 'SAT-based analysis of feature models is easy', in *Proc. SPLC'09*, pp. 231–240, San Francisco, (2009). Carnegie Mellon University.

[44] R. Michel, A. Classen, A. Hubaux, and Q. Boucher, 'A formal semantics for feature cardinalities in feature diagrams', in *Proc. Wks. VaMoS'11*, pp. 82–89, Namur, BE, (2011).

[45] S. Mittal and B. Falkenhainer, 'Dynamic constraint satisfaction problems', in *AAAI'90*, pp. 25–32, Boston, (1990).

[46] M. Pasanen, *Warnings and Pre-selection Packages in a Weight Constraint Rule Based Configurator*, Master's thesis, Helsinki University of Technology, Department of Computer Science and Engineering, Finland, 2003.

[47] K. Pohl, G. Böckle, and F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*, Springer, 2005.

[48] D. Sabin and E. C. Freuder, 'Configuration as Composite Constraint Satisfaction', in *Proceedings of the Artificial Intelligence and Manufacturing Research Planning Workshop (AIMRP), Albuquerque, New Mexico*, (1996).

[49] P-Y. Schobbens, P. Heymans, J-C. Trigaux, and Yves Bontemps, 'Generic semantics of feature diagrams', *Comput. Netw.*, **51**(2), 456–479, (2007).

[50] P. Simons, I. Niemelä, and T. Soininen, 'Extending and implementing the stable model semantics', *Artificial Intelligence*, **138**(1–2), 181–234, (2002).

[51] C. Sinz, A. Kaiser, and W. Küchlin, 'SAT-based consistency checking of automotive electronic product data', in *ECAI Workshop*, (2000).

[52] T. Soininen, *An Approach to Knowledge Representation and Reasoning for Product Configuration Tasks*, Ph.D. dissertation, 2000.

[53] T. Soininen, J. Tiihonen, T. Männistö, and R. Sulonen, 'Towards a general ontology of configuration', *AIEDAM*, **12**, 357–372, (1998).

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

[55] M. Stumptner and F. Wotawa, 'Model-based reconfiguration', in *Proc. AID'98*, pp. 45–64, (1998).

[56] E. Thorstensen, 'Capturing Configuration', in *Doctoral Program at the 16th International Conference on Principles and Practice of Constraint Programming (CP), St. Andrews, Scotland*, (2010).

[57] T. Thum, D. Batory, and C. Kastner, 'Reasoning about edits to feature models', in *Proc. ICSE'09*, pp. 254–264, Washington, DC, USA, (2009). IEEE Computer Society.

[58] T. T. Tun, Q. Boucher, A. Classen, A. Hubaux, and P. Heymans, 'Relating requirements and feature configurations: A systematic approach', in *Proc. SPLC'09*, pp. 201–210, San Francisco, CA, USA, (2009). ACM Press.

[59] Y. Xiong, A. Hubaux, and K. Czarnecki, 'Generating range fixes for software configuration', in *Proc. ICSE'12*, Zurich, Switzerland, (2012). IEEE Computer Society.

[60] T. Ziadi, L. Helouet, and J.-M. Jezequel, 'Towards a UML profile for software product lines', in *Software Product-Family Engineering*, Springer LNCS 3014, 129–139, (2004).

An Improved Constraint Ordering Heuristics for Compiling Configuration Problems

Benjamin Matthes and **Christoph Zengler** and **Wolfgang Küchlin**[1]

Abstract. This paper is a case study on generating BDDs (binary decision diagrams) for propositional encodings of industrial configuration problems. As a testbed we use product configuration formulas arising in the automotive industry. Our main contribution is the introduction of a new improved constraint ordering heuristics incorporating structure-specific knowledge of the problem at hand. With the help of this constraint ordering, we were able to compile all formulas of our testbed to BDDs which was not possible with an arbitrary constraint order.

1 INTRODUCTION

Since the early 80s, product configuration systems have been among the most prominent and successful applications of AI methods in practice [15]. As a result computer aided configuration systems have been used in managing complex software, hardware or network settings. Another application area of these configuration systems is the automotive industry. Here they helped to realize the transition from the mass production paradigm to present-day mass customization.

Besides CSP encodings [1] also propositional encodings [10] of configuration problems proved to be a viable alternative in the automotive industry. Specific queries to the configuration base can then be answered by a decision procedure for propositional logic, e.g. in many cases SAT solvers. Although modern SAT solvers prove to be very efficient in answering such queries, there are two major drawbacks: (1) Since decidability of a propositional formula is NP-hard, SAT solvers cannot guarantee certain runtime requirements required in online configuration applications; (2) there are some types of queries that cannot be handled by a SAT solver efficiently, e.g. restriction, model enumeration, or model counting. One approach to circumvent these limitations is the use of knowledge compilation.

The basic idea of knowledge compilation is to distinguish two phases: (1) an offline phase in which a given formula is compiled into the respective compilation format and (2) an online phase in which we query the compilation. Usually the offline phase is still NP-hard, but once compiled, there are a number of interesting polynomial time operations on the compilation. Well-known knowledge compilation formats for propositional logic are e.g. BDDs [3] or DNNFs [5].

For this paper we chose BDD as compilation format. Its use in configuration problems is well-studied [7, 11]. Hadzic et al. [7] focus on minimizing the final BDD for shorter response times; Narodytska et al. [11] try to establish a good static variable ordering for BDD compilation of configuration problems. They also present a constraint ordering based on the constraint graph. In contrast, in this paper we present an ordering of the constraints based on some structure knowledge of the problem which is not always deducible from the constraint graph. In most cases our test instances could only be compiled into BDDs with this new constraint ordering. With an arbitrary ordering we exceeded space or time limits.

In section 2 we will introduce propositional configuration problems and present the reader an overview of binary decision diagrams and some important properties. Section 3 shortly describes our test instances from the automotive industry. Our main contribution lies in section 4. We present an ordering of the constraints of the configuration problem with the help of which we could compile all formulas to BDDs.

2 PRELIMINARIES

2.1 Configuration problems

2.1.1 Propositional configuration problems

We use the definition of a configuration problem as given in [7, Definition 1]: a configuration problem is a triple $(\mathcal{V}, D, \Psi)$ where $\mathcal{V}$ is a set of variables $x_1, x_2, \ldots, x_n$, D is a set of their finite domains $D_1, D_2, \ldots, D_n$ and $\Psi = \{\psi_1, \psi_2, \ldots, \psi_m\}$ is a set of propositional formulas (constraints) over atomic propositions $x_i = v$ where $v \in D_i$, specifying conditions that the variable assignments have to satisfy. A *valid configuration* is an assignment α with $\mathrm{dom}(\alpha) = \mathcal{V}$ such that $\alpha \models \bigwedge_{\psi \in \Psi} \psi$, i.e. all constraints hold.

In this paper we consider the special case where we have only propositional variables in $\mathcal{V}$ and hence $D_i = \{1, 0\}$ for all $1 \leq i \leq n$. The set $\mathcal{O}$ is the finite set of all configuration options for a product. Each variable $x_o \in \mathcal{V}$ represents a configuration option $o \in \mathcal{O}$. The variable x_o is assigned to 1 if the option o is chosen, otherwise it is assigned to 0. Following this course, the resulting formulas $\psi \in \Psi$ are propositional formulas and hence $\varphi = \bigwedge_{\psi \in \Psi} \psi$ is a propositional formula describing all valid configurations. We will also refer to φ as *product overview formula (POF)* [10].

Remark. The restriction of the variables $x \in \mathcal{V}$ to propositional variables does not limit the expressiveness of our problem description. Since the domains D_i are finite and we only allow atomic propositions of the form $x = v$, we can use a reduction [4] from equality logic to propositional logic.

2.1.2 Structure of configuration problems

In many application domains (including the automotive product configuration), we can divide the set of constraints Ψ in three parts:

Unit Constraints Ψ_U constraints concerning only a single variable. These constraints enforce or forbid the selection of a single option.

[1] Symbolic Computation Group, WSI Informatics, Universität Tübingen, Germany, email: [matthesb, zengler, kuechlin]@informatik.uni-tuebingen.de

Cardinality Constraints Ψ_{CC} Constraints enforcing the selection of a certain number of options. In most cases the selection of exactly one option or at most one option is enforced.

Dependencies Ψ_D Constraints describing the dependencies between two or more options. These constraints are used to describe complex domain specific configuration knowledge.

Example. In the automotive industry we have the following examples for the aforementioned constraint sets:

- Ψ_U: Necessary or forbidden options in a production series of cars. E.g. 'EPS must be chosen in this series' or 'automatic transmission is not available for this series'.
- Ψ_{CC}: Enforcement that only one option from a certain option-family can be chosen at the same time. E.g. 'only one steering wheel in a car', or 'at most one navigation system'.
- Ψ_D: Description of complex dependencies in a car. E.g. 'navigation system enforces also board computer and forbids radio'.

2.2 Ordered binary decision diagrams

A binary decision diagram [3] is a directed acyclic graph which represents a propositional formula. Each inner node is labeled with a propositional variable and has two outgoing edges for negative and positive assignment of the respective variable. The leaves are labeled with 1 and 0 representing *true* and *false*. An assignment is represented by a path from the root node to a leaf and its evaluation is the respective value of the leaf. Therefore all paths to a 1-leaf are valid models for the formula. Ordered reduced BDDs (ROBDDs) are a subset with additional restrictions for the BDDs. Ordering guarantees the same variable ordering on all paths through the BDD; Reduction guarantees that equivalent subtrees of the BDD are compactified and redundant nodes are deleted. A ROBDD is a canonical representation of a propositional formula wrt. to a variable ordering, meaning the ROBDD of a formula is unique. From now on we will refer to ROBDDs simply as BDDs. Figure 1 presents the BDD for the formula $(x_1 \leftrightarrow x_2) \vee x_3$ with the variable ordering $x_1 < x_2 < x_3$. Solid edges represent the positive assignment, dashed edges the negative assignment.

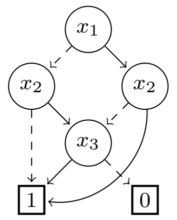


Figure 1. BDD for $(x_1 \leftrightarrow x_2) \vee x_3$ with ordering $x_1 < x_2 < x_3$

Once compiled, BDDs allow a large number of polynomial time operations on the represented formula. Among them are: satisfiability, general entailment, restriction or equivalence. Since satisfiability is a polynomial time operation on BDDs, it is obvious, that it is NP-hard to transform a given Boolean formula into a BDD. The size (number of nodes) of a BDD is strongly dependent on the variable ordering. There are many examples where bad orderings produce exponential size BDDs, whereas a good ordering produces a linear size BDD. So finding a good variable ordering is a crucial task in the compilation phase. Finding an optimal variable ordering is an NP-complete problem [2]. Different reordering heuristics for BDDs will be reviewed in section 2.2.1.

Since our input formulas are in CNF, the usual procedure of compiling the BDD is to generate BDDs for each clause and conjoin them. Here, the order in which the clauses are conjoined plays an important role. We will discuss the impact of this clause/constraint ordering in section 2.2.2.

2.2.1 *Reordering heuristics*

As already mentioned, finding the optimal variable order for a BDD is NP-complete. Modern BDD compilers use different heuristics to find a good variable ordering while compiling. We will present some of these heuristics which proved to be of interest for our real world applications.

The *sifting algorithm* by Rudell [14] is the foundation of various reordering heuristics. It is based on finding an optimum for each variable assuming all other variables remain fixed. Each variable is considered in sequence, beginning with the variable with most occurrences. The currently selected variable is sifted (moved) sequentially to both ends of the variable ordering and is finally fixed to the optimum position wrt. the size of the BDD. All variable movement can be done by a series of adjacent variable swaps. Swapping a variable with its direct predecessor or successor does *not* affect levels other than those of these two variables and therefore depends only proportionally on the size of the respective levels. This sifting process is repeated for each variable, in order of their occurrences. It is notable that the BDD size can increase heavily during sifting.

The sifting algorithm can be extended to a *symmetric sifting* [13], where symmetric variables (variables, that can be interchanged without changing the Boolean function) are kept close together. Symmetric sifting again can be generalized to *group sifting* [12]. Here, symmetry situations that go beyond the symmetry of two variables can be treated specially.

A different approach was suggested by Fujita et al. [6] and Ishiura et al. [8]. Instead of searching the optimal position of a variable in the whole variable ordering, the search space is restricted to a small *window*. Each variable is considered in sequence and permuted inside a window of size k. If x_i is considered and window size is 3, x_i, x_{i+1} and x_{i+2} have to be permuted. All $k!$ possibilities of arranging variables are exhaustively searched. After testing all permutations, the best one wrt. BDD size is used. The process is repeated for each variable. Due to the rapid growth of the faculty function, this approach is only practical for window sizes up to 5. Generally it performs better than sifting, but may not be able to overcome local minima.

For further comparison two random based algorithms have been used. The *random* variant randomly selects pairs of variables and transposes them with adjacent swaps. The best position wrt. BDD size is used. This step is repeated n times for n variables. The *random pivot* takes the same approach but requires that the first variable selected has a smaller index than a pivot element. This pivot element is the variable with most nodes in the BDD. Accordingly the second selected variable has to have a larger index than the pivot element.

2.2.2 *Clause orderings*

Given a CNF as input formula for the BDD compilation, the order in which clauses are added to the BDD is crucial. Consider a formula $\psi \wedge x \wedge \neg x$ where ψ is an arbitrary satisfiable CNF. If first all clauses of ψ are added to the BDD, the resulting BDD can be of large size (depending on ψ). If then at the end x and $\neg x$ are added, the formula turns unsatisfiable and the BDD degenerates to the 0-leaf. If on the other hand x and $\neg x$ are added to the BDD as first two clauses, all other clauses will have no more impact on the BDD. Obviously the second approach would perform much better in this case for a large ψ. In general we can not determine such a 'good' clause ordering but in our application we have specific structure knowledge which we can utilize. Since clauses in our application stem from various constraints, we will refer to this also as *constraint ordering*.

3 TEST CASES

As a testbed we used product configuration formulas for a series of cars of a major German car manufacturer. The series consists of 25 different car models, each with about 300 customer-selectable options in $\mathcal{O}$ and between 300 and 400 constraints in Ψ. Looking at the distinction in section 2.1.2 we have the following numbers:

- between 20 and 30 unit constraints in Ψ_U
- between 40 and 60 cardinality constrains in Ψ_{CC}
- about 300 dependencies in Ψ_D

The corresponding CNF translations of these formulas range between 200 and 350 variables and 500 and 3000 clauses.

We distinguish two different flavors of formulas: the first set represents a restriction of the formula to technical aspects, meaning only options are considered, which are really choosable by the customer; the second set models the full configuration space including some steering codes in the set of options which are used to guide certain processes during the manufacturing of the car. Table 1 presents an overview over all instances.

Table 1. Automotive product configuration instances

	technical		full configuration space	
	# variables	# clauses	# variables	# clauses
IA1	270	979	352	2796
IA2	262	895	344	2712
IA3	268	942	350	2759
IA4	262	898	344	2715
IB1	242	704	322	2519
IB2	236	667	316	2482
IC1	251	768	331	2583
IC2	242	704	322	2519
IC3	220	594	240	638
IC4	267	952	349	2769
IC5	257	853	339	2670
ID1	246	760	325	2575
ID2	237	696	317	2511
ID3	216	597	236	641
ID4	246	765	326	2580
ID5	238	669	318	2514
ID6	216	597	236	641
IE1	240	700	319	2514
IE2	236	662	315	2476
IE3	247	745	327	2560
IE4	241	695	321	2510
IE5	246	736	326	2551
IE6	241	697	321	2512
IE7	267	946	349	2763
IE8	257	859	339	2676

4 RESULTS

A framework for automated testing and evaluating of both static and dynamic variable orderings has been implemented. We used CUDD [2] as a foundation of our implementation. The framework can handle Dimacs CNF files and produces BDDs with different reordering heuristics and also a static variable ordering. It then evaluates the resulting BDDs wrt. compilation time and BDD size and generates comparison graphs for the different heuristics.

For the static variable ordering we solved the formula with a state-of-the-art SAT solver and used its assignment stack as ordering.

4.1 The impact of the constraint ordering

As mentioned in section 2.2.2 the clause/constraint ordering can play an important role in the compilation phase. In the first tests most of our instances could not be compiled into BDDs with an arbitrary constraint ordering. We can observe this effect in the first diagram of figure 2 for a test instance (IB2). Without structuring the constraints, we reached > 50 million internal nodes after adding 100 clauses. Due to memory restrictions (4 GB) we could not add more than 200 clauses.

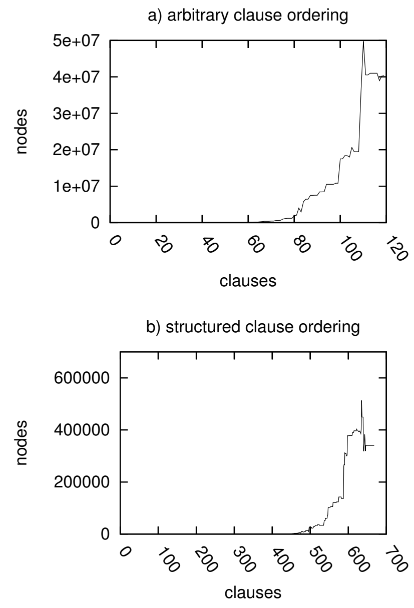


Figure 2. Impact of clause orderings (IB2)

To bypass this problem, we grouped our set of constraints according to section 2.1.2 in Ψ_U, Ψ_{CC} and Ψ_D and used the constraint ordering $\Psi_U < \Psi_{CC} < \Psi_D$, meaning first we add all the unit constraints, then we add all the cardinality constraints, and at last we add the dependencies. We will now take a closer look at the resulting BDD.

The conjoin of all the constraints in Ψ_U represents exactly one satisfying assignment. Therefore the resulting BDD is—independent of the variable ordering—a chain of n nodes for n constraints in Ψ_U. For each variable representing a necessary option, the negative edge goes to the 0-leaf and the positive edge goes to the next variable, and

[2] ftp://vlsi.colorado.edu/pub/cudd-2.5.0.tar.gz

vice versa for each variable representing a forbidden option. Figure 3 illustrates the BDD after adding all unit constraints for necessary options $n_1, \ldots, n_k$ and forbidden options $f_1, \ldots, f_l$.

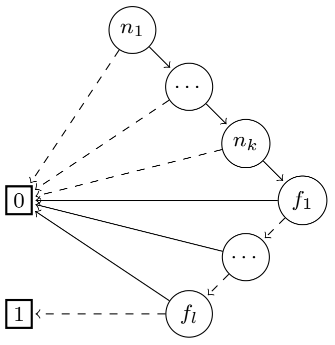


Figure 3. BDD after adding all unit constraints

Next we add all the cardinality constraints. In our context we only have to deal with 'exactly one' and 'at most one' constraints. The propositional translation of 'at most one option of $x_1, \ldots, x_n$ is chosen' is

$$\bigwedge_{i \in \{1,\ldots,n\}} \bigwedge_{j \in \{i+1,\ldots,n\}} (\neg x_i \vee \neg x_j). \tag{1}$$

For the encoding of 'exactly one variable of $x_1, \ldots, x_n$ is chosen' we simply conjoin $\left(\bigvee_{i \in \{1,\ldots,n\}} x_i\right)$ to (1). The resulting BDD has (independent of the variable ordering) $2n - 1$ nodes for an 'exactly one of n' constraint and $2n - 2$ nodes for an 'at most one of n' constraint. Figure 4 illustrates such a BDD for an 'exactly one of $x_1, \ldots, x_n$' constraint. In our application domain all constraints in Ψ_{CC} have disjoint variable sets (an option can only belong to one option-family). Therefore compiling all cardinality constraints to a BDD results in a chain of sub-BDDs as represented in figure 4. If one of the options in a cardinality constraint was also present in the unit constraints, the reduction property of the BDD guarantees immediate simplification. After adding the cardinality constraints and the unit constraints, the BDD size is still linear in the number of unit constraints and cardinality constraints and their respective variables.

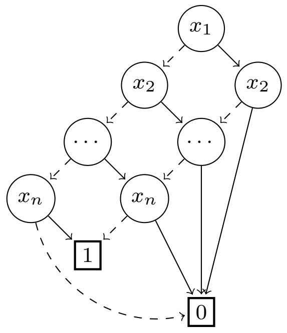


Figure 4. BDD for an 'exactly one' constraint

As a last step, the dependencies between the options Ψ_D are conjoined to the BDD. This step can enlarge the BDD significantly (exponential size in the worst-case). But our experiments show, that the knowledge already present due to the translation of the unit constraints and the cardinality constraints helps to a great extent to simplify the remaining constraints.

In the second diagram of figure 2 we can observe this effect: Adding the clauses representing unit and cardinality constraints (the first 500 clauses) goes smoothly and the resulting BDD is very small. First on adding the dependencies, the BDD size grows faster. But taking into account that we could not compile over 200 clauses with an arbitrary constraint ordering, this is a large improvement. With the help of this new constraint ordering, we were able for the first time, to compile all our industrial instances into BDDs with under two minutes per instance (most of them taking only a few seconds to compile).

4.2 Comparison of the reordering heuristics

We compared all reordering heuristics wrt. compilation time (execution time in user mode) and BDD size (total number of nodes). Our test system was a 64-Bit Linux running on an AMD Athlon 64 X2 Dual Core 4600+ with 4 GB of RAM. For each instance all 16 heuristics were tested. The results are denoted as follows:

- *var:* static variable ordering (assignment stack of SAT solver)
- *none:* ascending variable order $x_1 \ldots x_n$
- *sifting:* basic sifting algorithm
- *symsift:* symmetric sifting
- *gsift:* group sifting
- *windowX:* window permutation with window size X
- *random:* random selection algorithm
- *rpivot:* random selection with pivot element

A '-c' identifies the convergent variant of a heuristics, which means it is applied until no further improvement can be observed.

Figure 5 presents an evaluation for one test instance as automatically produced by our tool. Here you can observe a typical pattern we identified: the static variable ordering often has a short compilation time, but produces large BDDs. The windowing algorithms perform better than the sifting-based algorithms in most cases. The sifting algorithms yield by far the smallest BDDs.

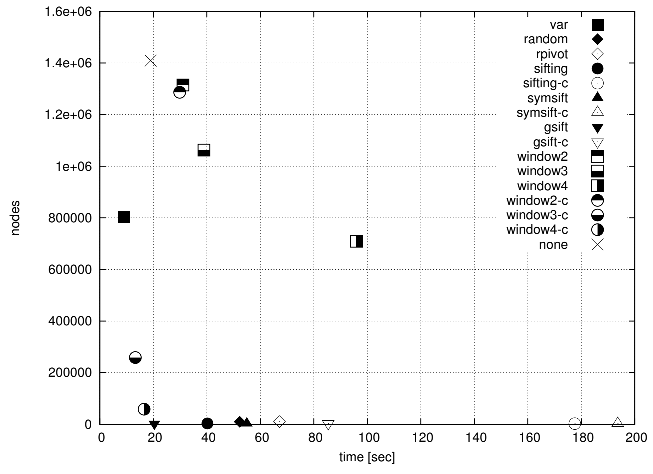


Figure 5. Heuristics comparison for ID5 (full configuration space)

Table 2 presents an overview what algorithm yielded the smallest BDD size for each test instance and what algorithm had the best compilation time. Times are noted in seconds. The table presents only instances of the full configuration space. The instances reduced to technical aspects showed similar results and are omitted here.

Table 2. Comparison of reordering algorithms (full configuration space)

	nodes	winner	time (in s)	winner
IA1	6133	gsift-c	36.41	none
IA2	3837	symsift-c	62.2	symsift
IA3	1974	symsift-c	28.3	window3-c
IA4	12016	gsift-c	48.61	window3-c
IB1	1820	gsift-c	3.37	var
IB2	2878	gsift-c	1.03	var
IC1	2539	symsift	7.6	window3-c
IC2	1411	gsift-c	8.17	var
IC3	844	symsift	0.99	var
IC4	4229	symsift-c	45.46	window3-c
IC5	2883	symsift-c	33.35	gsift
ID1	1781	symsift-c	18.18	window3-c
ID2	2702	gsift-c	10.87	var
ID3	1345	symsift-c	0.58	var
ID4	1343	symsift-c	3.54	var
ID5	2407	gsift	9.01	var
ID6	1345	symsift-c	0.6	var
IE1	1165	gsift-c	2.59	var
IE2	1313	symsift	0.93	var
IE3	3587	sifting-c	13.15	window3-c
IE4	2029	gsift	3.08	var
IE5	1853	symsift-c	10.89	var
IE6	1898	symsift-c	6.15	var
IE7	2308	symsift-c	37.91	window3-c
IE8	2233	gsift-c	34.66	window3-c

These results are summarized in table 3[3]. Here we show how many times (out of 50 instances—full and technical configuration space) each algorithm yielded the best result wrt. to size and time respectively. The aforementioned observations are justified: a static variable ordering or reordering algorithms based on windowing (especially with window size 3) have often the best compilation times at the expense of large BDD sizes. The various reorderings based on sifting, especially the convergent symmetric sifting variant, produce the smallest BDDs. Since performance in the offline phase is not too critical in knowledge compilation, sifting seems to be a viable choice for a reordering heuristics for our test instances in order to compile small BDDs with good query times.

5 CONCLUSION

In this paper we introduced a new constraint ordering for BDD compilation of industrial configuration instances. This constraint ordering uses structure-specific knowledge of the constraints at hand in order to optimize compilation time. With this ordering we were able for the first time to compile all configuration instances of our testbed—product configuration data of a major German car manufacturer—into BDDs. Most of these BDDs could be compiled in a few seconds and have surprisingly small representations (850 - 12.000 nodes). These results look very promising. There are some interesting questions like counting all constructible variants of a single car [9] or enumerating a certain number of cars with special features, that could be solved efficiently once we have a BDD representation.

[3] Complete benchmark results can be found at `http://www-sr.informatik.uni-tuebingen.de/research/confws2012-results.pdf`

Table 3. Summary of the winning heuristics

	# smallest size	# best time
var	0	26
random	0	0
randompivot	0	0
sifting	4	1
sifting-c	2	0
symsift	11	3
symsift-c	21	0
gsift	2	2
gsift-c	10	0
window2	0	0
window3	0	0
window4	0	0
window2-c	0	0
window3-c	0	15
window4-c	0	1
none	0	2

REFERENCES

[1] Jean Marc Astesana, Yves Bossu, Laurent Cosserat, and Helene Fargier, 'Constraint-based modeling and exploitation of a vehicle range at renault's: Requirement analysis and complexity study', in *Proceedings of the 13th Workshop on Configuration*, pp. 33–39, (2010).

[2] Beate Bollig and Ingo Wegener, 'Improving the variable ordering of OBDDs is NP-complete', *IEEE Transactions on Computers*, **45**(9), 993–1002, (1996).

[3] Randal E. Bryant, 'Graph-based algorithms for boolean function manipulation', *IEEE Transactions on Computers*, **35**(8), 677–691, (1986).

[4] Randal E. Bryant and Miroslav N. Velev, 'Boolean satisfiability with transitivity constraints', in *Proceedings of the CAD 2000*, volume 1855 of *Lecture Notes in Computer Science*, 85–98, Springer, Berlin, Heidelberg, Germany, (2000).

[5] Adnan Darwiche, 'Decomposable negation normal form', *Journal of the ACM*, **48**(4), 608–647, (2001).

[6] Masahiro Fujita, Yusuke Matsunaga, and Taeko Kakuda, 'On variable ordering of binary decision diagrams for the application of multi-level logic synthesis', in *Proceedings of the European Conference on Design Automation*, 50–54, IEEE Computer Society, (1991).

[7] Tarik Hadzic, Subbarayan Sathiamoorthy, Rune M. Jensen, Henrik R. Andersen, Jesper Møller, and Henrik Hulgaard, 'Fast backtrack free product configuration using precompiled solution space representations', in *Proceedings of the PETO 2004*, (2004).

[8] Nagisa Ishiura, Hiroshi Sawada, and Shuzo Yajima, 'Minimization of binary decision diagrams based on exchanges of variables', in *Proceedings of the ICCAD 1991*, 472–475, IEEE Computer Society, (1991).

[9] Andreas Kübler, Christoph Zengler, and Wolfgang Küchlin, 'Model counting in product configuration', in *Proceedings of LoCoCo 2010*, volume 29, 44–53, EPTCS, (2010).

[10] Wolfgang Küchlin and Carsten Sinz, 'Proving consistency assertions for automotive product data management', *Journal of Automated Reasoning*, **24**(1-2), 145–163, (2000).

[11] Nina Narodytska and Toby Walsh, 'Constraint and variable ordering heuristics for compiling configuration problems', in *Proceedings of the 20th International Joint Conference on Artifical Intelligence, IJCAI'07*, 149–154, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, (2007).

[12] Shipra Panda and Fabio Somenzi, 'Who are the variables in your neighbourhood', in *Proceedings of the ICCAD 1995*, 74–77, IEEE Compututer Society, (1995).

[13] Shipra Panda, Fabio Somenzi, and Barbard F Plessier, 'Symmetry detection and dynamic variable ordering of decision diagrams', in *Proceedings of the ICCAD 1994*, 628–631, IEEE Computer Society, (1994).

[14] Richard Rudell, 'Dynamic variable ordering for ordered binary decision diagrams', in *Proceedings of the ICCAD 1993*, 42–47, IEEE Computer Society, (1993).

[15] Daniel Sabin and Rainer Weigel, 'Product configuration frameworks-a survey', *IEEE Intelligent Systems*, **13**(4), 42–49, (1998).

Concurrent configuration and planning problems: Some optimization experimental results

Paul Pitiot[1], Michel Aldanondo[1], Elise Vareilles[1], Paul Gaborit[1]

Abstract. This communication deals with mass customization and the association of the product configuration task with the planning of its production process while trying to minimize cost and cycle time. We consider a two steps approach that first permit to interactively (with the customer) achieve a first product configuration and first process plan (thanks to non-negotiable requirements) and then optimize both of them (with remaining negotiable requirements). The communication concerns the second optimization step. Our goal is to evaluate a recent evolutionary algorithm (EA). As both problems are considered as constraints satisfaction problems, the optimization problem is constrained. Therefore the considered EA was selected and adapted to fit the problem. The experimentations will compare the EA with a conventional branch and bound according to the problem size and the density of constraints. The hypervolume metric is used for comparison..

1 INTRODUCTION

This paper deals with mass customization and more accurately with aiding the two activities, product configuration and production planning, achieved in a concurrent way. According to the preferences of each customer, the customer requirements (concerning either the product or its production) can be either non-negotiable or negotiable. This situation allows considering a two-step process that aims to associate the two conflicting expectations, interactivity and optimality. The first interactive step, that sequentially processes each non-negotiable requirement, corresponds with a first configuration and planning process that reduces the solution space. This process is present in many commercial web sites using configuration techniques like automotive industry for example. Then, a second process optimizes the solution with respect to the remaining negotiable requirements. As the solution space can quickly become very large, the optimization problem can become hard. Thus, this behavior is not frequent in commercial web sites. Meanwhile some scientific works have been published on this subject (see for example [1] or [2]) and the focus of this article is on the optimization problem. In some previous conferences we proposed an interesting adapted evolutionary algorithm for this problem [3]. However, the presentation was rather descriptive and experimentations were not significant. Therefore, the goal of this paper is to compare this algorithm with a classical branch and bound. This initial section introduces the problem and the organization of the paper.

1.1 Concurrent configuration and planning processes as a CSP

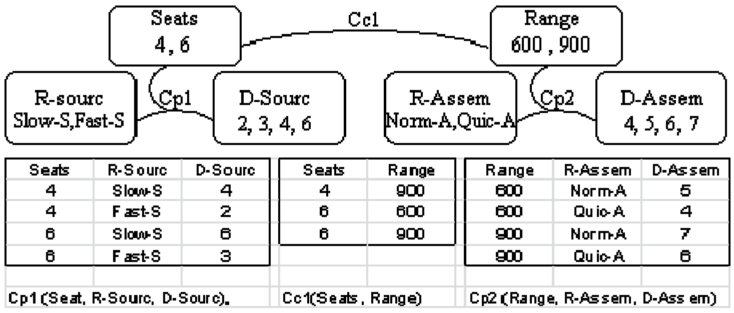
Deriving the definition of a specific or customized product (through a set of properties, sub-assemblies or bill of materials, etc…) from a generic product or a product family, while taking into account specific customer requirements, can define product configuration [4]. In a similar way, deriving a specific production plan (operations, resources to be used, etc...) from some kind of generic process plan while respecting product characteristics and customer requirements, can define production planning [5]. As many configuration and planning studies (see for example [6] or [5]) have shown that each problem could be successfully considered as a constraint satisfaction problem (CSP), we have proposed to associate them in a single CSP in order to process them concurrently.

This concurrent process and the supporting constraint framework present three main interests. First they allow considering constraint that links configuration and planning in both directions (for example: a luxury product finish requires additional manufacturing time or a given assembly duration forbids the use of a particular kind of component). Secondly they allow processing in any order product and planning requirements, and therefore avoid the traditional sequence: configure product then plan its production [7]. Thirdly, CSP fit very well on one side, interactive process thanks to constraint filtering techniques, and on the other side, optimization thanks to various problem-solving techniques. However, we assume infinite capacity planning and consider that production is launched according to each customer order and production capacity is adapted accordingly.

In order to illustrate the addressed problem we consider a very simple example dealing with the configuration and planning of a small plane. The constraint model is shown in figure 1. The plane is defined by two product variables: number of seats (Seats, possible values 4 or 6) and flight range (Range, possible values 600 or 900 kms). A constraint Cc1 forbids a plane with 4 seats and a range of 600 kms. The production process contains two operations: sourcing and assembling. (noted Sourc and Assem). Each operation is described by two process variables: resource and duration: for sourcing, the resource (R-Sourc, possible resources "Fast-S" and "Slow-S") and duration (D-Sourc, possible values 2, 3, 4, 6 weeks), for assembling, the resource (R-Assem, possible resources "Quic-A" and "Norm-A") and duration (D-Assem, possible values 4, 5, 6, 7 weeks). Two constraints linking product and process variables

[1] Toulouse University - Mines Albi, CGI lab, Albi, France
email: somename@mines-albi.fr

modulate configuration and planning possibilities: one linking seats with sourcing, Cp1 (Seat, R-Sourc, D-Sourc), and a second one linking range with the assembling, Cp2 (Range, R-Assem, D-Assem). The allowed combinations of each constraint are shown in the 3 tables of figure 1. Without taking constraints into account, this model shows a combinatory of 4 for the product (2x2) and 64 for the production process (2x4) x (2x4) providing a combinatory of 256 (4 x 64) for the whole problem. Considering constraints lead to 12 solutions for both product and production process.



Seats	R-Sourc	D-Sourc
4	Slow-S	4
4	Fast-S	2
6	Slow-S	6
6	Fast-S	3

Cp1 (Seat, R-Sourc, D-Sourc),

Seats	Range
4	900
6	600
6	900

Cc1(Seats, Range)

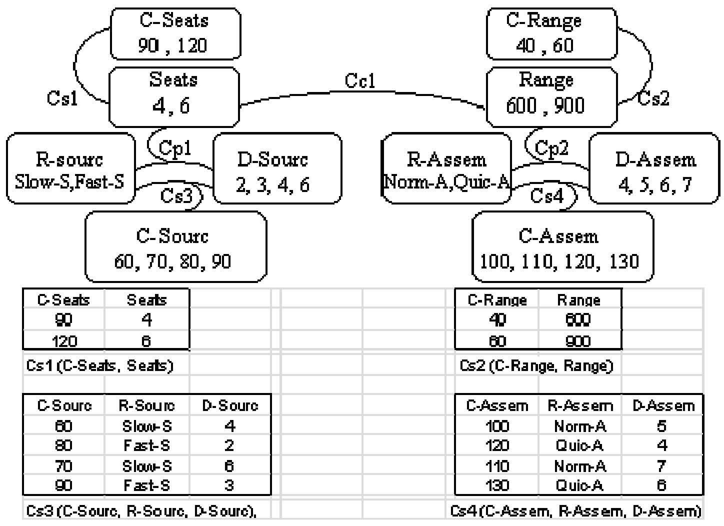
Range	R-Assem	D-Assem
600	Norm-A	5
600	Quic-A	4
900	Norm-A	7
900	Quic-A	6

Cp2 (Range, R-Assem, D-Ass em)

Figure 1 Concurrent configuration and planning CSP model

1.2 Optimizing configuration and planning concurrently

Given previous problem, various criteria can characterize a solution: on the product configuration side, performance and product cost, and on the production planning side, cycle time and process cost. In this paper we only consider cycle time and cost. The cycle time matches the ending date of the last production operation of the configured product. Cost is the sum of the product cost and process cost. We are consequently dealing with a multi-criteria optimization problem. As these criteria are in conflict, it is better for decision aiding to offer the customer a set of possible compromises in the form of Pareto Front.



C-Seats	Seats
90	4
120	6

Cs1 (C-Seats, Seats)

C-Range	Range
40	600
60	900

Cs2 (C-Range, Range)

C-Sourc	R-Sourc	D-Sourc
60	Slow-S	4
80	Fast-S	2
70	Slow-S	6
90	Fast-S	3

Cs3 (C-Sourc, R-Sourc, D-Sourc),

C-Assem	R-Assem	D-Assem
100	Norm-A	5
120	Quic-A	4
110	Norm-A	7
130	Quic-A	6

Cs4 (C-Assem, R-Assem, D-Assem)

Figure 2 Concurrent configuration and planning model to optimize

In order to complete our example, we add a cost and cycle time criteria as represented in figure 2. For cost, each product variable and each process operation is associated with a cost parameter and a relevant cost constraint: (C-Seats, Cs1), (C-Range, Cs2), (C-Sourc, Cs3) and (C-Assem, cs4) detailed in the tables of figure 2. The total cost is obtained with a numerical constraint and the cycle time, sum of the two operation durations, is also obtained with a numerical constraint as follow:
Total cost = C-Seats + C-Range + C-Sourc + C-Assem.
Cycle time = D-Sourc + D-Assem

The twelve previous solutions are shown on the figure 3 with the Pareto front gathering the optimal ones. In this figure, all solutions are present. When non-negotiable requirements are processed during interactive configuration and planning, some of these solutions will be removed. Once all these requirements are processed, the identification of the Pareto front can be launched in order to propose the customer a set of optimal solutions.

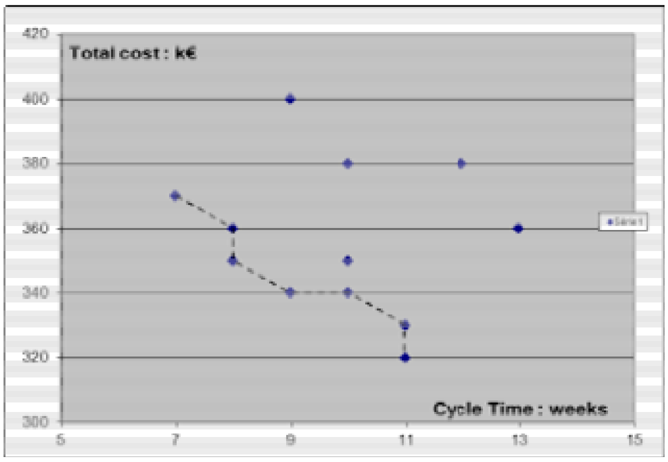


Figure 3 Optimal solutions on the Pareto Front

A strong specificity of this kind of problems is that the solution space is large. It is reported in [8] that a configuration solution space of more than 1.4* 10^{12} is required for a car configuration problem. When planning is added, the combinatorial structure can become huge. Specificity lies in the fact that the shape of the solution space is not continuous and in most cases shows many singularities. Furthermore, the multi-criteria aspect and the need for Pareto optimal results are also strong problem expectations. These points explain why most of the articles published on this subject (as for example [9]) consider genetic or evolutionary approaches to deal with this problem. However classic evolutionary algorithms have to be adapted in order to take into account the constraints of the problem as explained in [10]. Among these adaptations, the one we have proposed in [3] is an evolutionary algorithm with a specific constrained evolutionary operators and our goal is to compare it with a classical branch and bound approach.

In the following section we characterize the optimization problem and briefly recall the optimization techniques. Then experimentation results are presented and discussed in the last section.

2 OPTIMIZATION PROBLEM AND OPTIMIZATION TECHNIQUES

2.1 Optimization problem

The problem of figure 2 is generalized as the one shown in figure 4. The optimization problem is defined by the quadruplet <V, D, C, f > where V is the set of decision variables, D the set of domains linked to the variables of V, C the set of constraints on variables of V and f the multi-valuated fitness function. Here, the aim is to minimize both cost and cycle time. The set V gathers: the product descriptive variables and the resource variables. The set C gathers constraints (Cc and Cp). Cost variables and operation durations are deduced from the variables of the set V thanks to the remaining constraints.

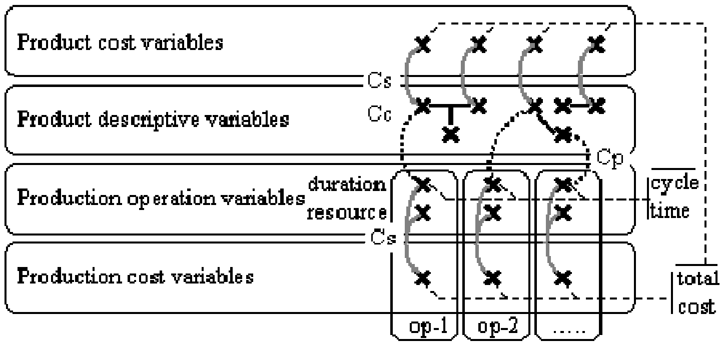

Product cost variables
Cs
Product descriptive variables
Cc
Cp
Production operation variables
duration
resource
cycle time
Cs
Production cost variables
total cost
op-1
op-2
.....

Figure 4 Constrained optimization problem

Experimentations will consider different problem sizes: different numbers of product variables, different number of production operations and different number of possible values for these variables. Different constraint densities (percentage of excluded combinations of values) will be also considered.

2.2 Optimization techniques

The proposed evolutionary algorithm is based on SPEA2 [11] with an added constraints filtering process that avoids infeasible individuals (or solutions) in the archive. This provides the six steps following approach:

1. Initialization of individual set that respect the constraints (thanks to filtering),
2. Fitness assignment (balance of Pareto dominance and solution density)
3. Individuals selection and archive update
4. Stopping criterion test
5. Individuals selection for crossover and mutation operators (binary tournaments)
6. Individuals crossover and mutation that respect constraints (thanks to filtering)
7. Return to step 2.

For initialization, crossover and mutation operators, each time an individual is created or modified, every gene (decision variable of V) is randomly instantiated into its current domain. To avoid the generation of unfeasible individuals, the domain of every remaining gene is updated by constraint filtering. As filtering is not full proof, inconsistent individuals can be generated. In this case a limited backtrack process is launched to solve the problem. For full details please see [3].

The key idea of the Branch and Bound algorithm is to explore a search tree but using a cutting procedure that stops exploration of a branch when a better branch has already been found. The first tool is a splitting procedure that corresponds to the selection of one variable of the problem and to the instantiation of this variable with each possible value. The second tool is a node-bound evaluation procedure. The filtering process is used to achieve this task with a partial instantiation and is able to evaluate if the partial instantiation is consistent with the constraints of the problem, and, if this is the case, to provide the lower bound of each criterion cycle time and cost. When the search reaches a leaf of the search tree, or complete instantiation, the filtering system gives the exact evaluation of the solution. Thus, the values of leaf solutions can be used to compute the current Pareto front and then to cut remaining unexplored branches that are dominated by any aspect of the Pareto front solution (e.g. the upper bounds of the leaf solution dominate the minimal bounds of the branch to cut).

3 EXPERIMENTATIONS

The optimization algorithms were implemented in C++ programming language and interacted with a filtering system coded in Perl language. All tests were done using a laptop computer powered by an Intel core i5 CPU (2.27 Ghz, only one CPU core is used) and using 2.8 Go of ram. These tests compared the behavior of our constrained EA algorithm with the exact branch-and-bound algorithm.

3.1 First experimentation: problem size and constraint densities

An initial first model, named "full model" is considered. It can be consulted and interactively used at http://cofiade.enstimac.fr/cgi-bin/cofiade.pl select model 'Aircraft-CSP-EA-10'. It gathers five product variables with a domain size between 4 and 6, six production operations with a number of possible resources between 3 and 25. Without constraints consideration, the solution space of the product model is 5,184, and the planning model is 96,000. The size of the global problem model is 497,664,000. A second model, named "small model", has been derived from the previous one with the suppression of a high combinatory task and a reduction of one domain size. This reduces the planning problem size to 12,000 and global model 6,220,800.

In order to evaluate the impact of constraints density, two versions of each model were produced: one with a "weak density" of constraints (20% of possible combinations are excluded in each constraint Cc and Cp) and the other with a "high density" of constraints (50% excluded). These values are frequently met in industrial configuration situations. This provides four models characteristics in table 1.

Table. 1 Problems characteristics

Solution quantity	Without constraints	Low density	High density
Small model	6 220 800	595 000	153 000
Full model	497 664 000	47 600 000	12 288 000

For the small models, evolutionary settings are tuned to: population size: 50; archive size: 40; P_{mut}: 0.4; P_{cross}: 0.8. The ending criterion used is a time limit of half an hour. For the full models, we adapt settings for a wider search: population size: 150; archive size: 100; P_{mut}: 0.4; P_{cross}: 0.8. The ending criterion used is a time required by the BB algorithm. In order to analyze the two optimization approaches, we compare the hypervolume evolution during optimization process. Hypervolum metric has been defined in [12]. It measures the hypervolume of space dominated by a set of solutions and is illustrated in Figure 5.

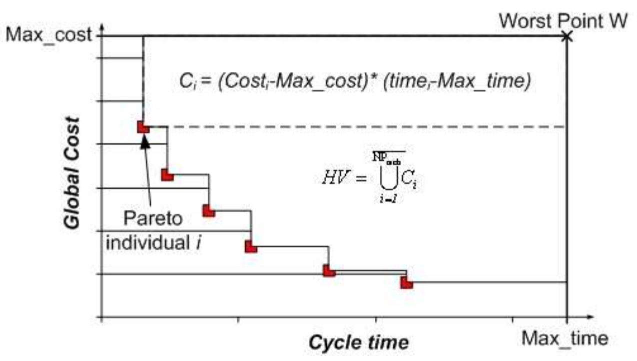

Max_cost
Worst Point W
$C_i = (Cost_i\text{-}Max\_cost)^* (time_i\text{-}Max\_time)$
$HV = \bigcup_{i=1}^{NP_{ech}} C_i$
Global Cost
Pareto individual i
Cycle time
Max_time

Figure 5 Hypervolume linked to a Pareto front

In our two criteria case, it is the upper right area of figure 5. It thus allows evaluation of both convergence and diversity properties because the fittest and most diversified set of solutions is the one that maximizes the hypervolume.

Results are presented in figure 6 where EA curves are average results for 30 executions. Both algorithms start with a lapse of time where performance is null. For the BB algorithm, this corresponds to the time needed to reach a first leaf on the search tree, while for the EA; it corresponds to the time consumed to constitute the initial population.

For the small models (first two curves), the BB algorithm reaches the optimal Pareto front much faster compared with EA performance. On the other hand, the EA is logically better than the BB algorithm on the full model. For example, on the low-constrained model, the BB algorithm took 20 times longer to reach a good set of solutions (less than 0.5% of the optimal hypervolume).

The impact of constraints density could also be discussed. As it can be seen, the BB algorithm performance is improved when the density of constraints is high. This is because the filtering allows more branches to be cut on the search tree, in such way that the algorithm reaches leaf solutions and, consequently, optimal solutions more quickly. The EA performance moves in the opposite way. The more the model is constrained, the more the random crossover operation will have to backtrack to find feasible solutions, and thus the time needed by the algorithm will be consequent.

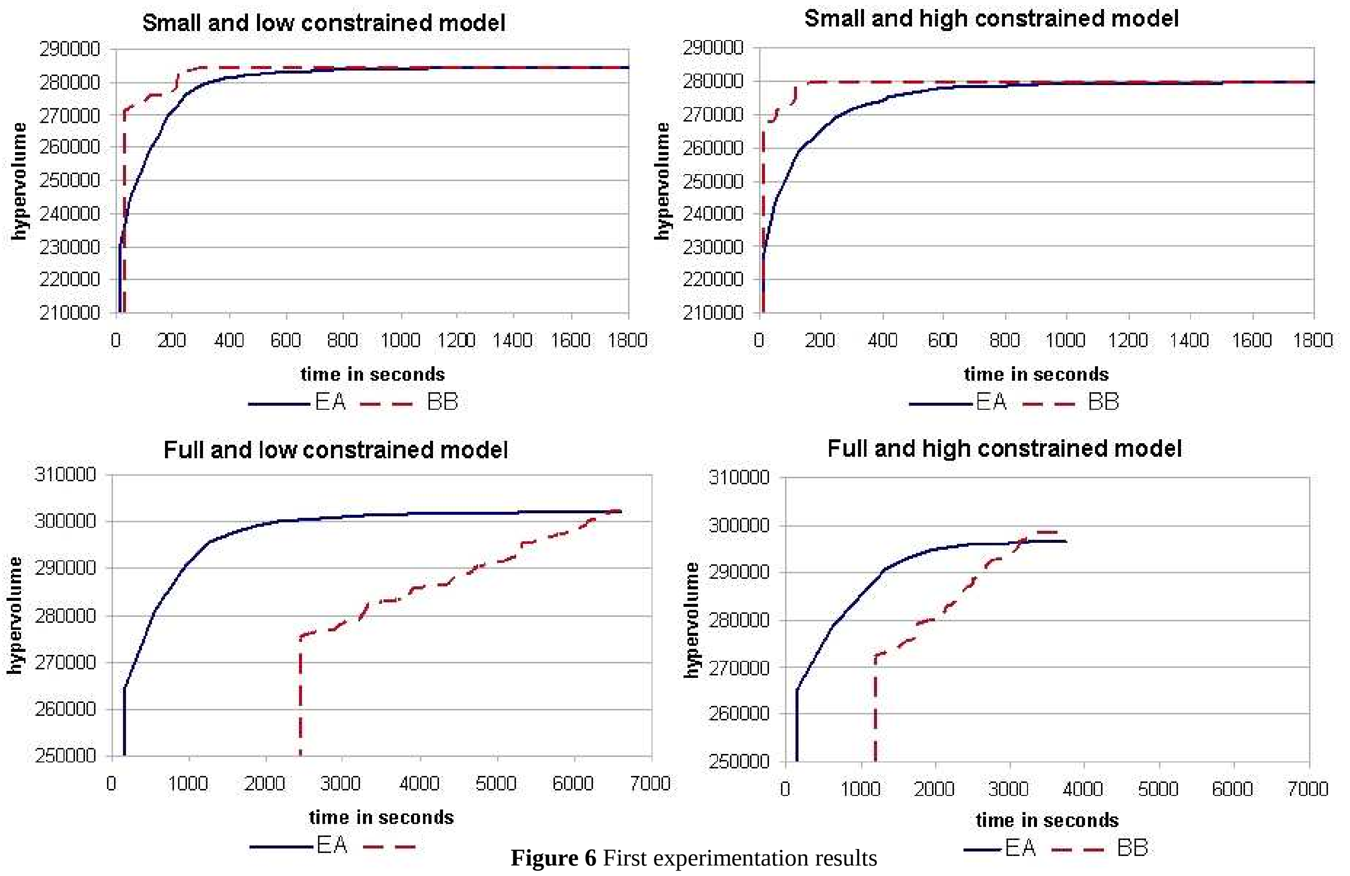


Figure 6 First experimentation results

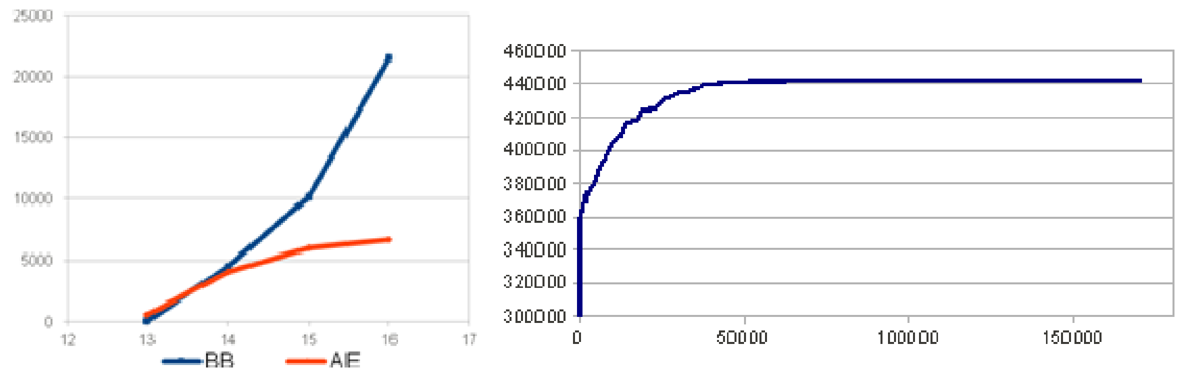


Figure 7 Experimentation results dealing with problem size

3.2 Experimentations on problem size

In order to try to identify the problem size where EA is more suitable than BB, we have modified the low constrained model as follows. We consider now a model gathering six product variables and six production operations with three possible values for each, and sequentially add either a product variable and or an operation. The range of study is between 12 and 16 decision variables with three possible values for each. Relevant solution spaces without constraint vary between $1.6*10^6$ and $43*10^6$.

The results are shown in the left part figure 7. The vertical axis corresponds with the computation time and the horizontal one with the number of decision variables. For BB curves, it shows the time to reach the optimal solution. For the EA curve it shows the time required for nine EA runs over ten to reach the optimal solution. Order of magnitude are close for both around 13 or 14 variables corresponding with a solution space around $2*10^6$ to $5*10^6$ comparable with our previous small model size.

As we already mention, industrial models are frequently larger than that. We therefore try our EA approach with a low constrained model with 30 variables and a solution space around 10^{16}. The stopping criterion is "2 hours without improvement". The right part of Figure 7 shows that the optimization process has stopped after 48 hours. It can be noticed that 90% of the final score was obtained after 3 hours and 99% in 10 hours. This allows underlining the good performance of our approach when facing large low constrained problems. Of course the idea is to use BB, if the first interactive configuration step has led to a rather small problem, less than 13 or 14 variables in our case, and EA otherwise.

Finally we also try to break optimization in two steps. The idea is: (i) compute quickly a low quality Pareto, (ii) select the area that interest the customer (iii) compute a Pareto on the restricted area. The restricted area is obtained by constraining the two criteria total cost and cycle time (or interesting area) and filtering these reductions on the whole problem. The search space is greatly reduced and the second optimization much faster. This is shown in figure 8 where the left part shows the single step process with 10 and 60 minutes Pareto and the right part shows the restricted area with the two previous curve and the one corresponding with a 10 minutes Pareto launched on the restricted area. It shows that the sequence of two optimization steps of 10 minutes provide a result equivalent to a 60 minutes optimization process.

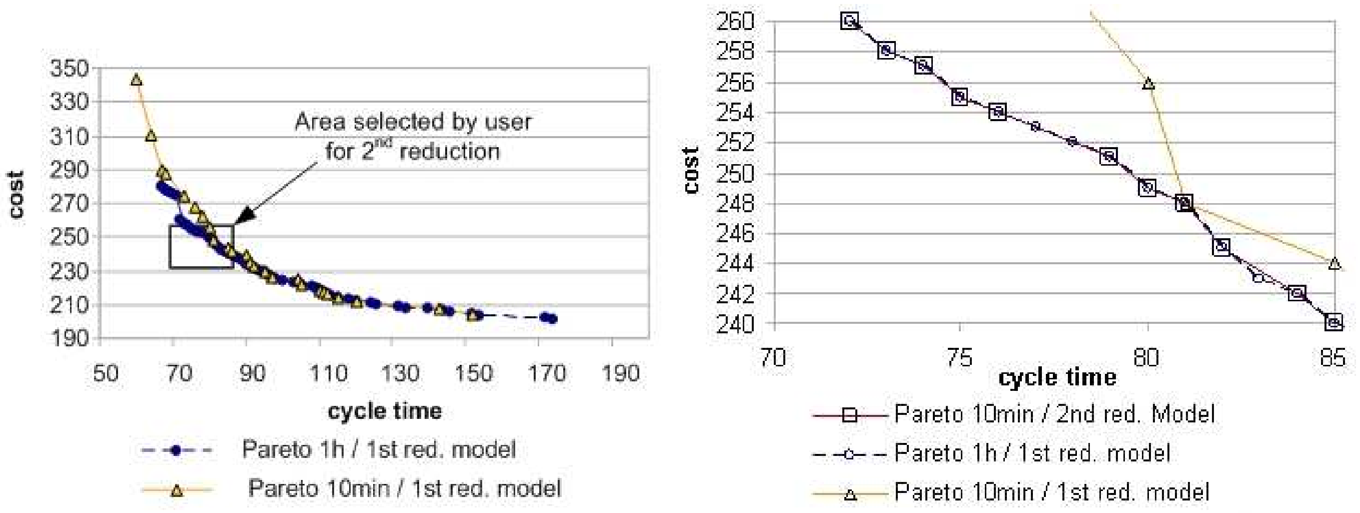


Figure 8 Experimentations with a two steps optimization process

4 CONCLUSIONS

The goal of this communication was to propose a first evaluation of an adapted evolutionary algorithm that deals with concurrent product configuration and production planning. The problem was recalled and the two optimization approaches (Evolutionary algorithm and branch and bound) where briefly presented. Various experimentations have been presented. A first result is that: (i) the proposed EA works fine when the size of the problem gets large compare to the BB, (ii) when problem tends to be more constrained the tendency goes to the opposite. When problem is low constrained (90% of excluded solutions) with 13-14 decision variables with 3 values each, they perform equally. When the problem gets larger, BB cannot be considered and EA can provide good quality results for the same problem with up to 30 variables (around 10^{16} solutions - 90% rejected). Finally some ideas about a two steps optimization process have shown that the proposed approach is quite promising for large problems. These are first experimentation results and we are now working on comparing our proposed EA with some penalty function approaches.

ACKNOWLEDGEMENTS

We would like to thank the referees for their comments, which helped improve this paper considerably.

REFERENCES

[1] Hong G., Hu L., Xue D., Tu Y, Xiong L,. Identification of the optimal product configuration and parameters based on individual customer requirements on performance and costs in one-of-a-kind production. Int. J. of Production Research, 46(12) 3297-3326 (2008)
[2] Aldanondo M., Vareilles E., Configuration for mass customization: how to extend product configuration towards requirements and process configuration, Journal of Intelligent Manufacturing, vol. 19 n° 5, p. 521-535A (2008)
[3] Pitiot P., Aldanondo M., Djefel M., Vareilles E., Gaborit P., Coudert T., Using constraints filtering and evolutionary algorithms for interactive configuration and planning. IEEE press, IEEM 2010, p.1921-1925, Macao China (2010)
[4] Mittal S., Frayman F., Towards a generic model of configuration tasks, proc of IJCAI, p. 1395-1401(1989)
[5] Barták R., Salido M., Rossi F., Constraint satisfaction techniques in planning and scheduling, in: Journal of Intelligent Manufacturing, vol. 21, n°1, p. 5-15 (2010)
[6] Junker U., Handbook of Constraint Programming, Elsevier, chap. 24, p. 835-875 (2006)
[7] Aldanondo M., Vareilles E., Djefel M.. Towards an association of product configuration with production planning, Int. J. of Mass Customisation, vol.3 n°4, p. 316-332 (2010)
[8] Amilhastre J., Fargier H., Marquis P., Consistency restoration and explanations in dynamic csps - application to configuration, in: Artificial Intelligence, vol.135, p. 199-234 (2002)
[9] Li L., Chen L., Huang Z., Zhong Y., Product configuration optimization using a multiobjective GA, I.J. of Adv. Manufacturing Technology, vol. 30, p. 20-29 (2006)
[10] Coello Coello C., Theoretical and numerical constraint-handling techniques used with EAs : A survey of the state of art, Computer Methods in Applied Mechanics and Engineering, vol. 191, n°11-12, p. 1245-1287 (2002)
[11] Zitzler E., Laumanns M., Thiele L., SPEA2: Improving the Strength Pareto Evolutionary Algorithm, Technical Report 103, Swiss Fed. Inst. of Technology (ETH), Zurich (2001)
[12] Zitzler E., Thiele L., Multiobjective Optimization Using Evolutionary Algorithms - A Case Study, 5th Conf. Parallel Problem Solving from Nature, Springer, p. 292-301 (1998)

Sales Configurator Capabilities to Prevent Product Variety from Backfiring

Alessio Trentin and **Elisa Perin** and **Cipriano Forza**[1]

Abstract. Firms offering high product variety and customization can paradoxically experience a loss of sales because customers feel overwhelmed by the number of product configurations offered. Sales configurators may be a solution for avoiding this paradox, but relatively few studies have focused on the characteristics they should have in order to overcome this problem. Furthermore, empirical investigation on the effectiveness of the recommendations made by these studies has been hindered by the lack of psychometrically sound measurement items and scales. This paper conceptualizes, develops and validates five capabilities that sales configurators should deploy in order to avoid the product variety paradox: namely, focused navigation, flexible navigation, easy comparison, benefit-cost communication, and user-friendly product-space description capabilities. The measurement instrument is hoped to support advancements in both research and practice.

1 INTRODUCTION

Many firms in diverse industries are increasing the product variety and customization offered to their customers [1-3]. By giving customers exactly what they want, or at least something closer to their ideal product solutions, companies expect to gain higher market shares and/or to be able to charge higher prices [4, 5], thereby increasing revenues.

There is a risk, however, that a strategy of product proliferation and customization backfires, leading to lower rather than greater revenues, as increasingly suggested in literature [5-11]. Potential customers, for example, may feel so confused and overwhelmed by the number of product configurations offered by a company that they choose not to make a choice at all [6] and the company loses potential sales. Firms offering product variety and customization may therefore experience what has been termed the "product variety paradox" [12]: offering more product variety and customization in an attempt to increase sales paradoxically results in a loss of sales.

An important role in alleviating the risk of experiencing this paradox can be played by sales configurators [12-14]. A sales configurator is a subtype of software-based expert systems (or knowledge-based systems) with a focus on the translation of each customer's idiosyncratic needs into complete and valid sales specifications of the product solution that best fits those needs within a company's product offer [15, 16]. The fundamental functions of a sales configurator include presenting a company's product space, meant as the set of product solutions that a firm offers [17], and guiding customers in the generation or selection of a product variant within that space, thus preventing inconsistent or unfeasible product characteristics from being defined [14, 18]. Additional functionalities of a sales configurator may include providing real-time information on price and/or delivery terms of a product variant, making quotations [19, 20] and recommending a product solution that can be further altered [13]. Sales configurators may be stand-alone applications or modules of other applications, known as product configurators, which support both sales specifications and the creation of product data necessary to build the product variant requested by the customer, such as bill of materials, production sequence, etc. [21].

Many studies on sales configurators and, more generally, on product configurators have investigated technical or application development issues, such as the modeling of configuration knowledge or the algorithms to make configurators faster and more accurate [e.g., 22, 23-28]. Many other studies have provided detailed accounts of the introduction and use of a configurator in a single company, focusing mainly on implementation challenges and operational performance outcomes from the company perspective [e.g., 19, 20, 29, 30-32]. In this vein, large-scale hypothesis-testing studies on the effects of product configurator use on a firm's operational performance have recently appeared as well [33, 34].

Instead, less attention has been given in literature to which characteristics of sales configurators reduce the effort involved in the specification process and drive users' satisfaction with this process [14], thereby alleviating the risk that companies experience the product variety paradox [12]. In particular, the empirical study of how sales configurators should be designed to ease the customer decision process and to increase configuration process-related value for the customer is still in its infancy [14, 35]. To help narrow this research gap, the present paper conceptualizes, develops and validates five sales configurator capabilities that are expected to motivate and facilitate further empirical investigation in the field.

2 BACKGROUND

Literature has suggested several mechanisms that can explain the product variety paradox [11]. In particular, four inter-related mechanisms link product variety and customization to the difficulty experienced by potential customers in configuring the product solutions that best fit their needs within a company's product space. Difficulty in the decision process may become a criterion for the potential customer's evaluation of the decision outcome itself [9, 11, 36, 37], leading to lower satisfaction with the

[1] Department of Management and Engineering, University of Padova, Padova, Italy, email: cipriano.forza@unipd.it

configured products and, eventually, reduced willingness to make a purchase [9, 11].

A first explanation for the product variety paradox relies on choice complexity, defined as the amount of information processing necessary to make a decision [9]. As product variety and customization increase, so too does choice complexity, since more alternatives have to be processed in order for a potential customer to make a decision based on rational optimization. The amount of information processing is a widely acknowledged source of decision difficulty [38]. If potential customers are provided with "too much" information at a given time, such that it exceeds their processing limits, information overload occurs [39]. Information overload, in turn, may lead potential customers to choose from competing brands that do not require such cognitive effort [5] thus reducing the company's revenues.

A related explanation for the product variety paradox relies on anticipation of post-decisional regret, which is a cognitively determined negative emotion that individuals experience when realizing or imagining that their present situation would have been better, had they acted differently [40]. When choice complexity becomes excessive, potential customers may become unable to invest the requisite time and effort in seeking the best option for them, thus basing their decision on heuristics which reduce information processing demands by ignoring potentially relevant information [38, 41, 42]. Furthermore, potential customers may have uncertain preferences because of poorly developed preferences or poor insight into their preferences [42-44]. When potential customers are unable to engage in rational optimization and/or have uncertain preferences, they may anticipate the possibility of post-decisional regret, due to poor fit between the selected product configuration and their preferences [7, 8, 45], and try to minimize this possibility during the decision process [8, 45]. This goal makes their decision processes more difficult [7] and may lead them to delay their purchase decisions [7, 45] or to prefer a standard product to a customized one [8].

A third related explanation for the product variety paradox relies on responsibility felt by potential customers for making a good decision. As product variety and customization increase, potential customers feel more responsible for their choices, given the greater opportunity of finding the very best option for them [7, 11]. These enhanced feelings of responsibility promote anticipated regret, as subjectively important decisions, for which individuals feel more responsible, will result in more intense post-decisional regret when things go awry [40, 45]. By amplifying anticipated regret and the resulting decision difficulty, responsibility for making a good decision magnifies the negative impact of choice complexity on customers' willingness to make a purchase.

Finally, a fourth mechanism relating product variety and customization to decision difficulty relies on conflict between product attributes that are highly valued by potential customers [5, 9, 38, 46]. To increase product variety and customization, companies need to broaden the range of product attributes on which they allow their potential customers to make a choice [47]. As the number of product-differentiation attributes increases, so too does the likelihood that potential customers have to face trade-offs among attractive attributes. This happens because offering all the possible combinations of all the different levels of the various product-differentiation attributes may be economically unfeasible, owing to insufficient manufacturing process flexibility and limited product modularity [48]. Explicit trade-offs among attractive attributes not only increase the cognitive effort required of potential customers to process all of the available information [5], but also cause potential customers to experience negative emotions such as anticipated regret [5]. This happens because trade-off resolution involves consideration of potential unwanted consequences and threatens one's reputation of self-esteem as a decision maker [49]. These negative emotions are another mechanism that increase subjective experience of choice task difficulty [9] and decreased satisfaction with the chosen product [11], thus explaining the product variety paradox.

3 CONSTRUCT DEVELOPMENT

In the following subsections, we propose five sales configurator capabilities that help companies avoid the product variety paradox by hindering operation of at least one of the mechanisms outlined in the previous section. These capabilities were identified based on a comprehensive literature review and the authors' experience in the design and implementation of product configurators.

3.1 Focused navigation capability

We define focused navigation capability as the ability to quickly focus a potential customer's search on a product space subset that contains the product configuration that best matches his/her idiosyncratic needs. A fundamental way of improving focused navigation capability is to allow potential customers to sequence their choices on product-differentiation attributes from the least uncertain choice to the most uncertain one [12]. This is because, in relation to the attribute being considered, a customer's preferences may be more or less uncertain [43] and preference uncertainty is an antecedent of anticipated regret [8, 50]. If the customer's early choices are those for which his/her preferences are best developed, then he/she is enabled to narrow down the search more quickly, as anticipated regret associated with those choices is lower. Noteworthy, a prerequisite for this way of structuring the customer-company interaction is the by-attribute presentation of the company's product space, meaning that the customer is asked which value he/she prefers for each product-differentiation attribute instead of being required to choose from among a set of fully-specified product configurations, as happens with the by-alternative presentation [6]. Another option to enhance focused navigation capability is to provide one or more starting points, that is, initial product configurations close to the customer's ideal solution and that may be further altered [13]. Starting points can be recommended with little or no effort on the customer's part, based on his/her past purchases and/or customer input concerning simple demographics, intended product usage and his/her best developed preferences [26, 51]. Noteworthy, this solution requires complementing the by-attribute presentation of the product space with the by-alternative presentation.

Focused navigation capability helps avoid the product variety paradox by reducing choice complexity and by mitigating anticipated regret. A sales configurator with this capability does not force potential customers to go through and evaluate a number of product options that they regard as certainly inappropriate for themselves. Therefore, this capability reduces the amount of information processing necessary to make a decision without potential customers experiencing anticipated regret [8, 40, 45, 50]. Furthermore, by quickly reducing the size of the search problem, this capability enables potential customers to invest more time and effort in exploring the product options for which their preferences are less certain. Potential customers can learn more about both these options and the value they would derive from them, especially when focused navigation capability is complemented with the capabilities discussed in the subsequent sections. In addition, a potential customer can rely on more time-consuming, compensatory decision strategies for the resolution of between-attribute conflicts [42], thus being more confident that the chosen solution is the one that best fits his/her needs within the company's product space. Reduced uncertainty on the superior fit of the selected product configuration with the customer's preferences, in turn, translates into less anticipated regret [45].

3.2 Benefit-cost communication capability

We define benefit-cost communication capability as the ability to effectively communicate the consequences of the available choice options both in terms of what the customer gets (benefits) and in terms of what the customer gives (monetary and nonmonetary costs). A fundamental way of improving benefit-cost communication capability is to explain what potential needs a given choice option contributes to fulfill and to what extent it does so [12]. This is especially important when choice options involve design parameters of the product, such as specifications of product components, because potential customers are often unable to relate design parameters to satisfaction of user needs [13]. Besides the benefits, it is also important to communicate monetary and nonmonetary costs of each option, for example by displaying the prices of the individual product components from among which potential customers can choose or by warning potential customers that certain options imply longer delivery lead-times [12].

Benefit-cost communication capability helps avoid the product variety paradox by mitigating anticipated regret. During the sales configuration process, potential customers seek to anticipate the value they will perceive from consumption of the product being configured [54]. Perceived product value is defined as the customer's "overall assessment of the utility of a product based on perceptions of what is received and what is given" [55: 14]. By delivering clear pre-purchase feedback on the effects of the available choice options, a sales configurator with high benefit-cost communication capability fosters potential customers' learning about the value they would derive from these options [56, 57]. This learning process makes a potential customer more confident that the product configuration he/she has selected is the one that best fits his/her needs within the company's product space. Reduced uncertainty on the superior fit of the chosen product configuration with the customer's preferences, in turn, translates into less anticipated regret [45], thus lowering choice task difficulty [7].

At the same time, however, higher benefit-cost communication capability may lead to greater choice complexity, with negative effects on decision difficulty. For instance, individual pricing of the available choice options may make cost-benefit trade-offs more salient and, hence, may increase information processing demands [58]. To fully realize the potential advantages of benefit-cost communication capability, therefore, this capability needs to be complemented with the focused navigation one, which lowers choice complexity by quickly reducing the size of the search problem for potential customers. As a result, the learning process enabled by benefit-cost communication capability focuses only on those choice options for which potential customers' preferences are less certain and, thus, the possible negative effects of this capability on choice complexity are mitigated.

3.3 Flexible navigation capability

We define flexible navigation capability as the ability to minimize the effort required of a potential customer to modify a product configuration that he/she has previously created or is currently creating. A fundamental way of improving flexible navigation capability is to allow sales configurator users to change the choice made at any previous step of the configuration process without having to start it over again [13]. Furthermore, after changing the choice made at a given step, potential customers should not be required to go through all the subsequent steps up to the current one. Instead, they should be asked to revise only those choices, if any, that are no longer valid because of the change they have just made [59]. Another option to enhance flexible navigation capability is to allow potential customers engaged in configuring their products to bookmark their works [13],to immediately recover a previous configuration in the case that they decide to reject the newly-created one.

Flexible navigation capability helps avoid the product variety paradox by mitigating anticipated regret. A sales configurator with this capability enables potential customers to quickly make and undo changes to previously created product configurations. Consequently, the number of product solutions a potential customer can explore in the time span he/she is willing to devote to the sales configuration task is larger. Stated otherwise, potential customers can conduct more trial-and-error tests to evaluate the effects of initial choices made and to improve upon them. Trial-and-error experimentation promotes potential customers' learning about the value they would derive from the product being configured [56, 57], especially when flexible navigation capability is complemented with the benefit-cost communication one as well as those discussed in the subsequent sections. This learning process makes potential customers more confident that the product configuration they have selected is the one that best fits their needs within the company's product space. This, in turn, translates into less anticipated regret for the customer [45].

3.4 Easy comparison capability

We define easy comparison capability as the ability to minimize the effort required of a potential customer to compare previously created product configurations. A fundamental way of improving easy comparison capability is to allow potential customers to save a product configuration they have just created and, then, to compare previously saved configurations side-by-side in the same screen [13]. The advantages of providing an overview of previous configurations can be enhanced by highlighting commonalities and differences among them, especially if the sales configuration process involves many choices. In this manner, a potential customer can immediately understand, for example, which configuration choices have caused the price or weight difference between two configurations he/she is comparing. Another solution to enhance easy comparison capability is to rank-order previously created configurations in terms of fit to the customer's preferences or profile [43]. This can be accomplished with little or no effort on the customer's part, based on his/her past purchases and/or customer input concerning simple demographics, intended product usage and his/her best developed preferences [26, 51].

Easy comparison capability helps avoid the product variety paradox by reducing choice complexity and by mitigating anticipated regret. A sales configurator with this capability fosters potential customers' learning about the value they would derive from consumption of the product being configured. This happens because, in assessing the value of a particular product solution, customers tend to rely on comparisons with other alternatives that are currently available or that have been encountered in the past [43, 60]. In particular, the possibility of easily comparing complete product configurations is of greatest assistance when global performance characteristics, which arise from the physical properties of most if not all of the product components [48], are important to potential customers. In brief, easy comparison capability gives potential customers practice at evaluating alternative configurations and provides anchors for the evaluative process [6]. Consequently, potential customers improve their confidence that the configuration they have eventually selected is the one that best fits their needs within the company's product space. In turn, reduced uncertainty on the superior fit of the chosen product configuration with the customer's preferences translates into less anticipated regret [45]. A sales configurator with high easy comparison capability also alleviates choice complexity, by reducing information processing necessary to make comparisons. Potential customers do not need to rely on their limited working memory to recover configurations they have previously created. Moreover, potential customers do not need to rely on their limited computational abilities to decompose the configurations they want to compare to find out similarities and differences among them.

3.5 User-friendly product-space description capability

We define user-friendly product-space description capability as the ability to adapt the product space description to the needs and abilities of different potential customers, as well as to different contexts of use. One way of improving user-friendly product-space description capability is to employ content adaptation techniques [cf. 61] to provide optional detailed information pertaining to the available choice options. In this manner, potential customers with higher involvement for the product, who are more interested in acquiring product information [62], are allowed to learn more about the choice options for which their preferences are less developed. Conversely, customers with lower involvement, who feel less responsible for making a good decision [45], are not forced to process product information they are not interested in. In this respect, a promising approach is to design multimedia-based interfaces that enable potential customers to retrieve rich information and explanations about specific product parts/features without breaking the continuity of their product evaluation processes [63]. Another option to enhance user-friendly product-space description capability is to adapt information content presented to potential customers according to their prior knowledge about the product [13, 52]. Particularly, novice customers should be allowed to use a needs-based interface, where the available choice options involve desired product performance and functions, while expert customers should be enabled to employ a parameter-based interface, where the available choice options include design parameters such as specifications of product components [12, 64].

User-friendly product-space description capability helps avoid the product variety paradox by reducing choice complexity and by mitigating anticipated regret. A sales configurator deploying this capability provides potential customers with the information content they value most according to their individual characteristics or usage contexts and does not bother users with communications they do not need [52]. In addition, a sales configurator with this capability augments or switches modalities of presentation of the same information content in such a way that each individual user's information processing is enhanced [67]. By tailoring both information content and information format, this capability reduces information overload and eases the customer decision process [68-70]. In particular, this capability allows for aligning the way in which the product space is presented to a potential customer with the way in which he/she is able or willing to express his/her requirements [56, 57]. As potential customers interact with a sales configurator in their customary language, they become able to assess the fit of the configured product with their needs more easily and in less time [71]. This means that, once a potential customer has selected his/her most preferred product configuration, he/she is more confident that the chosen solution is the one that best fits his/her needs within the company's product space. Reduced uncertainty on the superior fit of the selected product configuration with the customer's preferences, in turn, translates into less anticipated regret [45].

4 MEASURES DEVELOPMENT AND VALIDATION

We adopted a comprehensive, multi-step approach for the development, refinement and validation of the sales configurator capabilities measures. First, we generated a list of items based on both the relevant literature and subject matter experts' advice in order to ensure content validity of our instrument. Then, these items were reviewed by a focus group and through a field pretest, to reduce redundancy and ambiguity. Subsequently, we assessed and improved the reliability and the validity of the instrument by means of a Q-sort procedure. Finally, the resulting questionnaire (items are listed in Appendix A) was used to validate our measures, using large-scale data to assess the quality of the measures following the guidelines of O'Leary-Kelly and Vokurka [72].

4.1 Instrument development and refinement

The items for the five sales configurator capabilities were generated based upon the relevant literature, the authors' experience in industry, and extensive interviews with practitioners involved with the development and use of sales configurators. All the items were measured by means of a 7-point Likert scale. We used only positive statements, as negatively worded questions with an agree-disagree response format are often cognitively complex [73] and may be a source of method bias [74].

Then, the items were reviewed by a focus group of six people with different experiences and perceptions relative to sales configuration, who were questioned about the appropriateness and completeness of the instrument. Moreover, to replicate as closely as possible data collection procedures to be used in our large-scale study, we pretested the instrument with 20 engineering students from our university, who were asked to comment on any problems encountered while responding, such as interpretation difficulties, faulty instructions, typos, item redundancies, etc. Based on the feedback from the focus group and field pretesting, redundant and ambiguous items were either modified or eliminated. Finally, the resulting instrument was evaluated through a Q-sort procedure for establishing tentative indications of construct validity and reliability [75]. Each of ten practitioners who are experienced in developing or using sales configurators was given a questionnaire containing short descriptions of the proposed capabilities, together with a randomized list of the items. Subsequently, these expert judges were asked to assign each item to one or none of the defined capabilities. All the items were placed in the target construct by at least 75% of the judges and, therefore, were retained for our large-scale study [54].

4.2 Sample and data collection

Each of the proposed sales configurator capabilities indicates a fundamental benefit that potential customers should experience during the sales configuration process if the product variety paradox is to be avoided. Consistent with the capability perspective of routines, which sees routines as a "black box" [76], we do not focus on how such benefits are delivered, but rather on their purpose or motivation. Accordingly, to measure the proposed sales configurator capabilities, we needed to collect data on sales configurations experiences made by potential customers using sales configurators. Specifically, data for our large-scale study were gathered on a sample of 630 sales configuration experiences made by 63 engineering students at the authors' university (age range: 24-27; 29% females) using Web-based sales configurators for consumer goods. As a result, our data are biased in favor of young, male, and fairly adept persons who are familiar with the Internet. At the same time, however, young people adept at using Internet also represent the majority of business-to-consumer sales configurator users [35, 78].

The Web-based sales configurators used in the study largely varied in the graphical solutions deployed, in the complexity and length of the configuration process, and also in the size of the configuration space. They ranged from shoes configurators, where the customer could personalize simple product attributes (such as the colors of various parts of the product) with virtually no constrains, to cars configurators, where the customer had to choose among a set of predefined options with complex compatibility rules among them. Such differences in the selected sales configurators increased the variance of the sales configurators capabilities observed in our sample.

Each participant was pre-assigned 10 of these Web-based sales configurators. We assigned these configurators ensuring variance in the sales configurators capabilities to which each participant was exposed. Further, we ensured variance in the involvement of each participant in the products he/she had to configure, avoiding the assignment of products not of interest to him/her at all. Participants were then asked to configure a product on all these websites, according to their individual needs, and to fill out a questionnaire to rate the capabilities of each configurator.

4.3 Instrument validation

We decided to control for possible effects of participants' characteristics before assessing the psychometric properties of our measurement scales. Consequently, consistent with prior studies [79], we regressed our 17 indicators on 63 dummies representing the participants in our study and used the standardized residuals from this linear, ordinary least square regression model as our data in all the subsequent analyses.

Confirmatory factor analysis (CFA) was employed to assess unidimensionality, convergent validity, discriminant validity, and reliability of our measurement scales. In particular, we used LISREL 8.80 to conduct the analysis, with maximum likelihood estimation of the parameters in the model (factor loadings of the measurement items on their respective latent constructs, measurement errors, variance and covariance of the latent constructs). We estimated an a priori measurement model where the empirical indicators were restricted to load on the latent factor they were intended to measure. This model showed good fit indices (RMSEA (90% CI)= 0.047 (0.040; 0.054), χ^2/df (df) = 2.39 (109), CFI=0.991, NFI=0.984), meaning that our hypothesized factor structure reproduced the sample data well. Inspection of the standardized factor loadings further indicated that each of them was in its anticipated direction (i.e., positive correspondences between latent constructs and their posited indicators), was greater than 0.50, and was statistically significant at $p<0.001$. Altogether, these results suggested unidimensionality and good convergent validity of our measurement scales [80-83]. Unidimensionality implies that a set of empirical indicators reflect one, as opposed to more than one, underlying latent factor. Convergent validity ensures that the multiple items used as indicators of a construct significantly converge, or covary. Discriminant validity, which measures the extent to which the individual items of a construct are unique and do not measure other constructs, was tested using [84]'s procedure. For each latent construct, the square root of the average variance extracted (AVE) exceeded the correlation with all the other latent variables, thereby suggesting that our measurement scales represent distinct latent variables [84]. Reliability of a measurement scale, in turn, is established when the variance captured by the underlying latent factor is significantly larger than that captured by the error components. This was assessed using both AVE and the Werts, Linn, and Joreskog (WLJ) composite reliability method [85]. All the WLJ composite reliabilty values were greater than 0.70 and all the AVE scores exceeded 0.50, indicating that a large amount of the variance is captured by each latent construct rather than due to measurement error [84, 86].

Finally, we examined the predictive validity of our constructs by determining whether they exhibit relationships with other constructs in accordance with theory [87]. Our proposed sales configurator capabilities are posited to help firms avoid the risk that offering more product variety and customization to increase sales, paradoxically results in a loss of sales. Accordingly, these capabilities are hypothesized to positively influence both choice satisfaction (measured as in [9]) and purchase intention (measured following [88]). The structural model testing the hypotheses that the proposed sales configurator capabilities positively influence both choice satisfaction and purchase intention, showed a good fit to the data: RMSEA (90% CI) = 0.0432 (0.0372; 0.0493), χ2/df (df) = 2.18 (169), CFI=0.993, NFI=0.987. All the path coefficients are positive and statistically significant, indicating that each of the five sales configurator capabilities has a significant positive effect on both choice satisfaction and purchase intention and thus establishing the predictive validity of our constructs.

5 CONCLUSION

Drawing upon prior research concerning sales configurators and the customer decision process, the present paper conceptualizes five capabilities that sales configurators should deploy in order to help avoid the product variety paradox: namely, focused navigation, flexible navigation, easy comparison, benefit-cost communication, and user-friendly product-space description capabilities. Overall, these capabilities support personalization of the sales configuration experience according to each individual user's characteristics and context of usage. Benefit-cost communication capability combined with user-friendly product-space description capability supports personalization on the content and presentation levels [cf. 89], while focused navigation, flexible navigation, and easy comparison capabilities support personalization on the interaction level [cf. 89]. Personalization of the sales configuration experience is essential to build successful sales configurators, which improve fit between selected product configuration and customer needs while limiting search effort [cf. 89, 90]. The ultimate goal would be to simulate the adaptive and heuristic behavior that makes salespeople effective and aids in improving both the shopping experience and the final product choice [91, 92].

Another contribution of this study is the development and validation of an instrument to measure the proposed set of capabilities. The instrument was rigorously tested for content validity, unidimensionality, convergent validity, discriminant validity, predictive validity, and reliability. In particular, we found that each of the proposed capabilities significantly predicts both choice satisfaction and purchase intention, in accord with the theoretical argument that these capabilities help avoid the product variety paradox. Admittedly, our large-scale validation study involved hypothetical rather than real purchase experiences, only focused on sales configurators for consumer goods, and used students as subjects for research. Therefore, future studies should strengthen the proposed instrument through a series of refinements and tests across different populations and settings, including truly representative samples of potential customers, sales configurators for industrial goods, etc. In business-to-business contexts, for instance, the set of relevant sales configurator capabilities for avoiding the product variety paradox should be reconsidered. For technical and complex products, such as machinery, it may happen that all configurator users are experts with deep knowledge of the specific product. In such a context, user-friendly product-space description capability might be less relevant.

Though conscious that development of a measurement instrument is an ongoing process [93], we believe our instrument will be a useful diagnostic and benchmarking tool for companies seeking to assess their sales configurators to identify areas of improvement in order to ease the customer decision process and to increase his/her process-related value. This would help companies reduce the risk of developing high product and processes internal competences but still experiencing a loss of sales because customers feel confused and overwhelmed by the number of product configurations they are offered.

Further, we believe the instrument developed in this paper will be of use to researchers not only as a basis for refinement and extension, but also directly. Future studies could develop and test hypotheses linking the proposed capabilities to the various dimensions of the value of customization that have been discussed in literature [35, 54, 78]. In particular, further research is needed to empirically investigate complementarities among the proposed capabilities, meaning that the effects of one capability on the customer perceived value of customization is reinforced by another capability, as our paper suggests.

ACKNOWLEDGEMENTS

We acknowledge the financial support of the University of Padova, Project ID CPDA109359.

APPENDIX A

Benefit-cost communication capability: (1) Thanks to this system, I understood how the various choice options influence the value that this product has for me. (2)Thanks to this system, I realized the advantages and drawbacks of each of the options I had to choose from. (3) This system made me exactly understand what value the product I was configuring had for me.

Easy comparison capability: (1) The system enables easy comparison of product configurations previously created by the user. (2) The system lets you easily understand what previously created configurations have in common. (3) The system enables side-by-side comparison of the details of previously saved configurations. (4) The systems lets you easily understand the differences between previously created configurations.

User-friendly product-space description capability: (1) The system gives an adequate presentation of the choice options for when you are in a hurry, as well as when you have enough time to go into the details. (2) The product features are adequately presented for the user who just wants to find out about them, as well as for the user who wants to go into specific details. (3) The choice options are adequately presented for both the expert and inexpert user of the product.

Flexible navigation capability: (1) The system enables you to change some of the choices you have previously made during the configuration process without having to start it over again. (2) With this system, it takes very little effort to modify the choices you have previously made during the configuration process. (3) Once you have completed the configuration process, this system enables you to quickly change any choice made during that process.

Focused navigation capability: (1) The system made me immediately understand which way to go to find what I needed. (2) The system enabled me to quickly eliminate from further consideration everything that was not interesting to me at all. (3) The system immediately led me to what was more interesting to me. (4) This system quickly leads the user to those solutions that best meet his/her requirements.

REFERENCES

[1] B.J.II Pine, *Mass Customization: the New Frontier in Business Competition*, Harvard Business School Press, Boston, MA, 1993.

[2] L.F. Scavarda, A.Reichhart, S.Hamacher, and M. Holweg, 'Managing product variety in emerging markets', *International Journal of Operations & Production Management*, **30**, 205-224, (2010).

[3] M. Bils and P.J. Klenow, ''The acceleration in variety growth', *The American Economic Review*, **91**, 274-280, (2001).

[4] S. Kekre and K. Srinivasan, 'Broader product line: a necessity to achieve success?', *Management Science*, **36**, 1216-1231, (1990).

[5] J.T. Gourville and D. Soman, 'Overchoice and assortment type: when and why variety backfires', *Marketing Science*, **24**, 382-395, (2005).

[6] C. Huffman and B.E. Kahn, 'Variety for sale: mass customization or mass confusion?', *Journal of Retailing*, **74**, 491-513, (1998).

[7] S.S. Iyengar and M.R. Lepper, 'When choice is demotivating: can one desire too much of a good thing?', *Journal of Personality and Social Psychology*, **79**, 995-1006, (2000).

[8] N. Syam, P. Krishnamurthy, and J.D. Hess, 'That's what i thought i wanted? Miswanting and regret for a standard good in a mass-customized world', *Marketing Science*, **27**, 379-397, (2008).

[9] A. Valenzuela, R. Dhar, and F. Zettelmeyer, 'Contingent response to self-customization procedures: implications for decision satisfaction and choice', *Journal of Marketing Research*, **46**, 754-763, (2009).

[10] X. Wan, P.T. Evers, and M.E. Dresner, 'Too much of a good thing: the impact of product variety on operations and sales performance', *Journal of Operations Management*, **30**, 316-324, (2012).

[11] K. Diehl and C. Poynor, 'Great expectations?! Assortment size, expectations, and satisfaction', *Journal of Marketing Research*, **47**, 312-322, (2010).

[12] F. Salvador and C. Forza, 'Principles for efficient and effective sales configuration design', *International Journal of Mass Customisation*, **2**, 114-127, (2007).

[13] T. Randall, C. Terwiesch, and K.T. Ulrich, 'Principles for user design of customized products', *California Management Review*, 47, 68-85, (2005).

[14] M. Heiskala, J. Tiihonen, K.-S. Paloheimo, and T. Soininen, *Mass customization with configurable products and configurators: a review of benefits and challenges*, 1-32, in: Mass Customization Information Systems in Business, T. Blecker, G. Friedrich (Eds.), IGI Global, London, UK, 2007.

[15] C. Forza and F. Salvador, 'Application support to product variety management', *International Journal of Production Research*, **46**, 817-836, (2008).

[16] A. Haug, L. Hvam, and N.H. Mortensen, 'Definition and evaluation of product configurator development strategies', *Computers in Industry*, (in press).

[17] M.M. Tseng and T.F. Piller, *The Customer Centric Enterprise: Advances in Mass Customization and Personalization*, Springer Verlag, Berlin, Germany, 2003.

[18] N. Franke and F.T. Piller, 'Key research issues in user interaction with user toolkits in a mass customization system', *International Journal of Technology Management*, **26**, 578-599, (2003).

[19] J. Vanwelkenhuysen, 'The tender support system', *Knowledge-based systems*, **11**, 363-372, (1998).

[20] L. Hvam, S. Pape, and M.K. Nielsen, 'Improving the quotation process with product configuration', *Computers in Industry*, **57**, 607-621, (2006).

[21] C. Forza and F. Salvador, *Product Information Management for Mass Customization*, Palgrave Macmillan, London, UK, 2007.

[22] S.M. Fohn, J.S. Liau, A.R. Greef, R.E. Young, and P.J. O'Grady, 'Configuring computer systems through constraint-based modeling and interactive constraint satisfaction', *Computers in Industry*, **27**, 3-21, (1995).

[23] T. Soininen, J. Tiihonen, T. Männistö, R. Sulonen 'Towards a general ontology of configuration', *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing*, **12**, 357-372, (1998).

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

[25] S.K. Ong, Q. Lin, and A.Y.C. Nee, 'Web-based configuration design system for product customization', *International Journal of Production Research*, 44, 351-382, (2006).

[26] X. Luo, Y. Tu, J. Tang, and C.K. Kwong, 'Optimizing customer's selection for configurable product in B2C e-commerce application', *Computers in Industry*, **59**, 767-776, (2008).

[27] P.T. Helo, Q.L. Xu, S.J. Kyllönen, and R.J. Jiao, 'Integrated vehicle configuration system-Connecting the domains of mass customization', *Computers in Industry*, **61**, 44-52, (2010).

[28] G. Hong, D. Xue, and Y. Tu, 'Rapid identification of the optimal product configuration and its parameters based on customer-centric modeling for one-of-a-kind production', *Computers in Industry*, **61**, 270-279, (2010).

[29] J.R. Wright, E.S. Weixelbaum, G.T. Vesonder, K.E. Brown, S.R. Palmer, J.I. Berman, and H.H. Moore, 'A knowledge-based configurator that supports sales, engineering, and manufacturing at AT&T network systems', *AI Magazine*, **14**, 69-80, (1993).

[30] L. Hvam, 'Mass customisation in the electronics industry: based on modular products and product configuration', *International Journal of Mass Customisation*, **1**, 410-426, (2006).

[31] C. Forza and 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).

[32] C. Forza and F. Salvador, 'Product configuration and inter-firm co-ordination: an innovative solution from a small manufacturing enterprise', *Computers in Industry*, **49**, 37-46, (2002).

[33] 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).

[34] A. Trentin, E. Perin, C. Forza, "Organisation design strategies for mass customisation: an information-processing-view perspective", *International Journal of Production Research*, forthcoming.

[35] N. Franke and M. Schreier, 'Why customers value mass-customized products: the importance of process effort and enjoyment', *Journal of Product Innovation Management*, **27**, 1020-1031, (2010).

[36] G.J. Fitzsimons, 'Consumer response to stockouts', *Journal of Consumer Research*, **27**, 249-266, (2000).

[37] N. Novemsky, R. Dhar, N. Schwarz, I. Simonson, 'Preference fluency in choice', *Journal of Marketing Research*, **44**, 347-356, (2007).

[38] S. Chatterjee and T.B. Haeth, 'Conflict and loss aversion in multiattribute choice: the effects of trade-off size and reference dependence on decision difficulty', *Organizational Behavior and Human Decision Processes*, **67**, 144-155, (1996).

[39] N.K. Malhotra, 'Information load and consumer decision making', *Journal of Consumer Research*, **8**, 419-430, (1982).

[40] M. Zeelenberg, W.W. van Dijk, and A.S.R. Manstead, 'Reconsidering the relation between regret and responsibility', *Organizational Behavior and Human Decision Processes*, **74**, 254-272, (1998).

[41] J.W. Payne, J.R. Bettman, and E.J. Johnson, 'Adaptive strategy selection in decision making', *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **14**, 534-552, (1988).

[42] J.R. Bettman, M.F. Luce, and J.W. Payne, 'Constructive consumer choice processes', *Journal of Consumer Research*, **25**, 187-217, (1998).

[43] I. Simonson, 'Determinants of customers' responses to customized offers: conceptual framework and research propositions', *Journal of Marketing*, **69**, 32-45, (2005).

[44] I. Simonson, 'Regarding inherent preferences', *Journal of Consumer Psychology*, **18**, 191-196, (2008).

[45] M. Zeelenberg, 'Anticipated regret, expected feedback and behavioral decision making', *Journal of Behavioral Decision Making*, **12**, 93-106, (1999).

[46] R. Dhar, 'Consumer preference for a no-choice option', *Journal of Consumer Research*, **24**, 215-231, (1997).

[47] F. Salvador, C. Forza, and M. Rungtusanatham, 'Modularity, product variety, production volume, and component sourcing: theorizing beyond generic prescriptions', *Journal of Operations Management*, **20**, 549-575, (2002).

[48] K. Ulrich, 'The role of product architecture in the manufacturing firm', *Research Policy*, **24**, 419-440, (1995).

[49] M.F. Luce, 'Choosing to avoid: coping with negatively emotion-laden consumer decisions', *Journal of Consumer Research*, **24**, 409-433, (1998).

[50] J. Nasiry and I. Popescu, 'Advance selling when consumers regret', *Management Science*, (in press).

[51] A. De Bruyn, J.C. Liechty, E.K.R.E. Huizingh, and G.L. Lilien, 'Offering onlinerecommendations with minimum customer input through conjoint-based decision aids', *Marketing Science*, **27**, 443-460, (2008).

[52] S. Spiekermann and C. Parashiv, 'Motivating human-agent interaction: transferring insights from behavioral marketing to interface design', *Electronic Commerce Research*, **2**, 255-285, (2002).

[53] A.G. Sutcliffe, S. Kurniawan, and J.-E. Shin, 'A method and advisor tool for multimedia user interface design', *International Journal of Human-Computer Studies*, **64**, 375-392, (2006).

[54] A. Merle, J.-L. Chandon, E. Roux, and F. Alizon, 'Perceived value of the mass-customized product and mass customization experience for individual consumers', *Production and Operations Management*, **19**, 503-514, (2010).

[55] V. Zeithaml, 'Consumer perceptions of price, quality, and value: a means-end model and synthesis of evidence', *Journal of Marketing*, **52**, 2-22, (1988).

[56] E. von Hippel, 'PERSPECTIVE: User toolkits for innovation', *Journal of Product Innovation Management*, **18**, 247-257, (2001).

[57] E. von Hippel and R. Katz, 'Shifting innovation to users via toolkits', *Management Science*, **48**, 821-833, (2002).

[58] B.G.C. Dellaert and S. Stremersch, 'Marketing mass-customized products: striking a balance between utility and complexity', *Journal of Marketing Research*, **42**, 219-227, (2005).

[59] B. Yu and J. Skovgaard, 'A configuration tool to increase product competitiveness', *IEEE Intelligent Systems*, **13**, 34-41, (1998).

[60] I. Simonson and A. Tversky, 'Choice in contexts: tradeoff contrasts and extremeness aversion', *Journal of Marketing Research*, **29**, 281-295, (1992).

[61] A. Kobsa, J. Koenemann, and W. Pohl, 'Personalised hypermedia presentation techniques for improving online customer relationships', *The Knowledge Engineering Review*, **16**, 111-155, (2001).

[62] J.L. Zaichkowsky, 'Measuring the involvement construct', *Journal of Consumer Research*, **12**, 341-352, (1985).

[63] Z. Jiang, W. Wang, and I. Benbasat, 'Multimedia-based interactive advising technology for online consumer decision support', *Communications of the ACM*, **48**, 93-98, (2005).

[64] T. Randall, C. Terwiesch, and K.T. Ulrich, 'Principles for user design of customized products', *California Management Review*, 47, 68-85, (2005).

[65] J.H. Gerlach and F.-Y. Kuo, 'Understanding human-computer interaction for information systems design', *MIS Quarterly*, **15**, 527-549, (1991).

[66] L.M. Reeves, J.Lai, J.A.Larson, S.Oviatt, T.S. Balaji, S. Buisine, P. Collings, P. Cohen, B. Kraal, J.-C. Martin, M. McTear, T. Raman, K.M. Stanney, H. Su, and Q.-Y. Wang, 'Guidelines for multimodal user interface design', *Communications of the ACM*, **47**, 57-59, (2004).

[67] K. Stanney, S. Samman, L. Reeves, K. Hale, W. Buff, C. Bowers, B. Goldiez, D. Nicholson, and S. Lackey, 'A paradigm shift in interactive computing: deriving multimodal design principles from behavioral and neurological foundations', *International Journal of Human-Computer Interaction*, **17**, 229-257, (2004).

[68] A. Ansari and C.F. Mela, 'E-customization', *Journal of Marketing Research*, **40**, 131-145, (2003).

[69] T.-P. Liang, H.-J. Lai, and Y.-C. Ku, 'Personalized content recommendation and user satisfaction: theoretical synthesis and empirical findings', *Journal of Management Information Systems*, **23**, 45-70, (2006-7).

[70] H. Berghel, 'Cyberspace 2000: Dealing with information overload', *Communications of the ACM*, **40**, 19–24, (1997).

[71] T. Randall, C. Terwiesch, and K.T. Ulrich, 'User design of customized products', *Marketing Science*, **26**, 268-280, (2007).

[72] S. W. O'Leary-Kelly, J. R. Vokurka, "The empirical assessment of construct validity", *Journal of Operations Management*, **16**, 387-405, (1998).

[73] F.J. Fowler, *Survey Research Methods*, Sage Publications, Newbury Park, CA, 1993.

[74] H.W. Marsh, 'Positive and negative global self-esteem: a substantively meaningful distinction or artifactors?' *Journal of Personality and Social Psychology*, **70**, 810-819, (1996).

[75] J.K. Stratman and A.V. Roth, 'Enterprise Resource Planning (ERP) competence constructs: two-stage multi-item scale development and validation', *Decision Sciences*, **33**, 601-628, (2002).

[76] A. Parmigiani and J. Howard-Grenville, 'Routines revisited: exploring the capabilities and practice perspectives', *The Academy of Management Annals*, **5**, 413-453, (2011).

[77] L. D'Adderio, 'Configuring software, reconfiguring memories: the influence of integrated systems on the reproduction of knowledge and routines', *Industrial and Corporate Change*, **12**, 321-350, (2003).

[78] N. Franke and M. Schreier, 'Product uniqueness as a driver of customer utility in mass customization', *Marketing Letters*, **19**, 93-107, (2008).

[79] G.J. Liu, R. Shah, and R.G. Schroeder, 'Linking work design to mass customization: a sociotechnical systems perspective', Decision Sciences, **37**, 519-545, (2006).

[80] D.W. Gerbing and J.C. Anderson, 'An updated paradigm for scale development incorporating unidimensionality and its assessment', *Journal of Marketing Research*, **25**, 186-192, (1988).

[81] J.C. Anderson and D.W. Gerbing, 'Structural equation modeling in practice: a review and recommended two-step approach', *Psychological Bulletin*, **103**, 411-423, (1988).

[82] L. Menor and A.V. Roth, 'New service development competence in retail banking: Construct development and measurement validation', *Journal of Operations Management*, **25**, 825–846, (2007).

[83] J.F.J. Hair, R.E. Anderson, and R.L. Tatham, *Multivariate Data Analysis*, Macmillan Publishing Company, New York, 1992.

[84] C. Fornell and D.F. Larcker, 'Evaluating structural equation models with unobservable variables and measurement error', *Journal of Marketing Research*, **18**, 39-50, (1981).

[85] C.E. Werts, R.L. Linn, and K.G. Jöreskog, 'Intraclass reliability estimates: testing structural assumptions', *Educational & Psychological Measurement*, **34**, 25-33, (1974).

[86] S.W. O'Leary-Kelly and R. J. Vokurka, 'The empirical assessment of construct validity', *Journal of Operations Management*, **16**, 387-405, (1998).

[87] S. Li, S.S. Rao, T.S. Ragu-Nathan, and B. Ragu-Nathan, 'Development and validation of a measurement instrument for studying supply chain management practices', *Journal of Operations Management*, **23**, 618-641, (2005).

[88] A.E. Schlosser, T.B. White, and S.M. Lloyd, 'Converting Web site visitors into buyers: how Web site investment increases consumer trusting beliefs and online purchase intentions', *Journal of Marketing*, **70**, 133-148, (2006).

[89] G. Kreutler and D. Jannach, *Personalized needs acquisition in Web-based configuration systems*, 293-302, in: Mass Customization, Concepts - Tools - Realization, Proceedings of the International Mass Customization Meeting 2005 (IMCM'05), T. Blecker, G. Friedrich (Eds.), GITO-Verlag, Berlin, Germany, 2005.

[90] D. Jannach, A. Felfernig, G. Kreutler, M. Zanker, and G. Friedrich, *Research issues in knowledge-based configuration*, 221-236, in: Mass customization information systems in business, T. Blecker, G. Friedrich (Eds.), IGI Global, London, UK, 2007.

[91] D. Jannach and G. Kreutler, 'Rapid development of knowledge-based conversational recommender applications with advisor suite', *Journal of Web Engineering*, 6, 165-192, (2007).

[92] A.V. Lukas, G. Lukas, D.L. Klencke, and C. Nass, System and method for optimizing a product configuration, Patent Number US 7,505,921 B1, Finali Corporation, Westminster, CO (US), US, 2009.

[93] R.L. Hensley, 'A review of operations management studies using scale development techniques', *Journal of Operations Management*, 17, 343-358, (1999).

Author Index