Supporting Open Domains in Collaborative Modular Modeling

Albert Haag[1]

Abstract. In commercial practice, modeling configurable products is often collaborative and modular. The former refers to the fact that different teams in a company may work autonomously when contributing to a product model. The latter refers to the fact that some elements of a model may be intended for re-use in multiple product models. When maintaining such model fragments, the actual domains of the product properties, which are to be effectively applied in later configuration, may be only partially known to the modeler. For example, using a wildcard in a *variant table* (a list of valid combinations of product features) to express "anything goes" implies that an exact knowledge of the underlying domains is not important to the modeler. The same is true for *negative variant tables* that list exclusions. Domains are explicitly declared as open, when non-standard extensions of product features are foreseen – "additional values are allowed". This is meant to allow a sales-engineer to provide non-modeled features in an impromptu manner. In this short paper we propose a way to deal with open domains in configuration and present some results for maintaining arc-consistency with open domains. These results build on and extend insights from previous work [6, 13].

1 Introduction

Most current modeling of configurable products is based on the closed-world assumption that all problem variables and their domains are part of the model. This assumption is at the very heart of the definition of a *constraint satisfaction problem* (CSP), which postulates a finite set of variables, each with a finite domain [5][2]. However, there are practical situations, when the closed-world assumption does not hold. One example is an engineer-to-order scenario, where a sales-engineer is allowed to supply unforeseen *product features*[3] extempore during a sales configuration. The closed-world assumption is also limiting in modular modeling. The responsibility for modeling may be distributed among different teams, each working independently to produce a re-usable part of a model. Where products or solutions contain multiple components, each component might be modeled independently. Also, product features may be maintained centrally company-wide, not product by product. Central product features are typically subject to continual updates. Finally, modeling may be distributed subject to technical expertise. Constraints concerning electrical features require different expertise than those concerning hydraulics.

Extensional lists of feature combinations are important modeling elements that define variability in a very direct and intelligible manner. We refer to them as *variant tables* following the terminology of the *SAP Variant Configurator* (SAP VC) [2][4]. Variant table content may be updated frequently by product data engineers, decoupled from other changes to the model. These engineers have criteria for including or excluding feature combinations, but they do not need to know the complete product models.

This is a short paper based on observations about modeling in practice. The motivation and insights for needing to deal with open domains, which I look at here, come from discussions in the *Configuration Workgroup* (CWG) [3], an SAP centric user group, over the years. The paper discusses three cases where *open domains*[5] may need to be considered:

- Negative variant tables listing excluded feature combinations
- *Wildcards* (placeholders for allowing any feature)
- *Non-modeled*, impromptu product features[6]

This paper is organized as follows: Section 2 contains some background and notation. Section 3 is a reprise of the results in [6] concerning negative variant tables. Section 4 discusses the engineer-to-order requirement of allowing non-modeled features and feature combinations, introducing a symbol *'&other'* to refer to the potentially infinite set of non-modeled features of a product property. We refer to such a symbol as a *quasi-finite symbol* (qf-symbol), as proposed in [13]. Section 5 formalizes the use of wildcards. Section 6 deals with the handling of the qf-symbol *'&other'* in an interactive configuration. Finally, Section 7 gives conclusions and an outlook.

2 Background and Notation

This is a short paper and a review of literature will be brief. The author's insight into product configuration in practice stems from long personal involvement in the topic working for SAP and from discussions in the CWG [3]. Documentation of commercial configuration practice is still hard to come by. There exists a standard monograph about classic SAP Variant Configuration, now available in a second edition [2]. Furthermore, some chapters in [4] are devoted to commercial configurators, but the information there is far from

[1] PMH - Product Management Haag GmbH, Germany, email: albert@product-management-haag.de

[2] It is possible to relax the requirement of finiteness [13]. For example, the use of real (floating point) numbers and intervals is common.

[3] A *product feature* refers to a *value assignment* to a *product property*. In some industries, a single symbol (e.g. $'RED'$) has a company-wide interpretation (e.g. as *Color* = *'Red'*). Here, *Color* is a *product property*, which corresponds to a *CSP variable*.

[4] In SAP configurators, product properties are referred to as *characteristics* and product features either referred to as *features* or as *characteristic value assignments*.

[5] When the exact domain of a product property is not known to the modeler, it is considered to be *open*.

[6] Product properties may either have no features defined, or only have "standard" features defined, but allow additional features at run-time.

being technically complete. Also, there have been various contributions to this workshop series, two arbitrary examples being [18, 19]. In contrast, we consider both constraint programming [17] and its use in product configuration [4, 14] to be mature and well documented. The notation and results about arc-consistency with negative variant tables in Section 3 are adapted from [6].

We assume that all valid configurations of a product can be described at runtime using a finite set of k product properties[7], each property $p_j : 1 \leq j \leq k$ with a domain R_j, perhaps supplied dynamically at runtime. Thus, the space of all valid and invalid configurations can be represented as the Cartesian product:

$$\mathfrak{R} := R_1 \times R_2 \times \ldots \times R_k \tag{1}$$

$\mathfrak{R}$ can be seen as a k-dimensional space. Each point $\mathfrak{r} \in \mathfrak{R}$

$$\mathfrak{r} = (f_1, f_2, \ldots, f_k) \in \mathfrak{R} \tag{2}$$

represents either a valid or an invalid configuration, where f_j is a feature pertaining to property p_j. We call any point in $\mathfrak{R}$ an *r-tuple*, and any Cartesian product of product property subdomains a *c-tuple*. We can envision a c-tuple as a k-dimensional cuboid in $\mathfrak{R}$

$$(C_1 \times C_2 \times \ldots \times C_k) \subset \mathfrak{R} \tag{3}$$

The r-tuple in (2) can be considered as a special case of a c-tuple in (3), if we treat the individual features as singleton sets:

$$\{f_1\} \times \{f_2\} \times \ldots \times \{f_k\}$$

Treating the configurations listed in a given variant table $\mathcal{T}$ as a set of points in the k-dimensional solution space $\mathfrak{R}$, we define $\mathfrak{D}$ in (4) to be the c-tuple for the cuboid circumscribing $\mathcal{T}$.[8] We give an example of these sets in Section 3.

$$\mathfrak{D} := D_1 \times D_2 \times \ldots \times D_k \tag{4}$$

For notational simplicity we additionally define Q_j to refer to the complement of D_j with respect to R_j. $\mathfrak{Q}$ is the corresponding c-tuple of all Q_j:

$$Q_j := R_j \setminus D_j \tag{5}$$
$$\mathfrak{Q} := Q_1 \times Q_2 \times \ldots \times Q_k \tag{6}$$

We observe that the solution space $\mathfrak{R}$ in (1) can be represented as the disjoint union of $\mathfrak{D}$ and the k c-tuples $\mathfrak{C}_j$ in (7). Figure 1 illustrates a simple case of this decomposition.

$$\begin{aligned} \mathfrak{C}_1 &= Q_1 \times R_2 \times R_3 \times \ldots \times R_k \\ \mathfrak{C}_2 &= D_1 \times Q_2 \times R_3 \times \ldots \times R_k \\ &\ldots \\ \mathfrak{C}_k &= D_1 \times D_2 \times D_3 \times \ldots \times Q_k \end{aligned} \tag{7}$$

We remark that a variant table has a (non-unique) representation as a disjoint union of c-tuples (cf. [7, 15] and the examples in Section 3).

[7] If the product consists of only one component, its properties can readily be identified by name. If the product consists of multiple components with configurable properties, then a property must additionally be identified by a reference to the component.

[8] We consider $\mathcal{T}$ to be fixed and omit references to it to keep the notation simple.

3 Reprise: Negative Variant Tables

Let $\mathcal{U}$ be a negative variant table (listing excluded feature combinations). Any combination not excluded by $\mathcal{U}$ is implicitly allowed by $\mathcal{U}$. Therefore, it is natural to consider what is valid with respect to $\mathcal{U}$ at *runtime* (when actually configuring) and not when maintaining the table (*modeling time*). Previous work [6] deals with maintaining *arc consistency* [1, 16] in this situation. The main observation there is that if a feature, which does not occur anywhere in the table, is choosable at runtime, then all choosable features of all other properties remain choosable. An example: assume a T-shirt comes in various sizes and can sport an imprint. Suppose only one combination of imprint and size is known to be excluded: an imprint named 'STW' is not available in size 'S'.[9] Table 1 shows the corresponding negative variant table. Any imprint except 'STW' is available in all sizes. Any size except 'S' is available in all imprints. Figure 1 illustrates a disjoint decomposition (7) of the *solution set* $\mathfrak{R}$ (1) for the negative variant table in Table 1.

Table 1. $\mathcal{U}_1$: Excluded Imprints / Sizes

Size	Imprint
S	STW

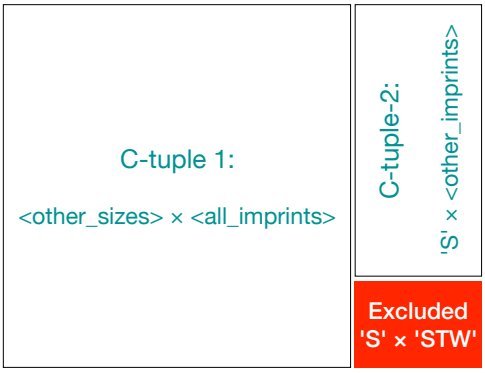

Figure 1. Runtime solution set consisting of two c-tuples

Assume that *Color* is also a property of the T-Shirts and that a particular imprint *'MIB'* is not available in the colors *'Blue'*, *'Red*, and *'White'*. Then both exclusions can be collectively expressed in a negative variant table as in Table 2:

Table 2. $\mathcal{U}_2$: T-Shirt Excluded Feature Combinations

Imprint	Size	Color
MIB	S;M;L;XL	Blue;Red;White
STW	S	Black;Blue;Red;White

To make the example more complete, we assume a further T-Shirt property *Fabric* which is not relevant for formulating exclusions. We order the properties as $vp_1 = Imprint$, $vp_2 = Size$, $vp_3 = Color$, and $vp_4 = Fabric$. We assume runtime domains $\mathfrak{R}$ as in (8) (the

[9] T-shirt examples with this restriction are elaborated more fully in [6, 13]

c-tuple $\mathfrak{D}$ circumscribing $\mathcal{U}$ (Table 2) is also given in (8)):

$$\begin{aligned} R_1 &= \{\textit{MIB,STW,EnvHero}\} & D_1 &= \{\textit{MIB,STW}\} \\ R_2 &= \{\textit{S,M,L,XL}\} & D_2 &= \{\textit{S}\} \\ R_3 &= \{\textit{Black,Blue,Red,White}\} & D_3 &= \{\textit{Blue,White,Red}\} \\ R_4 &= \{\textit{Cotton,Mixed,Synthetic}\} & D_4 &= \emptyset \end{aligned} \tag{8}$$

Then the decomposition (7) looks as follows:

$$\begin{aligned} \mathfrak{C}_1 &= \textit{EnvHero} \times R_2 \times R_3 \times R_4 \\ \mathfrak{C}_2 &= \textit{MIB, STW} \times \textit{M, L, XL} \times R_3 \times R_4 \\ \mathfrak{C}_3 &= \textit{MIB, STW} \times \textit{M, L, XL} \times \textit{Black} \times R_4 \\ \mathfrak{C}_4 &= \textit{MIB, STW} \times \textit{M, L, XL} \times \textit{Black} \times R_4 \setminus D_4 \end{aligned} \tag{9}$$

The set of valid T-Shirt configurations $\mathfrak{T}$ within $\mathfrak{D}$ is

$$\mathfrak{T} = (\mathfrak{D} \setminus \mathfrak{U}) \tag{10}$$

and the overall set of valid configurations $\mathfrak{T}^*$ within $\mathfrak{R}$ is

$$\mathfrak{T}^* = \mathfrak{C}_1 \cup \mathfrak{C}_2 \cup \ldots \cup \mathfrak{C}_k \cup T \tag{11}$$

Let $\pi_j(\mathfrak{R}) \subset \mathfrak{R}$ (12) be the reduction on $\mathfrak{R}$ that that can be achieved by enforcing arc-consistency using $\mathcal{U}$

$$\pi(\mathfrak{R}) := \pi(\mathfrak{R})_1 \times \pi(\mathfrak{R})_2 \times \ldots \times \pi(\mathfrak{R})_k \tag{12}$$

The following is a reformulation of the major results from [6]. We take $\mathfrak{R}$ to be decomposed as in (7).

Lemma 1 *If $Q_p \neq \emptyset$ for some p, then $\forall j \neq p : \pi(\mathfrak{R})_j = R_j$, i.e., no further reduction of any of the other domains is possible via arc-consistency with $\mathcal{U}$.*

Proof Without loss of generality, sort the columns such that $p = 1$. If $Q_1 \neq \emptyset$ then $\mathfrak{C}_1 = Q_1 \times R_2 \times R_3 \times \ldots \times R_k \neq \emptyset$ and any feature in Q_1 supports all features in R_j for $j \geq 2$. ∎

This has a trivial but important consequence (also taken from [6]):

Corollary 2 *If more than one non-empty Q_j exists, then no domain reduction at all is possible via arc-consistency with $\mathcal{U}$:*
$\exists Q_{j_1}, Q_{j_2} : j_1 \neq j_2 \wedge Q_{j_1}, Q_{j_2} \neq \emptyset \implies \pi(\mathfrak{R}) = \mathfrak{R}$ ∎

To summarize: Given a *negative variant table* $\mathcal{U}$, runtime domains $\mathfrak{R}$, and the space $\mathfrak{D}$, spanned by the table $\mathcal{U}$ (as in (4)). If more than one set $Q_j = R_j \setminus D_j$ is non-empty at runtime, then a *general arc consistency* (GAC) algorithm [1] does not need to be applied – no reduction would be obtained. Otherwise, a GAC algorithm should be applied on the potentially smaller solution space $\mathfrak{D}$, yielding $\pi(\mathfrak{D})$. If exactly one set Q_p is non-empty, then the only reduction achieved is $\pi(\mathfrak{R})_p = Q_p \cup \pi(\mathfrak{D})_p$. All features in Q_p remain choosable and for $j \neq p$ all features in R_j remain choosable. If there is no non-empty set Q_j, then $\pi(\mathfrak{R}) = \pi(\mathfrak{D})$, i.e. the reduction obtained by the GAC on $\mathfrak{D}$ holds.

4 Allowing Additional Non-Modeled Features

Some property domains may be left completely undefined explicitly. This delegates the definition of the domains to other means, such as variant tables. We treat features referenced in a variant table (not counting wildcard symbols)[10] to be *modeled* also. Sometimes, the "standard" features to be offered are modeled, but additional *non-modeled* features are allowed. Following an approach in previous work [13], we introduce a *quasi-finite symbol* (qf-symbol) $'\&other'_j$ to represent the non-modeled features of property p_j, which can be infinite, finite, or empty. The symbols $'\&other'_j$ cannot be explicitly referred to in a variant table.

A variant table allows partitioning R_j, the runtime domain of a property p_j, into D_j and Q_j as in (5). D_j collects all features referenced in the variant table for p_j and thus does not contain the symbol $'\&other'_j$. Where a runtime domain allows non-modeled features, i.e. $'\&other'_j \in R_j$, then $'\&other'_j$ is part of Q_j, not in $\mathfrak{D}$. For a negative variant table, the results of Lemma 1 and Corollary 2 apply directly, and $'\&other'_j$ will not be removed from consideration via arc-consistency due to the table.

In contrast, a positive table is a constraint and will restrict domains to lie within $\mathfrak{D}$ as defined in (4), removing all symbols $'\&other'_j$ from $\mathfrak{R}$. However, a positive table can also be seen to exclude any combination from $\mathfrak{D}$ not in the table. Formally, we could reverse the roles of $\mathcal{T}$ and $\mathcal{U}$ in (10), using the positive table to define a negative table (its complement). This would make sense if there were a way to allow *"additional combinations"*, which suggests itself for reasons of symmetry, but is not a modeling requirement the author has so far encountered in practice. Nevertheless, if we were to allow such an interpretation of a positive variant table *"pour le moment"*, the results of Lemma (1) and Corollary (2) apply, and $'\&other'_j$ will not be removed from consideration via arc-consistency due to the table.

5 Wildcards and C-Tuples

Variant tables can be represented in a significantly more compact form when using rows of c-tuples [7, 8, 15]. The practical relevance of a c-tuple representation is the topic of previous work [10, 11]. Table 3 is a positive formulation of Table 2 in three c-tuples. It takes the properties and their domains R_j to be defined as in (8). C-tuples can be represented naturally in a spreadsheet as rows with multi-valued cells, but most relational databases do not have straightforward support for a c-tuple representation.

Table 3. $\mathcal{T}_1$: Valid T-Shirt Variants

Imprint	Size	Color
MIB	S;M;L;XL	Black
STW	M;L;XL	Black;Blue;Red;White
EnvHero	S;M;L;XL	Black;Blue;Red;White

The use of a wildcard symbol is common in variant tables. Here, we treat a wildcard symbol in a c-tuple as a reference to the runtime domains, which may contain both modeled and non-modeled features. We denote the wildcard symbol for property p_j by Ω_j.[11] Table 4 is a representation of Table 3 using wildcards.

Table 4. $\mathcal{T}_2$: Valid T-Shirt Variants with Wildcards

Imprint	Size	Color
MIB	Ω_2	Black
STW	M;L;XL	Ω_3
EnvHero	Ω_2	Ω_3

[10] In this section, we assume that a variant table does not contain wildcards. Wildcards are dealt with in Section 5.

[11] The wildcard symbol used in practice varies, "$*$" is common. In the classic SAP VC an empty cell in a variant table is treated as a wildcard.

Table 4 lists the valid T-Shirt variants making use of wildcards. Regarding the wildcards, we assume that the modeled features are the R_j in (8).[12] Furthermore, the properties *Imprint* and *Color* allow (additional) non-modeled features. The runtime domains $\mathfrak{R}$ are depicted in (13).[13]

$$\begin{aligned} R_1 &= \{\textit{MIB,STW,EnvHero}\} \cup \&other_1 \\ R_2 &= \{\textit{S,M,L,XL}\} \\ R_3 &= \{\textit{Black,Blue,Red,White}\} \cup \&other_3 \end{aligned} \tag{13}$$

From the perspective of a GAC algorithm, the qf-symbols $'\&other'_j$ are just like any other symbol. A GAC algorithm can be applied in the usual way. The GAC algorithm may remove Ω_j as a whole, but it will not individually remove any features that are part of Ω_j at runtime.

6 Proposed Handling of Open Domains in Configuration

Just as a GAC algorithm would treat the qf-symbols $'\&other'_j$ like any other value, so would this symbol be presented to an agent performing a configuration task. It should be possible for this agent to

- to exclude non-modeled values by deselecting $'\&other'$
- to exclude all modeled values by selecting $'\&other'$
- to specify a set of non-modeled values to use for $'\&other'$

7 Conclusion and Future Work

This paper takes a look at some modeling techniques that are long established in practice but where a clear specification seems missing. The observations made here and in previous work [9, 11, 12, 13] are based on the author's personal experiences with the SAP configurators, as well as on general input from the product configuration community. The topics dealt with here are the use of *negative variant tables*, the use of *wildcards* in variant tables, and dealing with product property domains that allow the additional product features to be added to their domain at runtime in an impromptu manner. All have in common that they allow the domains at runtime to differ from what was known at modeling time (*open domains*).

The approach to arc-consistency with negative tables in Section 3, first introduced in [6], has been implemented in a prototype. Work is underway to verify this in a commercial setting. The handling of non-modeled features using a qf-symbol $'\&other'$ discussed in Section 4 has not yet been implemented. This is imminent future work. Wildcards have been in practical use for decades. The main contribution of Section 5 is to clarify that wildcards are meant to refer to the runtime domains and how they coexist with non-modeled features. The proposals of how to deal with non-modeled features in Sections 5 and 6 are new. Particularly, how and to what extent arc consistency can be established with open domains as discussed here, is seen as a contribution to practical configuration.

[12] These domains were introduced as runtime domains in (8), but are assumed to be the modeled domains here.

[13] Columns consisting only of wildcards can be added to or removed from a variant table at will, without affecting the meaning or processing of the table. For example, a column for the fourth T-Shirt property *Fabric* with a wildcard Ω_4 in all three rows can be arbitrarily added to Table 4 if opportune.

REFERENCES

[1] C. Bessiere, 'Constraint propagation', In Rossi et al. [17], chapter 3.

[2] U. Blumöhr, M. Münch, and M. Ukalovic, 'Variant tables in detail', in *Variant Configuration with SAP, second edition*, SAP Press, chapter 2.6.3, 152–158, Galileo Press, (2012).

[3] CWG. SAP Configuration Workgroup. `https://www.configuration-workgroup.com`, 2021. Accessed: 2021-06-04.

[4] *Knowledge-Based Configuration*, eds., Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, Morgan Kaufmann, Boston, 2014.

[5] Eugene C. Freuder and Alan K. Mackworth, 'Constraint satisfaction: An emerging paradigm', in *Handbook of Constraint Programming*, eds., Francesca Rossi, Peter van Beek, and Toby Walsh, volume 2 of *Foundations of Artificial Intelligence*, 13–27, Elsevier, (2006).

[6] Albert Haag, 'Arc consistency with negative variant tables', in *Proceedings of the 17th International Configuration Workshop, Vienna, Austria, September 10-11, 2015.*, eds., Juha Tiihonen, Andreas A. Falkner, and Tomas Axling, volume 1453 of *CEUR Workshop Proceedings*, pp. 81–87. CEUR-WS.org, (2015).

[7] Albert Haag, 'Managing variants of a personalized product', *Journal of Intelligent Information Systems*, 1–28, (2016).

[8] Albert Haag, 'Assessing the complexity expressed in a variant table', in *Proceedings of the 19th International Configuration Workshop, La Defense, France, September 14-15, 2017.*, pp. 20–27, (2017).

[9] Albert Haag, 'Quasi-finite domains: Dealing with the infinite in mass customization', in *20th Configuration Workshop (ConfWS 2018)*, eds., Alexander Felfernig, Juha Tiihonen, Lothar Hotz, and Martin Stettinger, number 2220 in CEUR Workshop Proceedings, pp. 77–84, Aachen, (2018).

[10] Albert Haag. Making more of variant tables in standard VC. Presentation at the CWG Virtual Spring Conference, Hamburg, Germany, May 26-June 19, 2020., 5 2020.

[11] Albert Haag, 'Toward data-driven modeling of configurable products', in *Proceedings of the 22th International Configuration Workshop, Vincenza, Italy, September 10-11, 2020.*, pp. 76–80, (2020).

[12] Albert Haag and Laura Haag, 'Empowering the use of variant tables in mass customization', in *Proceedings of the MCP-CE 2018 conference, Novi Sad, Serbia, September 19-21, 2018.*, pp. 180–189, (2018).

[13] Albert Haag and Laura Haag, 'Further empowering variant tables for mass customization', *International Journal of Industrial Engineering and Management (IJIEM)*, **10**(2), (2019).

[14] Ulrich Junker, 'Configuration', in *Handbook of Constraint Programming*, eds., Francesca Rossi, Peter van Beek, and Toby Walsh, volume 2 of *Foundations of Artificial Intelligence*, 837–873, Elsevier, (2006).

[15] G. Katsirelos and T. Walsh, 'A compression algorithm for large arity extensional constraints', in *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, ed., Christian Bessiere, volume 4741 of *Lecture Notes in Computer Science*, pp. 379–393. Springer, (2007).

[16] C. Lecoutre, 'STR2: optimized simple tabular reduction for table constraints', *Constraints*, **16**(4), 341–371, (2011).

[17] *Handbook of Constraint Programming*, eds., F. Rossi, P. van Beek, and T. Walsh, Elsevier, 2006.

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

[19] Juha Tiihonen, Ville-Valtteri Korppila, Jorma Heimonen, and Andreas Anderson, 'Structure oriented sales configuration of precast concrete production factories', in *Proceedings of the 22th International Configuration Workshop, Vincenza, Italy, September 10-11, 2020.*, pp. 1–8, (2020).