16th International Configuration Workshop

Proceedings of the 16th International Configuration Workshop

Edited by

Alexander Felfernig, Cipriano Forza, and Albert Haag

September 25-26, 2014

Novi Sad, Serbia

Organized by

UNIVERSITÀ DEGLI STUDI DI PADOVA

Graz University of Technology
Institute for Software Technology
Inffeldgasse 16b/2
A-8010 Graz
Austria

Alexander Felfernig, Cipriano Forza, and Albert Haag, Editors
Proceedings of the 16th International Configuration Workshop
September 25-26, 2014, Novi Sad, Serbia

Chairs

Alexander Felfernig, Graz University of Technology
Cipriano Forza, University of Padua, Italy
Albert Haag, SAP, Germany

Program Committee

Michel Aldanondo, Toulouse University, Mines Albi, France
Claire Bagley, Oracle Corporation, USA
David Benavides, University of Seville, Spain
Andreas Falkner, Siemens AG, Austria
Gerhard Friedrich, University of Klagenfurt, Austria
Paul Grünbacher, Johannes Kepler University
Alois Haselböck, Siemens AG, Austria
Mikko Heiskala, Aalto University, Finland
Lothar Hotz, University of Hamburg, HiTeC, Germany
Arnaud Hubaux, University of Namur, Belgium
Lars Hvam, Technical University of Denmark, Denmark
Dietmar Jannach, University of Dortmund, Germany
Thorsten Krebs, encoway, Germany
Tomi Männistö, Aalto University, Finland
Iulia Nica, Graz University of Technology, Austria
Rick Rabiser, Johannes Kepler University, Austria
Florian Reinfrank, Graz University of Technology, Austria
Stefan Reiterer, Graz University of Technology, Austria
Markus Stumptner, University of South Australia, Australia
Juha Tiihonen, Aalto University, Finland
Elise Vareilles, Toulouse University, Mines Albi, France
Franz Wotawa, Graz University of Technology, Austria
Linda Zhang, IESEG Business School Paris, France
Markus Zanker, University of Klagenfurt, Austria

Organizational Support

Martin Stettinger, Graz University of Technology, Austria
Nikola Suzic, University of Novi Sad, Serbia and University of Padova, Italy

Preface

Configuration problems have always been subject of interest for the application and the development of advanced Artificial Intelligence techniques. The selection of papers of this year's workshop demonstrates the wide range of applicable AI techniques including contributions on configuration knowledge representation, algorithms, theoretical approaches, and real-world configuration problems & applications.

The workshop is of interest for both, researchers working in the various fields of Artificial Intelligence as well as for industry representatives interested in the relationship between configuration technology and the business problem behind configuration and mass customization. It provides a forum for the exchange of ideas, evaluations, and experiences especially related to the use of Artificial Intelligence techniques in the configuration context.

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

Alexander Felfernig, Cipriano Forza, and Albert Haag

Contents

Using Answer Set Programming for Feature Model Representation and Configuration

Varvana Myllärniemi [1] and **Juha Tiihonen**[1] and **Mikko Raatikainen** [1] and **Alexander Felfernig**[2]

Abstract. Feature models are a wide-spread approach used for expressing variability in software product lines. Answer set programming (ASP) is nowadays an increasingly popular approach to configuration knowledge representation. In this paper, we study the similarities between feature modeling and configuration knowledge representation with ASP. We define the feature configuration problem utilizing ASP, and show two different ways using an example of translating the basic feature modeling concepts embodied in the graphical feature models into ASP programs. This way we want to emphasize the role of ASP as a means to tackle the feature configuration problem.

1 Introduction

Features and feature models [11, 17, 18] have been proposed as a means to represent the variability of a software system. Variability in software is defined as the ability of a system to be efficiently extended, changed, customized or configured for use in a particular context [27]. Correct and efficient management of variability is especially important for software product lines. A software product line is a set of products that share a common, managed set of features, a common architecture and a set of reusable assets, thus enabling the preplanned production of products with slightly varying capabilities [7, 10]. In fact, feature modeling has become the *de facto* means to represent and reason about variability in software product lines in academia [6]. Within software product lines, feature models can be used for two purposes: to manage and reason about commonality and variability at the domain engineering level, and to support the derivation of valid products at the application engineering level.

Software product line variability, and consequently, feature models, can grow large and complex. Due to the combinatorial explosion, analyzing feature models and finding a valid feature configuration is infeasible to do manually with large-scale feature models [3]. Thus, there is a need for automated analysis and reasoning of feature models [3]. However, it seems that current feature model analysis focuses on the analysis of the variability, that is, analysis at the domain engineering level, rather than on analysis of the derivation or configuration task. Out of the feature analysis operations listed in [3], only a few analyses are related to derivation: whether a given feature configuration is a valid product, and the operation to enumerate all possible valid configurations [3]. The problem of feature configuration has been studied to some extent, for example, for staged feature configuration [12] that elaborates several stages of making selections and pruning the variability space. Within this paper, we are interested in the simple configuration problem: given a set of requirements for a product, what are the valid feature configurations?

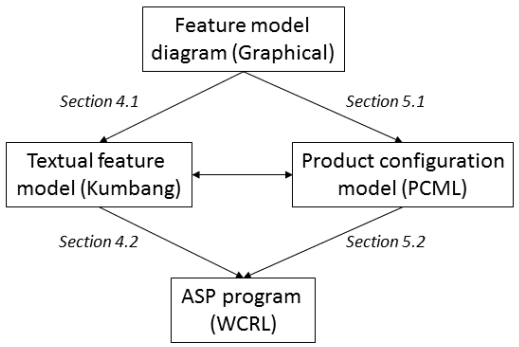


Figure 1. An illustration of how the research problem is addressed in this paper. The languages used to capture each model are marked in parenthesis.

In the field of mechanical and physical products, configuration has a long and successful history as a basis for mass-customization, see, e.g., [15]. The variability of the product is captured in a *configuration model* that represents the taxonomy and compositional structure of a product along with relevant constraints. The configuration task for a configuration model results in a *configuration*, a specification of a product individual [19, 30, 23] that meets the customer requirements.

As a supporting tooling, Answer Set Programming (ASP) is an increasingly important formalism for the representation of configuration models. Configuration is one of the first applications of ASP solving; the requirements of configuration problems were taken into account already in the development of the early ASP tool Smodels [25]. On the one hand, ASP programs have been applied directly to model configuration [24, 28] and reconfiguration [13, 24] problems in research systems. On the other hand, another approach is to model configuration models with a high-level language and to translate the resulting model into a corresponding ASP program [31, 29].

The two disciplines of software product lines and configurable products have similar goals and challenges in the variability management [16, 4]. A major goal of this paper is to show in an easily accessible manner and through concrete examples how ASP can be applied in the context of feature modeling. Previous work has described these aspects on a higher level of abstraction. Therefore, our research problem is to study the similarities between feature modeling and configuration knowledge representation with ASP. For this purpose, the following research questions are set:

- RQ1: How can the feature configuration problem be stated

[1] Aalto University, Finland, email: {firstname.lastname}@aalto.fi
[2] TU Graz, Austria, email: alexander.felfernig@ist.tugraz.at

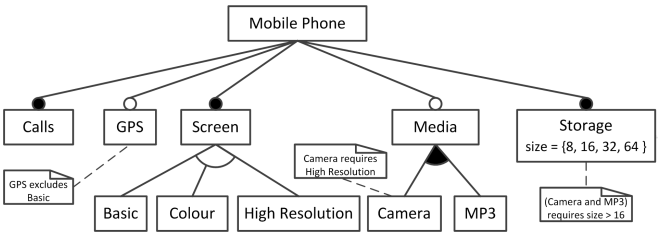


Figure 2. Example feature model slightly extended from [3].

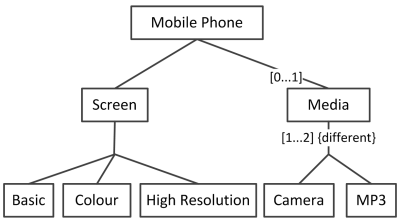


Figure 3. An excerpt from the feature model in Figure 1 modelled with cardinalities, following the notation used in [2].

through ASP?

- RQ2: What are the different ways to represent a feature model diagram as an ASP program?
- RQ3: What are the synergies in the variability management between feature modeling and product configuration?

Figure 1 illustrates the strategy that this paper utilizes to answer the research problem and questions. In particular, it shows how the graphical feature diagrams are represented with textual languages, and these textual languages are then automatically translated to ASP programs. Since the same graphical feature model can be represented both with the textual feature modeling language (Kumbang) as well as with the product configuration language (PCML), it is possible to compare and identify conceptual similarities and differences between software variability management and product configuration. Moreover, the figure illustrates the strategy of utilizing intermediate level languages: this omits the need to manually write ASP programs directly, and consequently, any inherent cognitive difficulties.

The contributions of this paper are the following. Firstly, we adapt the existing work [26] to define the feature configuration problem based on answer sets and stable model semantics. Secondly, we show how the basic concepts of feature models can be represented as ASP programs utilizing a concrete running example. This enables the use of existing ASP solvers to efficiently solve the feature configuration problem. Thirdly, for translating the feature models to ASP programs, we utilize two existing intermediate level languages; these languages enable the product line engineer to operate on domain-specific modeling constructs. Since these two languages originate from different paradigms, this highlights the conceptual similarities between software product line engineering and product configuration.

The remainder of this paper is organized as follows. Section 2 lays out the background as a previous work. Section 3 defines the feature configuration task and problem with ASP. Section 4 shows how graphical feature models can be represented as ASP programs by translating them through a textual feature modeling language called Kumbang (cf. Figure 1). Section 5 demonstrates that the same graphical feature model can be represented by Product Configuration Modeling Language (PCML) and its translation to ASP. Section 6 discusses the similarities of the software variability and traditional product configuration. Section 7 concludes.

2 Background

2.1 Feature modeling

A *feature* in a feature model can be seen as a characteristic of a system that is visible to the end-user [17]. For example, for a software product line for mobile phones, feature `MP3` might represent the capability to listen to and store audio files in MP3 format (see Figure 2). Since features can be used to capture also technological or implementation decisions [18], the definition of a feature has been extended to be a system property that is relevant to some stakeholder and is used to capture commonalities or discriminate among product variants [11].

Given a set of features, a *feature model* represents the variability and relations of those features. A feature model is represented as a hierarchically arranged set of features that consists of relations between a parent (or compound) feature and its child features (or subfeatures) and cross-hierarchy constraints [3]. Typically, feature models are presented as graphical diagrams. An example feature model for mobile phones is illustrated in Figure 2.

At least four basic relations between parent and child features can be identified [3]. Firstly, a child feature can be *mandatory* in relation to its parent feature: the child feature must be included in all products that include the parent feature. For example, feature `Calls` is mandatory in relation to feature `Mobile Phone` (see Figure 2). Secondly, a child feature can be *optional* in relation to its parent feature, for example, feature `GPS` can either be selected or left out for all mobile phones. Thirdly, a set of child features can be *alternative* in relation to their parent feature, which means that exactly one of the child features must be selected when the parent feature is in the product. As an example, exactly one of features `Basic`, `Colour`, and `High resolution` must be present in the product that has feature *Screen*. Fourthly, a set of child features can be in *or* relation to their parent feature, which means that one or more of them are present in the product that has the parent feature; this is exemplified by features `Camera` and `MP3` in Figure 2.

Additionally, there can be cross-hierarchy constraints. For example, features `GPS` and `Basic` are mutually exclusive, which means they cannot be in the same product, whereas feature `High resolution` must always be included in a product that contains feature `Camera`. These constraints are presented as annotations in Figure 2.

Various feature models and extensions to basic feature models have been proposed, as discussed in [3].

Firstly, there can be feature models with attributes [12, 5], as illustrated in Figure 2. Feature `Storage` has been characterized with attribute that describes the size in gigabytes, with an enumerated value range. Attributes are typically defined by stating a name and a specific range of values. Typically, a variation point that has a finite number of variants can be represented both as a set of features and as an attribute in a feature.

Secondly, there can be feature models with cardinality [11, 12]. It has been argued that cardinalities can be used to express similar relations as with basic feature relations. For example, Figure 3 illustrates

how a part of the model in Figure 2 is represented with cardinalities.

The usage of feature models varies from an informal documentation or visualization to more rigorous usages enabling even automated analysis. Respectively, the research has matured from the early notations [17] to various formalizations and analyses [3]. One possible usage of feature models is with configurable software product lines [8]: a product can be derived without further development [8] by configuring features, resulting in a model of a product individual.

2.2 Answer Set Programming

As summarized in [14], Answer Set Programming (ASP) has become a popular approach to declarative problem solving. The attractiveness of ASP stems from a combination of a rich and yet simple modeling language and the availability of high-performance solvers. The roots of ASP include knowledge representation, logic programming, (non-monotonic) reasoning, databases, and Boolean constraint solving.

ASP makes it possible to express the problem as a theory consisting of logic program rules with clear declarative semantics, and the *stable models*, i.e., the *answer sets* of the theory correspond to the solutions to the problem [25].

Programs that follow the Answer Set Programming paradigm are a generalization of normal logic programs. A generalized and unified syntax of ASP programs called *ASP-Core-2* has been defined [9]. This input language has been adopted by many ASP solvers [1]. Optimality criteria, variables and built-in functions can be defined. The syntax of ASP programs is close to Prolog, but the computation method via model generation is different [14].

There are a number of ASP solvers available, see [33], that can tackle a number of complex problems. The best ASP solvers perform well for a range of hard problems; see, for example, problems and results of the Fourth Answer Set Programming Competition [1]. The competition tasks included 3 problems in complexity class P, 15 problems in NP, 3 problems Beyond-NP ($\sum_2^P$), and 5 optimization problems; the domains of the tasks include combinatorial, database, diagnosis, graph, planning and scheduling problems. An example of current, well performing set of tools is Potassco, the Potsdam Answer Set Solving Collection[14], available from [22].

The authors of this paper have applied *weight constraint rule language (WCRL)* that is almost a genuine subset of ASP-Core-2. The languages ASP-Core-2 and WCRL are compatible enough so that the concrete WCRL logic programs generated by our tools are valid input to systems based on ASP-Core-2. This was verified with Clingo version 4.3, available from [22]. Thus, when describing WCRL, we actually describe a part of ASP-Core-2 that is sufficient for this paper. We can do this in a slightly more intuitive yet compact way than we could describe the full ASP-Core-2.

In the following, we describe the basic concepts of weight constraint rules focusing on the concepts needed in the rest of the paper. Instead of explaining the concepts utilizing a running example, these concepts are exemplified for Kumbang in Section 4 and for PCML in Section 5. For further details and examples, please see [25, 9].

Cardinality constraints are used as the primary basic building blocks of the product configuration rules. Cardinality constraints are of the form

$$l\{a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m\}u$$

where l and u are the lower and upper bounds of the constraint. Basic *atoms* are the smallest lexical units, for example a, or b. A literal is an atom b or a not-atom $not\ b$. A cardinality constraint is satisfied by a set of atoms S if the number of those literals in $\{a_1, \ldots, a_n, not\ b_1, \ldots,\ b_m\}$ that are satisfied by S is between the bounds l and u.

A *constraint rule* is an expression of the form

$$C_0 \text{ :- } C_1, \ldots, C_n$$

where the body of the rule consists of a number of cardinality constraints C_i, and the head C_0 cannot contain negated atoms. A program P is then a set of constraint rules.

For product configuration, the following rules are often useful. Firstly, in *choice rules* the number of satisfied atoms in the head must be between l and u:

$$l\{a_1, \ldots, a_n\}u \text{ :- } C_1, \ldots, C_n$$

Secondly, a rule with an empty head yields an *integrity constraint* :- $C_1, \ldots, C_n$, that is, an unsatisfiable constraint that allows specifying inconsistent situations where finding the answer is not possible. Finally, a rule with an empty body is called a *fact*. For example, a fact C_0 states that C_0 is always true.

Given a set of atoms S, a rule C_0 :- $C_1, \ldots, C_n$ is *satisfied* iff S satisfies C_0 whenever S satisfies each of $C_1, \ldots, C_n$. A program P is satisfied by S if each rule in P is satisfied by S. A *stable model* or answer set of a weight constraint rule program is defined as a set of atoms that 1) satisfies the program (is a classical model of the program) and 2) every atom in a stable model is *justified* (*grounded*) by the rules in the program. For example, consider the logical formula $b \wedge (b \wedge \neg c \rightarrow a)$ that has three (classical) models $\{b, c\}$, $\{a, b\}$ and $\{a, b, c\}$. The answer set program

$$b.\ a \text{ :- } b, not\ c.$$

has one stable model $\{a, b\}$. For the formalization of this definition, refer to [25].

Variable-free *ground* weight constraint rules discussed up to now become more practical by allowing the use of variables, function symbols, and predicates. A rule with variables is treated as a short hand for all its ground instantiations with respect to the Herbrand universe of the program. Decidability is retained by allowing only *domain-restricted* rules. Ignoring the details, each variable in a rule must appear in a *domain predicate* which occurs positively in the body of the rule. For example, $p(X)$:- $q(X)$ over constants $\{a, b, c\}$ is an abbreviation of

$$p(a) \text{ :- } q(a), p(b) \text{ :- } q(b), p(c) \text{ :- } q(c)$$

Given predicates and domains, rules with the so called *conditional literals* are frequently applied in product configuration. For example, a fact with predicate $chair$ and domain predicate $member$ states that every board must have exactly one chair that must also be a member:

$$1\ \{chair(X) : member(X)\}\ 1.$$

3 Feature Configuration Problem Utilizing ASP

Research question **RQ1** identified the need to address the feature configuration problem with ASP. In order to utilize ASP and existing solvers (see Section 2.2), one needs to define the basic concepts of the feature configuration problem. Figure 4 defines the feature configuration problem. Here, we adapt the definition of [26] to the domain of feature models in a straightforward manner. We describe each key concept in the definition informally and through examples from the domain of feature models. For further information about the configuration problem in more general terms, see [26].

Definition of the feature configuration task. Given	
CM	a feature configuration model CM translated to a set of rules,
GF	a set of ground facts representing the types in CM and unique identifiers for the instances of types, and
R	a set of rules R representing requirements,
is there a feature configuration C, that is,	
a stable model of $CM \cup S$, such that C satisfies R?	

Figure 4. The definition of the feature configuration task adhering to [26].

Firstly, a feature configuration model CM in Figure 4 specifies the entities, such as features; their properties, such as feature attributes; and composition structure, i.e. the feature tree structure; and the rules how the entities and their properties can be combined in a proper manner for a valid product. More informally, a feature configuration model represents the variability in the product line. For example, the feature model in Figure 2 is represented as one configuration model CM.

Within the definition in Figure 4, a distinction is made between types in a configuration model and instances in a configuration. Types in a configuration model define the properties of their individuals that can appear in a configuration. For example, in Figure 2, feature type storage defines the different attributes and their values, whereas feature instance storage in the actual product has a specific value for the size, for example 16 GB.

Ground facts GF in Figure 4 describe the possible feature instances and the attribute values of instances that can exist in a feature configuration. For example, for the feature `Storage` in Figure 2, a ground fact `featStorage(i).` indicates that feature instance with a unique identifier `i` is of feature type `Storage`. Additionally, a ground fact `hasattr(i,attrsizeGB,16).` tells that this instance has a specific attribute value assignment to indicate 16GB storage.

The set of rules R define requirements thus having a different status from the rules in the configuration model: these requirements represent the requirements that a specific product instance must satisfy. In a valid product configuration, the requirements must be satisfied by a configuration but cannot justify any elements in it. For a feature configuration problem, the requirements are stated as features that must be present in the configuration, or as attribute values that these features have. For example, for Figure 2, one requirement could be stated as `hasattr(i,attrsizeGB,16).`, meaning that there must be 16 GB storage in the product.

A feature configuration C consists of a set of positive and negative atoms. Positive atoms represent the feature instances and attribute values that are in the configuration. Due to the characteristics of ASP and stable models discussed in Section 2.2, the feature instances and attribute values in the configuration C, that is, the positive atoms in C, both *satisfy* the configuration model and its requirements, and are *justified* by them. For example, among the atoms that would be in the feature configuration for Figure 2, an atom `in(i)` indicates the inclusion of feature `Storage`. Further, if the storage is set to 16GB, an atom `hasattr(i,attrsizeGB,16)` is true, while atoms representing other attribute values, such as `hasattr(i,attrsizeGB,32)`, are false.

Consequently, the feature configuration C in the definition above is both consistent and complete. Informally, a *consistent* feature configuration is such that no rules of the configuration model are violated. A *complete* feature configuration is such that all the necessary selections have been made.

An ASP solver can be used to find consistent and complete configurations that meet a set of given requirements, given that such configurations exist. Therefore, the configuration problem definition above and its ASP solution can be used to support both domain and application engineering activities. At the domain engineering level, it can be checked whether the given feature configuration model CM doesn't have any consistent and complete configurations, which implies a self-contradictory model. At the application engineering level, the configuration task can support the finding of consistent and complete configurations, potentially even specifying the requirements R in an iterative manner.

For supporting the user in the configuration task, deducing the consequences of requirements is based on computing an approximation of the set of configurations satisfying the requirements that are valid but not necessarily all consequences are found. Intuitively, the consequences contain a set of facts that must hold for the configurations satisfying the requirements, a set of facts that cannot be true for the given requirements, and a set of unknown facts.

From the practical point of view, a product line engineer needs to capture the product line features and their commonality and variability into a configuration model CM. There are two options for this representation. The first option is to represent the informal feature model, for example, the visual notation in Figure 2, directly as an ASP program. However, this kind of a modeling task requires skills in logic programming, which may not be the case with an average product line engineer. The second option is to capture the feature model with a machine-processable, but human-readable textual language that utilizes directly the concepts known to a product line engineer, and then automatically translate the resulting middle-level model to an ASP program. This translation to ASP also gives the semantics to the middle-level representation language, as well as enables the use of existing ASP solvers for the configuration task. As is illustrated in Figure 1, this paper takes the latter approach.

In the following, we discuss how feature models can be represented as ASP programs, and consequently, how to represent the configuration model CM.

4 Representing Feature Models as ASP Programs through Textual Feature modeling Language

Section 3 presented the feature configuration problem utilizing ASP programs and identified the need to represent a given feature model as an ASP program. In the following, we show how the graphical feature model in Figure 2 and the basic feature modeling concepts can be represented as ASP programs. This is done in two phases, as illustrated in Figure 1: firstly, Section 4.1 shows how the feature model is represented as a textual model in Kumbang, and thereafter Section 4.2 shows how the textual model in Kumbang is translated to WCRL automatically with the Kumbang tool set [20]. Thus, for the purpose of this paper, we utilize WCRL as an example language to construct ASP programs (see also Section 2.2).

4.1 Representing the Feature Model in Kumbang

In order to enable the feature configuration with ASP, the feature model in Figure 2 needs to be represented in a form that is both understandable to a product line engineer, and can be unambiguously translated to an ASP program. For this purpose, we utilize Kumbang language [2], which is a modeling language and an ontology for modeling variability in software product line architectures from the

```
Forfamel model mobilephone
  root feature MobilePhone
feature type MobilePhone {
  contains
    Calls calls;
    GPS gps[0-1];
    Screen screen;
    Media media[0-1];
    Storage storage;
}
feature type Calls {}
feature type GPS {
  constraints not has_instances(Basic);
}
feature type Screen {
  contains (Basic,Colour,HighResolution) type;
}
feature type Basic {}
feature type Colour {}
feature type HighResolution {}
feature type Media {
  contains (Camera,MP3) apps[1-2] {different};
}
feature type Camera {
  constraints has_instances(HighResolution);
}
feature type MP3 {}
feature type Storage {
  attributes Size sizeGB;
  constraints
    (has_instances(Camera) and has_instances(MP3))
    => value(sizeGB) > 16;
}
attribute type Size = { 8, 16, 32, 64 }
```

Figure 5. Feature model from Figure 2 represented with the Kumbang language.

feature and component points of view. Kumbang is built on the product configuration concepts [26], on feature modeling approaches, and on the Koala architecture modeling language [32]. Kumbang is also supported by a set of tools that enable modeling and configuration tasks [20].

Figure 5 illustrates how the feature model in Figure 2 is represented with Kumbang language. In the following, we discuss the main characteristics and differences to the notation used in Figure 2.

Firstly, to adhere to the definition of the feature configuration task in Figure 4, Kumbang differentiates between a configuration model and a configuration. Variability in features is modelled explicitly in a configuration model (illustrated in Figure 5), whereas in a configuration, all variability has been resolved. The elements in a configuration model are referred to as types (for example, feature type `Storage` in Figure 5), while the elements in a configuration are referred to as instances. In contrast, traditional feature modeling notations do not usually make the conceptual distinction between feature types and instances. However, this may cause some difficulties in situations in which the definition of the features needs to be distinct from the feature compositional hierarchy. For example, if features need to be referred to in several places in the hierarchy (c.f., [12]), additional constructs, such as feature cloning or references may be needed. Thus, it seems that the distinction between types and instances allows more expressiveness in the model as such.

Secondly, traditional feature modeling uses a number of compositional relations between features, such as mandatory, optional, and alternative. As illustrated in Figure 3, the multitude of these relations can be expressed with one relation: cardinality. In order to define such relations in the configuration model, the cardinality needs a placeholder in the textual notation: such a placeholder in Kumbang is called a part definition. For example, the part definition `Media media[0-1]` in feature type `MobilePhone` states that `Media` is an optional feature i.e. has a cardinality from zero to one. Part definitions can be more complex: For example, part definition `apps` in type feature `Media` has two possible types of which one or two need to be in a configuration, and if two are selected, they need to be different. The use of part definitions with cardinalities is also advocated in [26].

```
% Definitions of feature types
featureType(featMobilePhone). featureType(featCalls).
featureType(featGPS).         featureType(featScreen).
featureType(featColour).      featureType(featBasic).
featureType(featHighResolution).
featureType(featMedia).       featureType(featMP3).
featureType(featCamera).      featureType(featStorage).

% Root feature MobilePhone
froot(X) :- featMobilePhone(X).
% The feature root is always in the configuration
1 { in(F) : froot(F) } 1.

% Some example part definitions (not all shown)
1{haspart(X1,X2,partDeftype):ppart(X1,X2,partDeftype,I)}1
:- featScreen(X1), in(X1).
1{haspart(X1,X2,partDefapps):ppart(X1,X2,partDefapps,I)}2
:- featMedia(X1), in(X1).

% Attribute definition for feature Storage
1 {hasattr(X,attrDefsizeGB,V):attrSize(V)} 1
:- in(X), featStorage(X).

% Definition of attribute value type Size
attrSize(8). attrSize(16). attrSize(32). attrSize(64).

% Constraint "Camera requires HighResolution"
% Other constraints omitted
constr5(X) :- in(X0),featHighResolution(X0),featCamera(X).
cf(5,X) :- featCamera(X), in(X), not constr5(X).
cff :- cf(5,X), featCamera(X).


% Possible feature instances in the configuration
% are enumerated with unique identifiers and
% corresponding possible parts are defined.
featMobilePhone(i0).
featCalls(i1).          ppart(i0,i1,partDefcalls,1).
featGPS(i2).            ppart(i0,i2,partDefgps,1).
featScreen(i3).         ppart(i0,i3,partDefscreen,1).
featBasic(i4).          ppart(i3,i4,partDeftype,1).
featColour(i5).         ppart(i3,i5,partDeftype,1).
featHighResolution(i6). ppart(i3,i6,partDeftype,1).
featMedia(i7).          ppart(i0,i7,partDefmedia,1).
featCamera(i8).         ppart(i7,i8,partDefapps,1).
featMP3(i9).            ppart(i7,i9,partDefapps,1).
featStorage(i10).       ppart(i0,i10,partDefstorage,1).

% A feature instance is in the configuration
% if it is both actual and possible part of something
in(X2) :- haspart(X1, X2, N), ppart(X1, X2, N, I).
```

Figure 6. The Kumbang representation of Figure 2 (Figure 5) translated to WCRL (some parts omitted and revised for clarity).

Thirdly, the constraints in Figure 2 need to be captured in an unambiguously defined, textual representation. In Figure 5, each constraint is defined in exactly one feature type, utilizing the existing constraint language [2] that supports logical expressions through, e.g., equivalence, implication, universal quantifiers, and references to the compositional hierarchy.

4.2 Representing the Kumbang Model in WCRL

Figure 6 illustrates how the feature model in Figure 5 is translated to WCRL. The translation has been performed automatically with the Kumbang tool set [20] and revised and organized for clarity.

Firstly, each feature type in the configuration model must be defined: for example, fact `featureType(featMobilePhone).` states that object constant `featMobilePhone` represents a feature type. Similarly, the attribute value types are defined, for example, fact `attrSize(8).` states that attribute value type named `Size` has 8 as one possible value.

Secondly, the root of the model must be defined. Rule

$$1\{in(F) : froot(F)\}1.$$

states that if feature type `F` is the root, a valid configuration must have exactly one feature instance selected (`in(F)`) that is instantiated from the root type, defined using predicate `froot`.

Thirdly, the compositional structure of the features must be defined. For each part definition, a rule with the following format is added:

$$n\{haspart(X_1, X_2, P) : ppart(X_1, X_2, P, I)\}m \text{ :- } F(X_1), in(X_1).$$

where F and P are replaced with feature and part names, and n, m replaced with the lower and upper bounds of the cardinality. Predicate `haspart` is used to indicate that a feature instance is instantiated as a part in the configuration, whereas predicate `ppart` is merely stating the possible parts. Together, these predicates justify the inclusion of a feature instance through composition:

$$in(X_2) \text{ :- } haspart(X_1, X_2, N), ppart(X_1, X_2, N, I).$$

Fourthly, attribute definitions are captured with the following rule:

$$1\{hasattr(X, A_d, V) : A_v(V)\}1 \text{ :- } in(X), F(X).$$

where A_d is replaced with the name of the attribute definition, A_v with the name of the attribute value type, and F with the name of the defining feature type.

Finally, the configuration model must also define the identifiers for each feature instance. This enables, for example, to state requirements R about the features that must be present in the configuration (see Figure 4). In Figure 6, the feature instances are given identifiers by enumerating all possible instances in the configuration, for example, fact `featMobilePhone(i0).` gives identifier `i0` to feature `MobilePhone`. Additionally, the identifiers are used to state the possible compositional relations between the instances with the predicate `ppart`. Using these identifiers, it is possible to state the requirements about the feature instances that must be in the configuration, for example, `in(i8).` requires that feature `Camera` must be present in the configuration.

5 Representing Feature Models as ASP Programs through Product Configuration modeling Language

In this Section, the example feature model of Figure 2 is represented with a configuration modeling language designed to model the variability of physical products. We also exemplify the corresponding ASP presentation.

5.1 Representing the Feature Model in PCML

For illustrating the application of a configuration modeling language, we apply *PCML*, Product Configuration Modeling Language [21]. PCML is used by the WeCoTin configurator [29] as the language for representing configuration models. PCML is object-oriented, declarative and has formal implementation-independent semantics.

The main concepts of PCML are *feature types*, their *compositional structure*, *attributes*, and *constraints*. Feature types define the subfeatures (parts) and attributes of their *individuals* that can appear in a configuration. In a configuration, subfeatures (parts) of a feature individual are *realized* with feature individuals. The realizing feature individual(s) "fill the role" created by the subfeature definition. If the cardinality includes 0, an empty realization is possible. A configuration is a non-empty tree of feature individuals and individuals representing attribute values. In addition, the compositional structure is explicitly presented.

The main modeling mechanism of this example is the compositional structure. Feature type `Mobile_Phone_t` in 7 serves as the root

```
configuration model MyProduct
 feature Mobile_Phone_t
   subfeature Screen_p allowed features
     Basic_t, Colour_t, High_resolution_t
     cardinality 1
   subfeature Calls_p
     allowed features Calls_t cardinality 1
   subfeature GPS_p
     allowed features GPS_t cardinality 0 to 1
   subfeature Media_p
     allowed features Media_t cardinality 0 to 1
   subfeature Storage_p
     allowed features Storage_t cardinality 1
   constraint GPS_excludes_Basic not ((present(
     GPS_p)) and (Screen_p individual of Basic_t))
 feature Basic_t
 feature Colour_t
 feature High_resolution_t
 feature Media_t
   subfeature Camera
     allowed features Camera_t cardinality 0 to 1
   subfeature MP3
     allowed features MP3_t cardinality 0 to 1
   constraint Camera_requires_High_resolution
     (present(Camera)) implies
    ($config.Screen_p individual of High_resolution_t)
   constraint Media_requires_Camera_or_MP3
     (present(Camera)) or ( present ( MP3 )  )
   constraint Camera_and_Mp3_require_min_32GB
     ((present(Camera)) and (present(MP3))) implies
     ($config.Storage_p.Size_GB >= 32)
 feature Camera_t
 feature MP3_t
 feature GPS_t
 feature Calls_t
 feature Storage_t
   attribute Size_GB value type integer
     constrained by $ in list(8,16,32,64)
 configuration feature Mobile_Phone_t
```

Figure 7. The feature model of Figure 2 represented with PCML.

of the compositional structure '*configuration type*', see Figure 7. An individual of the type serves as the root of the configuration.

Feature type `Mobile_Phone_t` defines it's compositional structure through a set of *subfeature definitions*. A subfeature definition specifies a *subfeature name*, a non-empty set of *possible subfeature types* (*allowed types* for brevity) and a *cardinality* indicating the valid number of subfeatures. Note that the example of Figure 7 applies a naming convention where the names of feature types end with `_t` and names of subfeatures (parts) with `_p`.

A mandatory subfeature is represented by specifying cardinality `1` and by specifying exactly one allowed type. An example is the mandatory feature `Calls_p`. An optional subfeature is modeled with a subfeature definition whose cardinality is `0 to 1`, e.g. the feature `GPS_p`. Alternative features are modeled with cardinality `1` and more than one allowed type. E.g., feature `Screen_p`. Or-subfeatures are not directly supported by PCML, because with large cardinalities individuals of the same type would be allowed. Therefore for modeling `Media_t`, further subfeatures were defined and a constraint added that enforces the presence of at least one subfeature.

The only attribute of the example is `Storage_t` defining an enumerated integer attribute `Size_GB`.

5.2 Representing the PCML Model in WCRL

Figure 8 shows a partial WCRL/ASP representation of the example feature model. When studying the WCRL/ASP presentation of Figure 8, it is visible that early versions of PCML and WeCoTin applied terminology where feature types were called *component types* and *subfeatures* were called *parts*.

Figure 8 shows the corresponding WCRL presentation (partial). The comments explain the predicates. For a more complete explanation, see [29].

Figure 9 shows one of the 52 answer sets. It represents a feature configuration with `Colour`, `Calls`, `Storage`, Storage `size=16` GB.

```
% if an individual C2 is as part of C1 -> in(C2)
in(C2) :- pa(C1,T,C2,Pn), ppa(T,C1,C2,Pn).
% exclusive parthood: same individual cannot
% be a part of several whole individuals
:- 2{pa(C1,T,C2,Pn):ppa(T,C1,C2,Pn)}, compT_Feature(C2).
%transitivity of is-a hierachy
isa(X,Z):- isa(X,Y), isa (Y,Z),
  compTDom(X), compTDom(Y), compTDom(Z).
% reflexivity of is-a
isa(X,X):- compTDom(X).

%Example types
% Screen_t is a component type and a subtype of 'Feature'
compTDom(compT_Feature).
%Screen types are direct subtypes of 'Feature'
compTDom(compT_Basic_t).
compT_Feature(C) :- compT_Basic_t(C).
isa(compT_Basic_t,compT_Feature).
compTDom(compT_Colour_t).
compT_Feature(C) :- compT_Colour_t(C).
isa(compT_Colour_t,compT_Feature).
compTDom(compT_High_resolution_t).
compT_Feature(C) :- compT_High_resolution_t(C).
isa(compT_High_resolution_t,compT_Feature).
% Storage_t
compTDom(compT_StoraStorage_t).
compT_Feature(C) :- compT_Storage_t(C).
isa(compT_Storage_t,compT_Feature).
% attribute Size_GB of Storage_t
1{prop_Storage_t_Size_GB(X,compT_Storage_t,Y):prSpec(Y)}1
  :- in(X),compT_Storage_t(X).
prSpec(8).
prSpec(16).
prSpec(32).
prSpec(64).

%part name Screen_p
pan(part_Screen_p).
%cardinality 1
1{pa(C1,compT_Mobile_Phone_t,C2,part_Screen_p):
  ppa(compT_Mobile_Phone_t,C1,C2,part_Screen_p)}1 :-
    in(C1),compT_Mobile_Phone_t(C1).
% assignment of possible part individuals of allowed
% types for part screen_p with helper predicate for.
% The automated translation makes such an allocation
% for symmetry breaking, which this example
% does not need
ppa(compT_Mobile_Phone_t,C1,C2,part_Screen_p) :-
  compT_Mobile_Phone_t(C1),compT_Basic_t(C2),
  for(compT_Mobile_Phone_t,C1,C2,part_Screen_p).
ppa(compT_Mobile_Phone_t,C1,C2,part_Screen_p) :-
  compT_Mobile_Phone_t(C1),compT_Colour_t(C2),
  for(compT_Mobile_Phone_t,C1,C2,part_Screen_p).
ppa(compT_Mobile_Phone_t,C1,C2,part_Screen_p) :-
  compT_Mobile_Phone_t(C1),compT_High_resolution_t(C2),
  for(compT_Mobile_Phone_t,C1,C2,part_Screen_p).

% Constraint compilation omitted for brevity.
% it is performed by subexpression.
```

Figure 8. PCML representation of Figure 2 (Figure 7) translated to WCRL (some parts omitted for brevity).

6 Discussion

In this paper, we showed two ways to represent feature models as ASP programs by utilizing existing textual modeling languages designed for feature modeling and product configuration modeling. The use of an intermediate, textual language between the graphical feature models and logic programs is not that common: it seems typical that graphical feature diagrams are directly translated, e.g., to propositional logic [3], rather than utilizing an intermediate textual language.

```
in(ind_compT_Colour_t_1)
pa(ind_compT_Mobile_Phone_t_1,compT_Mobile_Phone_t,
  ind_compT_Colour_t_1,part_Screen_p)
in(ind_compT_Calls_t_1)
pa(ind_compT_Mobile_Phone_t_1,compT_Mobile_Phone_t,
  ind_compT_Calls_t_1,part_Calls_p)
in(ind_compT_Storage_t_1)
pa(ind_compT_Mobile_Phone_t_1,compT_Mobile_Phone_t,
  ind_compT_Storage_t_1,part_Storage_p)
in(ind_compT_Mobile_Phone_t_1)
prop_Storage_t_Size_GB(ind_compT_Storage_t_1,
  compT_Storage_t,16)
```

Figure 9. An answer set representing a feature configuration with `Colour`, `Calls`, `Storage`, Storage `size_GB=16`. Ground atoms were derived from the WCRL of Figure 8. Long atoms are split into two lines.

The benefit of using such intermediate languages and models is that they may be more approachable to product line engineers: they utilize modeling concepts that more or less directly correspond to the concepts used to represent software variability. Such intermediate languages can serve a multitude of purposes: they can be represented graphically and modelled with the aid of graphical tools; they can be created or edited directly if need arises; and they can be automatically translated to ASP programs.

Another option would have been to directly represent or encode the entities and relations in feature models as ASP programs. The benefit of writing directly ASP programs is that the resulting ASP programs most probably are more compact and directly human-readable. The drawback is that logic programming even in the form of ASP programs might be challenging for a product line engineer not trained in computational logic programming.

For simplicity, our representation in this paper covered some basic concepts of feature models. Nevertheless, the languages discussed in Sections 4 and 5 cover much richer sets of modeling constructs. For example, the capability to represent feature inheritance was not utilized in the examples. Similarly, the literature contains numerous proposed extensions of feature models. Some of them are included in our conceptualizations and corresponding tools (e.g. attributes, cardinalities) while some are not. In any case, a detailed discussion about the needed modeling concepts is a future work item.

By mapping the feature modeling notation to both Kumbang and PCML, we demonstrated that both approaches, one tailored for feature modeling and one for product configuration, can be utilized for modeling software variability. A specific addition to the traditional feature modeling concepts done in this paper is to differentiate between feature instances and feature types. This dichotomy, however, parallels with domain and application engineering in software product families and is, therefore, quite natural for software variability although it has not been applied explicitly in feature modeling.

The product configuration community has applied configuration modeling and configuration techniques in full scale production use for decades. It may be that some modeling constructs and approaches related to managing variability could be carried over to describe and analyze feature models. In such a case, existing analyses and respective tools could be readily utilized.

However, the derivation of product lines is not just about configuration: feature models are applicable to a wide range of settings, not just to configurable software product lines. Because of this, the tools intended for product configuration do not necessarily support all the relevant activities in the application engineering phase of software product lines.

In general, due to the availability of a variety of different efficient ASP solvers, it seems beneficial to represent feature models as ASP programs. Despite the fact that the theoretical computational complexity inherent in the feature configuration problem is NP-hard, the current ASP solvers are efficient in calculating the stable models even for programs that represent real-life feature models. We believe that it is more important to find and utilize real problems in testing scalability instead of generated random problems. Consequently, we have configured real problems interactively, without no noticeable delay: see the configuration model with slightly less than 500 variation points [29] and the configuration model with dozens of different types [2] as examples.

7 Conclusions

This study shows how feature models can be represented as ASP programs by means of two different mappings of a graphical feature diagram through intermediate languages. The representation of feature models as ASP programs enables utilizing existing inference engines that are efficient for practical problems. Moreover, the mapping shows significant similarities between feature modeling and product configuration, in particular demonstrating how a feature model diagram can be presented using a product configuration language. This is one concrete step towards better unification between these two similar disciplines of research.

Acknowledgment

We acknowledge the financial support of TEKES as part of the Need 4 Speed (N4S) program of DIGILE and the Austrian Research Promotion Agency (Casa Vecchia, 825889).

REFERENCES

[1] Fourth (open) answer set programming competition - 2013. https://www.mat.unical.it/aspcomp2013, 2013. retrieved 2014-05-05.

[2] Timo Asikainen, Tomi Männistö, and Timo Soininen, 'Kumbang: A domain ontology for modelling variability in software product families', *Advanced engineering informatics journal*, **21**(1), (2007).

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

[4] David Benavides, Alexander Felfernig, JosA. Galindo, and Florian Reinfrank, 'Automated analysis in feature modelling and product configuration', in *Safe and Secure Software Reuse*, eds., John Favaro and Maurizio Morisio, volume 7925 of *Lecture Notes in Computer Science*, 160–175, Springer Berlin Heidelberg, (2013).

[5] David Benavides, Pablo Trinidad Martín-Arroyo, and Antonio Ruiz Cortés, 'Automated reasoning on feature models', in *International Conference on Advanced Information Systems Engineering*, (2005).

[6] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki, 'A study of variability models and languages in the systems software domain', *Software Engineering, IEEE Transactions on*, **39**(12), 1611–1640, (Dec 2013).

[7] Jan Bosch, *Design and Use of Software Architectures: Adapting and Evolving a Product-Line Approach*, Addison-Wesley, 2000.

[8] Jan Bosch, 'Maturity and evolution in software product lines: Approaches, artefacts and organization', in *Proc. of Software Product Line Conference*, (2002).

[9] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Francesco Ricca, and Torsten Schaub. ASP-Core-2: Input language format (v.2.03b). ASP Standardization Working Group, https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.03b.pdf, 2012. retrieved 2014-05-05.

[10] Paul Clements and Linda Northrop, *Software Product Lines—Practices and Patterns*, Addison-Wesley, 2001.

[11] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker, 'Formalizing cardinality-based feature models and their specialization', *Softw. Proc. Improv. Pract.*, **10**(1), 7–29, (2005).

[12] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker, 'Staged configuration through specialization and multilevel configuration of feature models', *Software Process: Improvement and Practice*, **10**(2), 143–169, (2005).

[13] Gerhard Friedrich, Anna Ryabokon, Andreas A. Falkner, Alois Haselböck, Gottfried Schenner, and Herwig Schreiner, '(Re)configuration using Answer Set Programming', in *22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), Workshop on Configuration*, eds., Kostyantyn Shchekotykhin, Markus Zanker, and Dietmar Jannach, pp. 17–25, (2011).

[14] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider, 'Potassco: The potsdam answer set solving collection', *AI Communications*, **24**(2), 107–124, (2011).

[15] L. Hotz, A. Felfernig, A. Günter, and J. Tiihonen, 'A Short History of Configuration Technologies', in *Knowledge-based Configuration – From Research to Business Cases*, eds., A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, chapter 2, 9–19, Morgan Kaufmann Publishers, (2013).

[16] Arnaud Hubaux, Dietmar Jannach, Conrad Drescher, Leonardo Murta, Tomi Mnnist, Krzysztof Czarnecki, Patrick Heymans, Tien Nguyen, and Markus Zanker, 'Unifying software and product configuration: A research roadmap', in *Proceedings of the workshop on configuration (confws12)*, (2012).

[17] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, and A.S. Peterson, 'Feature-oriented domain analysis (foda) feasibility study', Technical Report CMU/SEI-90-TR-21, ADA 235785, Software Engineering Institute, (1990).

[18] K.C. Kang, Jaejoon Lee, and P. Donohoe, 'Feature-oriented product line engineering', *IEEE Software*, **19**(4), 58–65, (2002).

[19] S. Mittal and F. Frayman, 'Towards a Generic Model of Configuration Tasks', in *11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, volume 2, pp. 1395–1401, Detroit, Michigan, USA, (1989).

[20] Varvana Myllärniemi, Mikko Raatikainen, and Tomi Männistö, 'Kumbang tools', in *Software Product Line Conference*, volume 2, pp. 135–136, (2007).

[21] Hannu Peltonen, Juha Tiihonen, and Andreas Anderson. Configurator tool concepts and model definition language. Unpublished working document of Helsinki University of Technology, Software Business and Engineering Institute, Product Data Management Group, Espoo, Finland, 2001.

[22] Potassco. Potassco, the Potsdam Answer Set Solving Collection, bundles tools for answer set programming developed at the university of potsdamanswer set programming. SourceForge project `http://potassco.sourceforge.net/`. Accessed 2014-05-06.

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

[24] Gottfried Schenner, Andreas Falkner, Anna Ryabokon, and Gerhard Friedrich, 'Solving object-oriented configuration scenarios with asp', in *15 th International Configuration Workshop*, eds., Michel Aldanondo and Andreas Falkner, pp. 55–62, (2013).

[25] Patrik Simons, Ilkka Niemelä, and Timo Soininen, 'Extending and implementing the stable model semantics', *Artificial Intelligence*, **138**, 181–234, (2002).

[26] 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 (AAAI Technical Report SS-01-01)*, eds., Alessandro Provetti and Tran Cao Son, pp. 195–201, (2001).

[27] Mikael Svahnberg, Jilles van Gurp, and Jan Bosch, 'A taxononomy of variability realization techniques', *Software—Practice and Experience*, **35**(8), 705–754, (2005).

[28] Tommi Syrjänen, 'Including diagnostic information in configuration models', in *First International Conference on Computational Logic (CL 2000)*, eds., John Lloyd, Veronica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, LuísMoniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, volume LNCS 1861, pp. 837–851. Springer, (2000).

[29] Juha Tiihonen, Mikko Heiskala, Andreas Anderson, and Timo Soininen, 'Wecotin–a practical logic-based sales configurator', *AI Communications*, **26**(1), 99–131, (2013).

[30] Juha Tiihonen and Timo Soininen, 'Product Configurators - Information System Support for Configurable Products', Technical Report TKO-B137, Helsinki University of Technology, Laboratory of Information Processing Science, (1997). also publsihed in: Increasing Sales Productivity through the Use of Information Technology during the Sales Visit, Hewson Consulting Group.

[31] Juha Tiihonen, Timo Soininen, Ilkka Niemelä, and Reijo Sulonen, 'A practical tool for mass-customising configurable products', in *Proceedings of the 14th International Conference on Engineering Design*, eds., A.Folkeson, K. Gralén, M. Norell, and U. Sellgren, pp. CDROM, paper number 1290, 10 pp., (August 19-21, 2003 2003).

[32] Rob van Ommering, Frank van der Linden, Jeff Kramer, and Jeff Magee, 'The Koala component model for consumer electronics software', *Computer*, **33**(3), 78–85, (March 2000).

[33] Wikipedia. Answer set programming. `http://en.wikipedia.org/wiki/Answer_set_programming`. Accessed 2014-05-06.

Integrating Distributed Configurations with RDFS and SPARQL

Gottfried Schenner[1] and **Stefan Bischof**[1] and **Axel Polleres**[2] and **Simon Steyskal**[1,2]

Abstract. Large interconnected technical systems (e.g. railway networks, power grid, computer networks) are typically configured with the help of multiple configurators, which store their configurations in separate databases based on heterogeneous domain models (ontologies). In practice users often want to ask queries over several distributed configurations. In order to reason over these distributed configurations in a uniform manner a mechanism for ontology alignment and data integration is required. In this paper we describe our experience with using standard Semantic Web technologies (RDFS and SPARQL) for data integration and reasoning.

1 INTRODUCTION

Product configuration [9] is the task of assembling a system from predefined components satisfying the customer requirements. Large technical systems are typically configured with the help of *multiple* configuration tools. These configurators are often specific to a technology or vendor and therefore use heterogeneous domain models (ontologies).

For large interconnected systems (e.g. railway networks, power grid) the configuration of the overall system may be stored across separate databases, each database containing only the information for a sub-system.

The domain models and databases of these configurators are a valuable source of information about the deployed system. But there must be a way to access the information in an uniform and integrated manner in order to exploit this.

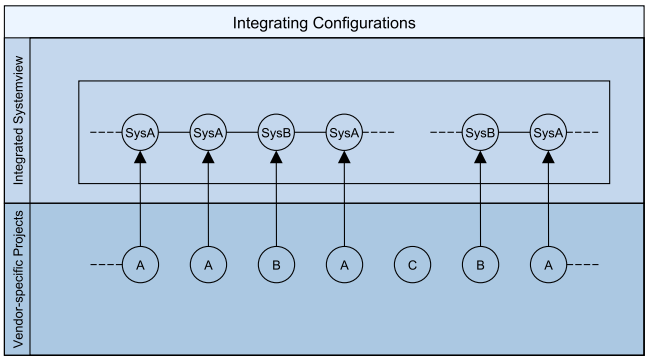


Figure 1: Data integration approach

Figure 1 shows a typical scenario from the railway domain. The individual stations of a network are built by different vendors (A, B, C). Vendors A and B use proprietary configurators (A, B) and store the configurations of these stations in separate projects. Vendor C does not use a configurator, therefore there is no (digital) data available to integrate.

In the railway scenario the railway company owning the railway network wants to obtain information about the whole network in a vendor-independent way. To achieve this, some form of ontology and data integration is necessary. We can identify three steps: (i) create a vendor-independent ontology, (ii) map or align the vendor-specific ontologies or schemas to the vendor-independent ontology, and (iii) provide the vendor-specific data in terms of the vendor-independent ontology.

This paper investigates, how to use standard Semantic Web technologies (RDFS, SPARQL and OWL) for data integration. Our approach uses SPARQL CONSTRUCT queries to generate a linked system view of the distributed configurations as depicted in Figure 2. This system view can then (*i*) be queried in a uniform manner, (*ii*) be checked for contraint violations taking all relevant configurations into account and (*iii*) be used for reasoning and general consistency checks (cf. Figure 3).

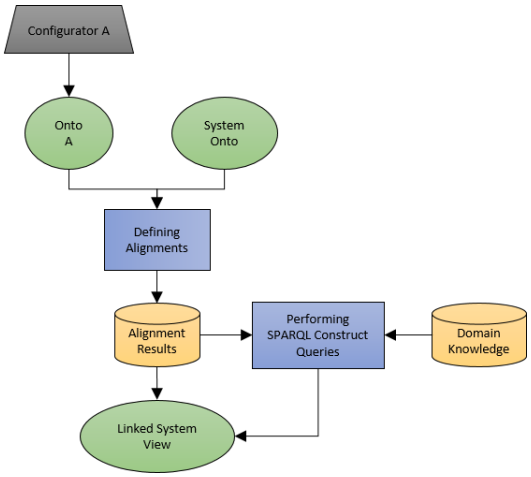


Figure 2: Integrating configurations with SPARQL CONSTRUCT queries into a linked system view.

The remainder of this paper is structured as follows: Chapter 2 discusses the preliminaries of this paper, especially the used Semantic Web technologies. Chapter 3 introduces the working example of this paper, Chapter 4 shows how to derive an integrated view of the system from the individual configurator specific databases, in Chapter 5 we

[1] Siemens AG Österreich, Siemensstrasse 90, 1210 Vienna, Austria {gottfried.schenner|bischof.stefan}@siemens.com
[2] Vienna University of Economics & Business, 1020 Vienna, Austria {axel.polleres|simon.steyskal}@wu.ac.at

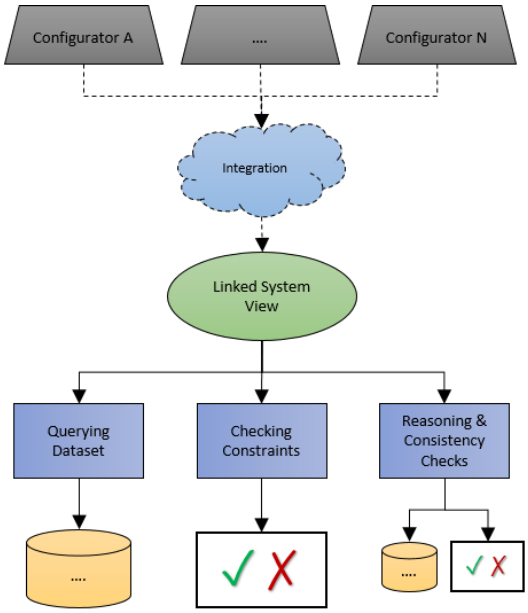


Figure 3: Using a linked system view for querying and reasoning over distributed configurations.

discuss, how to reason about the overall system with SPARQL queries and we discuss related work in Chapter 6. Finally, we conclude our paper in Chapter 7.

2 PRELIMINARIES

The proposed approach builds heavily on Semantic Web standards and technologies. Instance data is represented as RDF triples, domain models are mapped to domain dependent ontologies/vocabularies and queries are formulated in SPARQL.

2.1 Data representation with RDF

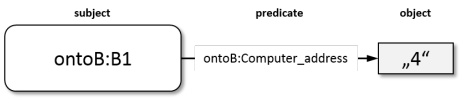


Figure 4: A simple RDF triple.

The *Resource Description Framework* (RDF) [15] is a framework for describing and representing information about resources and is both human-readable and machine-processable. These abilities offer the possibility to easily exchange information in a lightweight manner among different applications.

In RDF every resource is identified by its URI and represented as `subject - predicate - object` triples, where subjects and predicates are URIs and objects can either be literals (strings, integers, . . .) or URIs as shown in Figure 4. Additionally, subjects or objects can be defined as *blank nodes*, these *blank nodes* do not have a corresponding URI and are mainly used to describe special types of resources without explicitly naming them. For example the concept *mother* could be represented as a *female person having at least one child*.

2.2 Querying with SPARQL

SPARQL Protocol And RDF Query Language (SPARQL) [14] is the standard query language for RDF, which has become a W3C Recommendation in version 1.1 in 2013. Its syntax is highly influenced by the previous introduced RDF serialization format Turtle [1] and SQL [4] a query language for relational data[3].

Besides basic query operations such as union of queries, filtering, sorting and ordering of results as well as optional query parts, version 1.1 extended SPARQL's portfolio by aggregate functions (`SUM, AVG, MIN, MAX, COUNT,` . . .), the possibility to use subqueries, perform update actions via SPARQL Update and several other heavily requested missing features [23].

Furthermore, it is possible to create entirely new RDF graphs based on the variable bindings constituted in graph patterns which are matched against one or more input graphs, using SPARQL CONSTRUCT queries. Using such CONSTRUCT queries offers the possibility to easily define transformations between two or more RDF graphs/ontologies, which serves as a basic building block for the present paper.

2.3 Semantic heterogeneity

In order to be able to integrate two or more ontologies into one integrated knowledge base, it is mandatory to define correspondences between the elements of those ontologies to reduce semantic heterogeneity among the integrated ontologies [8].

The problem of semantic heterogeneity can be caused by several facts, e.g. that different ontologies model the same domain in different levels of precision or use different terms for the same concepts [26] (e.g. a concept `Computer` is equivalent to another concept `Device`). Such "simple" differences can be detected by most of the current state-of-the-art ontology matching systems like *YAM++* [21] or *LogMap* [18]. However more complex heterogeneities (e.g. a concept `Subnet` is equivalent to the union of the concepts `Computer` and `Switch`; or a property `hasPort`, which links a `Computer` to its `Port`, is equivalent to an attribute `ownsPort`, which contains the respective port as string representation) are not only more difficult to detect but also not supported by the majority of ontology matching tools [13,27], although a few approaches to tackle those problems exist [5,6,26]. A slightly different approach was followed by [24], where the authors propose a framework which defines executable semantic mappings between ontologies based on SWRL [16] rules and string similarity.

Nevertheless, based on the absence of ontology matching tools which are capable of detecting such complex correspondences, we assume the presence of already known correspondences between entities of the ontologies for our integration scenario.

3 WORKING EXAMPLE

As working example[4] a fictitious computer network is used and represented as UML class diagrams. Figure 5 shows the customer view (system view) of the network.

The following additional constraints hold for the system view:

- In the computer network every computer has a unique address
- A computer can be part of 1-2 subnets
- A computer is part of exactly one project

[3] All listings within this paper are serialized in Turtle syntax.
[4] The example ontologies and queries are available upon request from the first author.

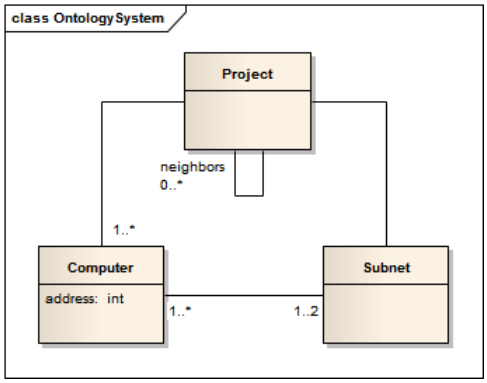


Figure 5: System Ontology

- A project is some arbitrary subdivision of the whole network (e.g. building)
- A subnet can be part of multiple projects

In the example there are 2 vendors (A and B), each providing their own configurator. A project can be configured either with configurator A or configurator B. In both cases there is one configurator database for every project. None of the domain models contains the concept of a subnet as found in the system view.

Figure 6a shows the domain model of configurator A. In the domain model of configurator A computers are called devices. Internal devices are the devices configured in the current project. External devices are devices of other projects that are directly connected to a internal device. These are needed to configure the network cards of the internal device.

Figure 7a shows the domain model of configurator B. Vendor B realizes the computer network with switches. Computers can have 1 or 2 ports, which must be connected to a port of an switch. The attribute external is set to 'true' for elements that are external to the current project.

3.1 Converting object-oriented models to ontologies

Although using Description Logics for configuration has a long history [10, 20, 28] in our experience large scale industrial configurators mostly use some form of UML-like object-oriented formalisms. For this paper we use the approach for converting object-oriented data models and their instance data into RDF/OWL shown in Table 1. Because of the clear correspondance between UML class diagrams and OWL ontologies we depict ontologies also as UML class diagrams.

This conversion captures the bare minimum that is required for our data integration approach. See [29] for a more elaborate approach for representing product configurator knowledge bases in OWL.

Listing 1 shows a fragment of the class model of Figure 6a and the instance data of Figure 6b in RDF & OWL[5].

Listing 1: Ontology A with instance data

```
# object model
ontoA:Device rdf:type owl:Class .

ontoA:InternalDevice rdf:type owl:Class ;
   rdfs:subClassOf ontoA:Device .

ontoA:Device_address rdf:type
    owl:DatatypeProperty ,
    owl:FunctionalProperty ;
   rdfs:domain ontoA:Device ;
   rdfs:range xsd:unsignedInt .

ontoA:Device_slot1Connected rdf:type
    owl:ObjectProperty ;
   rdfs:range ontoA:Device ;
   rdfs:domain ontoA:Device .

# instance data
ontoA:A1 rdf:type ontoA:InternalDevice ;
   ontoA:Device_address "1"^^xsd:unsignedInt ;
   ontoA:Device_slot1Connected
           ontoA:A2 , ontoA:A3 ;
   ontoA:Device_slot2Connected
           ontoA:B3 , ontoA:B4 .

ontoA:A3 rdf:type ontoA:InternalDevice ;
   ontoA:Device_address "3"^^xsd:unsignedInt ;
   ontoA:Device_slot1Connected
           ontoA:A1 , ontoA:A2 .

ontoA:B1 rdf:type ontoA:ExternalDevice ;
      ontoA:Device_address "4"^^xsd:unsignedInt ;
      ontoA:Device_slot1Connected
           ontoA:B2 , ontoA:A1 .

ontoA:B2 rdf:type ontoA:ExternalDevice ;
      ontoA:Device_address "5"^^xsd:unsignedInt ;
      ontoA:Device_slot1Connected
           ontoA:B1 , ontoA:A1 .
```

[5] For the sake of simplicity, we omitted `owl:DatatypeProperty` and respective project definitions.

Table 1: Convert object-oriented data models to ontologies

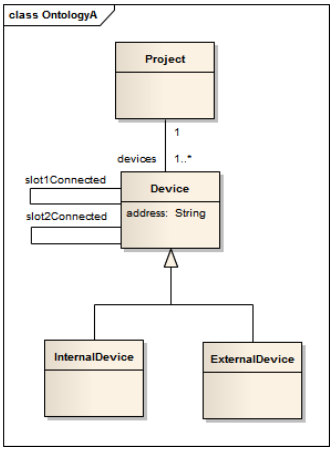
UML	RDF/OWL
class C	URI(C) rdf:type owl:Class .
C1 extends C	URI(C1) rdfs:subClassOf URI(C) .
attribute A	URI(A) rdf:type owl:DatatypeProperty , owl:FunctionalProperty ; rdfs:domain URI(C); rdfs:range TYPE(A) .
assoc A(C1,C2)	URI(A) rdf:type owl:ObjectProperty; rdfs:range URI(C1); rdfs:domain URI(C2) .
object O of class C	URI(O) rdf:type URI(C) .
attributevalue A	URI(O) URI(A) VALUE(A) .
for every tuple(O1,O2) in assoc A	URI(O1) URI(A) URI(O2).

3.2 Unique Name Assumption and Closed World Assumption

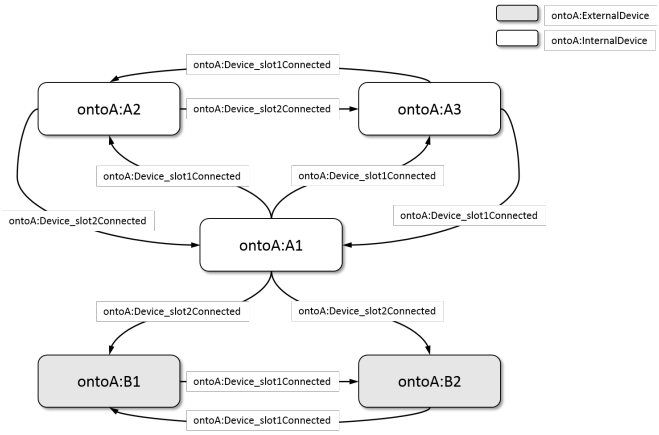
When converting the instance data of a configurator to RDF an identifier (URI) for every object must be generated. Most product configurators impose the Unique Name Assumption, i.e. objects with different object-ID refer to different objects of the domain. In the example above we therefore know that `ontoA:A1` and `ontoA:A2` refer to different Devices.

RDF/OWL does not impose the Unique Name Assumption. This is a desirable feature when reasoning about linked data. If one wants to integrate instance data from different sources using heterogeneous ontologies, these ontologies will often refer to the same entity under different URIs. The same can happen, when we integrate multiple interconnected configurations into one configuration.

Figures 6b and 7b show the configurations of two projects (A and B). Although every computer/device is only represented once in each configuration, some computers/device are known in both projects

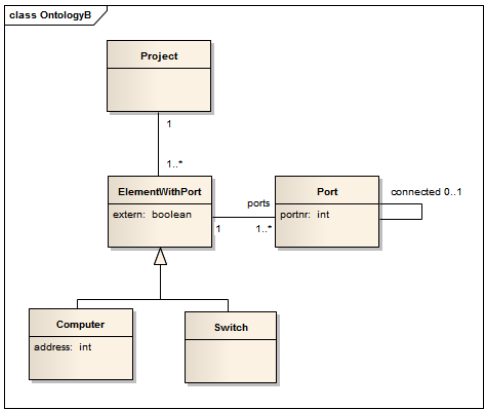


(a) Ontology A

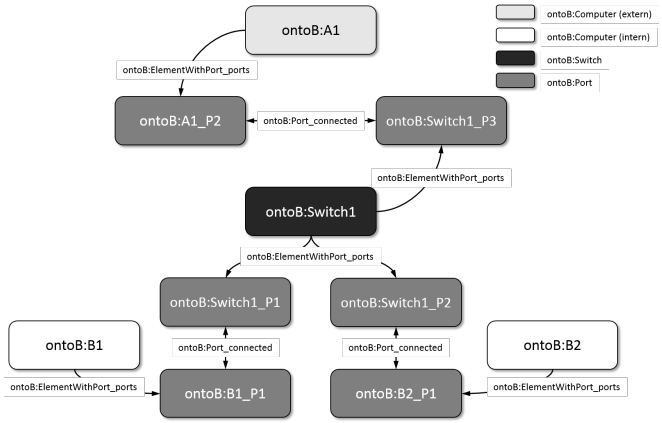


(b) Instance data of Project A

Figure 6: Ontology and instance data of Project A (Ontology A)



(a) Ontology B



(b) Instance data of Project B

Figure 7: Ontology and instance data of Project B (Ontology B)

i.e. the ExternalDevice `ontoA:B1` and the Computer `ontoB:B1` are referring to the same real world object under different URIs.

As a pragmatic solution for the Unique Name Assumption for this paper all URIs are treated as different, unless explicitly stated by `owl:sameAs`.

Similar considerations apply to the Closed World Assumption. In a configurator database one assumes that all components relevant to the current context are known. For instance in our example all the computers in the current project are known and one can use the Closed World Assumption to conclude that there are no other internal computers. The same applies to external computers that are directly connected to a internal computer. But we cannot apply the Closed World Assumption to the whole computer network, since we have no information about how many projects and computers there are in total.

4 DATA INTEGRATION WITH SPARQL

We followed an approach proposed in [7] which motivates the use of SPARQL CONSTRUCT queries to perform data integration (i.e. based on known correspondences between ontologies, we are able to translate their instance data to be conform with the structure of the integrated ontology).

4.1 Creation of the system view

As a first step in our data integration approach a system view of the configurator specific instance data is created. This system view reflects the view of the owner of the configured system and is completely self contained i.e. does not contain any URIs of the domain specific ontologies. To derive the system view from the proprietary configurator data we use SPARQL CONSTRUCT queries. Figure 6b shows a configuration of configurator A, Figure 7b shows a configuration of configurator B. The projects of the two configurations are connected via the subnet containing A1(C1), B1(C4) and B2(C5).

4.1.1 Creating instances

To map an instance of the source ontology to a new instance of the target ontology we can either generate a new URI in the namespace of the target ontology or use blank nodes.

The following example (cf. Listing 2) creates a computer in the system ontology for every device of the source ontology A by creating a new unique URI using a unique identifier of the target object (in this case the attribute address).

One advantage of using that approach is that for every instance only one URI will be created in the instance data and the order of

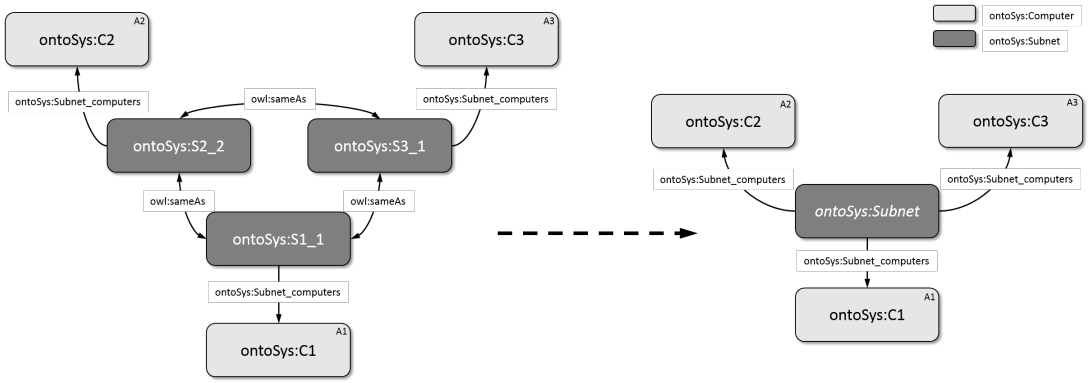


Figure 8: Equivalence relations of subnets derived from Ontology A

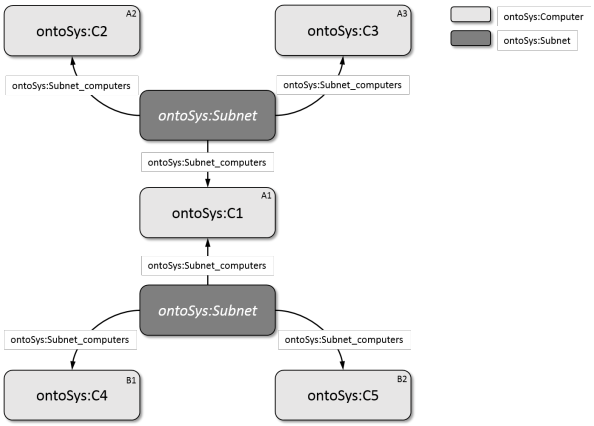


Figure 9: Instance data of Project A and Project B (System Ontology)

executing the CONSTRUCT queries does not matter.

Listing 2: Instance creation with new URI

```
CONSTRUCT {
  ?computer rdf:type ontoSys:Computer.
  ?computer ontoSys:Computer_address ?address.
}
WHERE{
  ?device ontoA:Device_address ?address.
  BIND(URI(CONCAT(URISYS,STR(?address)))
      AS ?computer)
}
```

If in contrast blank nodes are used, every CONSTRUCT query generates a new blank node for a source object. Therefore we use blank nodes only, when it is not possible or inconvenient to create a unique URI for an instance. In our example, since there is no identifier for projects in the source ontology, new projects can be created with the CONSTRUCT query shown in Listing 3.

Listing 3: Instance creation with blank node

```
CONSTRUCT {
    _:p rdf:type ontoSys:Project .
    _:p ontoSys:origin ?project .
}
WHERE{
    ?project rdf:type ontoA:Project .
}
```

By using the special object-property `ontoSys:origin`, we can keep track what led to the construction of the blank node. This information can then reused in subsequent CONSTRUCT queries.

4.1.2 Complex mapping

Sometimes it is more to convenient to use multiple URIs for the same instance, especially if there is no explicit representation of the concept of the object in the source ontology. These multiple URIs will then be related using `owl:sameAs`.

In our example the concept of a subnet is not directly represented in ontology A. To create the subnets for instance data of ontology A a more complex query is necessary as depicted in Listing 4.

Listing 4: Creating subnets from instance data

```
# C = abbreviation for URI of Computer
# SIRI = abbreviation for URI of subnets
CONSTRUCT {
  ?sub1 ontoSys:Subnet_computers ?c1 .
  ?sub2 ontoSys:Subnet_computers ?c2 .
  ?sub1 rdf:type ontoSys:Subnet .
  ?sub2 rdf:type ontoSys:Subnet .
  ?sub1 owl:sameAs ?sub2 .
}
WHERE {
  { ?d1 ontoA:Device_slot1Connected ?d2 .
    ?d1 ontoA:Device_address ?a1 .
    BIND(CONCAT(STR(?a1),"_1") AS ?sid1)
  } UNION {
    ?d1 ontoA:Device_slot2Connected ?d2 .
    ?d1 ontoA:Device_address ?a1 .
    BIND(CONCAT(STR(?a1),"_2") AS ?sid1) . }
  { ?d2 ontoA:Device_slot1Connected ?d1 .
    ?d2 ontoA:Device_address ?a2 .
    BIND(CONCAT(STR(?a2),"_1") AS ?sid2)
  } UNION {
    ?d2 ontoA:Device_slot2Connected ?d1 .
    ?d2 ontoA:Device_address ?a2 .
    BIND(CONCAT(STR(?a2),"_2") AS ?sid2) . }
  BIND(URI(CONCAT(C,STR(?a1))) AS ?c1)
  BIND(URI(CONCAT(C,STR(?a2))) AS ?c2)
  BIND(URI(CONCAT(SIRI,STR(?sid1))) AS ?sub1)
  BIND(URI(CONCAT(SIRI,STR(?sid2))) AS ?sub2)
}
```

5 USING THE INTEGRATED MODEL

The data of the different systems is available and expressed in terms of a common ontology. We can now access the data in a uniform manner and perform different kinds of operations. This section presents two classes of use cases, namely posing queries over the whole system and checking constraints concerning several systems.

5.1 Queries

After the data-integration the former heterogeneous data can now be queried in a uniform manner using only concepts of the system ontology.

Listing 5: Example Quering the system model

```
# return all the addresses used in project
SELECT ?p ?address
WHERE {
 ?p ontoSys:Project_computers ?c .
 ?c ontoSys:Computer_address ?address .
}
```

5.2 Checking constraints

If one wants to query information specific to a domain ontology, this data is still accessible via the `ontoSys:origin` link. One use case for using the `ontoSys:origin` property, is to detect inconsistencies in the source data. For example if a subnet is part of two projects, for every computer in that subnet, there must be two representations in the source ontologies (In one of these projects the computer is external). The following query checks this property.

Listing 6: Checking constraints

```
SELECT ?c ?o
WHERE {
 ?project ontoSys:Project_computers ?c .
 ?sub ontoSys:Subnet_computers ?c .
 ?sub ((owl:sameAs|^owl:sameAs)*) ?other .
 ?project2 ontoSys:Project_subnets ?other .
 FILTER(?project!=?project2)
 {
   ?c ontoSys:origin ?o .
 } MINUS {
   ?c ontoSys:origin ?o1 .
   ?c ontoSys:origin ?o2 .
   FILTER(?o1!=?o2)
 }
}
```

So far we checked the integrity of the instance data by writing special SPARQL queries. Whenever these queries are not empty a constraint violation is detected. Alternatively SPARQL CONSTRUCT queries can be used to derive a special property `ontoSys:constraintviolation` and record the reason for the inconsistencies.

Listing 7: Constraint violations

```
CONSTRUCT {
  _:cv ontoSys:constraintviolation ?c .
  _:cv ontoSys:description
    "inconsistent data" .
}
...
```

5.3 Special treatment of `owl:sameAs`

As discussed in Chapter 3.2 OWL does not impose the unique name assumpion (UNA). Therefore it is common to have different names (URIs) refer to the same real-world object. In that case they can be linked via `owl:sameAs`. SPARQL is unaware of the special semantics of `owl:sameAs`. This can be a problem, especially when using counting aggregates, since one usually wants to count the number of real-objects and not the number of URIs referring to it. Take for example a query counting the number of subnets. Our construction of subnet-URIs creates a URI for every connected port of a computer (Figure 8). A naive SPARQL query would count all distinct URIs that refer to the same subnet ($ontoSys : S1_1$, $ontoSys : S2_2$, $ontoSys : S3_1$) i.e. resulting in 3 instead of the expected answer 1. To fix this, one has to choose one representative for every element equivalence class induced by `owl:sameAs` and count the number of representatives. In our approach this is done by choosing the lexicographically smallest element.

Listing 8: Example counting predicates

```
# query without special treatment of sameAs
SELECT (COUNT(DISTINCT ?subnet) AS ?numberofsubnets)
WHERE {
  ?subnet a ontoSys:Subnet .
}
# result: numberofsubnets = 6

# query with special sameas treatment
# chooses the lexicographic first element
# as representation of the equivalence class
SELECT (COUNT(DISTINCT ?first) AS ?numberofsubnets)
WHERE {
  ?subnet a ontoSys:Subnet .
  # first subquery
  { SELECT ?subnet ?first
    WHERE {
     ?subnet ((owl:sameAs|^owl:sameAs)*) ?first .
    OPTIONAL {
     ?notfirst ((owl:sameAs|^owl:sameAs)*) ?first .
     FILTER (STR(?notfirst) < STR(?first))}
     FILTER (!BOUND(?notfirst))}
  }
}
# result: numberofsubset = 2
```

An alternative approach would be to replace all cliques of the RDF-graph linked by `owl:sameAs` with a new unique URI. We did not consider that because it requires a proprietary implementation and by replacing URIs, one loses information about the source of information. For instance, if the instance data of two different configurators refer to the same real-world object but have conflicting data-values for that object, both values and their sources must be communicated to the end-user.

6 RELATED WORK

In order to successfully perform data or information integration using Semantic Web technologies two main issues have to be addressed, namely:

Ontology Mapping Tackling the difficulties of *Ontology Mapping* (i.e. defining alignments between ontologies) extensive studies have been taken out over the last couple of years [2, 11, 12, 19], mainly focusing on resolving heterogeneity among different ontologies or data sources by detecting similarities amongst them.

Ontology Integration Two main approaches can be identified for integrating different ontologies [22], (i) define an upper ontology which contains general concepts and properties for those in the underlying more specific ones and define mappings between them and (ii) define alignments directly between underlying ontologies and use query rewriting for query support [3, 25].

With its W3C Recommendation for version 1.1 in 2013 [14], introducing e.g. UPDATE queries and a revised entailment regime, SPARQL has become more feasible to be used within information integration scenarios and not only as query language for RDF data.

Our approach can be used for data integration of distributed configurations and for reasoning about the consistency of the integrated system. It can not be used to solve (distributed) configuration problems. For a CSP-based approach on how to solve distributed configuration problems see [17].

7 CONCLUSIONS

When we started out writing this paper, we were looking for a lightweight approach for data integration for distribute configurations using standard Semantic Web technologies.

In the present paper we show that using solely SPARQL and RDFS is sufficient for an approach that relies only on standards and makes it easy to introduce new concepts and individuals on the fly using SPARQL queries. This is especially important for practical use cases, where it is unpredictable which information a customer will request about the configured system.

We tested our approach with real-world data. On a standard Windows-7 laptop with 8 GB using the SPARQL-API of JENA 2.11.1 a large database (>50000 instances) can be integrated in less than 5 minutes resulting in a RDF-graph with more than 500000 triples.

We also considered using OWL reasoners but could not find a solver-independent way of creating new individuals. Nevertheless, for future work we plan to look into using richer OWL ontologies, which would offer the possibility to use configurator specific concepts such as part-subpart, resource, (hardware-)component etc.

As can be seen in the example SPARQL queries in this paper, some queries (especially the ones that take into account `owl:sameAs` properties), are only understandable for a SPARQL expert. One approach for making queries more accessible for a SPARQL beginner would be to hide the special treatment of `owl:sameAs` from the inexperienced user by using query rewriting.

ACKNOWLEDGEMENTS

Stefan Bischof and Simon Steyskal have been partially funded by the Vienna Science and Technology Fund (WWTF) through project ICT12-015.

Simon Steyskal has been partially funded by ZIT, the Technology Agency of the City of Vienna (Austria), in the programme ZIT13 plus, within the project COSIMO (Collaborative Configuration Systems Integration and Modeling) under grant number 967327.

REFERENCES

[1] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. Turtle – Terse RDF Triple Language. W3C Candidate Recommendation, February 2013. `http://www.w3.org/TR/2013/CR-turtle-20130219/`.

[2] Namyoun Choi, Il-Yeol Song, and Hyoil Han, 'A survey on ontology mapping', *ACM Sigmod Record*, **35**(3), 34–41, (2006).

[3] Gianluca Correndo, Manuel Salvadores, Ian Millard, Hugh Glaser, and Nigel Shadbolt, 'Sparql query rewriting for implementing data integration over linked data', in *Proceedings of the 2010 EDBT/ICDT Workshops*, p. 4. ACM, (2010).

[4] Chris J Date and Hugh Darwen, *SQL. Der Standard.: SQL/92 mit den Erweiterungen CLI und PSM.*, Pearson Deutschland GmbH, 1998.

[5] Robin Dhamankar, Yoonkyong Lee, AnHai Doan, Alon Halevy, and Pedro Domingos, 'imap: discovering complex semantic matches between database schemas', in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pp. 383–394. ACM, (2004).

[6] AnHai Doan and Alon Y Halevy, 'Semantic integration research in the database community: A brief survey', *AI magazine*, **26**(1), 83, (2005).

[7] Jérôme Euzenat, Axel Polleres, and François Scharffe, 'Processing ontology alignments with sparql', in *Complex, Intelligent and Software Intensive Systems, 2008. CISIS 2008. International Conference on*, pp. 913–917. IEEE, (2008).

[8] Jérôme Euzenat, Pavel Shvaiko, et al., *Ontology matching*, volume 18, Springer, 2007.

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

[10] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, Markus Stumptner, and Markus Zanker, 'Configuration knowledge representations for semantic web applications', *AI EDAM*, **17**(1), 31–50, (2003).

[11] Chiara Ghidini and Luciano Serafini, 'Mapping properties of heterogeneous ontologies', in *Artificial Intelligence: Methodology, Systems, and Applications*, 181–193, Springer, (2008).

[12] Chiara Ghidini, Luciano Serafini, and Sergio Tessaris, 'On relating heterogeneous elements from different ontologies', in *Modeling and Using Context*, 234–247, Springer, (2007).

[13] Bernardo Cuenca Grau, Zlatan Dragisic, Kai Eckert, Jérôme Euzenat, Alfio Ferrara, Roger Granada, Valentina Ivanova, Ernesto Jiménez-Ruiz, Andreas Oskar Kempf, Patrick Lambrix, et al., 'Results of the ontology alignment evaluation initiative 2013', in *Proc. 8th ISWC workshop on ontology matching (OM)*, pp. 61–100, (2013).

[14] Steve Harris and Andy Seaborne, 'Sparql 1.1 query language', *W3C Reccomendation*, **14**, (2013).

[15] Patrick Hayes and Brian McBride. Rdf semantics. W3C Recommendation, February 2004. `http://www.w3.org/TR/rdf-mt/`.

[16] Ian Horrocks, Peter F Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, Mike Dean, et al., 'Swrl: A semantic web rule language combining owl and ruleml', *W3C Member submission*, **21**, 79, (2004).

[17] Dietmar Jannach and Markus Zanker, 'Modeling and solving distributed configuration problems: A csp-based approach', *Knowledge and Data Engineering, IEEE Transactions on*, (99), 1–1, (2011).

[18] Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau, 'Logmap: Logic-based and scalable ontology matching', in *The Semantic Web–ISWC 2011*, 273–288, Springer, (2011).

[19] Yannis Kalfoglou and Marco Schorlemmer, 'Ontology mapping: the state of the art', *The knowledge engineering review*, **18**(01), 1–31, (2003).

[20] Deborah L. McGuinness and Jon R. Wright, 'An industrial strength description logics-based configurator platform', *IEEE Intelligent Systems*, **13**(4), 69–77, (July 1998).

[21] DuyHoa Ngo and Zohra Bellahsene, 'Yam++: a multi-strategy based approach for ontology matching task', in *Knowledge Engineering and Knowledge Management*, 421–425, Springer, (2012).

[22] Natalya F Noy, 'Semantic integration: a survey of ontology-based approaches', *ACM Sigmod Record*, **33**(4), 65–70, (2004).

[23] Axel Polleres, 'Sparql1. 1: New features and friends (owl2, rif)', in *Web Reasoning and Rule Systems*, 23–26, Springer, (2010).

[24] Han Qin, Dejing Dou, and Paea LePendu, 'Discovering executable semantic mappings between ontologies', in *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, 832–849, Springer, (2007).

[25] Bastian Quilitz and Ulf Leser, 'Querying distributed rdf data sources with sparql', in *The Semantic Web: Research and Applications*, 524–538, Springer, (2008).

[26] Dominique Ritze, Christian Meilicke, O Sváb-Zamazal, and Heiner Stuckenschmidt, 'A pattern-based ontology matching approach for detecting complex correspondences', in *ISWC Workshop on Ontology Matching, Chantilly (VA US)*, pp. 25–36. Citeseer, (2009).

[27] Pavel Shvaiko and Jérôme Euzenat, 'Ontology matching: state of the art and future challenges', *Knowledge and Data Engineering, IEEE Transactions on*, **25**(1), 158–176, (2013).

[28] Timo Soininen, Juha Tiihonen, Tomi Männistö, and Reijo Sulonen, 'Towards a general ontology of configuration', *Artif. Intell. Eng. Des. Anal. Manuf.*, **12**(4), 357–372, (September 1998).

[29] Dong Yang, Rui Miao, Hongwei Wu, and Yiting Zhou, 'Product configuration knowledge modeling using ontology web language', *Expert Syst. Appl.*, **36**(3), 4399–4411, (April 2009).

Configuring Decision Tasks

Martin Stettinger[1] and **Alexander Felfernig**[1] and **Michael Jeran**[1] and **Gerald Ninaus** [1] and **Gerhard Leitner** [2] and **Stefan Reiterer** [3]

Abstract. In most cases, decision tasks are individual and different decision tasks require different combinations of features. Features can be, for instance, special preference visibilities during the decision process or specific heuristics that support the recommendation of decisions. To find the right features for a decision task it is essential to offer a corresponding configuration functionality. In this paper we illustrate how the design of a decision task can be represented as a configuration problem. The underlying configuration knowledge is already integrated in a tool called CHOICLA.

1 Introduction

Decisions have to be taken in different situations - for example a decision about the destination for the next holidays or a decision about which restaurant to choose for a dinner with friends. Decision scenarios can differ from each other in terms of their process design. Some decision scenarios rely on a preselected decision heuristic that defines the criteria for taking the decision, for example, a group decides to use majority voting for deciding about the next restaurant visit. Furthermore, the visibility of the preferences of other users is an important feature that can be configured by the creator of a decision task.

In this paper we show how the design of decision tasks (the underlying process) can be defined as a configuration problem. The major advantage of this approach is that making the process design of decision tasks configurable introduces the flexibility that is needed due to the heterogenity of decision problems. This way we are able to build a model that is flexible with regard to the implementation (generation) of problem-specific decision applications. The knowledge representations introduced in the following are included in the CHOICLA decision support environment (see www.choicla.com).

The remainder of this paper is organized as follows. In the next section (Section 2) we discuss features that are essential to the design of a decision task. In Section 3 we introduce dependencies that exist between features. In Section 4 we provide insights into group recommendation approaches integrated in the CHOICLA environment. We then discuss related and future work and thereafter conclude the paper.

2 Configuring a decision task

In the following we discuss different features that are relevant when designing (configuring) a decision task. On a formal level, we represent a *decision task configuration problem* as a constraint satisfaction problem [12] and [5] (CSP – see Definition 1).

Definition 1 (Constraint Satisfaction Problem). A CSP consists of (1) a set of finite-domain variables $X = \{x_1, x_2, ..., x_n\}$ and (2) a set of constraints $C = \{c_1, c_2, ..., c_m\}$. For each variable x_i out of X there exists a finite set D_i (domain of the variable) of possible assignments. Possible variable assignments can be limited via constraints. A complete assignment (every variable has a corresponding value) which is consistent with the constraints in C is denoted as a solution for a CSP.

For the purpose of better understandability we use a feature model notation to express variability properties of decision tasks. A feature model (FM) represents a set of possible features and relationships between them. Features are arranged hierarchically which is basically a tree structure with one root feature [2]. Within this tree structure the nodes are the features and the edges are the relationships (constraints). A more detailed discussion of different feature model representations can be found in [1], [2] and [3].

Six different types of constraints (relationships) are typically used for the construction of feature models ([1], [2]): *mandatory, optional, alternative, or, requires* and *excludes*. Feature models are representing configurable products which can be formalized in the form of a CSP. A feature f is *included* if the value is set to 1 - otherwise it is said to be *excluded*. We will exemplify this formalization on the basis of feature model depicted in Figure 1. Figure 1 shows a fragment of the CHOICLA feature model [4].
The CSP representation of the feature model depicted in Figure 1 is the following:

$V = \{f_1, f_2, ..., f_{21}\}$

$dom(f_1) = dom(f_2) = ... = dom(f_{21}) = \{0, 1\}$

$c_1 : f_1 \leftrightarrow f_2$
$c_2 : f_1 \leftrightarrow f_3$
$c_3 : f_1 \leftrightarrow f_4$
$c_4 : f_1 \leftrightarrow f_5$
$c_5 : f_6 \rightarrow f_1$
$c_6 : (f_7 \leftrightarrow (\neg f_8 \wedge f_3)) \wedge (f_8 \leftrightarrow (\neg f_7 \wedge f_3))$
$c_7 : (f_9 \leftrightarrow (\neg f_{10} \wedge \neg f_{11} \wedge f_4)) \wedge (f_{10} \leftrightarrow (\neg f_9 \wedge \neg f_{11} \wedge f_4))$
$\wedge (f_{11} \leftrightarrow (\neg f_9 \wedge \neg f_{10} \wedge f_4))$

[1] Graz, University of Technology, Austria, email: firstname.lastname@ist.tugraz.at
[2] Alpen Adria University, Austria, email: gerhard.leitner@aau.at
[3] SelectionArts Intelligent Decision Technologies GmbH, Austria, email: s.reiterer@selectionarts.com

[4] A more in-depth discussion of the CHOICLA decision support environment can be found in [18].

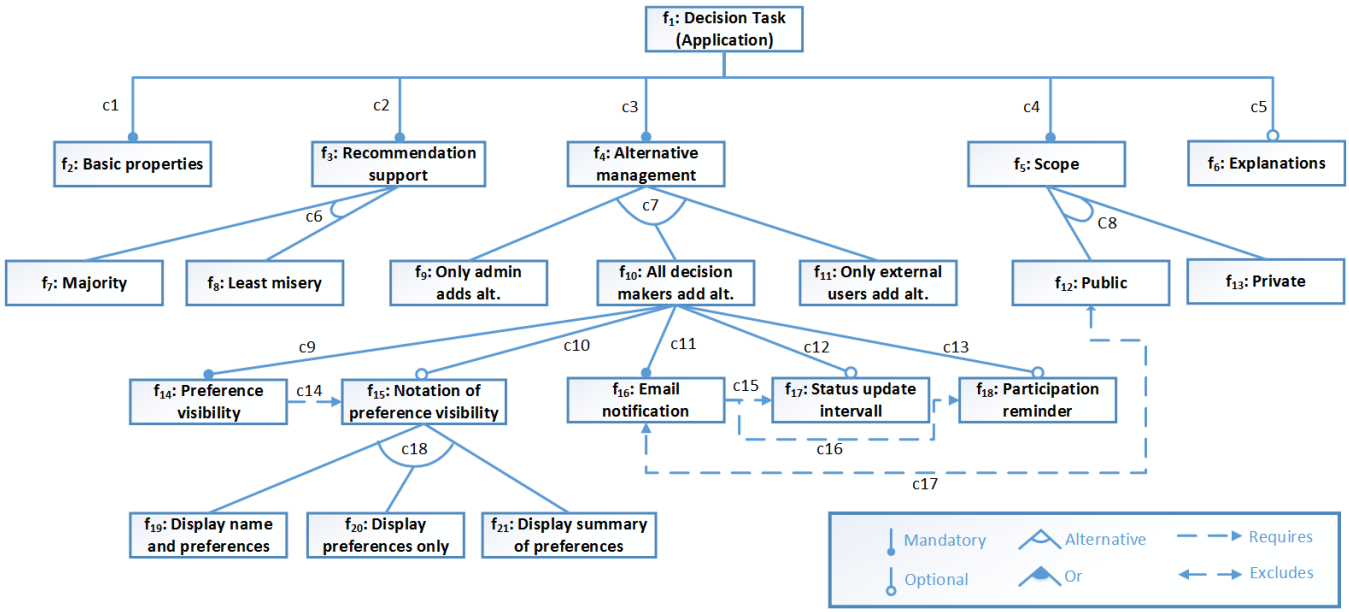


Figure 1. Fragment of the CHOICLA feature model. In this model, f_i are used as abbreviation for the individual features, for example, f_1 is the short notation for feature *Decision Task (Application)*.

$$c_8 : (f_{12} \leftrightarrow (\neg f_{13} \wedge f_5)) \wedge (f_{13} \leftrightarrow (\neg f_{12} \wedge f_5))$$
$$c_9 : f_{14} \leftrightarrow f_{10}$$
$$c_{10} : f_{15} \rightarrow f_{10}$$
$$c_{11} : f_{16} \leftrightarrow f_{10}$$
$$c_{12} : f_{17} \rightarrow f_{10}$$
$$c_{13} : f_{18} \rightarrow f_{10}$$
$$c_{14} : f_{14} \rightarrow f_{15}$$
$$c_{15} : f_{16} \rightarrow f_{17}$$
$$c_{16} : f_{16} \rightarrow f_{18}$$
$$c_{17} : \neg(f_{16} \wedge f_{12})$$
$$c_{18} : (f_{19} \leftrightarrow (\neg f_{20} \wedge \neg f_{21} \wedge f_{15})) \wedge (f_{20} \leftrightarrow (\neg f_{19} \wedge \neg f_{21} \wedge f_{15})) \wedge (f_{21} \leftrightarrow (\neg f_{19} \wedge \neg f_{20} \wedge f_{15}))$$

We will now discuss different basic properties of decision task configuration problems. In this context we explain the individual features and constraints depicted in Figure 1.

Basic properties. Each decision task is characterized by a name, a corresponding description, and a picture that represents the decision task (summarized in the feature *Basic Properties* for simplification purposes).

Management of alternatives. There are different possibilities to support alternative management within the scope of a decision task. *First*, only the creator of a decision task is allowed to add alternatives – this could be the case if a person is interested to know the opinions of his/her friends about a certain set of alternatives (e.g., alternative candidates for the next family car). Another related scenario are so-called "Micro-Polls" where the creator is only interested in knowing the preference distribution of a larger group of users. *Second*, in some scenarios it should be possible that all decision makers can add alternatives – a typical example of such a scenario is the group-based decision regarding a holiday destination or a hotel [10]. In this context, each user should be allowed to add relevant alternatives. An example scenario of the *third* case (only external users can add alternatives) is the support of group-based personnel decisions – in this context it should be possible that persons apply for a certain position (the application itself is interpreted as the addition of a new alternative to the decision task).

Scope. The scope of a decision task denotes the external visibility. The scope "private" allows only invited users to participate, i.e., the task is not visible for other users except those who have been invited. If the scope is "public", the decision task is visible to all users – this is typically the case in the context of so-called Micro-Polls. The selection of the scope has an impact on other features – related aspects will be discussed in Section 3.

Preference visibility. The visibility of individual preferences of the other participants involved in a decision process can have an impact on decision quality (see [6], [10], and [11]). There occur some decision scenarios where all participants should exactly know which person articulated a rating of an alternative. If, for example, a date for a business meeting is the topic of the decision task it is very essential to find a date where all division managers can attend the meeting and therefore it is important to know the individual preferences of the participants in that case. But there are of course decision scenarios where preference visibility can lead to disadvantages for some participants but still some kind of transparency of the preferences is helpful to come to the best decision. In such cases a summary of all given preferences of an alternative is a good way to support the participants best during the decision process. A summary prevents all participants from statistical inferences but still can help participants who are not sure about which rating to select.

Email notification. If this feature is set, emails can be used to exchange information about the current state of the decision process.

For example, the status update interval specifies in which intervals participants of a decision process receive a summary of the current status of the decision process. The active participation reminder is a feature which helps to trigger need for closure. If this feature is set, a maximum inactive time (without looking at the current status of the decision task) for the participants can be set. After this time is elapsed an email will be sent to the corresponding participants to encourage an active participation at the decision task.

Recommendation support. In context of group decision tasks another very essential aspect is the aggregation function (recommendation heuristic). Aggregation functions can help to foster consensus in a group decision process, furthermore, user studies show that these functions also help to increase the degree of the perceived decision quality (see, for example [6]). Preferences of individual users can be aggregated in many different ways and there exists no standard heuristic which fits for every decision scenario. To support groups of users in different scenarios the selection of recommendation heuristics is a necessary feature which has to be configured by the creator of a decision task. Some basic aggregation heuristics which can be used in such cases are described below. For an in-depth discussion of basic types of aggregation heuristics see, for example, the overview of Masthoff [14]. The example given in Table 1 represents the individual ratings of the participants for the defined alternatives. The results of applying the decision heuristics discussed below are depicted in Table 2.

restaurant	Martin	Dave	George	Ben
Clocktower	5	3	5	4
Häuserl im Wald	3	3	5	3
La Botte	5	3	3	3
El Gaucho	4	3	4	4

Table 1. Examples of user-specific ratings with regard to the available decision alternatives (restaurants).

Majority Voting (see Formula 1) determines the value (d) that a majority of the users selected as voting for a specific solution s where $eval(u, s)$ denotes the rating for solution s defined by user u. For example, the majority of votings for *Clocktower* is 5 (see Table 2).

$$MAJ(s) = maxarg_{(d \in \{1..5\})}(\#(\bigcup_{u \in Users} eval(u, s) = d)) \quad (1)$$

Least Misery (see Formula 2) returns the lowest voting for solution s as group recommendation. For example, the LMIS value for the s = *Clocktower* is 3.

$$LMIS(s) = min(\bigcup_{u \in Users} eval(u, s)) \quad (2)$$

Most Pleasure (see Formula 3) returns the highest voting for solution s as group recommendation. For example, the MPLS value for the s = *Clocktower* is 5.

$$MPLS(s) = max(\bigcup_{u \in Users} eval(u, s)) \quad (3)$$

Group Distance (see Formula 4) returns the value d as group recommendation which causes the lowest overall change of the individual user preferences. For example, the GDIS value for s = *Clocktower* is 5 (or, alternatively 4).

$$GDIS(s) = minarg_{(d \in \{1..5\})}(\sum_{u \in Users} |eval(u, s) - d|) \quad (4)$$

Finally, *Ensemble Voting* (see Formula 5) determines the majority of the results of the individual voting strategies H = {MAJ, LMIS, MPLS, GDIS}. For example, the ensemble-based majority voting for *Clocktower* is 5.

$$ENS(s) = maxarg_{(d \in \{1..5\})}(\#(\bigcup_{h \in H} eval(h, s) = d)) \quad (5)$$

solution	*MAJ*	*LMIS*	*MPLS*	*GDIS*	*ENS*
Clocktower	5	3	5	5	5
Häuserl im Wald	3	3	5	3	3
La Botte	3	3	5	3	3
El Gaucho	4	3	4	4	4

Table 2. Results of applying the aggregation functions to the user preferences shown in Table 1. MAJ = Majority Voting; LMIS = Least Misery; MPLS = Most Pleasure; GDIS = Lowest Group Distance; ENS = Ensemble Voting. This example is based on the preference information in Table 1.

Explanations. Explanations can play an important role in decision tasks since they are able to increase the trust of users in the outcome of a decision process [4]. When configuring a decision task in CHOICLA, explanations can be selected as a feature of the decision process. In the current version of CHOICLA, explanations are supported by simply allowing the creator of the decision process to include textual argumentations as to why a certain decision alternative has been selected as "the final decision". If this feature is selected, the administrator of a decision task has to enter some explanatory text, if not, the entering of such a text remains just an option.

3 Dependencies among features

We now discuss examples of constraints that restrict the combinations of features as shown in the feature model of Figure 1. The constraint-based representation of these constraints is shown in the CSP definition of the feature model given in Section 2.

Scope of a decision. If a decision task is public, there are restrictions regarding the support of message interchange (e.g., via email) and the visualization of the preferences of other users. In the case that a decision task is private, it is in both cases possible to choose. Preferences can (but must not) be made visible to other users and the type of possible message interchange can be specified. The differentiation between public and private decision tasks also has an impact on other system properties. For example, if a decision task is defined as private, the corresponding decision application can not be reused by other users, i.e., found as a result via the CHOICLA search interface.

Preference visibility. A dependency of type 'requires' exists between the feature *preference visibility* and the corresponding notation of visibility. Preference visibility denotes a functionality where the individual preferences of other users are made visible for the current user. The type of visualization can only be selected in the case that the *preference visibility* feature is has been selected by the designer of a decision task.

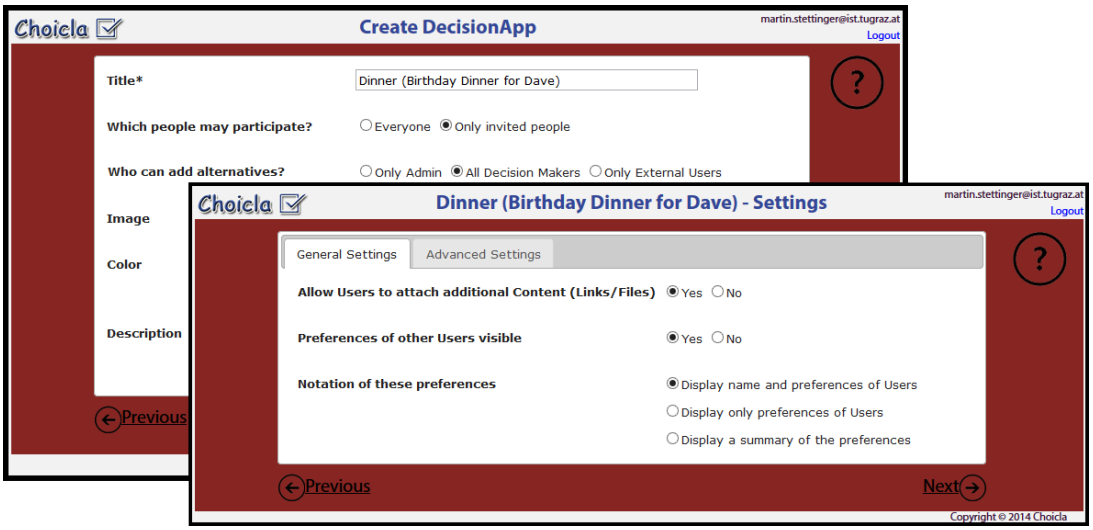


Figure 2. CHOICLA: definition of a decision task. Basic settings & further configurable features in case the decision makers are allowed to contribute own alternatives during the decision process.

Email notification. Similar to the visibility of preferences, the type of supported message exchange (e.g., via email) can only be specified in the case that the creator of the decision task decided to support email notifications. As already mentioned, email communication is only supported if the scope of the decision task is private.

These simple examples already show the need to manage decision task related variability in a structured fashion. Our knowledge representation approach allows for a product line oriented development of decision support functionalities and makes systems much more flexible for future requirements and corresponding extensions.

4 Configuring decision tasks in CHOICLA

In the following we give an example of how a decision task can be configured in the CHOICLA decision support environment (www.choicla.com). The application knowledge base of CHOICLA is currently rule-based. For reasons of easier maintenance and adaptability we apply reasoning and CSP for future versions of CHOICLA.

Parts of the user interface that supports a creator of a decision task are depicted in Figure 2. The possible parametrizations correspond to the features in the model of Figure 1. If, for example, a specific feature A depends on the inclusion of another feature B, this is taken into account in the user interface, i.e., such a feature (feature A) can only be selected, if the other feature (feature B) is also selected. In the example of Figure 2, the *scope* of the decision task is private (only invited users can participate), all decision makers are allowed to add alternatives, and for all participants of the decision process the preferences of other users are visible (names as well as preferences). Note that in the CHOICLA environment there are many additional features that can be selected within the scope of a decision task configuration process.
For understandability reasons we kept our working example simple and focused on aspects that give the reader an impression of the basic underlying configuration problem. The user interface for the inclusion of alternatives is depicted in Figure 3.
Figure 4 shows how the decision alternatives can be voted by the individual users of a decision task.

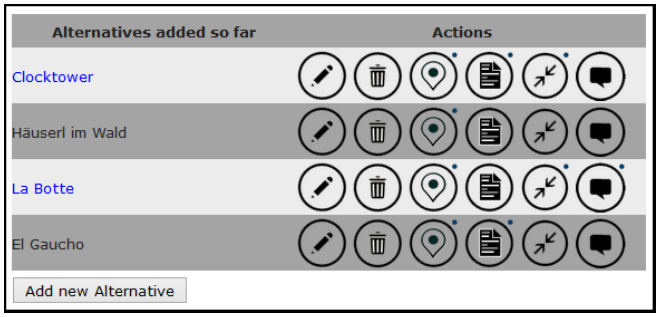


Figure 3. CHOICLA: user interface for addition of decision alternatives. The dots in the upper right corner of every symbol indicate whether there is an information in this category available or not. The meaning of the used symbols is (from left to right): *edit*, *delete*, *geographical information*, *files*, *links* and *comments*.

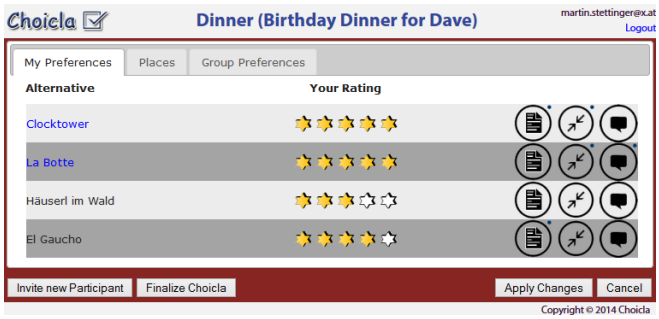


Figure 4. CHOICLA: user interface for individual voting of decision alternatives. Each alternative can be voted by a five-star scale. The tab *Places* shows the geographical distribution of the decision alternatives (if available). In tab *Group Preferences* the actual group recommendation as well as the individual preferences of the other users (if feature $f14$ is set) is presented to the users. The process where the "final decision" can be set is triggered by the button *Finalize Choicla*.

5 Related and Future Work

There exist a couple of online tools which support different types of decision scenarios. *The Decider*[5] is a tool that allows the creation of

[5] labs.riseup.net.

issues and decision alternatives – the corresponding recommendation is provided to users who are articulating their preferences regarding the given decision alternatives. Rodriguez et al. [17] introduce *Smartocracy* which is a decision support tool which supports the definition of tasks (issues or questions) and corresponding solutions. Solution selection (recommendation) is based on exploiting information from an underlying social network which is used to rank alternative solutions. *Dotmocracy*[6] is a method for collecting and visualizing the preferences of a large group of users. It is related to the idea of participatory decision making – it's major outcome is a graph type visualization of the group-immanent preferences. Doodle[7] focuses on the aspect of coordinating appointments – similarly, VERN [19] is a tool that supports the identification of meeting times based on the idea of unconstrained democracy where individuals are enabled to freely propose alternative dates themselves. Compared to CHOICLA these tools are not able to customize their decision processes depending on the application domain and are also focused on specific tasks. Furthermore, no concepts are provided which help to improve the overall quality of group decisions, for example, in terms of integrating explanations, recommendations for groups, and consistency management for user preferences.

The support of group decision processes on the basis of recommendation technologies is a new and upcoming field of research (see, e.g., Masthoff et al. [14]). The application of group recommendation technologies is still restricted to specific domains such as interactive television [13], e-tourism [9, 15], software requirements engineering [6], and ambient intelligence [16].

Future Work. Our future work will focus on the analysis of further application domains for the CHOICLA technologies. Our vision is to make the design (implementation) of group decision tasks as simple as possible. The resulting decision task should be easy to handle for users and make group decisions in general more efficient. Within the scope of our work we will also focus on the analysis of decision phenomena within the scope of group decision processes. Phenomena such as decoy effects [7] and anchoring effects [8] are well known for single-user cases but are not investigated in group-based decision scenarios. Finally, we will also focus on the development of further group recommendation heuristics. In this context, our major goal is to make the CHOICLA datasets available to the research community in an anonymized fashion for experimentation purposes.

6 Conclusions

In this paper we have shown how to represent the design of decision tasks as a configuration problem. In this context, we gave a short introduction to the CHOICLA group decision environment which supports the flexible design and execution of different types of group decision tasks. Compared to existing group decision support approaches, CHOICLA provides an end user modelling environment which supports an easy development and execution of group decision tasks.

ACKNOWLEDGEMENTS

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

[6] dotmocracy.org.
[7] doodle.com.

REFERENCES

[1] Don Batory, 'Feature models, grammars, and propositional formulas', in *Proceedings of the 9th International Conference on Software Product Lines*, SPLC'05, pp. 7–20, Berlin, Heidelberg, (2005). Springer-Verlag.
[2] David Benavides, Sergio Segura, and Antonio Ruiz-Corts, 'Automated analysis of feature models 20 years later: A literature review', *Information Systems*, **35**(6), 615 – 636, (2010).
[3] A. Felfernig, D. Benavides, J. Galindo, and F. Reinfrank, 'Towards Anomaly Explanation in Feature Models', *Workshop on Configuration, Vienna, Austria*, 117–124, (2013).
[4] A. Felfernig, B. Gula, and E. Teppan, 'Knowledge-based Recommender Technologies for Marketing and Sales', *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, **21**(2), 1–22, (2006).
[5] Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, *Knowledge-based Configuration – From Research to Business Cases*, Elsevier, 2014.
[6] Alexander Felfernig, Christoph Zehentner, Gerald Ninaus, Harald Grabner, Walid Maalej, Dennis Pagano, Leopold Weninger, and Florian Reinfrank, 'Group decision support for requirements negotiation', in *Advances in User Modeling*, eds., Liliana Ardissono and Tsvi Kuflik, volume 7138 of *Lecture Notes in Computer Science*, 105–116, Springer Berlin Heidelberg, (2012).
[7] J. Huber, J. Payne, and C. Puto, 'Adding Asymmetrically Dominated Alternatives: Violations of Regularity and the Similarity Hypothesis', *The Journal of Consumer Research*, **9**(1), 90–98, (1982).
[8] K. Jacowitz and D. Kahneman, 'Measures of Anchoring in Estimation Tasks', *Personality and Social Psychology Bulletin*, **21**(1), 1161–1166, (1995).
[9] A. Jameson, S. Baldes, and T. Kleinbauer, 'Two methods for enhancing mutual awareness in a group recommender system', in *ACM Intl. Working Conference on Advanced Visual Interfaces*, pp. 48–54, Gallipoli, Italy, (2004).
[10] Anthony Jameson, 'More than the sum of its members: challenges for group recommender systems', in *Proceedings of the working conference on Advanced visual interfaces*, AVI '04, pp. 48–54, New York, NY, USA, (2004). ACM.
[11] Anthony Jameson and Barry Smyth, 'Recommendation to groups', in *The Adaptive Web*, eds., Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, volume 4321 of *Lecture Notes in Computer Science*, 596–627, Springer Berlin Heidelberg, (2007).
[12] Alan Mackworth, 'Consistency in networks of relations', *Artificial Intelligence*, **8**(1), 99–118, (1977). Reprinted in Readings in Artificial Intelligence.
[13] J. Masthoff, 'Group modeling: Selecting a sequence of television items to suit a group of viewers', *User Modeling and User-Adapted Interaction (UMUAI)*, **14**(1), 37–85, (2004).
[14] J. Masthoff, 'Group Recommender Systems: Combining Individual Models', *Recommender Systems Handbook*, 677–702, (2011).
[15] K. McCarthy, M. Salamo, L. Coyle, L. McGinty, B. Smyth, and P. Nixon, 'Group recommender systems: a critiquing based approach', in *2006 International Conference on Intelligent User Interfaces (IUI 2006)*, pp. 282–284, Sydney, Australia, (2006). ACM.
[16] I. Perez, F. Cabrerizo, and E. Herrera-Viedma, 'A Mobile Decision Support System for Dynamic Group Decision-Making Problems', *IEEE Transactions on Systems, Man, and Cybernetics*, **40**(6), 1244–1256, (2010).
[17] M. Rodriguez, D. Steinbock, J. Watkins, C. Gershenson, J. Bollen, V. Grey, and B. deGraf, 'Smartocracy: Social networks for collective decision making', in *HICSS 2007*, p. 90, Waikoloa, Big Island, HI, USA, (2007). IEEE.
[18] Martin Stettinger, Gerald Ninaus, Michael Jeran, Florian Reinfrank, and Stefan Reiterer, 'We-decide: A decision support environment for groups of users', in *Recent Trends in Applied Artificial Intelligence*, eds., Moonis Ali, Tibor Bosse, KoenV. Hindriks, Mark Hoogendoorn, CatholijnM. Jonker, and Jan Treur, volume 7906 of *Lecture Notes in Computer Science*, 382–391, Springer Berlin Heidelberg, (2013).
[19] S. Yardi, B. Hill, and S. Chan, 'VERN: Facilitating Democratic group Decision Making Online', in *International ACM SIGGROUP Conference on Supporting Group Work (GROUP 2005)*, pp. 116–119, Sanibel Island, Florida, USA, (2005). ACM.

A backtrack-free process for deriving product family members

Homero M. Schneider[1]

Abstract. In this paper, we present a new approach for the customisation of product families. It is based on a knowledge framework for representing product families that combines a generic product structure and an extension of the classical constrain network model by the attachment of design functions to the variables. We also present a method for deriving family members from this framework, which consists of a two-stage process. First, a solution to the constraint network is found which is consistent with the set of customer requirements. Second, the solution is used to transform the generic structure into a specific one corresponding to a product family member that meets the customer requirements. One major outcome of the design functions is the establishment of instantiation patterns that guide the problem-solving process. Moreover, if a few modelling conditions are satisfied, it can be proved that finding solutions becomes a backtrack-free process. As a practical example, this approach is used for the implementation of a prototype configurator for a solar powered pumping system.

1 INTRODUCTION

Since the proposal made by Mittal and Frayman [1] to represent product configuration as a CSP problem, many extensions have been put forward to cope with the specificities of configuration problems [2]. Moreover, to improve the efficiency of the product configuration process, it is a practice to use knowledge about the problem domain to guide the search process [3]. Following this rationale, this paper presents an approach to derive members of a product family that exploits the specificities intrinsic to this concept.

It is well known that the design of a product family is a "difficult and challenging task" [4], for it requires the development of multiple products at the same time. However, after the product family is designed, it should not be a surprise that the process of deriving its members can be turned into a routine design task. This claim follows from the fact that during the design process, designers acquire a great amount of knowledge regarding the product family architecture, how the variable aspects depend on each other and their range of variability.

The approach presented in this paper is based on a knowledge framework which combines two general models. A generic product structure (GPS) that represents the product family architecture, and a constraint network model extended with design functions (CN-F) to complement the GPS in the definition of the product family members. The CN-F model is an extension of the classical constraint network (CN) model by the attachment of design functions to its variables. The primary role of these functions is to generate the values for the variables to which they are attached during the customisation process. However, design functions are also used to elicit the dependencies between the variables to form dependency patterns.

In our approach, members of the product family are derived from the knowledge framework as instantiations into two stages. First, a solution to the CN-F model has to be found from the customer requirements. This process is guided by dependency patterns. Then, the solution obtained is used to transform the GPS into a specific physical model that corresponds to a product family member, one that meets the customer requirements.

Although the instantiation patterns can restrict the design space to relatively few variables, they cannot avoid backtracking. Thus, another important contribution of this work is the setting up of modelling conditions such that if the CN-F model satisfies them, the instantiation process becomes backtrack-free. These conditions eliminate the sources of inconsistencies during the execution of the instantiation algorithm proposed for the CN-F model.

In contrast to other approaches that claim to be backtrack-free [5, 6], which typically resort to a pre-processing stage and to computational power, our approach resort to the structuration of the customization process of product families. As a result, it is possible to implement very efficient configurators based on the data flow principle.

As for the remaining of this work, in the next section we review the related literature. In Section 3, we present the SPPS system, which will be used along the paper as our practical example, the solar powered pumping system. In Section 4, we introduce our knowledge framework, by defining the elements of the GPS and CN-F models. In Section 5, we introduce our method for deriving product family members. First, we present our instantiation algorithm. After that, we introduce the conditions for which this algorithm is backtrack-free. Then, we present the method for transforming the GPS into a specific product model. In Section 6, we present the implementation of our prototype configurator. Finally, in Section 7, we make our concluding remarks.

2 RELATED WORK

One early proposal to extend the CSP model was made by Mittal and Falkenhainer [7], who proposed a dynamic constraint satisfaction problem (DCSP) to deal with the fact that the set of variables that are relevant for the solution of a configuration problem may change dynamically during the problem solving. To deal with the structural aspect of configuration problems, Sabin and Freuder [8] proposed a composite CSP. In their approach, the variables are allowed to represent an entire sub problem, such as

[1] Centre for Information Technology Renato Archer, Campinas, Brazil, email: homero.schneider@cti.gov.br

the constituent parts of the final product or the internal structure of components. In [2], Veron et al. proposed to model the configurable product as a tree with internal nodes representing sub-configurable components and leaf nodes corresponding to elementary configurable or standard components. The attributes of the configurable components are represented as variables and each component is associated to a state variable. The configuration process works on two levels. First, the state variables are used to manage the tree structure. Then, the CSP problem is addressed to define the attributes of the active components. The user expresses his choices by adding/retracting unary constraints.

The CSP approaches have been focused mostly on discrete variables and binary constraints. However, in the configuration of engineering products, it is quite common to have continuous variables and constraint on multiple variables. Thus, Gelle et al. [9] introduced local consistency methods to handle discrete and numerical variables and in the same framework to address engineering products represented as a CSP.

With a few exceptions, dependencies have been largely neglected in product configuration approaches. In [10], Xie et al. proposed the Dependent CSP. In this approach, the variables can be related by dependencies or constraints and are divided into independent and dependent by means of the relation of dependency. The independent variables are assigned values from their associated domains, while the values of the dependent variables are assigned values from the values of the independent variables through the relations of dependency. A solution is an assignment to the variables such that all dependencies and constraints are satisfied. The search for solutions is made by a backtracking method of the type "backjunping". The updating of values and the verification of constraints is organized by a directed acyclic graph. This graph is defined based on the dependencies between variables and of constraints in relation to the independent variables. Heuristics are used to establish the order in which variables are considered.

To avoid response delay and dead-ends associated to search-based methods, some recent works resorted to a two-stage process, by precompiling all the solutions using some form of efficient representation. Although these methods still have to solve a hard problem to find all the solutions, this is done offline and only once. Then, the interactive part of the configuration process can be done efficiently. For instance, Hadzic et al. [5] proposed a method to compile all the solutions of the problem using binary decision diagrams. Although they claim that the method has very good practical results, depending on the size of the configuration problem it may run out of space. A different pre-processing method is proposed by Freuder et al. in [6]. Unlike other conventional approaches that add constraints to the problem, thus making them susceptible to space limitation, they remove values from the domain of the variables to make their representation of the problem backtrack-free. The disadvantage of this method is that solutions are lost.

3 THE SOLAR POWERED PUMPING PRODUCT FAMILY

At the core of a solar powered pumping system (SPPS) product family, there is a water pump system and a photovoltaic (PV) array, which provides power to the pump. To improve the pump performance, a pump controller is used to condition the power and to control the pump. A float switch (S_T) is used to turn the pump off when the water tank is full, and another switch (S_W) is used to turn the pump off when the water level at the well is low, thus avoiding that it runs dry. The components of an SPPS are connected by wires to transmit power and control signals. The water is carried from the well to the tank through a piping system. A battery bank may be added to the system if the customer requires the system to have some autonomy, so that water may be pumped at night or during heavily clouded days. A charge controller is used to manage the charging of the battery bank.

Although a typical SPPS is composed of a few components, the product family may have a very large number of variants. For example, the water pump may have many options, each one operating optimally within a narrow window of water head and flux with a specified power, and the PV array can be configured in many ways, based on the choice of the PV model and the arrangement of the components.

Hence, configuring an SPPS to meet the customer requirements and optimizing its performance and cost is far from trivial, demanding a lot of expertise. This precludes most of the potential customers of participating interactively on the decision making along the configuration process, except for providing the application requirements at the beginning of the process.

4 THE PRODUCT FAMILY KNOWLEDGE FRAMEWORK

In the following subsections, we will present our knowledge framework for representing product families. In this approach we assume that the product family has already been developed. However, with this framework we will abstract all the relevant knowledge about the product family for deriving its members.

4.1 The generic product structure

The GPS is a modular architecture composed of component types, which stands for classes of components with the same functionality. In our approach, component types belong to four possible categories: common/generic, optional/generic, common/specific and optional/specific. Figure 1 illustrates schematically the concept of component types and their classification. A component type is specific if the corresponding class has only one component. However, if the corresponding class has two or more components, then the component type is generic.

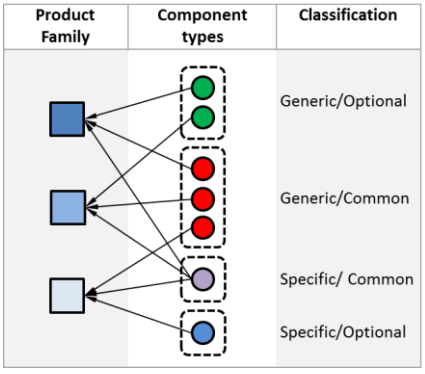


Figure 1. Classification of component types

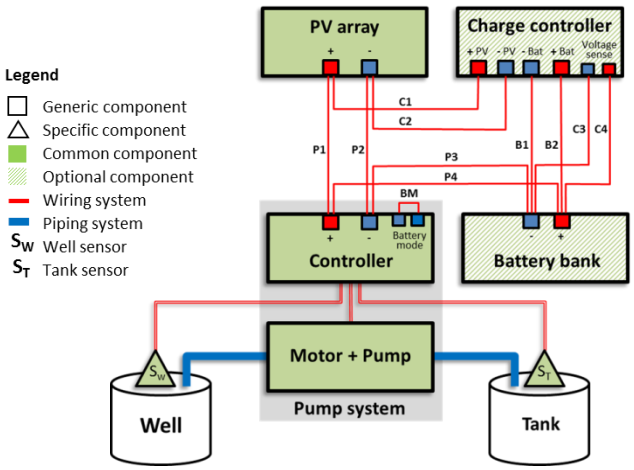


Figure 2..The GPS for the SPPS product family

If all members of the product family have a component in the corresponding class, the component type is common. Otherwise, if at least one member of the product family does not have a corresponding component in the class, it is optional. Note that the component types form a partition on the set of components that is used to derive all the members of the product family.

In Figure 2, it is shown the GPS for the SPPS product family. The PV array, Pump system, Sensors, Wiring and Piping systems are common component types, i.e., they are present in every member of the SPPS product family. However, the Battery bank and Charge controller are optional component types. The Well and Tank sensors are assumed to be specific component types, i.e., they do not vary among applications. All the other components are of the generic type, i.e., they can vary among applications and have two or more variants. It should be noted that, according to our classification, to be a common component type in the product family architecture does not imply that it is fixed. Actually, in our example, most of the product family variability happens on the common part of the GPS. Hence, although the optional components in a product family are one main source of variability, another important source of diversity can be the common part of the product family GPS. This is the case only if it is composed of generic components types.

Formally, we say that a GPS represents the architecture of a given product family if and only if the architecture of each member of that family is isomorphic to a substructure of the GPS and collectively the members of the product family are coherent to the classification of the component types on the GPS.

Hence, given a sample of SPPS, the GPS can be used to decide which of them belong to the product family. On the other hand, the GPS is not enough to determine which configuration of components can lead to a member of the product family, and let alone, which specific configuration will meet the requirements of a given application. To achieve this goal, we combine the GPS with the CN-F model.

4.2 The Constraint Network Extend with Design Functions

The CN-F model used in our approach can be regarded as an extension of the traditional CN model. It is defined by the tuple (V, C, F), where V is a set of variables, C is a set of constraints on subsets of V, and F is a set of design functions (which will be abbreviated as *d*-function), such that, every variable in V has at least one *d*-function attached to it that can generate its values. In what follows, we will define each of these elements and show how they apply to the SPPS product family in complement to the GPS.

Variables – Variations between the members of the product family are identified by variables in V. Consequently, these variables can be mapped on the GPS. Their scope of variation can vary widely, since they may be related from a specific feature to a whole component. For example, the configuration of the PV array is completely specified by three variables: *PV module model*, *PV modules in series* and *PV module strings in parallel*. The pump is associated only to the variable *Pump model*. The range of values that can be assigned to a variable is called its domain. For example, the domain for the variable *Pump model* is composed by the set of pumps {HR-03, HR-03H, HR-04, HR-04H, HR-07, HR-14, HR-20, C-SJ5-8, C-SJ8-7}.

Since all the variability of the product family is related to optional and generic components, only these types of components are associated with variables. These variables will be referred to as output variables because after their values are assigned, a product family member is specified. A special type of output variable is the inclusion variable associated to optional component types (e.g., *Battery inclusion*). These are binary variables that define if the component is included or not in the derived product.

However, variations can also be related to the application environment. For the SPPS example, the amount of *Daily water* needed, the *Well yield*, the *Tank capacity*, the *System autonomy*, etc., are variables that express the customer requirements and are referred to as input variables. Input and output variables are not necessarily disjoint subsets of V. Besides these two classes, the set V may contain auxiliary variables, which are neither input nor output variables. For example, the variable *Total dynamic head* is defined in terms of input variables, and although it is an essential variable for the choice of the pump system, it is not used to specify directly any of the components in the GPS. Therefore, it is classified as an auxiliary variable. In the SPPS example, we have identified 32 mixed discrete and continuous variables. In Figure 3, they appear as nodes of the constraint network, numbered from 1 to

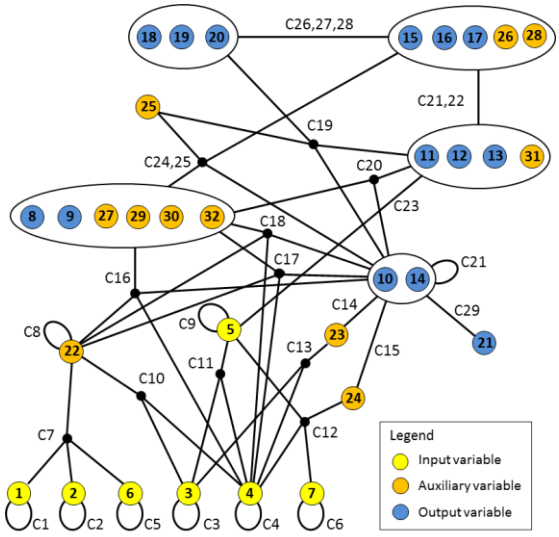


Figure 3. Constraint network for the SPPS product family

32. Some of these variables have been named explicitly within the text. As it will be discussed below, for convenience, variables can be grouped to form a composite variable. The encircled nodes in Figure 3 represent composite variables.

Constraints – Constraints define how subsets of variables in V are related to each other, thus restricting the possible combinations of values that can be assigned to them simultaneously. For example, the following sample of constraints describes how the auxiliary variable *Total dynamic head* is related to some variables in V:

C7: *Total dynamic head* (22) is equal to the sum of the *Water level* (1), *Water drawdown* (2), *Tank elevation* (6) and the friction loss of the piping system.
C8: *Total dynamic head* (22) must be less or equal than the head of the pump system (defined by the combination of the pump and its controller).
C18: If there is a *Battery inclusion* (10), the *Daily water* (4) requirement must be equal or less than 24 hours of pumping with the maximum available *Pump output flux* (32) at the required *Total dynamic head* (22).

Note that while the constraint C8 is defined over one variable, the other two relate four variables. Actually, in our approach, constraints can involve any subset of V. To satisfy a constraint, the values assigned to the variables in the expression defining it must render the expression true. However, if a constraint involves an inclusion variable and the corresponding optional component will not be included in the custom product, it can be disregarded.

Figure 3 depicts the complete constraint network for the SPPS product family. Note that, when nodes are the composition of variables, they may involve more than one constraint, each one relating a different subset of those variables.

Design Functions – The *d*-functions have been introduced as an extension to the CN model to capture the necessary knowledge to generate the values for the variables in V. Generically, *d*-functions will be represented by $f_x(y,z,\ldots,w)$, where x is the depended variable to which the *d*-function is attached and $y,z,\ldots,w$ are the independent variables from which the value for x is generated. As an example, Figure 4 shows the specification of *d*-function F4, which generates the values for *Total dynamic head* as a function of *Water drawdown*, *Water level* and *Tank elevation*.

As we shall see in more details below, an important consequence of *d*-functions is the dependency relation between variables that they establish. However, if the value generated by a *d*-function is to be consistent with the values of the variables it depends on, it must incorporate all the constraints involving these variables. We say that a *d*-function incorporates a constraint if and only if every combination of the values of the independent variables (for which the *d*-function is defined) and the value generated from them, satisfy that constraint. For example, from lines 1, 2, 3 and 4, it can be verified that constraints C7 and C8 are incorporated by F4.

F4 (*Water drawdown*, *Water level*, *Tank elevation*)
Begin
1. *Geometrical head* = *Water level* + *Water drawdown* + *Tank elevation*;
2. *Friction Loss* = 0.05 × *Geometrical head*; } <=> C7
3. *Total dynamic head* = *Geometrical head* + *Friction Loss*;
4. If *Total dynamic head* ≤ maximum head of the available pump systems } => C8
5. Then return *Total dynamic head*
6. Else notify "The configuration process cannot proceed because there is no
7. available pump system that can overcome the total dynamic head.";
8. Abort configuration;
End

Figure 4. The *d*-function F4 attached to *Total dynamic head*

In general, not all the variables related (by constraints) to the variable which a *d*-function is attached to will be involved in the dependency. For example, the variables *Daily water*, *Battery inclusion* and *Pump output flux* are related to *Total dynamic head* by the constraint C18 but are not required for the generation of its values. Consequently, C18 is not incorporated by F4. If a *d*-function does not incorporate a constraint involving the variable to which it is attached, we say that the constraint is free regarding that *d*-function. However, a free constraint may be incorporated by another *d*-function attached to the same variable or to a related variable.

Input variables are attached with special *d*-functions that request the user to assign a value chosen from a delimited range of values, which may be generated dynamically as a function of values assigned to other variables. Hence, except possibly for the input variables, all variables in V will necessarily depend on some other variable due to the *d*-function attached to them, forming a network of dependencies on V, as discussed in more detail below.

The *d*-function F4 specified in Figure 4 is relatively simple. The CN-F model for the SPPS also contains much more complex ones. For example, to define the values of the variables that specify the component type PV array (related above), the *d*-function F16 finds the best module arrangement to cope with the power requirements of the SPPS without violating the voltage and current restrictions imposed by the pump or battery controller. As another example, the *d*-function F12 selects the pump system from a performance table which correlates the total dynamic head, the output flux and the input power for the optimal performance of the pump systems.

If a set of variables is strongly coupled, i.e., the value of any one variable cannot be assigned independently of the others, as in the two cases just discussed, they are be grouped together to form a composite variable and the same *d*-function will generate the values for all of them. Otherwise, attaching a single *d*-function to each of those variables would form dependency loops between them, a condition that is undesirable in our approach.

Since only values generated by the *d*-functions are taken into account in the configuration process, in our approach the domain of a variable in V can be defined as the set of all values that can be generated by the *d*-functions attached to it. An important consequence of this definition is that the domains need not to be defined explicitly. Moreover, they can be either discrete or continuous without distinction.

Before introducing the instantiation process for the CN-F model, we note that the dependency between variables in V induces a dependency between *d*-functions in F. For example, the *d*-function F4 attached to *Total dynamic head* depends on the *d*-functions that generate the values to variables *Water drawdown* and *Water level*, *Tank elevation*.

5 DERIVING PRODUCT FAMILY MEMBERS

Members of the product family are derived from the knowledge framework. This process is divided into two stages. First, a solution to the CN-F model is found from the values of the input variables.

Second, this solution is used to transform the GPS into a specific model representing the desired product family member.

5.1 Finding solutions to the CN-F model

An assignment of values to all the variables in V such that no constraint in C is violated is said to be a solution to the CN-F model. The set of all solutions will be denoted by S. As we will argue below, solutions in S correspond to members of the product family.

The instantiation process begins with the assignment of values to the input variables and proceeds towards the output variables, through the auxiliary variables. This process is guided by the dependencies established over V by the d-functions. In Figure 5, we present an instantiation algorithm to carry out this process. In that algorithm, a d-function is enabled if all the variables it depends on have been assigned their values. The set G represents the variables for which the values have already been generated and $L(f)$ represents the set of free variables in relation to f. For this algorithm to work properly, it is necessary to rule out loops between d-functions. Thus, we assume that $F = \{f_1, f_2, \dots, f_k\}$ can be order by the dependency relation induce over F, that is to say, for $i = 1,2, \dots, k$, the element f_i is an input d-function or all d-functions it depends on precedes it in that order.

Every time a d-function $f_x(y, z, \dots, w)$ from F is executed (line 2 of the instantiation algorithm), a value is assigned to variable x from the values of the variables $y, z, \dots, w$. If we represent this dependency by a directed graph, with arrows from the independent variables toward the dependent one, the execution of the instantiation algorithm can be represented by a dependency graph as the one shown in Figure 6. The nodes represent variables (single or composite), the same ones shown on the constraint network in Figure 3. Near to each node, it is indicated the d-function that was used to set its dependency (the incoming arrows). The dependency graph can be organized into dependency levels. At level 0 are the input variables whose values have been assigned by the customer and that do not depend on other variables. In general, a variable is localized at level n if it depends on at least one variable at level $n - 1$. Note that the input variables *Daily water* and *System autonomy*, represented by nodes 4 and 5, appear at levels 2 and 3, respectively. Although it is the customer who assigns their values,

The instantiation algorithm:
Begin
1. Following the order in F, remove the first f which is enabled
2. If f successfully assigns a value to x
3. Then add x to G, remove from F all alternative functions attached to x
4. and make $L = L \cup L(f)$
5. Else If there is another generative function attached to x
6. Then go to step 1
7. Else go to End and return "Failure" ⇒ inconsistency of type I
8. For every $R_X \in L$
9. If $X \subseteq G$
10. Then If R_X is satisfied
11. Then remove it from L
12. Else go to End and return "Failure" ⇒ inconsistency of type II
13. If $F \neq \emptyset$
14. Then go to step 1
15. Else go to End and return "Success"

End

Figure 5. The instantiation algorithm to find solutions to the CN-F model

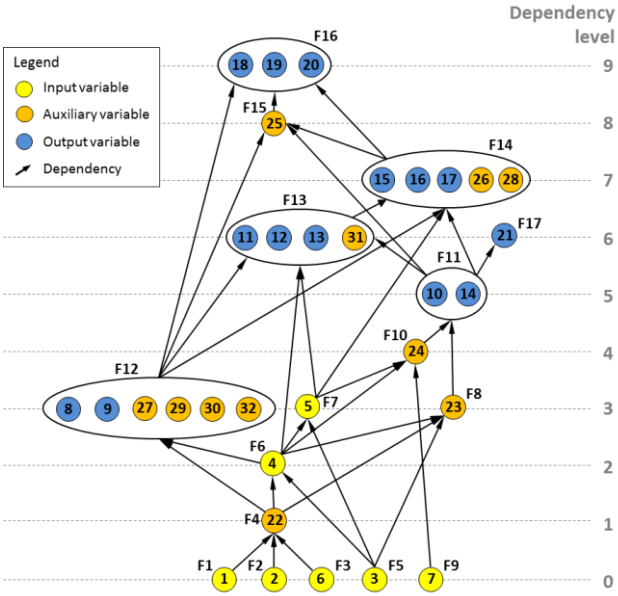


Figure 6. The instantiation graph for the SPPS product family

they also depend on other variables for checking the consistency of the values assigned by the customer.

Now, every instantiation graph can be associated to a subset of F, composed of exactly those d-functions used to generate it. Since the same set of d-functions can be elicited for a variety of inputs, we will call this set an instantiation pattern, represented by P. More specifically, every subset $P \subseteq F$ satisfying the ordering condition and such that, for every $x \in V$, there is only one $f_x \in P$ is an instantiation pattern. If a variable in V is attached with more than one d-function, the CN-F model will be associated to more than one instantiation pattern. However, in general, one should not expect many instantiation patterns. In the modelling of the SPPS example, there is only one instantiation pattern composed of 17 d-functions, number from F1 to F17 in Figure 6.

As indicated in Figure 5, there are only two points during the execution of the instantiation algorithm where it can terminate without finding a solution. Each one is associated to a different type of inconsistency. Type I arises when the d-functions attached to a variable cannot generate its value. The inconsistency of type II, arises if there is a free constraint in $L(f)$ that is violated by the values assigned to the variables in G. If the values assigned to the input variables are not part of a solution in S, then there is some inconsistency embedded in the input and the algorithm will fail.

As it is well known, local consistency in a CN model does not guarantee global consistency [11]. Therefore, although the values generated by the d-functions are locally consistent, the instantiation pattern does not guarantee that an input without an embedded inconsistency will lead to a solution. Thus, in what follows we will introduce two consistency conditions to the CN-F model such that our instantiation algorithm will always be able to find a solution.

Consistency condition 1 – For every $x \in V$, there is at least one $f_x \in F$ which is defined for every instantiation of the variables it depends on.

Consistency condition 2 – Let $P \subseteq F$ be an instantiation pattern. Every constraint in R is incorporated by some d-function belonging to P.

It can be proved that, if the CN-F model satisfies the Consistency conditions 1, no inconsistency of type I will arise during the execution of the instantiation algorithm, and if all its instantiation patterns satisfy the Consistency condition 2, no inconsistencies of type II will arise. However, if the CN-F model satisfies the two conditions, lines 4-12 of the algorithm in Figure 5 can be eliminated, since the inconsistency testing is no longer required. Therefore, the resulting instantiation algorithm becomes extremely simple.

The CN-F model for the SPPS satisfies the second condition state above; however, it fails the first one. The problem is with the *d*-function attached to *Total dynamic head* shown in Figure 4. According to its specification, only after all three inputs variables it depends on have being assigned their values is that the *Total dynamic head* is calculated and the result is compared to the head of the available pumps. If the condition on line 4 is not satisfies, there is no solution to the application and the configuration has to be aborted. To satisfy the Consistency condition 1, an alternative approach is to restrict the range of values for the input variable *Water level* dynamically, so that the resulting total dynamic head of the application is always within the range of the available pump systems. Nevertheless, this restriction is equivalent to the abort condition in a disguised form. On the other hand, because the decision to abort is taken at the very start of the configuration process, and we can give explanations for why the configuration process cannot proceed, this modelling approach was preferred. However, to cope with this abort condition, it was necessary to add a control mechanism in the implementation of the instantiation algorithm, not present in its description in Figure 5. Note that the risk of having to abort the configuration is reduced as the maximum head of the available pumps is increased.

5.2 Transforming the GPS into physical models

Once the solution to the CN-F model has been found, all the output variables on the GPS have their values assigned, and its transformation into a specific physical model can start. This process is carried out in two steps. First, it is necessary to remove the optional components types from the GPS that are not required in view of the customer requirements. For example, if the customer does not require any system autonomy, there is no need for batteries in the SPPS. To determine if an optional component type have to be removed we refer to the value of the associated inclusion variables. In our example, if the value is 0, the component is removed. Otherwise, if it is 1 the component is kept in the structure. After the GPS has been stripped of the unnecessary components, the second step of the transformation process is carried out with the substitution of the generic components by specific ones from their correspondent class of components. The definition of which component will be selected is made based on the values of the output variables on the generic component type. For example, besides the inclusion variable, the Charge controller is associated to three other variables. One of these variables specifies the model of the charger, and the other two the configuration of two switches to set the output voltage of the charger. After all the generic component types have been substituted by specific ones, a physical model of the custom SPPS will emerge from the GSP.

Based on the transformation process described above, every solution in S leads to a specific physical model. Obviously, the resulting physical model is isomorphic to the GPS of the product family and is coherent to the component types by construction. Now, if every relevant design constraint has been elicited and introduced in the CN-F model, we can conclude that every solution in S corresponds to a member of the product family.

6 IMPLEMENTATION OF THE CONFIGURATOR

The SPPS configurator has been conceived as a tool to support the sales force of a company that provides water pumping solutions to the rural area. The configurator requires the sales force to have only enough technical knowledge about SPPS to make some assessments at the customer site to input the customer requirements. This process is interactive with the configurator requesting specific information. To avoid inconsistencies embedded in the input, the configurator makes a few checks, suggesting appropriate corrections if necessary. But in case no solution can be provided to the customer, the configurator notifies the impossibility as early as possible.

In Figure 7, it is shown the implementation of SPPS configurator using LabVIEW. At the centre, it can be seen the *d*-functions (numbered F1 to F17), each one representing a subVI (a kind of routine in LabVIEW), with the variables to which they are attached at the right of the diagram. The variables to which the *d*-functions depend on are indicated by the lines coming from below. Thus, this diagram arrangement clearly reveals the dependency between the *d*-functions. At the left of the diagram, it can be seen the control structure which operates in conjunction with the loop structure (the outer structure encompassing the whole program). Initially, only the first four *d*-functions will be executed. If the abort condition in the *d*-function F4 (specified in Figure 4) is true there is no solution for the configuration problem and the program ends. Otherwise, the abort variable is set to false and the other *d*-functions are executed. As the *d*-functions are executed, the values for the correspondent variables are generated, and they are set to inactive. The *d*-functions attached to variables (other than the inclusion variable) on optional component types, which will not be included in the custom product, can be set to inactive without generating values. When no abortion happens and all the functions are inactive (which is equivalent to $F = \emptyset$ in the control algorithm in Figure 5), a solution has been found and the program ends. This happens in exactly three iterations of this configurator program.

It is interesting to note that, if the CN-F model satisfies the two consistency conditions, the configurator can be implemented a data flow program by the concatenation of *d*-functions. Moreover, if it were not for the abort condition, the iteration structure in Figure 7 could have been dismissed.

7 CONCLUSIONS

In this paper, we have proposed a new approach to the customisation of product families. It is based on a knowledge framework which combines a GPS and a CN-F model to represent product families. Members of the product family are derived from this knowledge framework by a two-stage process. First, a solution to the CN-F model is found from the customer requirements through an instantiation process. Then, in the second stage, the solution is used to transform the product family GPS into a specific model which represents the desired product family member.

A number of contributions to the area of product configuration are introduced by this approach. It is provided a formal definition for the product family GPS and an extension to the classical CN model by attaching *d*-functions to the variables to generate their values. Since the domains of the variables are defined through the *d*-functions, their values need not to be predefined explicitly. As a consequence, we can deal with mixed discrete and continuous variables.

Moreover, the *d*-functions provide a method to establish the dependency between variables as part of the modelling of the customisation process. Dependency patterns can reduce the design space for finding solution considerably. However, despite their local consistent, they do not avoid backtracking. To achieve this goal we have set up a few conditions for the CN-F model, such that, if satisfied, deriving product family members becomes a backtrack-free process. The remarkable aspect about this

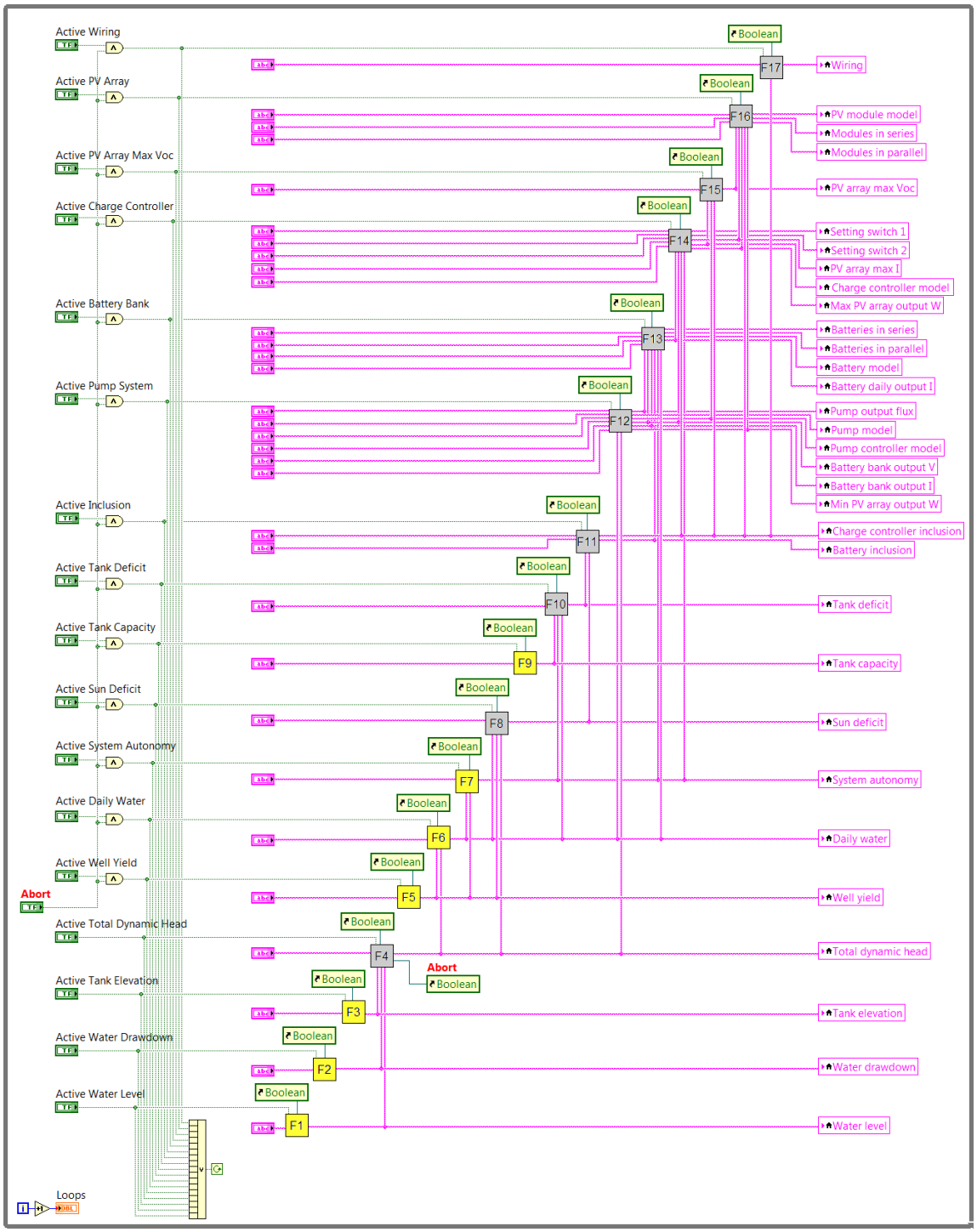


Figure 7. A view of the SPPS configurator program implemented in LabVIEW

achievement is that it does not depend on pre-processing, but can be obtained by the systematization of the knowledge about product families.

It is also interesting to note that through the *d*-functions it may be possible to design components during the customization process, thus providing great flexibility to the customization process. However, this is a capability which requires further investigation, because making changes to components without the proper delimitation of the design space can compromise the manufacturability or performance of the product being derived.

Our approach is suited for the configuration of complex product families for which the customers do not have the necessary expertise to participate directly during all the configuration process. It can deal with configuration problems for which the constraints between the variables are highly complex, since they are incorporated by the *d*-functions and dealt with in the form of procedures. The complexity of the configurator is not particularly affected by the number of variables, since this amounts to adding new *d*-functions. In case some of the variables are attached with more than one *d*-function, this will generate multiple instantiation patterns. However, the proposed instantiation algorithm is enough to deal with this condition, since at every moment only one instantiation pattern is being followed. As for the verification of the compliance to the consistency conditions, this is largely an analysis of the *d*-function individually. (The same is true for maintenance, because *d*-functions are high modular.) Now, if the CN-F model satisfies our assumption on the ordering of the set of *d*-function and the two consistency conditions, the configurators can be implemented in the form of dataflow programs by the concatenation of the *d*-functions.

Despite the advantages related above, to exploit all the potential of our approach in practical applications, there are a number of issues that must be further developed. For example, concerning the integration of our approach into a mass customisation system, it will be necessary to have a more elaborate representation of the GPS to support the generation of customer quotations and production orders [13]. However, at least for a mass customization systems based on 3D printing, we have shown that our approach can be integrated with CAD tools, and that the generation of 3D models for the custom products can be made automatically, in a seamless way [14].

ACKNOWLEDGEMENTS

The author wish to gratefully acknowledge the financial support of FINEP for the realization of this work.

REFERENCES

[1] S. Mittal and F. Frayman, "Towards a Generic Model of Configuration Tasks," in *Proceedings of the 11th International Joint Conference of Artificial Intelligence*, San Francisco: Morgan Kaufman, 1989, pp.1395–1401.

[2] M. Veron, H. Fargier and M. Aldanondo, "From CSP to Configuration Problems," in *AAAI-99 Workshop on Configuration*, Orlando, Florida, July 18–19, 1999.

[3] B. Wielinga and G. Schreiber, "Configuration design problem solving," *IEEE Expert*, vol. 12, no. 2, pp. 49–56, 1997.

[4] T. W. Simpson, B. Aaron, L. A. Slingerland, S. Brennan, D. Logan and K. Reichard, "From user requirements to commonality specifications: an integrated approach to product family design," *Research in Engineering Design*, vol. 23, no. 2, pp. 141–153, 2012.

[5] T. Hadzic, S. Subbarayan, R. M. Jensen, H. R. Andersen, H. Hulgaard and J. Moller, "Fast backtrack-free product configuration using a precompiled solution space representation," in *International Conference on Economic, Technical and Organizational aspects of Product Configuration Systems*, Technical University of Denmark, Lyngby, Denmark, June 28–29, 2004.

[6] E. C. Freuder, T. Carchrae and J. C. Beck, "Satisfaction Guaranteed," in *Workshop on Configuration, Eighteenth International Joint Conference on Artificial Intelligence*, 2003.

[7] Mittal, S. and Falkenhainer, B., "Dynamic Constraint Satisfaction Problems," in *Proceedings of the 8th National Conference on Artificial Intelligence*, 1990, pp. 25-32.

[8] D. Sabin and F. Freuder, "Configuration as Composite Constraint Satisfaction," in *Technical Report FS-96-03, Workshop on Configuration*, Menlo Park: AAAI Press, 1996, pp. 28–36.

[9] E. Gelle, B. V. Faltings, D. E. Clement, and I. F. C. Smith, "Constraint Satisfaction Methods for Applications in Engineering," *Engineering with Computers*, vol. 16, no. 2, pp. 81–85, 2000.

[10] H. Xie, P. Henderson, J. Neelankavil and J. Li, "A Systematic search strategy for product Configuration," in *17th International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems Manufacturing* (IEA), Ottawa, Ontario, January 1, 2004.

[11] R. Dechter, "Constraint Networks," in *Encyclopedia of Artificial Intelligence*, S. C. Shapiro, Ed. New York, Wiley, pp. 276–285, 1992.

[12] C. Forza and F. Salvador, "Managing for variety in the order acquisition and fulfillment process: The contribution of product configuration systems," *International Journal of Production Economics*, vol. 76, pp. 87–98, 2002.

[13] A. Haug, L. Hvam and N. H. Mortensen, "A layout technique for class diagrams to be used in product configuration projects," *Computers in Industry*, vol. 61, pp. 409–418, 2010.

[14] H. M. Schneider, D. T. Kemmoku, P. Y. Moritomi, J. V. L. da Silva, Y. Iano, "Matching the Capabilities of Additive Technologies with a Flexible and Backtrack-free Product Family Customisation Process," in *Proceedings of the Fraunhofer Direct Digital Manufacturing Conference 2014*, Berlin, Germany, March 12-13, 2014.

Optimization based framework for transforming automotive configurations for production planning

Tilak Raj Singh [1] and **Narayan Rangaraj** [2]

Abstract. A product (e.g. automobile, computer) can be configured using different combinations of its available attributes (features). However, selection of attributes may not be independent of the selection of other attributes. In practice, each attribute implies a selection rule (dependency) for other sets of attributes in order to generate a valid configuration. Due to dynamic changes in the product design, miniaturization, legislation etc., product attributes and their selection rules get changed. This implies that variants produced in the past may not be valid for future product design. Nevertheless, customer history contains important information related to customer buying behaviour which is an essential input for future planning activities. In order to achieve efficient adaption of past customer orders to a changed product design, we propose a fully automated optimization based framework. The methodology is demonstrated using an industry size example.

1 Introduction

Manufacturing companies are currently focusing on mass customization. In this environment customer mix and match different available product attributes to get desired configurations. Selection of any attribute implies certain conditions on other set of attributes. For example, if driving assistance system is selected in a car then the customer may only be able to select steering types which have required control options. These engineering dependencies are available in the product's technical documentation (e.g. Bill-Of-Material) and each valid configuration must satisfy these restrictions in order to be producible [13]. Manufacturers enable their customers to select and order constructible product variants by offering sales manuals and web-based product configurators. The product configurator guarantees that attributes selected by the customer must satisfy all dependency rules at the time configuration is created. If any combination of attributes violates the product configuration rules (constraints), then, this will be an invalid configuration and cannot be produced [5]. We will use the term configurations rule (or rules in short) in the meaning of all restrictions imposed on configuration problem and by fulfilling all rules configuration will be considered feasible for production.

In order to provide short lead time for complex engineering products (e.g. Automobiles, Computers) often hybrid manufacturing philosophies like assemble-to-order is used. The production is setup based on forecast demand and final assembly is done for the real customer orders. The effectiveness of this method depends upon the quality of the forecasted demand. Most manufacturers use data about product variants produced in the past to get suitable estimates for the future customer demand [10]. Continuous changes in product design and market conditions imply that product variants which have been produced in the past may not be valid according to the changed product. However, changes are incremental in nature which means that past variants can be upgraded (by dropping and/or adding some features) to new changed model once the required changes are incorporated [4]. In this paper our aim is to develop methods to upgrade base configuration (configuration produced in the past) in such a way that 1) new configuration satisfies required product configuration rules 2) new configuration should be as similar as possible to the base configuration. The similarity measure can be monitored by using some distance (e.g. Hamming distance) or cost function.

In contrast to the above problem, another requirement to transform existing configuration to the new configuration arises from the *Reconfiguration* problem [11]. In this case the previously selected base configuration is still valid with respect to configuration model however the customer may want to make some explicit changes with respect to the earlier choice - for example, adding or dropping some of product's features. Most of these reconfiguration problems are motivated by the customer's request to change the previously selected variants. This is not an uncommon situation in premium customizable products. However, the reconfiguration problem can also be driven from the manufacturer point of view. For example due to capacity limitations, production of customer orders may be shifted from one country/plant to another. Then the production feasibility need to be checked as configuration rules may vary between production plants and counties.

In this paper we propose an integrated solution framework where: 1) user can update any given configuration by changing configuration variables (adding or removing product attributes) 2) Feasibility

[1] IT-Production Tools, Mercedes-Benz R & D India, Bangalore, Email: tilak.singh@daimler.com

[2] Indian Institute of Technology, Bombay, Powai Mumbai, India, email: narayan.rangaraj@iitb.ac.in

of desired configuration can be checked at any point of time 3) In case of conflicts with underlying configuration rules, the solution is computed through solving an optimization model which ensures that the modification to the base configuration is done with minimal change cost. In section 2 and section 3 we will discuss characteristics of the problem and the available data. Section 4 will focus on the development of an optimization based configuration transformation model. In section 5 the solution procedure will be discussed with initial computational results.

2 The planning problem

A product can be configured using different combinations of its attributes (features). In case of automobiles, attributes could be: body style, transmission type, sunroof, parking assistance etc. If we describe a product as an exhaustive list of attributes then the product configuration can be expressed as a 0-1 vector over the attribute set, where 0 (zero) represents the absence of any attribute and 1 (one) represents its presence in the configuration. A feasible configuration can be achieve by satisfying predefined set of rules (Boolean formulas) monitoring interdependencies among attributes.

Let us define our product configuration problem as per [7, Definition 1]: the configuration problem C can be expressed through a triple (X, D, F), where:

- X is a set of product attributes (configuration variables) lets say $\{1, ...n\}$. Where n is the total number of attributes.
- D is the set of attributes finite domains $d_1, d_2, ..., d_n$.
- $F = \{f_1, f_2, ..., f_m\}$ is set of propositional formulas (rules or restrictions) over attribute set X.

In this paper the configuration variables X are boolean, hence domain $d_i \in \{0, 1\}, \forall i \in X$. A configuration is said to be feasible if an assignment for all attributes ($i \in X$) is found which fulfils each and every propositions in F. For configuration problem C a solution space $S(C)$ can be built by finding all assignment of configuration variable X which satisfy rules F. The problem we have in hand, the size of solution space $S(C)$ could be in the ranges of thousands of billion [6].

For a customizable product which changes with respect to time (due to introduction of new attributes, discontinuation of existing attributes or change in attributes dependencies) the configuration problem at any given time t can be expressed as $C_t = (X_t, D_t, F_t)$, where X_t,D_t and F_t are configuration variable, its domain and underlying propositional formulas respectively at time t. In this paper the domain D_t is fixed (boolean for all variables/attributes) so changes in configuration problem are possible by changing configuration variables X, changing rule set F or both.

In the scenario shown in Figure 1, let us assume that at time t the manufacturer wants to make some planning estimate for time $t + T$ (mid to long term planning, typically T = 6 months - 3 years) to support various planning activities such as production planning,

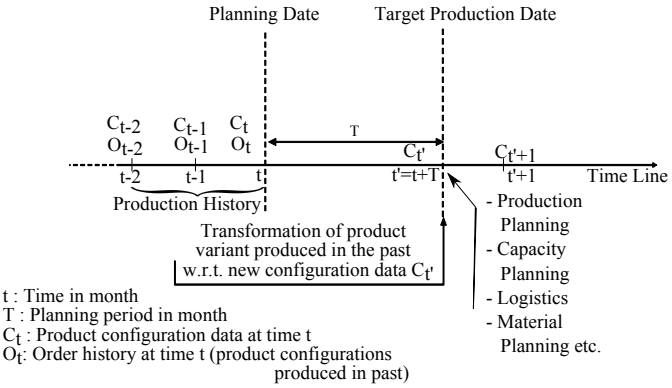


Figure 1: Product variant produced in the past need to be transformed w.r.t. new product design for use in future planning

capacity planning, material requirement, supplier selection. At time t the manufacturer has information about its current and past product configurations data ($C_t, C_{t-1}, ...$, capturing list of attributes/features (X) and its dependencies/rules (F)) and order history O_t, which is an 0-1 assignment of attributes. At any time t the validity of the product configurations will be checked according to rules written in $C_t = \{X_t, F_t\}$. As the product changes with respect to time, for every time instance we will have a corresponding product configuration problem instance. In practice, process of engineering change starts much before (typically 5-7 years) the start of production. This gives possibility to know the product configuration data for future time i.e. $C_{t+T} = \{X_{t+T}, F_{t+T}\}$ at given time t.

Now, for given set of configurations (O_t, will also be called base configuration) which are derived from configuration model $C_t = (X_t, F_t)$ we are required to validate their feasibility with respect to $C_{t+T} = (X_{t+T}, F_{t+T})$. In case of infeasible configurations we are required to find the new configuration in the solution space of $S(C_{t+T})$ with the minimal change to its base configuration. As the configurations variables are Boolean in nature, change in the configuration can be performed either by adding new attributes, or removing old attributes. The distance between two configurations (base and transformed) can be expressed through sum of the changes in the attribute assignment, which can be expressed through the Hamming distance. However, changing any arbitrary attribute in the base configuration in order to make them feasible may not be practically desired. For example, some of the product attributes may have high cost of change such as engine, special body style or sophisticated optional equipment, and changing these attributes may be difficult to handle as compared to changes in some simple options such as cup holder or some alarm features. Thus, a change cost can be associated with each attribute and transformation of base configuration to new configuration can be sought to be achieved by minimizing the total change cost. Change cost will only be associated to configuration if certain attribute is either added or removed in the configuration. One may consider two different quantities of change cost for an attribute such as attribute addition cost and attribute removal cost.

In case of *Reconfiguration* problem, some attributes are fixed by customer (attributes on which modification is asked) or may have very high change cost as they may be customer's most preferred attributes. Then the solution is sought only by changing the remaining set of attributes. The reconfiguration problem can be defined as a special case of configuration problem where certain configuration variables are set to predefined values (true or false). The aim is to fix certain attributes in base configuration (either by replacing some previously selected attribute or adding new) and then look for a new configuration which has minimal changes with respect to the base configuration.

In our case, the changes in the configuration can only be made either by adding new attributes or removing previously selected attributes from the configuration. As configuration changes are associated with change in attributes thus a change cost can be associated with each attribute to measure the impact of change.

In our work we propose an optimization model for transforming invalid configurations to valid ones as well as transforming configurations with predefined settings over attributes (Reconfiguration). We develop a framework which can incorporate information from different data sources such as configuration rules, sales program (cost associated with attributes) and planning expert's knowledge (to change configuration in some guided way). As most of the information is available or can be converted in the form of logical propositional formulas, we develop an optimization based framework after a required transformation of the logical propositions. In the next section we discuss various input data for the planning problem.

3 Input Data and its characteristics

3.1 The configuration data

A variant rich customizable product can be defined on the basis of attributes (features) in order to facilitate aggregate level of planning for components and modules [14]. Customer configurations can be created by combining different attributes that are permitted by the corresponding configuration data. It is important that while combining different attributes, we must fulfil the interdependencies between attributes, so that a feasible product configuration can be generated [13]. For instance, if in the USA some engines require special transmission types, this condition must hold while configuring a car of that type. A product document captures the technical, market and legal restrictions and provides an important data source for the configuration feasibility check.

Interdependencies among attributes are documented and maintained in the configuration data by a rule system. These rules are basically Boolean expressions imposed against each attribute. Selection of attributes in a configuration is done through evaluating the respective Boolean expression. Table 1 shows an example of such a data.

A customer configuration consists of a list of attributes. Each attribute is represented as a Boolean variable in the configuration data.

Attribute	Name	Rule	Description
1	Automatic climate control	$(2) \wedge (3 \vee 4)$	attribute 1 only when attribute 2 is present and either attribute 3 or 4 is present
2	Air condition	TRUE	must be present in every variant
3	Comfort package	$\neg(4)$	attribute 3 is not with attribute 4
4	Performance package	$\neg(3)$	attribute 4 is not with attribute 3

Table 1: Example: Rule based configuration data

The value of the attribute will be set to TRUE, if particular attribute is selected by the customer. The selection of the attribute is controlled by the logical rule system as shown in rule column of table 1. The logical rule system is built from usual Boolean operators $\vee$(OR), $\wedge$(AND), $\neg$(NOT) and an attribute serving as a proportional variable. The customer order processing is controlled by evaluating the rule's formulae under the variable assignment induced by the customer order and executing suitable actions based on whether the formula evaluates to TRUE or FALSE.

As discussed in section 2 configuration problem (C) can be defined by triple (X, D, F). For configuration data shown in table 1. $X = \{1, 2, 3, 4\}$,$D \in \{0, 1\} \forall X$, and $F = \{f_1, f_2, f_3, f_4\}$ where

$$f_1 = \{1 \rightarrow (2) \wedge (3 \vee 4)\}$$
$$f_2 = \{2\}$$
$$f_3 = \{3 \rightarrow \neg(4)\}$$
$$f_4 = \{4 \rightarrow \neg(3)\}$$

where $a \rightarrow b$ means attribute a implies attribute b, if a is selected (or set to true) then b has to be selected in the configuration. Propositional formulas in F can also be expressed as $F = \{((2) \wedge (3 \vee 4)) \vee \neg(1), 2, \neg(4) \vee \neg(3), \neg(3) \vee \neg(4)\}$. In the given example, associated rule with f_3 and f_4 have the same boolean expression so only one can be evaluated and also $f_2 = \{2\}$ says that attribute 2 will be the part of every configuration. As all rules written in the configuration rule set F has to be satisfied. All element of F can be combined with AND operator, $\varphi = \wedge_{f \in F} f$. Thus φ will be the boolean formula whose Truth value will represent an configuration. φ is also called as *product overview formula* [9]. Our configuration variable set X contains all possible attributes which can be the part of the product configuration either from customer point of view of manufacturer. For example, some plant and production related attribute may not be relevant to the customer but is required to handle feasibility of production at certain planning stage. In the next section we discuss different changes in the configuration data which may result in modification or upgradation of configurations.

3.2 Changes in the configurations

As a customizable product can be defined based on different features offered by the manufacturer, product changes can be studied based on

the change in the offered product attributes. In this section we will outline various changes in product attributes which can make certain product variants to invalid. The changes in the product attributes can be cause by one or more of reasons described below:

1. Deletion of old attributes: All past configurations containing attributes which are discontinued will become invalid according to changed product. If discontinued attributes have no dependencies with remaining attributes we can simply remove these attributes to restore the validity (feasibility) of the product variant. For a complex engineering product this is very unlikely. In general, product attributes have complex dependencies among each other and modification of one attribute needs to be validated with the remaining set of attributes.
2. Change in rule: The technical rules pertaining to an attribute that are expressed in configuration data may get changed due to various reasons such as design modification, legal changes. For some practical product instances, a single attribute may depend on hundreds of other attributes by a complex Boolean expression. Change in some part of a rule may affect feasibility of certain attribute combinations.
3. Inclusion of new attributes: As a product evolves, some new features get added. These may not have been present in the past, but a customer may select them in the future. As newly introduced features may have some dependencies with other available attributes, variants produced in the past have to be modified in such a way, that transformed configurations also contain new features (according the estimate of new feature).
4. Attribute fragmentation/atomization: In some cases an attribute is split into more attributes. For example, let us assume that a car was produced with the option off-road package which includes features as high battery capacity, heavy duty suspension, hi-fi music system and a sunroof. Customers were not allowed to select above features individually but selection can be made through package. Now, due to some change, the manufacturer has decided to divide the off-road package into two new packages. The first package includes the features high battery capacity and heavy duty suspension, the second package includes the hi-fi music system and sunroof. Both new packages can be selected individually, which means that the customer has more choice than before which may effect the distribution of packages from the past. Some input form sales in-terms of demand estimates of new package may help here to adapt past configurations according to new product offerings.
5. Replacement of attributes: Most often due to technology and other changes, some old attributes are replaced by new attributes. For example, some old telematic features are replaced by the new generation touchscreen based systems. Therefore historic product variants should also be upgraded to the new generation to use them for planning of a future production system.

Apart from the above changes there also exists some desire to change attributes of a past product variant according to new product offerings. For example, due to market changes, the demand for a certain engine type may decrease in comparison to other available engines. In this case the changes in the engine distribution across all transformed historic orders have to be considered in the transformation process. This information is not documented in the configuration rules but can be accessible through planning experts or through some sales forecast. During the development of the automatic configuration transformation system we try to accommodate these kinds of requests.

3.3 Customer history

Let a product be defined by a set of 4 attributes $\{1, 2, 3, 4\}$. According to Table 1, the configuration can be listed as described in Table 2 and 3. As shown in Table 3, customer configuration can be presented as a 0/1 vector over attributes, any change in the configuration can be made by changing attributes from 0 to 1 and vice-versa. While transforming the configuration, one objective will be to be as close as possible to the old configuration. This can be done by minimizing the Hamming distance between the old and new configurations.

No.	Configurations
i	1,2,3
ii	1,2,4
iii	2,3
iv	2,4

Table 2: Configuration based on attributes set

No.	1	2	3	4
i	1	1	1	0
ii	1	1	0	1
iii	0	1	1	0
iv	0	1	0	1

Table 3: Configuration as 0/1 matrix over attributes

4 Formulation of the optimization model

During the transformation of product configurations, we need to evaluate each rule written in the corresponding configuration data. At the same time, we also need to ensure, that the changes in the given product variant are done with minimal cost. Cost can vary based on deviation from the base configuration and the type of changes done. In this section we explore an optimization based framework to find a solution for the above problem. To create an optimization based transformation procedure, all information included in the product configuration process need to be considered in the model. To do this, in the following section we first transform rules from the configuration data to the corresponding 0-1 discrete programming equivalent forms.

4.1 Transformation of logical rules to linear inequalities

Constraint programming approach is a well-used methodology inside the many product configuration systems [2]. Restrictions on product configurations are modelled as constraints and a solution is a total assignment satisfying each of the constraints. Most of the proposed framework rely of the transformation of boolean formulas to special structure such as conjunctive normal form (CNF) before writing the

final constraint set [3]. We developed an alternate method to avoid the initial conversion of the input to CNF. Our formulas are so large that naive CNF conversion by applying the distributive law failed for lack of memory and time. Also, CNF conversion steps involves introduction of large number of new variables which increases the complexity of the problem.

4.1.1 Data structure for configuration rules

Using the normal precedence operators and the conventional evaluation of expressions, the logical rule from configuration data (F) can be presented in form of a tree structure. For example, let's say selection of an attribute 1 is controlled by following Boolean expression:

$$f_1 = (2) \wedge (3 \vee 4) \tag{1}$$

The tree representation of above expression can be shown as Figure 2. We used *Stack* for storing binary tree for implementation of algorithm for transforming logical rules to algebraic inequalities [12].

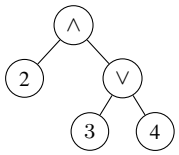


Figure 2: Representation of attribute selection rule in a binary tree

index	Elements
0	∧
1	2
2	∨
3	3
4	4

Figure 3: Rule in a stack

4.1.2 Transforming propositional formula's to 0-1 LP

In this section we describe the transformation of logical propositions to its equivalent linear integer constraint through an example. The procedure to obtained required transformation is discussed in [12] and [1]. Linear inequalities over Boolean variables are a widely used modelling technique. The main task during transformation of an attribute selection rule into a system of linear constraints is to maintain the logical equivalence of the transformed expressions. The resulting system of constraints must have the same truth table as the original statement. For every attribute we introduce a binary decision variable, denoted by x_i. The connection of these variables to the propositions is defined by the following relations:

$$x_i = \begin{cases} 1 & \text{iff attribute i is TRUE} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Imposition of logical conditions linking the different actions in a model is achieved by expressing these conditions in the form of linear constraints connecting the associated decision variables.

Let us assume that a product is defined by five different attributes as shown in table 4. Our task is to write a set of linear constraints which represents same information as described for configuration

Attribute	Name	Selection Rule
1	Rear-view camera	$1 \rightarrow \neg(4 \vee 5) \wedge (\neg 6)$
2	Parking assistant system	$2 \rightarrow (1) \wedge (\neg(4 \vee 5))$
3	Cruise control	$3 \leftarrow (1 \vee (4 \wedge 5))$

Table 4: Example: attributes and their selection rule

problem. In this example attribute 1, 2 and 3 imposes a selection rule criteria while attribute 4 and 5 do not have explicit dependencies.

Our approach, in principle, involves identification of precise compound attribute rules of the problem and then processing it with identified equations. The logical rule is represented by a tree graph (as per Section 4.1.1), where attributes are associated with their common operator node. We traverse through the tree and prune it in such a way, that the standard transformation equation can be applied [12]. Figure 4 shows the final expression tree for configurations rule written in Table 4.

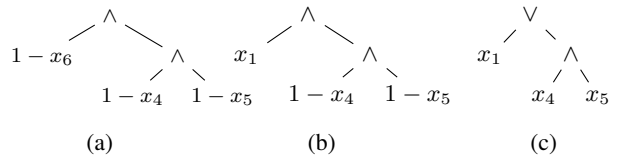


Figure 4: Example: Final expression tree for (a) Attribute 1 (b) Attribute 2 (c) Attribute 3

$$\begin{bmatrix} 3 & 0 & 0 & 1 & 1 & 2 & 0 \\ -1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & -2 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 & -1 & 0 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \leq \begin{bmatrix} 4 \\ 2 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\mathbf{B} \times [x] \leq \mathbf{b} \tag{3}$$

Where: B = Coefficient matrix over attributes and b is the right-hand side values. In order to transform the given Boolean expressions to liner constraints we introduced new variable x_7 corresponding to attribute 3. Attribute x_7 controls boolean expression $4 \vee 5$. Resulting constraint system is shown in Eq. 3.

4.2 The configuration transformation model

In this section we present a mathematical model for the transformation of a base configuration (configuration produced in past) to the new configuration. The new configuration should satisfy all restrictions imposed by product document and should have maximum correlation with its base configuration.

Let

i be i^{th} attribute, $i \subseteq \{1...n\}$, where n is the total number of attributes

Data

$$a_i = \begin{cases} 1 & \text{if } i^{th} \text{ attribute is present in base configuration} \\ 0 & \text{otherwise} \end{cases}$$

c_i = Change cost associated with attribute i. We assume that c_i is given as input either from user or derived from sales planning data (e.g. cost of attribute)

Decision variables

$$x_i = \begin{cases} 1 & \text{if } i^{th} \text{ attribute is in transformed configuration} \\ 0 & \text{otherwise} \end{cases}$$

Objective Function

$$\textbf{Z = Minimize} \sum_i c_i \times |a_i - x_i| \tag{4}$$

Subject to

$$B[x] \leq b \tag{5}$$

The Hamming distance between base and new configuration for attribute i calculated by $|a_i - x_i|$. Constraints in Eq. 5 is the set of linear inequalities derived from configurations rules (restrictions). Objective function Z is used to minimize the mismatch cost associated with each attribute so that the transformed configuration will match the base (old) configuration as close as possible. Change cost associated with each attribute is assumed here as an input data provided either by planning experts (sales) or by user. Usually for automobile change cost for complex attributes such as power train, production country is high compare to other attributes. In this case user can specify relative cost (such as weight factor or priorities) among attributes. Constraint 5 is a set of linear constraints originating by transforming logical conditions written in the product document to linear inequalities using the procedure described in Section 4.1. Any new configuration $[x]$ from the above optimization model will guarantee that the configuration is feasible according to the product document and the objective function will ensure its minimum cost deviation from the base configuration. As the configuration transformation model transforms one configuration at a time, for every transformation of non-feasible (according to given product document) configuration, this model needs to be run. A typical practical instance of this problem contains around 500-1000 decision variables and some tens of thousands of constrains.

5 Solution framework

Our aim is to provide an automated system which can interpret information from configuration data and planning experts. The system should consider given information in the best possible way while transforming the base (given) product variants to new (upgraded) variants. For this, we will create a knowledge database, where information from planning experts can be stored and used during the configuration transformation. The term *planning experts* is used to present collective information/rules specified by engineers/product managers or the user of the our application. For the reconfiguration problem, change information can be described by the customer and same can be applied during updating the base configuration. The expert database will collect the changes of attributes from one stage of product to another. Table 5 shows an excerpt of such a knowledge database. In the expert database we want to maintain an explicit set of rules which can be applied in a guided way to base configuration. For example, in the past, a car was produced only with one type of entertainment system. Due to some enhancement in the product, the manufacturer now provides three different entertainment systems. The challenge will be to distribute new entertainment systems over configurations produced in the past. In this case, the knowledge of the planning expert plays an important role in achieving a realistic transformation of past products.

Situation	**in Past**	**in Future**
An old attribute is replaced by new attribute (one to one mapping)	i	j
An attribute has been replaced by number of new attributes (one to many mapping)	i	$i = j$ for 70% configurations produced in past; $i = k$ for 30% configurations
Group of individual attribute replaced by new Package (many to one mapping)	i, j, k	$p = [i, j, k]$ add package p if at least two attribute from $\{i, j, k\}$ is present
An old package is divided in more than one packages	$p = [i, j, k, l]$	$p_1 = [i, j]$, $p_2 = [k, l]$

Table 5: An excerpt of expert's knowledge database

Knowledge from the expert database is applied to every configuration that we want to transform. It may happen that the modifications from the expert database do not suffice to meet all the configuration rules. In that case, we use the configuration transformation model presented in section 4.2. A solution is sought automatically that is valid under the new model, but which differs minimally (in the "Hamming distance") from the "old" configuration. The flow diagram in Figure 5 shows the solution framework.

The configuration transformation process starts with analyzing the product configuration rules. In this step, we can get the list of all available attributes and attribute dependencies in terms of logical rules. These rules can be converted into a set of linear inequalities as discussed in Section 4.1. Once the configuration rules are modelled as constraints, we will look into the expert database and apply all possible attribute mappings described in the expert database. All discontinued attributes will be removed from the base configuration because they will not be valid for the new model. At this stage, we will check if this configuration is feasible according to given configuration rules. If the answer is YES, we proceed with transforming the next configuration. If the answer is NO, we call the configuration transformation model as defined in Section 4.2. We repeat the above

procedure till all configurations are transformed.

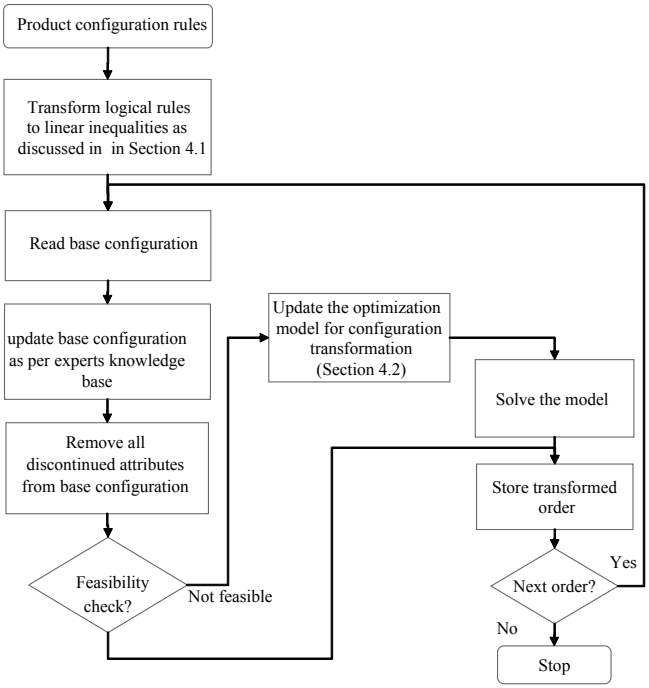


Figure 5: Flow diagram for transformation of product variants from past to given document information

5.1 Computational Experiments

We have tested our solution approach with various industry size problems. In this section, we will present two different experiments created out of practical scenarios in the automotive industry.

Sr	Scenario	total attributes	total base configurations
Exp_1	Transforming past configurations as per changes in configuration rules	695	2200
Exp_2	Upgrading base configurations with new Engine	705	1000

Table 6: Excerpt of computational scenarios

Table 6 shows the computational set up for two experiments. In the first experiment (Exp_1) our aim is to utilize customer orders produced in the past for future production planning. For this, 2200 past configurations are taken which are 6 months old from new production date. As the product has undergone engineering changes, our aim is to upgrade the given configurations as per the new configuration rules. The new configurations are defined with 695 different attributes.

Figure 6 shows the plot of time versus Hamming distance for transformed configurations. The transformation is done after analysing new configurations rules which results in information such

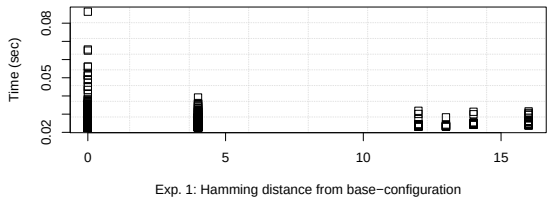


Figure 6: $Experiment_1$ Transforming base configurations as per new configuration rules

as discontinuation of some old attributes. Removing of barred attributes and application of information from expert's knowledge as discussed in section 5, we found that a large number of configurations become feasible (Hamming distance zero in figure 6) . For other configurations, solutions are found by solving the optimization model as discussed in section 4.

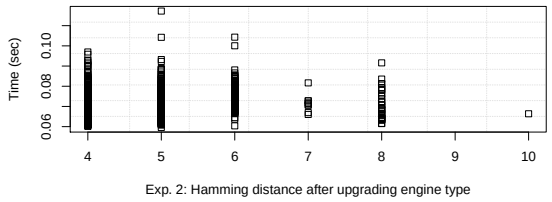


Figure 7: $Experiment_2$ Hamming distance vs time plot of engine upgradation problem

In experiment 2, we solved the reconfiguration problem by upgrading the engine type. Given 1000 configurations were upgraded to a new engine type. First, attributes related to the old engine type were replaced with the new engine and some related attribute replacements were done through expert's knowledge base. For example, associating the right gear box for the new engine. After user's modification, we transformed the given configurations as per the model shown in Figure 7. A large number of given configurations are transformed with minimal changes (4-8 attributes) to its original values. The optimization model out of configuration rules has a few thousand decision variables and thousands of linear constraints.

We used the optimization solver IBM Ilog Cplex 12.2 to solve the order transformation model. For simplicity, the following assumptions were made: 1) the attribute change cost is assumed to be one in $Experiment_1$. 2) In $Experiment_2$ we used a relatively high change cost for new engine and in all transformed configurations, the attributes related to new engine remained unchanged. On applying expert knowledge and the mathematical model that we have developed, the initial computational results shows that the given configurations can be transformed as per the desired objective in reasonable computation time (a few seconds).

6 Related work

Product configuration systems have been a key enabler for mass customization. One main contribution of configurations system is to support mass customization at various key processes such as product configuration, product data management (PDM) and customer relationship management (CRM) for effective product and process variety management [5]. The effect of configuration process can be seen on the customization responsiveness when information from sources such as customer requirements, product characteristics, production process and logistics network are considered in the configuration systems [8].

In a variant rich customizable product, finding customer focused configurations out of enormous choices is a challenging task [16]. Failing to access market needs has an adverse effect in product quality of product configurators [15]. Enabling production planning with customer historical demand (configurations produced in the past) may help to retain aspects of customer buying behaviour. However, to use past configurations for future production planning, an upgradation is required. Fichter et. al. [4] considered some of the product change conditions in their work of transforming configurations between two different product document rules. They proposed a knowledge based framework to transform invalid product variants according to change of rules in a configurator. However, in their heuristic approach it is not clear whether the transformed configuration has small deviation (minimal cost/distance) from original configuration. Walter et. al. [17] have discussed MaxSAT based approach for reconfiguration problem. In our paper we translated configurations propositional rules to set of linear constraints and the configuration transformation problem. An optimization based model has some advantages and the results of this formulation can be extended to support the generation or the transformation of sets of configurations [12].

7 Conclusion

In order to adapt the customer configurations produced in the past to the latest engineering design and market conditions we have discussed an optimization based framework. Design related changes are captured in our optimization model by transforming the product configuration rules to a set of linear inequalities. Market and expert knowledge during configuration transformation are captured by maintaining a knowledge database to transform configurations according to the best available information. The method will facilitate future planning activities based on consistent and constructible configuration sets (order sets), which will have maximum correlation with the past customer demand. For a complex product which changes dynamically with respect to time, production planning activities will improve gradually with the effective adaption of design and market changes.

REFERENCES

[1] Egon Balas, 'Logical constraints as cardinality rules: Tight representation', *Journal of Combinatorial Optimization*, **8**(2), 115–128, (2004).

[2] Caroline Becker and Hélène Fargier, 'Maintaining alternative values in constraint-based configuration', in *IJCAI*, ed., Francesca Rossi. IJCAI/AAAI, (2013).

[3] Hachemi Bennaceur, 'A comparison between sat and csp techniques', *Constraints*, **9**(2), 123–138, (2004).

[4] Michael Fichter, Michael Klein, and Andreas Schmidt, 'Transformation of product between various version of the rule world of a product configurator', *IEA/AIE, Springer-Verlag Berliun Heidelberg*, **5579**(1), 721 – 730, (2009).

[5] C. Forza and F. Salvador, 'Application support to product variety management', *International Journal of Production Research*, **46**(3), 817–836, (2008).

[6] H. Graf, 'Innovative logistics is a vital part of transformable factories in the automotive industry', in *Reconfigurable Manufacturing Systems and Transformable Factories*, ed., AnatoliI. Dashchenko, 423–457, Springer Berlin Heidelberg, (2006).

[7] T. Hadzic, S. Sathiamoorthy, R. M. Jensen, H. R. Andersen, J. Møller, and H. Hulgaard, 'Fast backtrack free product configuration using precompiled solution space representations', in *Proceedings of the International Conference on Economic, Technical and Organisational aspects of Product Configuration Systems*, (2004).

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

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

[10] Andrew Kusiak, M. R. Smith, and Zhe Song, 'Planning product configurations based on sales data', *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, **37**(4), 602–609, (2007).

[11] Peter Manhart, 'Reconfiguration - A problem in search of solutions', in *IJCAI'05 Configuration Workshop*, eds., Dietmar Jannach and Alexander Felfernig, pp. 64–67, (2005).

[12] Tilak Raj Singh and Narayan Rangaraj, 'Generation of predictive configurations for production planning', in *15 th International Configuration Workshop*, p. 79, (2013).

[13] C. Sinz, A. Kaiser, and W. Küchlin, 'Formal methods for the validation of automotive product configuration data', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **17**(1), 75–97, (JAN 2003). Special issue on configuration.

[14] R. Srinivasan and J. M. Swaminathan, 'Managing configurable products in the computer industry: Planning and coordination issues', volume 22, pp. 33–43. Sadhna:Academy Proceedings in Engineering Sciences, (February 1997).

[15] Alessio Trentin, Elisa Perin, and Cipriano Forza, 'Product configurator impact on product quality', *International Journal of Production Economics*, **135**(2), 850 – 859, (2012). Green Manufacturing and Distribution in the Fashion and Apparel Industries.

[16] Alessio Trentin, Elisa Perin, and Cipriano Forza, 'Sales configurator capabilities to avoid the product variety paradox: Construct development and validation', *Computers in Industry*, **64**(4), 436 – 447, (2013).

[17] Rouven Walter, Christoph Zengler, and Wolfgang Küchlin, 'Applications of maxsat in automotive configuration', in *15 th International Configuration Workshop*, volume 1, p. 21, (2013).

Testing Configuration Knowledge-Bases

Franz Wotawa and **Ingo Pill**[1]

Abstract. Writing tests for configuration knowledge-bases is a difficult task. One not minor reason is the huge search space. For exhaustive testing, all possible combinations of configuration parameters must be considered. In practice, exhaustive testing is thus often impossible, due to the sheer, exponential, number of combinations. Consequently it becomes necessary to focus on the most important configurations first. This abstract challenge is well-known in the testing community, and can be addressed by exploiting combinatorial testing. Combinatorial testing deals with reducing the number of test inputs by aiming at exhaustive combinations of parameter subsets. That is, ensuring that a test-suite contains tests covering all value combinations for all parameter subsets for (or up to) a given size. In this paper, we formulate the configuration-testing problem and show how combinatorial testing can be used in a corresponding test case generation process, in order to achieve a huge reduction in the number of required test cases.

1 INTRODUCTION

A configuration, i.e., *something that results from a particular arrangement of parts or components* (according to the Merriam Webster dictionary[2]), can be considered as a system aggregating specific parts in order to implement a desired functionality or behavior. In model-based configuration, we use a knowledge-base in order to represent those components' functionality, given user requirements, and any other knowledge that is necessary for defining or constructing the system. Such additional knowledge encompasses, for instance, constraints prohibiting physically impossible (and thus conflicting) arrangements. Obviously, the outcome of any configuration algorithm depends heavily on the model's quality. In some sense, quality in this case can be considered as being "as close as necessary (and possible) to reality", so that we need to capture the "appropriate" knowledge and do that in the right way.

In case of faults in the knowledge base, e.g., when we miss some constraint that prohibits some impossible configuration, a derived configuration might be incorrect for at least some specific scenarios or corner cases. Thus, testing, which is basically unavoidable for verification and validation problems, is not only essential for hardware and programs, but also for knowledge-bases. We certainly have to ensure that a configuration behaves as desired. The evidence is even stronger when moving from static configuration, e.g., configuring a product based on user needs, to dynamic configurations where the system might adapt itself for a certain situation. For example, a robot might adapt its control behavior in case of a broken wheel, that is, on its view of the world that it stores in an internal knowledge-base as foundation for its reasoning. In such cases, a reliable and, to a certain degree, expected and "safe" behavior has to be ensured.

[1] Technische Universität Graz, email: {wotawa,ipill}@ist.tugraz.at

[2] http://www.merriam-webster.com/dictionary/configuration

In this paper, our focus is on such faults in configuration knowledge-bases an their consequences. Of course, another source for failure is in the configuration algorithm's implementation, i.e., the reasoning engine, itself. While such faults are outside our paper's focus, the generated tests can also be used to test the reasoner.

Regarding fault detection and isolation, the size of a knowledge-base is of certain interest. That is, if the knowledge-base itself, or the configuration space, is very small, exhaustive testing might even be feasible and a valid option for specific situations. However, in case of huge knowledge-bases or huge configuration spaces, exhaustive testing is practically impossible. For instance, and without loosing generality, let us assume the example application of parameter configuration. There the purpose is to find a value assignment to all available system parameters, in order to receive a setup that implements a desired functionality. If we have parameters $p_1, \ldots, p_n$, each taking values from a domain D with size k, an exhaustive search would require us to test k^n possible configurations, which, for the obvious reasons, is most likely infeasible for those values for k and n experienced in practice. Therefore, we require effective alternatives that allow us to systematically focus our testing efforts.

In system testing, where we often have to test in the context of alternative "environments', we suffer from a similar problem. For instance, if we want to test a web page, we have to consider various hardware platforms from PCs to smart phones and tablets, a variety of operating systems, a set of web browsers commonly used, and so on. Testing the web page in the context of all the possible "configurations" is of course an achy task that requires a lot of resources. An empirical study (see e.g., [12]) showed, however, that not all combinations of parameter value assignments are necessary for revealing a bug. Rather, it seems sufficient to consider local parameter configurations. Implementing the concept of combinatorial testing (see also Section 4), we aim to cover all local parameter combinations up to, or of, a given size in a test suite. That is, all the combinations for (all possible) chosen "local" subsets of parameters, which allows us to dramatically reduce the number of required tests. Of course the choice of the subset size directly influences the "locality" of the test case generation process.

In this paper, we discuss the testing problem for configuration knowledge-bases and propose the use of combinatorial testing for automated test input generation. We introduce our preliminary definitions using a simplified example from the e-vehicle domain, and furthermore discuss two different testing aspects. First, we consider testing of different configurations. And second, when considering the desired functionality as being changeable, there arises the question of whether there actually is a valid configuration for a certain combination of functionalities.

Our paper is organized as follows. First, we discuss some related research with a focus on testing of knowledge-based systems in general. Afterwards, we introduce the foundations of configuration using

a running example. We then use the same example to discuss combinatorial testing. After the introduction into combinatorial testing, we discuss testing of configuration knowledge-bases in more detail. Finally, we conclude the paper and outline future research directions.

2 RELATED RESEARCH

Knowledge-based systems are used for various purposes like configuration, diagnosis, and also decision support, e.g., for high-level control of systems. For all these application areas, systems have to be predictable, that is, they have to behave as expected and do not cause any trouble leading to a loss of resources or even harm people. Despite this fact, it is interesting to note that there has not been a huge number of papers dealing with testing, verification, and validation of knowledge-based systems. Robert Plant [17, 18] was one of the first dealing with verification, validation, and testing of expert systems and knowledge-based systems in general. There is also an earlier survey available (see [13]) that deals with tools for validation and verification of knowledge-based systems.

Regarding testing of knowledge-based systems, it is also worth mentioning El-Korany and colleagues' work [5], where their focus is on the testing methodology. There the authors distinguish different cases where testing is required, i.e., inference knowledge testing and task knowledge testing. The objective behind their work was to increase the level of correctness of knowledge-based systems. Other work includes [9], where Hartung and Håkansson discuss test automation for knowledge-based systems. Their approach works for production rules that are extracted from the knowledge-bases.

Hayes and Parzen [10] focused more on the question of "to what degree a knowledge-based systems fulfills its purpose", that is, as indicated in the title of their publication, on achieving the desired behavior. In order to answer the question about the quality of decisions coming from a knowledge-based system, Hayes and Parzen introduced a special metric (QUEM) to judge the quality of the solutions. The proposed approach is essential for measuring the overall performance of a knowledge-based system.

To the best of our knowledge, there is only little work on testing configuration motors or motors that make use of configuration methods like recommenders. Felfernig and colleagues [8, 7] discuss the use of testing, i.e., white-box testing, and development environments in the context of recommender applications. Other work from Felfernig and colleagues [6] mainly focuses on the second step of debugging, i.e., fault localization and correction, but still requires test cases for finding inconsistencies between the behavior coded in a knowledge base and the expected behavior, which originates from knowledge engineers or customers of the configuration system.

Tiihonen and colleagues [20] described a rule-based configurator and also introduced a more or less model-independent testing method. In their approach the configurator is tested using randomly generated requirements given to the configurator. Besides discussing the underlying methodology Tiihonen et al. also presents empirical results gained from 4 different configuration models. In contrast to Tiihonen and colleague the testing approach proposed in this paper is not a random testing approach. Moreover, our focus is more on testing the configuration knowledge-base and not the whole configurator. Although, the obtained tests can be used later for testing concrete implementations.

In our paper, we rely on previous research in the domain of testing knowledge-based systems, but focus on the specific case of knowledge-based systems for configuration. We distinguish different cases for testing and suggest to use a specific testing methodology, i.e., combinatorial testing, which seems to suit configuration very well.

3 THE CONFIGURATION PROBLEM

For illustration purposes, let us consider the following simplified example from the domain of vehicle configurations. In Figure 1, we illustrate an example comprising an electric vehicle that contains an electric motor, electric consumers like an air-condition, and a battery that delivers the required electricity. Battery size and other factors, like the driving mode, substantially influence the range of the vehicle. The configuration knowledge base for our example comprises four components, i.e., an air-condition, a motor, the driving mode, and the battery - each of them offering some options, which then vary from configuration to configuration. Let us now assume that there are two engine types (*standard* and *powerful*), three types of air-condition (*none*, *manual*, and *electronic*), two driving modes (*leisure* and *race*), and three different batteries (*type1*, *type2*, *type3*), each providing a different electric capacity. Clearly, the configured vehicle's range depends heavily on the battery and actual power consumption. That is, for instance, if there is too much power consumption, some particular range can never be achieved. The range, however, reflects an important part of a customer's needs. While customer A is satisfied when she can drive the car for one day in a city for no more than 100 km, customer B expects his car being able to cover more than 200 km before it has to be recharged. Other customer requirements might concern air-conditioning, or the availability of a particular driving mode.

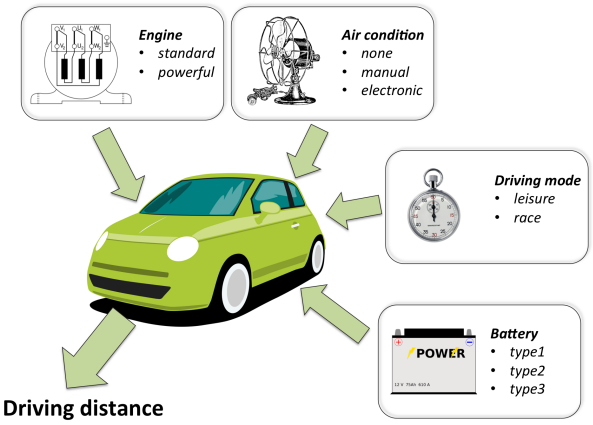


Figure 1. Configuration problem of an electric vehicle

In the following, we discuss the formalization of our e-vehicle configuration example, but let us introduce the definition of a configuration problem first.

Definition 1 (Configuration problem) *A configuration problem is a tuple* $(SD \cup REQ, PARTS, MODES)$ *where SD is the system description, REQ are the requirements, and PARTS are the configurable parts that are allowed to be set to particular modes from MODES.*

We assume that *SD* and *REQ* are first order logic formulae. Other formalisms might also be used requiring the existence of consistency

checks and reasoning capabilities. Our definition of the configuration problem assumes that the functionality or behavior of the parts (from *PARTS*) are defined in *SD* for a particular mode (from *MODES*). For our e-vehicle example, we consider 4 different parts: the electric motor (*emot*), the air-condition (*ac*), the driving mode (*dm*) and the battery (*bat*), i.e., $PARTS = \{emot, ac, dm, bat\}$.

What is missing, is the configuration knowledge and the requirements. Regarding the latter, we assume that we want to distinguish slow acceleration cars (*slowacc*) from fast acceleration cars (*fastacc*), as well as the availability of air-condition cooling (*cooling*). Moreover, a user might specify the maximum distance before recharging, which might be *city* for less than or equal 100 km, *interurban* for distances up to 250 km, and *max*, otherwise. In the following, we depict *SD* for our running example. In the system description, we make use of the predicate *mode* that assigns a component a certain parameter value, *cons* for fixing the power consumption of a part, and *avpow* for stating the available electrical power for batteries.

Electric motor: The standard engine provides slow acceleration only, but draws less electrical power. The powerful motor provides fast acceleration but consumes more electricity as a downside.

$$mode(emot, standard) \rightarrow (cons(emot, 300) \wedge slowacc)$$
$$mode(emot, powerful) \rightarrow (cons(emot, 400) \wedge fastacc)$$

Air-condition: If there is no air-condition, then there is no power consumption and also no cooling. The manual air-condition draws less power than the electronic one. Both provide cooling.

$$mode(ac, none) \rightarrow (cons(ac, 0) \wedge \neg cooling)$$
$$mode(ac, manual) \rightarrow (cons(ac, 100) \wedge cooling)$$
$$mode(ac, electronic) \rightarrow (cons(ac, 150) \wedge cooling)$$

Driving mode: A leisure driver consumes no additional electricity on top of the power required to drive the motor. A racy driver draws more power due to higher acceleration.

$$mode(dm, leisure) \rightarrow (cons(dm, 0))$$
$$mode(dm, race) \rightarrow (cons(dm, 100))$$

Battery: The three battery types have varying capacities.

$$mode(bat, type1) \rightarrow (avpow(bat, 450))$$
$$mode(bat, type2) \rightarrow (avpow(bat, 600))$$
$$mode(bat, type3) \rightarrow (avpow(bat, 800))$$

Other constraints: There are several further domain-dependent constraints: The racy driving mode can only be obtained when having a powerful motor, i.e., it is not possible to have fast acceleration without the right motor.

$$\neg\,(mode(dm, race) \wedge \neg fastacc)$$

In addition, we have to ensure that a component cannot be in more than one mode simultaneously,, and that some available functions are in contradiction, e.g., slow and fast acceleration.

$$\neg\,(mode(emot, standard) \wedge mode(emot, manual))$$
$$\neg\,(mode(none, standard) \wedge mode(manual, manual))$$
$$\neg\,(mode(none, standard) \wedge mode(manual, electronic))$$
$$\neg\,(mode(none, manual) \wedge mode(manual, electronic))$$
$$\neg\,(mode(dm, leisure) \wedge mode(dm, race))$$
$$\neg\,(mode(bat, type1) \wedge mode(bat, type2))$$
$$\neg\,(mode(bat, type1) \wedge mode(bat, type3))$$
$$\neg\,(mode(bat, type2) \wedge mode(bat, type3))$$
$$\neg\,(slowacc \wedge fastacc)$$

The power consumption of all vehicle parts should never exceed the available power, so that we add an integrity constraint:

$$avpow(bat, B) \wedge cons(emot, E) \wedge cons(ac, A) \wedge cons(dm, D) \rightarrow B > (E + A + D)$$

Finally, we have to map power consumption and available power to the vehicle's maximum distance (without recharging) class.

$$avpow(bat, B) \wedge cons(emot, E) \wedge cons(ac, A) \wedge cons(dm, D) \wedge B - (E + A + D) > 99 \rightarrow city$$

$$avpow(bat, B) \wedge cons(emot, E) \wedge cons(ac, A) \wedge cons(dm, D) \wedge B - (E + A + D) > 200 \rightarrow interurban$$

$$avpow(bat, B) \wedge cons(emot, E) \wedge cons(ac, A) \wedge cons(dm, D) \wedge B - (E + A + D) > 400) \rightarrow max$$

It is worth noting that the above definition allows to derive different maximum distances at the same time. If the distance is larger than 400 *city*, *interurban*, and *max* become valid. This could be avoided via integrity constraints or chaining to constraints, in order to get non-overlapping definitions. However, this definition is intended such as to allow to specify a minimum capability.

Now let us we define formally what we understand about a configuration. Intuitively, a configuration has to do with a mode assignment, which corresponds to choosing a certain part, e.g., setting the battery to *type1* means that we want this battery in our configuration.

Definition 2 (Configuration)
Let $(SD \cup REQ, PARTS, MODES)$ be a configuration problem. A configuration is an assignment of a particular mode to each of the parts, i.e., a set C is a configuration, if and only if $|C| = |PARTS|$ and $\forall\, p \in PARTS : \exists\, mode(p, m) \in C$ where $m \in MODES$.

As this definition ignores *REQ* and *SD*, it induces the whole configuration space. Being interested only in valid configurations, i.e., those that do not contradict *REQ* and *SD*, we define them as follows:

Definition 3 (Valid configuration) *Let C be a configuration for the configuration problem $(SD \cup REQ, PARTS, MODES)$. Configuration C is* valid *if and only if $SD \cup REQ \cup C$ is satisfiable.*

Clearly Definition 3 does not ensure that a valid configuration meets the requirements. Hence, we define a suitable configuration.

Definition 4 (Suitable configuration) *Let C be a valid configuration for the configuration problem $(SD \cup REQ, PARTS, MODES)$. C is* suitable *iff the requirements REQ can be derived from the system description and the configuration, i.e., $SD \cup C \models REQ$.*

The user requirements have a direct impact on the space of suitable configurations. Clearly, $REQ = \{city\}$ has more suitable configurations than the requirements $REQ = \{city, cooling\}$. The given definitions of configuration are close to those of reconfiguration and parameter configuration, e.g. from [19, 15]. However, to some extent, generative configuration, e.g., [19], can also be handled, when assuming a boundary for involved components and connections. Each potential component and connection has to be defined in the system description having two modes. One is for indicating the use of a component or connection in a configuration, and the other for stating that the component or connection is not used. In addition, some integrity constraints have to be specified, in order to ensure that in a

final configuration there is no connection without the corresponding components. Note that for larger systems and configurations such a bounded variant might lead to a description that cannot be used for computing configurations in reasonable time, which does not contradict the observation that the given definitions - in principle - allow for specifying different configuration problems.

Let us come back to our running example and the definition of suitable configurations. When stating $REQ = \{city, cooling\}$ we can obtain the suitable configuration

$$\left\{ \begin{array}{c} mode(emot, standard), mode(ac, manual), \\ mode(dm, leisure), mode(bat, type1) \end{array} \right\}$$

but also

$$\left\{ \begin{array}{c} mode(emot, powerful), mode(ac, electronic), \\ mode(dm, leisure), mode(bat, type2) \end{array} \right\}$$

among others. The configuration

$$\left\{ \begin{array}{c} mode(emot, powerful), mode(ac, none), \\ mode(dm, leisure), mode(bat, type1) \end{array} \right\}$$

would be a valid one, but is not suitable as $cooling$ is not established. For computing configurations meeting requirements, we refer the interested reader to [19] or [15].

What remains now, is the question whether the formalized configuration problem represents reality and results in the desired configurations. Hence, we need to test the configuration knowledge-base. To this end, in the next section we introduce a certain testing methodology suitable for this task.

4 COMBINATORIAL TESTING

Combinatorial testing is a method for the algorithmic computation of tests and in particular test input data for a system under test (SUT).

An answer to the question of how much test input data we should generate in order to reveal undetected faults is of great practical importance. As mentioned before, for n inputs with k possible values, an exhaustive approach would require us to test k^n combinations. When missing an important combination, so that a fault remains in the source code, the consequences might be catastrophic, especially for safety-critical systems. Recently, researchers suggested not to consider all input value combinations, but only certain ones focusing on an exhaustive "local" search (see e.g. [3, 23, 24]). The underlying idea is that while input combinations might be required in order to reveal a bug, in practice, we can restrict the size of considered combinations and consider multiple "local" combinations in a test case.

Combinatorial testing formalizes this idea of considering a certain combination of inputs - in our case parameter subsets of size -, e.g., 2 or 3, where all possible value combinations are tried. Regarding the considered combination of inputs we distinguish the strength of combinatorial testing, e.g., strength 2 or 3. Each strength t (where $t \geq 2$) requires that each t-wise tuple of values of the different system parameters is covered at least once in the test suite, which reduces the necessary number of test cases substantially. Of course, the strength t could also be set to the maximum in order to do an exhaustive search. The natural question is then if this method is sufficient. In [12], for example, the authors report on an empirical study considering various programs from different domains and showed that it was enough to consider six-way interactions in order to detect all faults.

We now illustrate combinatorial testing in the context of our running example, where we restrict our focus purely on the testing methodology. Even more details are offered in the next section. For brevity let us consider the component modes as inputs:

input	values
$emot$	$standard, powerful$
ac	$none, manual, electronic$
dm	$leisure, race$
bat	$type1, type2, type3$

When searching for all two-way combinations, i.e., combinations of values for two particular inputs, we would obtain results similar or equivalent to the one depicted in Table 1. There for each combination of two inputs, all possible value combinations are given, which results in 9 test cases. For comparison reasons, we also depict the test cases for strength 3 in Table 2. It is worth noting that, when considering all combinations, we would finally obtain 36 test cases.

Table 1. All two-way interactions for the e-vehicle example

	$emot$	ac	dm	bat
1	$powerful$	$none$	$race$	$type1$
2	$standard$	$none$	$leisure$	$type2$
3	$powerful$	$none$	$leisure$	$type3$
4	$standard$	$manual$	$race$	$type1$
5	$powerful$	$manual$	$leisure$	$type2$
6	$standard$	$manual$	$race$	$type3$
7	$powerful$	$electronics$	$leisure$	$type1$
8	$standard$	$electronics$	$race$	$type2$
9	$powerful$	$electronics$	$race$	$type3$

The significant advantage of combinatorial testing is that the number of test cases can be reduced while still considering combinations of input values. In order to implement combinatorial testing as a test case generation method, the following steps are required:

1. First, someone has to write a model of the input space, comprising the inputs and their value domains.
2. The combinatorial design procedure takes this input space and generates an array where each row is simple a test case describing the value for each input considering the given strength t.
3. Every row is delivered back as a single test case describing potential input data (but not the expected output).

Another benefit of combinatorial testing is that Steps 2 and 3 can be automated completely. There are tools available for computing the test cases, e.g., the ACTS combinatorial test generation tool [16]. ACTS has been developed jointly by the US National Institute Standards and Technology (NIST) and the University of Texas at Arlington and currently has more than 1,400 individual and corporate users.

A drawback of combinatorial testing is that only test *input* data is generated. Hence, the oracle problem, i.e., classifying the computed output as being correct or not, still remains for combinatorial testing. However, at least, combinatorial testing offers a structured and well defined method for test input data generation that can be effectively used in practice.

Regarding an algorithm for computing test cases using combinatorial testing, we refer the reader to the available literature. The underlying data structure for computing the test is the mixed-level covering array which can be defined as follows (see [4]).

Definition 5 *A mixed-level covering array which we will denote as* $MCA(t, k, (g_1, \ldots, g_k))$ *is an* $k \times N$ *array in which the entries of*

Table 2. All three-way interactions for the e-vehicle example

	$emot$	ac	dm	bat
1	$standard$	$none$	$leisure$	$type1$
2	$powerful$	$none$	$race$	$type1$
3	$standard$	$none$	$race$	$type2$
4	$powerful$	$none$	$leisure$	$type2$
5	$standard$	$none$	$leisure$	$type3$
6	$powerful$	$none$	$race$	$type3$
7	$standard$	$manual$	$race$	$type1$
8	$powerful$	$manual$	$leisure$	$type1$
9	$standard$	$manual$	$leisure$	$type2$
10	$powerful$	$manual$	$race$	$type2$
11	$standard$	$manual$	$race$	$type3$
12	$powerful$	$manual$	$leisure$	$type3$
13	$standard$	$electronics$	$leisure$	$type1$
14	$powerful$	$electronics$	$race$	$type1$
15	$standard$	$electronics$	$race$	$type2$
16	$powerful$	$electronics$	$leisure$	$type2$
17	$standard$	$electronics$	$leisure$	$type3$
18	$powerful$	$electronics$	$race$	$type3$

the i-th row arise from an alphabet of size g_i. Let $\{i_1, \ldots, i_t\} \subseteq \{1, \ldots, k\}$ and consider the subarray of size $t \times N$ by selecting rows of the MCA. There are $\prod_{i=1}^{t} g_i$ possible t-tuples that could appear as columns, and an MCA requires that each appears at least once. The parameter t is also called the strength of the MCA.

The mixed level covering array defines all possible combinations of t inputs having a finite value domain of g_i for an input i. It is worth noting that in combinatorial testing we have to have finite domains (which is perfectly fine in case of configuration knowledge-bases). We might also remark that the technique for discretizing the parameter values is referred to as input parameter modeling in combinatorial testing [11]. After discussing combinatorial testing, we show how combinatorial testing can be effectively used for testing configuration knowledge-bases in the next section.

5 TESTING KNOWLEDGE-BASES

The obvious purpose of testing is to reveal a SUT's faults. To this end, the SUT is executed using certain input values, and the resulting behavior is logged. This behavior is compared with the expected one. In case of deviations, a fault is detected and we certainly get interested in the corresponding root causes. In his ACM Turing Lecture 1972, Edsger W. Dijkstra mentioned that "*program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence*". Hence, someone might be interested in efficiently detecting deviations, i.e., finding the right input that causes the misbehavior. Finding such an input might be like finding a needle in a haystack. Testing methods like combinational testing help in this respect.

For a more detailed view on testing, we recommend Myers book [14], where he - aside covering other issues - introduces 10 testing principles. In the 5th one, Myers mentions that "*test cases must be written for input conditions that are invalid and unexpected as well as for those that are valid and expected*". Hence, there is a requirement not only to test for expected results, but also to execute a SUT using input values for which the SUT was not designed. In case of a configuration knowledge base, this means that we have to use also queries where we expect no solution due to inconsistencies arising during resolution.

Testing is based on test cases. We formalize test cases in a simplified form appropriate for our purposes.

Definition 6 (Test case) *A test case for a SUT is a tuple (IN, OUT) where IN is a formalization of the input values, and OUT defines the expected output when executing the SUT using IN.*

We say that a test case (IN, OUT) is a passing test case for a SUT if the execution of SUT using *IN* returns an output that is not in contradiction with *OUT*. Otherwise, we say that the test case is a failing test case. A test suite is a set of test cases. In order to test a SUT, we are interested in having a test suite that comprises at least one failing test case. If there is no such test case, we assume the SUT to be correct with respect to the test suite.

After discussing some basic testing principles, the question remains of how to actually test configuration knowledge-bases. According to Definition 1, the formalized knowledge covers the system description *SD* and the requirements *REQ*. What we actually want to ensure is that when querying the knowledge-base using a certain request, we obtain the expected result. Hence, for testing purposes, we are interested mainly in testing *SD* and not *REQ*.

There are some additional aspects when discussing testing configuration knowledge-bases. For testing ordinary programs, the role of input and output variables is well known. For configuration problems, someone might, however, also consider *REQ* as input and the set of suitable configurations *SCONF* as output. It might also be desirable to ask for the requests to be obtained when assuming a certain configuration. In terms of configurations, most likely there are some valid configurations that are not suitable. Others are not even valid. According to Myers 5th testing principle, however, we also have to check the invalid and unexpected cases.

We now formalize these two testing problems. Let us assume a system description *SD* that describes configuration knowledge regarding *PARTS* and *MODES*. The first testing problem is for checking whether the derived suitable configurations are the correct ones.

Definition 7 (Testing configuration) *The* testing configuration *problem concerns testing the knowledge-base in its capabilities for deriving the expected configurations, and can be characterized as follows:*

Input: *SD, PARTS, and MODES*
Objective: *Finding test cases of the form $(REQ, SCONF)$, where REQ are requirements, and CONF is a set of expected configurations for the configuration problem $(SD \cup REQ, PARTS, MODES)$. Note that CONF might be empty in case of inconsistencies. Otherwise, CONF is expected to comprise suitable configurations only.*

The second testing problem is related to checking whether given configurations lead to the derivation of the correct requirements, if there are any.

Definition 8 (Testing requirement derivation) *The* testing requirement derivation *problem captures the case where we are interested in testing the capabilities of the knowledge base to derive requirements from conflicts. It can be characterized as follows:*

Input: *SD, PARTS, and MODES*
Objective: *Finding test cases, of the form (C, R) where C is a configuration and R is the expected result. Obviously, R might be $\bot$ in case the configuration itself lead to an inconsistency, i.e., $SD \cup C \models \bot$. R might comprises all requests for which C is a suitable configuration, or might be empty if there are no requests for which C is suitable.*

In order to solve both configuration specific testing problems, we need a method for computing input values, i.e., requirements respectively configurations, and the resulting values. For the first part, we can easily make use of combinatorial testing with the advantage of a reduced number of test cases to be computed while still retaining the capabilities for revealing a faulty behavior. Computing the expected outcome in an automated fashion, however, is not directly possible, because of a missing specification. Hence, we have to rely on the knowledge engineer to provide this information. In the testing community, this problem is referred to as the *oracle problem*. There are some related methods like model-based testing (e.g., see [22, 21]) or metamorphic testing (e.g., see [1, 2]). The latter uses symmetries in the functions or systems to be tested in order to gain information about the correct behavior. For example, when testing the sinus function implementation, we can make use of the property $\sin(x) = \sin(2\pi + x)$. If available, such techniques can be also used for testing configuration knowledge-bases. However, in the following we discuss the overall testing process ignoring metamorphic testing.

Algorithm 1 TEST_CONF(*SD*, *PARTS*, *MODES*, *CM*)

Input: A system description *SD*, its component set *PARTS*, their modes *MODES*, and a combinatorial testing model *CM* for requirements.
Output: A test suite *TS* where also the result of the test is stored for each test case

1: *TS* := ∅
2: $t := 2$
3: *flag* := *FALSE*
4: **repeat**
5: Call the combinatorial testing algorithm using *CM* and t and store the result in T.
6: **for all** $t \in T$ **do**
7: Convert t to its corresponding requirements representation *REQ*.
8: Call the configuration engine on (*SD* ∪ *REQ*, *PARTS*, *MODES*) and store the result in *SCONF*.
9: Present *REQ* and *SCONF* to the user for obtaining a classification *UC* ∈ {**PASS**, **FAIL**, ?}
10: **if** *UC* = **FAIL** **then**
11: Ask the user for *SCONF*
12: *flag* = *TRUE*
13: **end if**
14: *TS* := *TS* ∪ {(*REQ*, *SCONF*, *UC*)}
15: **end for**
16: $t := t + 1$
17: **until** $flag$ or no more t-way combinations are possible
18: **return** *TS*

In the proposed testing methodology for configuration knowledge-bases, we make use of combinatorial testing for generating the inputs for both problems, the *testing configuration* as well as the *testing requirement derivation* problem. We use these inputs, and a configuration engine (respectively a theorem prover) for generating the current output. The input and the corresponding output is given to the user (e.g., the knowledge engineer) for classifying the result as **FAIL** or **PASS**. Note that we also have to consider that the user has no clear understanding of the expected outcome. In this case, the classification inconclusive (i.e., ?) can be used. This test input generation and classification process that keeps the user in the loop, is started considering 2-way combinations. If no **FAIL** is obtained, the process can be continued for 3-way combinations or even stronger ones, of course re-using previously obtained classifications. The process can definitely stop when strength t in combinatorial testing (for obtaining t-way combinations) reaches the number of variables used. Experimental surveys suggest that it seems enough to consider 6-way combinations (see [12]).

Algorithm 1 summarizes the steps necessary for computing a test suite in order to solve the *testing configuration* problem. The algorithm for solving *testing requirement derivation* problem is very similar. Algorithm 2 shows the necessary steps. The only differences are in the for-loop of the algorithm, where we have to take care of the different situations. Both algorithms terminate assuming a finite set of requirements and configurations. When ignoring the time required for theorem prover, computing a configuration, and user interaction, the time required for executions is mainly bound by the time required for combinatorial testing.

Algorithm 2 TEST_REQ(*SD*, *PARTS*, *MODES*, *CM*)

Input: A system description *SD*, its component set *PARTS*, their modes *MODES*, and a combinatorial testing model *CM* for configurations.
Output: A test suite *TS* where also the result of the test is stored for each test case

1: *TS* := ∅
2: $t := 2$
3: *flag* := *FALSE*
4: **repeat**
5: Call the combinatorial testing algorithm using *CM* and t and store the result in T.
6: **for all** $t \in T$ **do**
7: Convert t to its corresponding configuration representation C.
8: **if** $SD \cup C \models \bot$ **then**
9: $R := \bot$.
10: **else**
11: Call the the theorem prover with input $SD \cup C$ and store the derivable requirements in R.
12: **end if**
13: Present C and R to the user for obtaining a classification *UC* ∈ {**PASS**, **FAIL**, ?}
14: **if** *UC* = **FAIL** **then**
15: Ask the user for R
16: *flag* = *TRUE*
17: **end if**
18: *TS* := *TS* ∪ {(C, R, *UC*)}
19: **end for**
20: $t := t + 1$
21: **until** *flag* or no more t-way combinations are possible
22: **return** *TS*

Finally, it is worth discussing the computation of combinatorial tests in Algorithm 1 and Algorithm 2. For Algorithm 2, we already computed the test cases in the previous section. See, for example, Table 1 for all two-way combinations. There, test case 4 would lead to an inconsistency when calling the theorem prover, because the *standard* motor would lead to *slowacc* which contradicts the rules $\neg(mode(dm, race) \wedge \neg fastacc)$ in combination with $\neg(slowacc \wedge fastacc)$. Hence, we would be able to detect the case

where a knowledge-base is missing some of the mentioned rules.

For obtaining the combinatorial tests for Algorithm 1, the situation is a little different (but not much). There, we are interested in requirement combinations. As discussed before, there might be cases where we do not want to specify all requirements. Hence, we have to find a model for the combinatorial testing algorithm where we are able to take not care on a certain requirement. For our e-vehicle example, we have three different requirement categories: cooling, driving distance, and acceleration, each of them with the following possible values:

input	values
cooling	*true*, *false*, _
driving distance	*city*, *interurban*, *max*, _
acceleration	*slowacc*, *fastacc*, _

Note that the value _ is used to indicate that this requirement is currently not active. When using this model as input to the ACTS tool, we are able to obtain 12 combinatorial tests of strength 2 depicted in Table 3. Each row comprises requirements for our configuration model. Some of the requirements may lead to suitable configurations, some may not. This clarification has to be performed when considering the test cases in Algorithm 1.

Table 3. All two-way interactions for the requirements of the e-vehicle

	driving distance	acceleration	cooling
1	*city*	*slowacc*	*false*
2	*city*	*fastacc*	_
3	*city*	_	*true*
4	*interurban*	*slowacc*	_
5	*interurban*	*fastacc*	*true*
6	*interurban*	_	*false*
7	*max*	*slowacc*	*true*
8	*max*	*fastacc*	*false*
9	*max*	_	_
10	_	*slowacc*	*true*
11	_	*fastacc*	*false*
12	_	_	_

From the results obtained using our running example we are able to conclude that combinatorial testing – in principle – can be used to solve the two testing problems, which correspond to configuration knowledge-bases. These two problems correspond to the two different questions someone would ask during and after the development of configuration knowledge-bases. The first question, deals with the challenge of ensuring whether a knowledge-base is able to derive expected configurations. The second question is related to the evaluation whether a knowledge-base allows for deriving configurations that fulfill the given requirements. Both questions have to be addressed within the development of configurators and their knowledge-bases in order to gain trust in their correctness.

6 CONCLUSION

In this paper, we raised the question of how to test configuration knowledge-bases. We focused on model-based configuration and defined two testing problems. One for checking whether obtained configurations are in line with the requirements, and the other for testing whether the correct set of configurations is returned for given requirements. The proposed testing method relies on combinatorial testing for computing input data needed. We argued that combinatorial testing is very well suited for configuration testing, because of ensuring a good fault detection capability while still reducing the number of input combinations to consider. In practice, limited combinations, i.e., five- to six-way combinations have turned out to be sufficient for revealing faults that have not been found before. The question whether, e.g., six-way combinations are enough for configuration knowledge-base testing, will have to be addressed by future research and corresponding experiments.

In future research also the proposed approach has to be empirically evaluated. For such an evaluation, large configuration knowledge-bases should be used. Moreover, by introducing faults in the knowledge-bases someone would be able to check, whether the proposed approach is capable of detecting faults. Ideally, the fault detection capabilities should be compared with other approaches, e.g., random testing. Another interesting question is due to the testing capabilities of existing knowledge-based configuration tools. Do they support testing? Which testing strategies do they suggestion? These two questions among others can be answered, when carrying out a case study with the objective of evaluating existing configuration solutions. It is worth noting that we focussed more on the principles of testing configuration knowledge-bases in this paper and provided a solution. We leave a detailed empirical analysis of the proposed approach for future research.

ACKNOWLEDGEMENTS

The research presented in this paper has been carried as part of the eDAS project funded by the European Commission FP-7 grant agreement number: 608770.

REFERENCES

[1] T.Y. Chen, S.C. Cheung, and S.M. Yiu, 'Metamorphic testing: a new approach for generating next test cases', Technical report, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, (1998). Technical Report HKUST-CS98-01.

[2] T.Y. Chen, J. Feng, and T.H. Tse, 'Metamorphic testing of programs on partial differential equations: a case study', in *Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC '02)*, pp. 327–333, Los Alamitos, CA, (2002). IEEE Computer Society.

[3] David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, and Gardner C. Patton, 'The AETG system: An approach to testing based on combinatorial design', *IEEE Trans. Softw. Eng.*, **23**(7), 437–444, (1997).

[4] Charles J. Colbourn, 'Covering arrays', in *Handbook of Combinatorial Designs*, eds., Charles J. Colbourn and Jeffrey H. Dinitz, Discrete Mathematics and Its Applications, 361–365, CRC Press, Boca Raton, Fla., 2nd edn., (2006).

[5] Abeer El-Korany, Ahmed Rafea, Hoda Baraka, and Saad Eid, 'A structured testing methodology for knowledge-based systems', in *11th International Conference on Database and Expert Systems Applications (DEXA)*, pp. 427–436. Springer, (2000).

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

[7] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker, 'An integrated environment for the development of knowledge-based recommender applications', *International Journal of Electronic Commerce (IJEC)*, **11**(2), 11–34, (2006).

[8] A. Felfernig, K. Isak, and T. Kruggel, 'Testing knowledge-based recommender systems', *OEGAI Journal*, **4**, 12–18, (2005).

[9] Ronald Hartung and Anne Håkansson, 'Automated testing for knowledge based systems', in *Knowledge-Based Intelligent Information and Engineering Systems*, eds., Bruno Apolloni, RobertJ. Howlett, and Lakhmi Jain, volume 4692 of *Lecture Notes in Computer Science*, 270–278, Springer Berlin Heidelberg, (2007).

[10] Caroline C. Hayes and Michael I. Parzen, 'Quem: An achievement test for knowledge-based systems', *IEEE Transactions on Knowledge and Data Engineering*, **9**(6), 838–847, (November/December 1997).
[11] D.R. Kuhn, R.N. Kacker, and Y. Lei, *Introduction to Combinatorial Testing*, Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series, Taylor & Francis, 2013.
[12] D.R. Kuhn, R.N. Kacker, Y. Lei, and J. Hunter, 'Combinatorial software testing', *Computer*, 94–96, (August 2009).
[13] Stephen Murrell and Robert T. Plant, 'A survey of tools for the validation and verification of knowledge-based systems: 1985-1995', *Decision Support Systems*, **21**(4), 307–323, (1997).
[14] Glenford J. Myers, *The Art of Software Testing*, John Wiley & Sons, Inc., 2 edn., 2004.
[15] Iulia Nica and Franz Wotawa, '(re-)configuration of communication networks in the context of m2m applications', in *Proceedings of the 15th Workshop on Configuration*, Vienna, Austria, (2013).
[16] NIST, *User Guide for ACTS*. which is online available at csrc.nist.gov/groups/SNS/acts/documents/acts_user_guide_v2_r1.1.pdf; last visited on June 20th, 2014.
[17] Robert Plant, 'Rigorous approach to the development of knowledge-based systems', *Knowl.-Based Syst.*, **4**(4), 186–196, (1991).
[18] Robert T. Plant, 'Expert system development and testing: A knowledge engineer's perspective', *Journal of Systems and Software*, **19**(2), 141–146, (1992).
[19] Markus Stumptner, Gerhard Friedrich, and Alois Haselböck, 'Generative constraint-based configuration of large technical systems', *AI EDAM*, **12**, 307–320, (9 1998).
[20] Juha Tiihonen, Timo Soininen, Ilkka Niemelä, and Reijo Sulonen, 'Empirical testing of a weight constraint rule based configurator', in *ECAI 2002 Configuration Workshop*, pp. 17–22, (2002).
[21] J. Tretmans, 'Model-based testing and some steps towards test-based modelling', in *Proceedings of the 11th International School on Formal Methods for Eternal Networked Software Systems (SFM 2011)*, (2011).
[22] M. Utting and B. Legeard, *Practical Model-Based Testing - A Tools Approach*, Morgan Kaufmann Publishers Inc., 2006.
[23] Cemal Yilmaz, Myra B Cohen, and Adam A Porter, 'Covering arrays for efficient fault characterization in complex configuration spaces', *Software Engineering, IEEE Transactions on*, **32**(1), 20–34, (2006).
[24] Linbin Yu, Yu Lei, R.N. Kacker, and D.R. Kuhn, 'Acts: A combinatorial test generation tool', in *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*, pp. 370–375, (2013).

Calpinator: A Configuration Tool for Building Facades

Andrés F. Barco and **Elise Vareilles** and **Michel Aldanondo** and **Paul Gaborit**[1]

Abstract. Reducing energy consumption of residential and commercial buildings is a major challenge nowadays. One of the strategies to achieve a significant reduction lies on building renovation. On this regard, a project targeting the industrialization of high performance thermal renovation for apartment buildings is been executed. The renovation is based on an external envelop composed by rectangular wood-made panels that cover the whole building. Two concurrent configuration tasks within the project have been identified: The configuration of each one of the panels w.r.t. to the facade and the configuration of the entire facade using a set of these panels. We focus our efforts on the development of a decision support system for the configuration of panels and facades. In this paper we introduce Calpinator, a Java-based configuration tool which is the heart of the decision support system for the project. The tool uses the notion of Constraint Satisfaction Problems as underlying model and implements a smart greedy-recursive algorithm to find a feasible configuration. In this communication we present the tool's design, its features and its implemented algorithm. We use a real-world scenario to illustrate the kind of facades the system can deal with.

1 INTRODUCTION

Energy consumption of residential and commercial buildings is constantly growing and currently it exceeds industry and transport sectors. It represents more than a third of the energy consumption in developed countries: 44% in France[2], 37% in Europe [19], 36% in North America [7] and 31% in Japan [5]. The increase in population, the enthusiasm for new technologies and the improvement of living comfort combined with the domestic habits creates an energy demand of buildings that will continue to increase in the coming years. Therefore, reducing energy consumption of buildings is now a priority in national and international levels.

According to Falcon et al. [8] one of the strategies to achieve a significant reduction lies on thermal building renovation. However, old methods involving by hand configuration, human scheduling and craft assembly, are expensive both in time and costs (bill of materials). It is therefore essential to assist this massive renovation of buildings with decision support systems [13].

Our work is part of project called CRIBA (for its acronym in French of Construction and Renovation in Industrialized Wood Steel) [8]. This project focuses on the industrialization of energetic renovation for residential buildings. The challenge, very ambitious, is to have a building energetic performance under $25kWh/m^2/year$ after the renovation. To do this, the building is completely covered with a new envelope composed of rectangular panels that are prefabricated in factories. The core of our work lies on the two concurrent configuration tasks that have been identified: To configure each one of the panels w.r.t. to the facade and to configure the entire facade using a set of these panels [23, 24]. We focus our efforts on the development of a decision support system for the configuration of panels and facades.

In this paper we introduce Calpinator, a Java-based configuration tool which is the heart of the decision support system for the CRIBA project. The tool uses the notion of Constraint Satisfaction Problems as underlying model and implements a smart greedy-recursive algorithm to find *one* feasible configuration of panels and facades. In this communication we present the tool's design, its features and briefly describe the implemented algorithm. It is worth noting that the algorithm, whose details can be fond in [2], is not part of the contribution of the present work. Instead, we focus our efforts on the implementation of the algorithm.

1.1 Related work

Layout synthesis, also known as space planning, techniques have been used within different contexts and scenarios. For instance, finding solutions for room configurations [25], apartment layouts [15] and activities within a business office [12]. Also, some tools have been implemented using different approaches, here we name a few of them. For example, in [22] Shikder et al. present a prototype for the interactive layout synthesis of apartment buildings including design information and an iterative design process. In [4] is introduced WRIGHT, a constraint-based layout generation system that exploits disjunctions of constraints to manage the possibilities on positioning two-dimensional objects in a two-dimensional space. Another system, LOOS [9], is able to configure spaces using rectangles that can not be overlaped but that may have holes. It uses test rules applied by steps to the rectangles in order to reach a good configuration based on its orientation and relation with other rectangles. The same authors have developed SEED [11]: A system based on LOOS used for early stages on architectural design. A comparison between WRIGHT and LOOS can be found in [10]. The system HeGel [1] (for Heuristic Generation of Layouts) is yet another space planning tool that simulates human design based on experimental cases. Finally, Medjdoub et al. presents in [17] the system ARCHiPLAN which integrates geometrical and topological constraints to apartment layout planning.

2 PROBLEM CONTEXT

In order to achieve the CRIBA project goals and ensure the sealing of the building, each facade of the renovated building must be

[1] Université de Toulouse, Mines d'Albi, Route de Teillet Campus Jarlard, 81013 Albi Cedex 09, France, email: abarcosa@mines-albi.fr

[2] http://www.developpement-durable.gouv.fr/IMG/pdf/Rep_-_chiffres_energie.pdf

completely covered by rectangular configurable panels, i.e., it is necessary a *configuration* of panels to cover the facade. Configuration is the task of designing a given product (here facades) from predefined generic components (here panels) [14, 21]. Components, which are described in terms of its functions, characteristic and prices, are usually arranged in a catalog. Customized solutions, are built from the combination of this catalog components and users requirements and preferences.

In our context, a configuration solution for a facade layout is therefore finding a spatial positioning of panels that covers the whole facade front, without overlapping nor holes. Keep in mind that, whereas components (i.e., panels) in our catalog have well-defined geometric shapes, dimensions and relations, their number is *not known in advance*.

2.1 Layout elements

The following elements are part of the renovation. We include the description of facades because its composing elements are important in the accurate configuration of panels.

- Facades: A facade is represented by a 2D coordinate plane, with origin of coordinates (0,0) as the bottom-left corner of the facade, containing rectangular zones defining:
  - Perimeter of facade to renovate with its dimensions (height and width).
  - Frames (windows and doors) with their dimensions (height and width) positioned in the reference plane.
  - Supporting areas (place to fix panels), with their permissible load, positioned in the reference plane.
  - Zones labeled as "out of configuration" which are areas that can not be covered by configured panels and therefore require specific panels design.
- Rectangular panels (shown in Figure 1): Panels are rectangular, of varying dimensions (from 1 to $45.5m^2$) and may include different equipment (joinery, solar modules, etc.). These panels are designed one at a time, when the definition of the layout configuration has been done, and manufactured in the factory prior to shipment and installation on the building site.

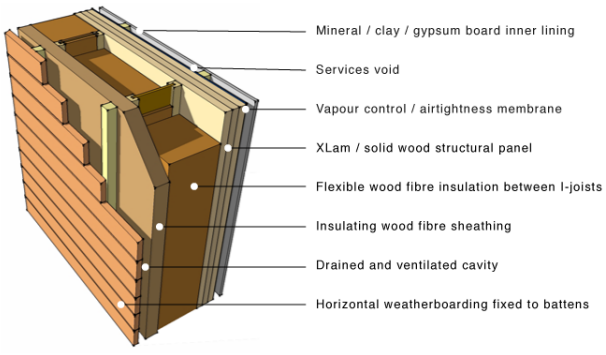


Figure 1. Prefabricated rectangular panels

2.2 Configuration process

The renovation process follows a series of steps going form the building site through the elaboration of panels and ending in its assembly [23]. At each level, a series of descriptive questions are asked to the user. Each answer has a potential impact on the permissible dimensions of panels. For example, the inaccessibility of a given facade may limit the dimensions of panels and therefore the surface covered by each one of them.

Once the descriptions of the site, building and facade are completed, the layout configuration of each facade can begin. Facades must wear a set of panels that must be the greatest as possible while respecting the architectural constraints, supporting areas, manufacturing and accessibility limitations. A rectangular panel is well configured if it meets the following conditions:

C1 It should cover the greatest possible area given the accessibility and the geometric position of frames.
C2 It can be installed in facade and supported by one or more supporting areas.
C3 It does not overlap with any other panel.
C4 It does not block the definition and configuration of the rest of the facade.

2.3 Configuration example

Consider the facade to renovate in literal (a) of Figure 2. The horizontal and vertical lines represent the places in which we are allowed to attach panels, i.e., the supporting areas. They correspond to various possible locations for the fasteners supporting the weight of panels. In this article, we assume that these places are capable of supporting a large enough weight to not constrain the surface of the panels.

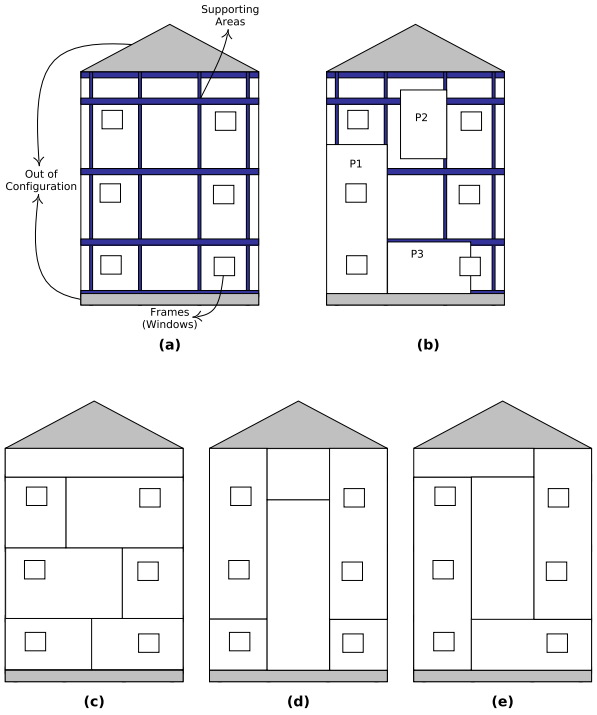


Figure 2. Well and ill-configured facades

Fasteners consist of two parts: One fixed directly onto the facade (wall bracket) and one installed on the panel at the factory. On the facades, the fasteners are positioned in the center of the supporting areas. At the level of the panels, brackets are fixed to the lower edge of the panels at equidistant (from 0.9 to 4 meters) from each other: These minimum and maximum distances allow to better distribute the weight of supported panels. A wall bracket can support a single panel (if it is on the perimeter of the panel) or two panels (if it is at the junction between two consecutive panels).

Small rectangles present on the facade to renovate in Figure 2 literal (a), correspond to the locations of frames (doors and windows).

Two areas of the facade are considered "out of configuration": The gable and the bottom part before the first horizontal supporting area. Two specific panels will be designed, one triangular for the gable and a square one for the specific building foot.

Figure 2 literal (b) presents a facade with three ill-configured panels: Due to the impossibility to place another panel north to the already placed panel $P1$, because there are no supporting areas at the corners of panel $P2$ and because panel $P3$ partially overlaps a frame. None of these configurations are valid. Facades in literals (c), (d) and (e) of Figure 2 present layout configurations where all panels meet the four conditions. From these, the facade (e) is preferred over the other two because it uses less panels.

3 UNDERLYING MODEL

Following the CSP model, we have identified 6 constraint variables, presented in Table 1, that allow us to represent the core of the layout configuration for a given facade: The spatial positioning of panels. Recall that a CSP problem is described in terms of a tuple $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{V}$ is a set of variables, $\mathcal{D}$ is a collection of potential values associated for each variable, also known as domains, and $\mathcal{C}$ is a set of relations over those variables, referred to as constraints [18].

Table 1. 6 variables used in the Calpinator implementation.

Variable	Description	Domain
(p_{x0},p_{y0})	Origin (bottom-left) of panel p	$x0 \in [0, w_{fac}]$, $y0 \in [0, h_{fac}]$
(p_{x1},p_{y1})	End (top-right) of panel p	$x1 \in [0, w_{fac}]$, $y1 \in [0, h_{fac}]$
w_p	Width of panel p	$[0.9, 13.5]$
h_p	Height of panel p	$[0.9, 13.5]$

The algorithm implemented in the tool uses the following parameters to set domains and to link variables: Width of facade (w_{fac}), height of facade (h_{fac}), environmental property (e_{fac}), for each frame f its origin point (f_{x0},f_{y0}) and its end point (f_{x1},f_{y1}) and, a collection of horizontal and vertical supporting areas each one of them with its origin point (sa_{x0},sa_{y0}) and its dimensions (sa_w,sa_h).

In what follows we briefly describe five of the six constraints that are part of the model and that are constraints in the Calpinator tool, more details about the model can be found in [2]. The sixth constraint, dealing with weight restrictions, is not presented because it is not yet included in the implementation.

Environmental The width w_p and height h_p of panels may be constrained because accessibility difficulties to the facade (e.g. trees, water sources, high voltage lines, etc), transportation issues (e.g. only small trucks available) or even climatological aspects (e.g. wind speed more than a given threshold).

Dimension Considering the panels suppliers and panel fabrication specifications, the width w_p and height h_p of each panel is in the range $[0.9, 13.5]$. However, this is actually a combination of values. In other words, it is possible to configure a panel with dimensions 0.9×13.5, 3×8.4 or 13.5×0.9, but it is not possible to configure one with dimensions 13.5×13.5, this is due to fabrication and transportation constraints.

Area A correct facade configuration is one in which the whole facade area is covered by prefabricated panels. Thus, a constraint forcing the sum of panel areas ($\mathrm{w}_p \times \mathrm{h}_p$) to be equal to the facade area ($\mathrm{w}_{fac} \times \mathrm{h}_{fac}$) is needed.

Non-Overlap In addition, we must ensure that the panels do not overlap so we can have a valid configuration. Thus, for each pair of panels p and q we apply the non-overlap constraint (also known as ndiff in different CSP tools).

Panel vs. Frames We adjust the width or height of a given panel if there exists a frame near to it. Either the panel overlaps the frame or the panel is right, left, up or down of the frame. In any case, due to the internal structure of the panel, borders of frames and borders of panels must be separated by a minimum distance given as input.

4 CALPINATOR: A FACADE CONFIGURATOR

Using the aforementioned model, we have developed two algorithms for solving the problem of facades configuration. The first algorithm is an attempt to find one layout configuration in a greedy fashion (more information can be found in [2]). The second algorithm uses global constraints and a constraint engine to find all possible panel configurations for covering the facades (more information can be found in [3]). In the current state of development of our tool, however, only the greedy-recursive algorithm has been implemented (Section 4.2). The constraint-based solution is planned for forthcoming releases of the tool and will, probably, use the constraint solver Choco [20] version 3 as underlying engine.

The result of our work is a Java-based tool that we call Calpinator[3]. It allows the user to input a building specification with an undefined number of facades and throws a solution for each of the facades if there is any. An intuitive view of the process is available by means of a friendly graphical user interface. In this Section we present the internal design of Calpinator, its implemented algorithm, the input and output formats, and the current options for customization.

It is worth mentioning that currently the user is suppose to be an architect, the building owner or a third-party contractor that is in charge of mapping the building data into the appropriated input format. Nevertheless, the goal, in a different stage of the project, is to automate the renovation process in every possible way. Thus, one of the partners in the CRIBA project is working on the automatic generation of the input for the configurator. In essence, they will use drones with pattern and image recognition to obtain most [4] of the facade related information.

4.1 Design

Calpinator has a very basic and modular design. The main characteristic of Calpinator is the implementation of a greedy algorithm for

[3] The name Calpinator is the combination of the French word *calpinage*, which means layout, and the word *configurator*. `https://bitbucket.org/anfelbar/calpinageprototype/wiki/Home`

[4] Some aspects can not be managed by drones. This is the case of the supporting areas maximum load, which is data that is recorded by the building constructors.

finding panels and facades configuration. Besides, we have enhanced the tool with an intuitive graphical user interface and provide a standard storage format (JSON) to allow a transparent communication with other software. Figure 3 presents the internal design of calpinator at first glance.

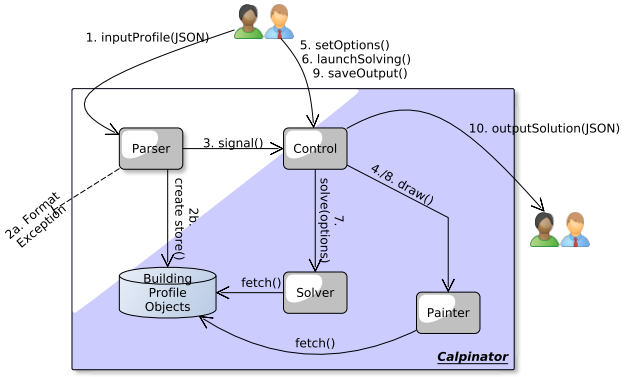


Figure 3. Calpinator internal design.

Let us explain further the execution and interaction between objects in the figure. Initially, the user inputs its building profile specification as a JSON file (Step 1). As expected, if the input file is not well formed, an exception is thrown (Step 2a). Alternatively, the system creates a data base (Step 2b) that stores all objects of the building, i.e., facades, frames, etc. Once the parsing is done, it informs the control it can enable the solving process (Step 3). The first task of the Control (Step 4) is to send the Painter object to draw the facades and its elements. Afterwards, (Step 5) the user may customize the solving process as explained in Section 4.4. If no user-parameters are given, Calpinator uses the default options (see Section 4.4). Next, when the user asks for the solution (Step 6), the Control calls the Solver (Step 7) which executes the greedy-recursive algorithm presented in Section 4.2. If a solution is found, the Control tells the Painter (Step 8), by user's demand, to draw one panel of the solution at a time. Finally, the user may save the solution to another JSON file (Steps 9-10).

Take into account that each time the user opens a new building profile, the data base with the profile objects is re-instantiated. This is done in order to avoid conflicts between elements of different building profiles.

4.2 Algorithm internals

Using the elements description in Section 3, we have developed an algorithm that solves the layout configuration in a greedy fashion [2]. This means that the algorithm makes local decisions for positioning panels following a well-known approach in layout synthesis field called *constructive* [12, 16]. Such decision making process is opposite to previous works where a search space is explored using backtracking search (see [6, 25] for instance). The implemented algorithm exploits recursion, simulating backtracking, when positioning a panel is not possible due to constraint conflicts. In what follows, we present the algorithm which an adaptation of the original one presented by the authors in [2]. The difference between this algorithm and the original one resides in the non-implementation of the weight constraint (postponed for further releases of the tool).

Step 1-: It begins by retrieving an available origin point and finding an end point given the heuristic for panel orientation. At this point, consistent with dimensions upper bounds, the panel is as big as possible.

Step 2-: It proceeds by generating a new valid point by means of solving conflicts between panels and frames. If dimensions of the panel violate dimensions constraints then it fails at positioning the panel.

Step 3-: It checks whether it is possible to install it using an horizontal or vertical supporting areas.

Step 4-: To install the panel, either in an horizontal or a vertical supporting area, it checks if the corners of the panel match supporting areas. This ensures that the panel can be installed as well as panels above it and at its right.

Step 5-: In the case it is not possible given the absence of supporting areas, it reduces the dimensions of the panel until the corners are matched with supporting areas.

Step 6-: Finally, if the panel is well positioned, it proceeds by computing new origin points and adding the next panel recursively.

Step 7-: If the next panel can not be placed, dimensions for current panel are reduced and another check is run. Otherwise we have found a solution so add it to the solution list and return.

4.3 Profiles and solutions

In order to use Calpinator, the user must know how to input the information and how to retrieve solutions. In this section we present the formats used by the tool.

4.3.1 Input

At the current state of development, Calpinator tool receives as input a building description that we call a *profile*. A building profile is, in essence, a table with alphanumeric values describing each of the facades in the building. In order to input this data into the tool, we have adopted a well-known format called JSON which is a composition of entries in the form *key:value*. This decision is attractive given that many formats (such as excel sheets and XML files) can be mapped to JSON files and vice versa. For instance, a simple excel sheet can be easily mapped into a JSON file using the open source program *Mr. Data Converter*[5]. Support for other formats, such as excel sheets and XML files, will be provided in forthcoming versions of the tool.

In order to avoid ambiguity, Calpinator is able to read only a particular set of values stored in a JSON file. The JSON input file for Calpinator is described in what follows.

- **type:** This key represents the type of element described by the entry. Allowed values are: *'facade'* which informs that there is a new facade in the building: *'floor end'* which is an horizontal supporting area: *'cross wall'* which is a vertical supporting area: *'crossing'* which describes the place in which an horizontal and vertical supporting areas meet: *'window'* a new window in the facade: *'door'* a new door in the facade and: *'out'* a zone out of configuration. There can be any number of elements in the building profile. Furthermore, elements do not follow any particular order inside the JSON file.

[5] The program is available online at `http://shancarter.github.io/mr-data-converter/`

- **id:** Each element is associated with an unique alphanumeric value that distinguishes the element from any other.
- **ref:** Each element, except from facades, belongs to another element. The key *'ref'* is an alphanumeric value referring to the element that the current element belongs to.
- **x:** Origin coordinate in x-axis.
- **z:** Origin coordinate in z-axis.
- **width:** Width of the element (in meters).
- **height:** Height of the element (in meters).

It is worth mentioning that Calpinator makes a distinction of all elements in a building profile. To do so, it uses the element identifier and the reference the element belongs to. Simply stated, all elements in a given facade must have different identifiers. However, elements of different facades may have the same identifiers provided they have different references. A given element will be part of the facade referenced by the field *'ref'* regardless the *'id'* value of the element.

Given that most users are used to excel sheets, we present an input example using an excel table and show its corresponding JSON translation. Table 2 presents a building with one facade, one window, one door, one zone out of configuration and three different supporting areas. Table 3 shows the corresponding translation into JSON.

Table 2. Building profile example using excel sheet.

type	id	ref	x	z	width	height
facade	fac1		0	0	18,95	10,64
floor end	1	fac1	0,16	0	18,79	0,16
cross wall	1	fac1	0	0	0,16	10,64
crossing	1	fac1	0	0	0,16	0,16
window	1	fac1	0,92	1,11	1,4	1,3
door	1	fac1	9,69	0,16	0,8	2,25
out	1	fac1	5,88	0	2	2

Table 3. Building profile example using JSON format.

```
[
{'type':'facade', 'id':fac1,  'ref':'',  'x':0,  'z':0,
                        'width':18.95,'height':10.64},
{'type':'floor end','id':1,'ref':'fac1','x':0.16,'z':0,
                        'width':18.79,'height':0.16},
{'type':'cross wall','id':1,'ref':'fac1', 'x':0, 'z':0,
                        'width':0.16,'height':10.64},
{'type':'crossing','id':1,'ref':'fac1',  'x':0,  'z':0,
                        'width':0.16,'height':0.16},
{'type':'window','id':1,'ref':'fac1','x':0.92,'z':1.11,
                        'width':1.4,'height':1.3},
{'type':'door','id':1,'ref':'fac1', 'x':9.69, 'z':0.16,
                        'width':0.8,'height':2.25},
{'type':'out','id':1,  'ref':'fac1',  'x':5.88,  'z':0,
                        'width':2,'height':2}
]
```

Recall that this is the first version of the Calpinator tool and thus the input data is limited to that used by the greedy-recursive algorithm. In consequence, important data as the y-coordinate (for a 3D model), facade adjacency and facade inclination have been currently left out of the configurator's input. Forthcoming developments will take into account these values but will have, necessarily, to be implemented with other versions or algorithms of that presented in Section 4.2.

4.3.2 *Output*

The output of a configuration is another JSON file containing the information of each one of the panels. Additionally, the output contains all information concerning frames inside panels. In short, each frame (e.g., window or door) covered by a panel has a relative position w.r.t. the origin of the panel. This is necessary for the fabrication of the panel. i.e., each panel must be fabricated with the corresponding holes for frames. Thus, for each panel or frame the output specify: **type:** Type of element(*'panel'* or *'frame'*), **id:** Panel or frame identifier, **ref:** Facade id or panel id that the element belongs to, **x:** Origin x-coordinate (relative to facade origin or the panel origin), **z:** Origin z-coordinate (relative to facade origin or the panel origin), **width:** Width of the element, **height:** Height of the element.

4.3.3 *Facades with no solution*

Calpinator tool allows for any kind of facade to be used as input. Nonetheless, it is not the case that any facade has a valid configuration given the constraints in our model or given the user preferences. For instance, literal (a) in Figure 4 does no have supporting areas in necessary places (no supporting areas at meter 15). Or perhaps, a given facade has no possible configuration because there is not enough distance between frames and supporting areas which is the case of literal (b) in Figure 4. Lastly, a facade may not be configured with Calpinator because an ill definition of zones out of configuration, as presented in literal (c) of Figure 4: No supporting areas at the top of the zone. As a workaround, the user should extend the zone out of configuration until the next horizontal supporting area. In the figure, the doted square shows the result of extending the zone.

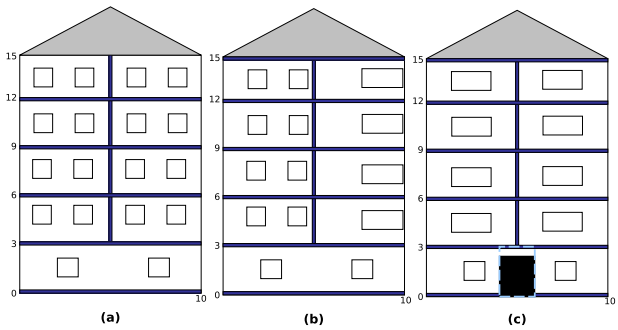


Figure 4. Three facades with no solution.

4.4 Parameterization

In its current state, our configurator is customizable in two ways. On the first hand, the user may choose an heuristic that defines a preference in the orientation of panels. On the other hand, the user may change the lower and/or upper bound for panel dimensions. As a consequence of such parameterization, the tool finds different solutions for the same facade. Nevertheless, as the implemented algorithm is deterministic, any given customization will result in the same configuration for a given input.

4.4.1 *Orientation heuristic*

When we talk about orientation we refer to relation between width and height which have an impact on the internal structure of the panel. In essence, if the width of the panel is bigger than its height, we consider the panel as *horizontally oriented*. Conversely, if the

panel height is bigger than its width, we consider it as *vertically oriented*. The user, for instance, may prefer to use horizontal panels in its facade. Calpinator will try then to put each panel horizontally, i.e., $w_p \in [0.9, 13.5] \wedge h_p \in [0.9, 3.5]$ (see the constraint *Dimensions* in Section 3). If a given panel can not be placed in the preferred orientation due to constraints conflicts, calpinator tries to place it using the other orientation. At the model level we consider the heuristic as a soft constraint, i.e., it can be violated without causing failure. This is why we do not include soft constraints in the core of our model.

4.4.2 *Dimensions range*

Recall that given the environmental aspects of the facades, the dimensions for panels may be reduced to a given interval. In addition, the user may, optionally, further constrain the dimensions for all panels in the facade according to its preferences. This is done by changing the lower and upper bound of the panel dimensions. As expected, the tool will respect the consistency between environmental constraints and the user preference. For instance, if the environmental properties constrain the width of a panel to be in the interval $[0.9, 8]$ and the user preferred upper bound is 9.5, the tool will set the upper bound in 8. This is due to the monotonic properties of CSPs. For this customization the tool presents three options:

- **Manually:** The user may change either the lower bound, the upper bound or both values.
- **Random:** The system chooses a random value for the upper bound. This constraints only one dimensions, the width for horizontal orientation and the height for vertical orientation. Note that the random strategy is applied for each panel in the facade. Thus, it is likely that most of the panels have different dimensions. This is interesting because, on the one hand, each time the user runs the algorithm it will find a different configuration of panels. On the other hand, it is more likely that the algorithm finds a valid configuration because it will try new values until exhaustion.
- **Square:** Try square panels only, i.e., constraints the upper both of vertical and horizontal orientation to be in the range of $[0.9, 3.5]$

Keep in mind that a given facade may have no configuration solution given its properties. Thus, constraining dimensions may reduce the number of chances to find one feasible facade configuration.

5 USING CALPINATOR

In this section we present a brief description of how Calpinator works in practice using some examples in real-world scenarios. As Calpinator is implemented in Java, the user needs to count with an updated version of the Java Virtual Machine. In addition, several dependencies are necessary in order to run the application. The libraries[6] used by the tool are *Oracle Commons* libraries (beanutils, collections, io, lang and logging) and *Maven* libraries (ezmorph and json-lib).

After launching the application, the user opens a JSON file specifying a building profile with any number of facades and elements (see Section 4.3.1). Then, all facades inside the building profiles are shown in the application, each facade in one tab. For instance, a building with two facades will be visualized as presented in the Initial State of Figure 5.

[6] For simplicity, these libraries are included in the distribution of Calpinator. Recall that these libraries are free software but each may have its own License agreement. Calpinator is distributed under General Public License version 3 and can be fount at `https://bitbucket.org/anfelbar/calpinageprototype/wiki/Home`

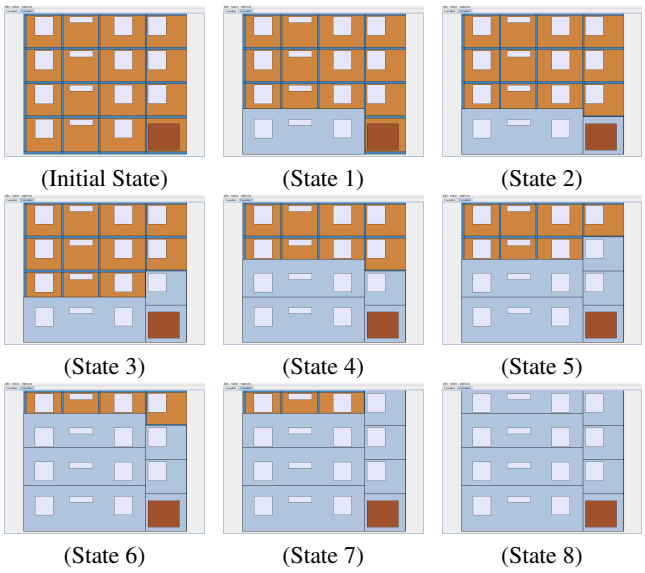


Figure 5. View of the configuration evolution.

Next, a customization may be done by changing the panels dimensions and choosing an heuristic as explained in Section 4.4. Afterwards, selecting the `solve` entry in the menu bar, the tool will try to find one feasible configuration for the facade in the current selected tab. For instance, Figure 5 presents a configuration solution for a facade with w_{fac} equals 12.59 meters and height equals 10.907. The customization for this facade is horizontal panels with maximum width of 13.5 meters for each panel. Each of the states in the figure presents different views reached by making left click on the canvas of Calpinator. Additionally, if the user wants to go back and see a partial configuration he may do so by using the right click on the canvas. Ultimately, the tool allows to save the configuration solutions by choosing `save` in the menu bar. Note that only those solved facades will be saved in the output. Given that this is work in progress and that the greedy algorithm is a deterministic one, the tool will only find one solution (if there exist) that satisfies the four conditions presented in Section 2.2. In consequence, the potentially many solutions for the facade layout are not found by Calpinator and thus no heuristic or criteria for choosing the best one is necessary. Ongoing investigation is looking into the possibility of finding different solutions by combining the greedy approach and search trees.

5.1 Examples

In this section we present some examples with different panel orientation and panel dimensions. The illustrated facades are part of the working site La Pince in the commune Saint Paul-lès-Dax in the department of Landes, France. Each of the columns of Figure 6 presents one facade of La Pince. The original facades, i.e., its frames, doors and supporting areas, are presented in literals (1a) and (2a).

Literals (1b) and (1c) in Figure 6, for the facade on the left, show configurations thrown by Calpinator using horizontal panels, with 3 meters as width upper bound for literal (1b) and 9.5 meters for literal (1c). Next, in literal (1d) and (1e) we present the configurations of the same facade using vertical orientation, with 6 meters as height upper bound for literal (1d) and 13.5 meters for literal (1e).

Conversely, the right column of Figure 6 presents some configuration configurations for the facade in literal (2a). The first two config-

Figure 6. La Pince facade 1 (right) and facade 2 (left).

urations present an horizontal orientation of panels and width upper bound of 8 and 13.5 meters for literals (2b) and (2c), respectively. Finally, in literals (2d) and (2e) of Figure 6 we present the configurations with vertical panels and height upper bound of 8 meters and 13.5 meters, respectively.

6 CONCLUDING REMARKS

Controlling energy consumption in buildings is one of the major challenges of the 21^th century. Reducing energy consumption in buildings is now focused on the renovation of existing buildings. To achieve renovation goals set by the French Government in 2009 and 2013, it is essential to assist massive renovation with technological tools and industrial methods rather than artisanal ones.

We presented in this paper a tool dedicated to the definition of layout configuration for building facades. The novelty of the tool lies on the implementation of a greedy-recursive algorithm that takes into account the many constraints inherited by facades in order to find a feasible configuration of panels. This work falls under the project CRIBA which aims to industrialize the renovation from the outside of buildings of residential housing in order to achieve an energy performance close to $25kWh/m^2/year$.

We have presented our first problem of layout configuration describing the specifics details related to the insulation of facades outside. In a second step, we have brefly described the knowledge model supporting this configuration problem based on constraints. The set of constraints was formalized by CSP in [2]. These formalize both manufacturing constraints and transportation, but also constraints relating to the geometry and structure of building and the internal structure of rectangular panels. The first version of the layout configuration tool incorporating all of these constraints is then presented and illustrated on an example from the pilot project site. The solutions proposed by our algorithm are all consistent with the constraints of the layout problem.

However, not the algorithm nor the tool take into account aesthetics preferences of users (e.g. architects' preferences). To avoid the generation of non-compliant solutions, additional "business" knowledge should be added to the (constraint) knowledge model. They are mainly related to the building after aesthetic renovation, such as an alignment constraint of connection joints between panels.

6.1 Future work

We acknowledge that our work is still in its infancy. Different efforts in crucial aspects will improve results in the model, algorithms and the tool. On this regard, the following objectives are strategic directions within the project.

a. Implement the constraint-based algorithm introduced in [3] is a priority. The algorithm is conceived to throw all possible panel configurations for the facade. This goal includes finding a constraint solver with appropriated filtering and search capabilities.
b. Improve greedy-algorithm with pre-processing and post-processing capabilities. Intuitively, a human configuration takes advantages of the facade dimensions and positions of frames to find a solution. Thus, it is adequated to add new constraints consequence of previous structural analysis of the facade.
c. Add more variables, hence constraints, to the model and improve or create new algorithms. For instance, there exists a constraint for fasteners and panel's edges distances which is important for the panel's stability. Also, there are some constraints over inclination of the facade, or the building itself, and panels positions. These and other relations will increase both the detail and the complexity of the problem, but are mandatory steps for the industrialization of the renovation.
d. Implement in Calpinator tool the weight constraint. The weight constraint to be implemented involves a new constraint variable, fai_{load}: Maximum weight load of fastener which is in the range of $[0, 500]$ kilograms. The constraint is is defined as follows.

   Weight Constraint A given fastener in a supporting area is defined by its coordinates and its maximum weight load. Let ATP_i be the panels attached to the fastener fa_i and let

$computeWeight(p)$ be a function[7] that returns the weight of panel p. Constraint over panels weight is defined by

$$\sum_{j=1}^{|ATP_i|} computeWeight(ATP_i[j]) \leq fai_{load}$$

This constraint is not implemented yet because we have not extracted and validated knowledge on how to distribute the panel's weight in the supporting areas. Up-to-now, we know that half of the panel's weight have an impact on a supporting area if there is only one fastener interacting between the panel and the supporting area. Otherwise all the panel's weight will be supported in area. Figure 7 shows some examples of this knowledge.

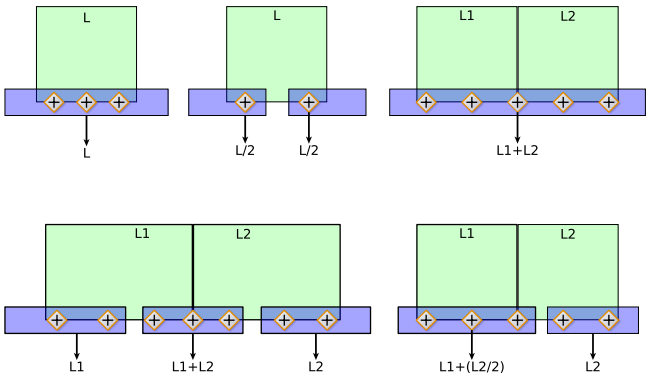


Figure 7. Distribution of weight in supporting areas.

e. Finally, a big challenge is to model and implement the concurrent renovation of multiple-adjacent facades. This particular scenario introduce different problems. Consider, for instance, a vertical supporting area at the right edge of a facade which is, in fact, the first vertical supporting area in the next facade. A given configuration has to take into account the weight in both facades over the same supporting area. Another issue is the angle between two adjacent facades and its implications for the width of panels.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the TBC Générateur d'Innovation company, the Millet and SyBois companies and all partners in the CRIBA project, for their contributions to the CSP model. Special thanks to the referees for their comments and to Philippe Chantry from École des Mines d'Albi for his contribution to the tool's GUI and some graphics in the paper.

REFERENCES

[1] Ö. Akin, B. Dave, and S. Pithavadian, 'Heuristic generation of layouts (hegel): based on a paradigm for problem structuring', *Environment and Planning B: Planning and Design*, **19**(1), pp. 33 – 59, (1992).

[2] A. F. Barco, E. Vareilles, M. Aldanondo, and P. Gaborit, 'A recursive algorithm for building renovation in smart cities', in *21st International Symposium on Methodologies for Intelligent Systems. To appear.* Springer-Verlag, (June 2014).

[7] This function uses the next values to calculate the weight of a panel: dimensions of the panel, insulation type of the panel, weight of the frames within the panel (if any) and weight of any other component (e.g. solar modules).

[3] A. F. Barco, E. Vareilles, M. Aldanondo, P. Gaborit, and M. Falcon, 'Constraint-based decision support system: Designing and manufacturing building facades', in *Join Conference on Mechanical, Design Engineering and Advanced Manufacturing. To appear.* Springer-Verlag, (June 2014).

[4] Can A. Baykan and Mark S. Fox, 'Artificial intelligence in engineering design (volume i)', chapter WRIGHT: A Constraint Based Spatial Layout System, 395–432, Academic Press Professional, Inc., San Diego, CA, USA, (1992).

[5] The Energy Conservation Center, *Energy Conservation Handbook*, The Energy Conservation Center, Japan, 2011.

[6] P. Charman. Solving space planning problems using constraint technology, 1993.

[7] U.S. Green Building Council, *New Construction Reference Guide*, 2013.

[8] M. Falcon and F. Fontanili, 'Process modelling of industrialized thermal renovation of apartment buildings', *eWork and eBusiness in Architecture, Engineering and Construction*, 363–368, (2010).

[9] U. Flemming, 'Knowledge representation and acquisition in the LOOS system', *Building and Environment*, **25**(3), 209 – 219, (1990).

[10] U. Flemming, C.A. Baykan, R.F. Coyne, and M.S. Fox, 'Hierarchical generate-and-test vs constraint-directed search', in *Artificial Intelligence in Design '92*, eds., J.S. Gero and Fay Sudweeks, 817–838, Springer Netherlands, (1992).

[11] U. Flemming and R. Woodbury, 'Software environment to support early phases in building design (seed): Overview', *Journal of Architectural Engineering*, **1**(4), 147–152, (1995).

[12] M. M. D. Hassan, G. L. Hogg, and D. R. Smith, 'Shape: A construction algorithm for area placement evaluation', *International Journal of Production Research*, **24**(5), pp. 1283–1295, (1986).

[13] Y. Juan, P. Gao, and J. Wang, 'A hybrid decision support system for sustainable office building renovation and energy performance improvement', *Energy and Buildings*, **42**(3), 290 – 297, (2010).

[14] U. Junker, *Configuration.*, Chapter 24 of Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York, NY, USA, 2006.

[15] K.J. Lee, H.W. Kim, J.K. Lee, and T.H. Kim, 'Case-and constraint-based project planning for apartment construction.', *AI Magazine*, **19**(1), pp. 13–24, (1998).

[16] R. S. Liggett, 'Automated facilities layout: past, present and future', *Automation in Construction*, **9**(2), pp. 197 – 215, (2000).

[17] B. Medjdoub and B. Yannou, 'Separating topology and geometry in space planning', *Computer-Aided Design*, **32**(1), 39 – 61, (2000).

[18] U. Montanari, 'Networks of constraints: Fundamental properties and applications to picture processing', *Information Sciences*, **7**(0), 95 – 132, (1974).

[19] L. Pérez-Lombard, J. Ortiz, and C. Pout, 'A review on buildings energy consumption information', *Energy and Buildings*, **40**(3), 394 – 398, (2008).

[20] C. Prud'homme and JG. Fages, 'An introduction to choco 3.0 an open source java constraint programming library', in *CP Solvers: Modeling, Applications, Integration, and Standardization. International workshop.*, Uppsala Sweden, (2013).

[21] D. Sabin and R. Weigel, 'Product configuration frameworks-a survey', *IEEE Intelligent Systems*, **13**(4), 42–49, (July 1998).

[22] S Shikder, A Price, and M Mourshed, 'Interactive constraint-based space layout planning', *W070-Special Track 18th CIB World Building Congress May 2010 Salford, United Kingdom*, 112, (2010).

[23] E. Vareilles, A. F. Barco, M. Falcon, M. Aldanondo, and P. Gaborit, 'Configuration of high performance apartment buildings renovation: a constraint based approach', in *Conference of Industrial Engineering and Engineering Management (IEEM)*. IEEE., (2013).

[24] E. Vareilles, C. Thuesen, M. Falcon, and M. Aldanondo, 'Interactive configuration of high performance renovation of apartment buildings by the use of csp', in *15th International Configuration Workshop*, pp. pp. 29 – 34. CEUR Workshop Proceedings, (aug 2013).

[25] M. Zawidzki, K. Tateyama, and I. Nishikawa, 'The constraints satisfaction problem approach in the design of an architectural functional layout', *Engineering Optimization*, **43**(9), pp. 943–966, (2011).

Towards More Flexible Configuration Systems: Enabling Product Managers to Implement Configuration Logic

Klaus Pilsl and **Martin Enzelsberger** and **Patrick Ecker**[1]

Abstract. Developing a configurator requires a deep understanding of the configurable product. The configuration logic must encompass the way product components may be combined and customized, as well as how the integrity of a configuration can be verified. When products evolve over time, the configurator must be adapted accordingly. Product Managers are intimately familiar with the features and capabilities of a product and drive its development. By enabling them to specify the configuration logic of a product, the time required to introduce new products or respond to changes in existing ones can be reduced significantly. In order to achieve this, an environment must be provided that facilitates the implementation of configuration logic in an efficient and intuitive manner. In this paper we try to identify the key aspects of such an environment and present our experiences in realizing a product configuration system based on our findings.

1 INTRODUCTION

Creating and maintaining a product configurator is usually a complex task [1]. The configurator database report 2014 [2] said, that 14% of the 900 configurators running 2013 disappeared 2014. Keeping a configurator running and up to date is often more time consuming than expected.

For reducing the creation and maintenance expenses for configurators, it is the target to give product managers the tools to create configurators themselves.

This article focuses on the very practical problems product managers face in building product configurators. Further on we identify key elements of an optimal configuration system environment. This leads to new approaches in the way the data is being entered, calculated and presented through the web. Notable findings will be presented in the last section.

2 DRAWBACKS OF COMMON DATA STRUCTURES USED IN CONFIGURATION SYSTEMS

As described in Ecker [3] and Šormaz [4] there are different ways of how the product data can be provided:

1. Relational form
2. Code / macros / scripts
3. Object oriented form
4. Mixture

Due to the penetration of relational database management systems [5] and the software solutions the configurators are built with (i.e. ERP-Systems) most configurator system use that relational data structure.

Each of the above listed data structure has its own problems and drawbacks. The following paragraphs will highlight those.

2.1 Relational form

Relational data structures need to image the product data in tables. These tables follow either a predefined [6] or a user generated schema. It is obvious, that product managers have more flexibility if they can define their own schema, which is necessary for certain kinds of products [7][8], but it also demands a higher skill level from the product managers, which most of them do not have.

But even in predefined schemata the problem remains, that the data of the product need to be squeezed in this predefined form. The challenge here is, to find a way to maximize functionality and readability which are adversary.

Our experience has shown that the majority of the product managers quickly lose sight on the complexity of the data structures they develop.

Problems:

- The product managers are forced to establish an extra documentation layer to keep the system manageable.
- The complexity especially for rule driven visualizations outruns many product managers' capabilities.
- The chronology the system has to calculate the rule needs to be defined by the product manager. This is discovered as a major weak point in terms of error and debugging expenses.

2.2 Code / macros / scripts

Code allows the product managers to transcript even most complex rules. The product manager does not have to follow a predefined schema at all.

Problems:

- The period of vocational adjustment is very high.

[1] IndiValue GmbH, Sarleinsbach, Austria, email: klaus.pilsl@combeenation.com, martin.enzelsberger@combeenation.com, patrick.ecker@combeenation.com

- This informal freedom, however, quickly leads to a lack of lucidity, which makes this form of product data unpopular for product managers who are not software developers.
- It demands a high skill level in software development from the product manager
- Developing a configurator by coding is time consuming and expensive
- Future adapting and enhancements are also expensive and time consuming

2.3 Object oriented form

Just like with the relational form there is a schema, but here they are called classes. The product manager can define an individual classes for every product. The main difference to the relational form is that the rules create dependencies, not the entities of the database.

That gives the product manager the ability to freely transcript the product rules into the system without the need of creating a database structure or a schema fitting his requirements first.

Problems:

- The period of vocational adjustment is moderate.
- This form demands a very sophisticated user interface to guide the user properly.

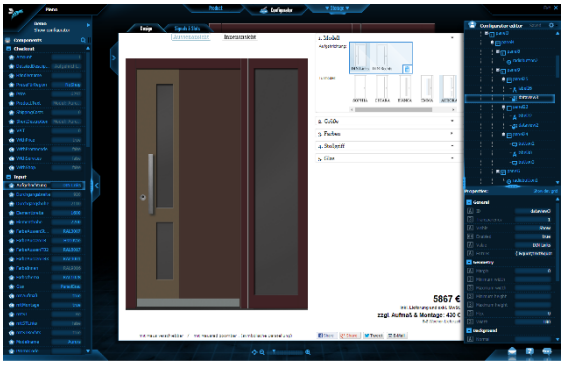
The advantages of this approach are:

- Because of the fact, that the class is individual for each product, the system can create it, while the product manager models the product.
- Classes can also be derived (inherited). So the product manager can easily create variations of the product data.

3 KEY ASPECTS OF THE OPTIMAL ENVIRONMENT

There are several key aspects which define an optimal product configuration environment which support the product manager in building the product configurator.

As a result of our experience and findings we created a configurator management system which fulfills the requirements described in this paper. This system is call Combeenation[9].



The Configurator Management System *Combeenation* from IndiValue GmbH. Showing the user interface designer of a front door configurator.

3.1 Instant feedback

The process of creating product data is usually defined by the following steps:

- Modeling the structure and rules
- Saving and compiling
- Testing

This iteration is continuously repeated, until the product data is finished for publishing. The described process is called progressive evaluation [10].

Progressive evaluation can be very time consuming. Especially testing (opening a test configuration, navigating through the configurator to the point of interest, collecting the test data, closing the test configuration) is very expensive.

If a system would save the data instantly give instant feedback even into every displayed data affected by the changes including an always open test configurator, the time needed for modelling product data would greatly be reduced.

Leitner et al. [11] state that testing and adapting the layout of the configurator interface plays a major role in developing a suitable user interface for configurators. Rapid prototyping processes can be implemented with Combeenation, an innovative configuration environment that supports application development on a graphical level and enables immediate user testing.

3.2 Simple user interface structure

Concerning the user interfaces of the configurator management system many user interfaces are IT oriented. That means the interface is built on the necessities of the system, the data structure and the underlying technology that runs the configurator system.

As described in Ko et. al. [12] most of the product managers are experts in the field of their product, but aren't very practiced in the use of integrated development environments (IDEs) or other user interfaces which are mainly designed for software developers.

The key aspect in this area is to create a user interface for a configurator management system that focuses on the product itself instead of the technical system. The product in its actual visual appearance should always be visible to give instant feedback of the changes made. The components, the properties, the rules and the controls the product manager has to work with needs to be presented in a continuous and integrated way, regardless how these items are used for. This reduces the times needed to jump between screens, menus or pages.

The user interface of a configurator plays a key role for both, consumers and product managers. Leitner et al. [11] identified five key principles for developing user interfaces for configurators which are suitable for both types of users.

- Customize the customization process: Adaption of the user interface depending on the type of customer.
- Provide starting points: Initial design from which the customer can continue the configuration process.
- Support Incremental Refinement: Tradeoff analysis (i.e. product comparison functionalities).

- Exploit prototypes to avoid surprises: Development and Teach the customer: Increasing the user's knowledge about product properties.

Besides these key principles, the arrangement of user interface elements of the configurator as well as the kind of process navigation (i.e. handling, ease of use, guidance through the configuration process) affects the customer's satisfaction with the configurator [13].

3.3 Separation of data and rules

In mass production industries it is common to integrate values (sizes, angles, weights, etc.) into the structure data (i.e. CAD systems). That is okay as long as the products do not change much after they are released to market.

In mass customization [15], however, the product and with it its values change continuously using product configurators. It is also often needed to start configurations of a specific product via different starting values (several presets for the same product).

These requirements can be met by separating the configurable values from the rules and structure data. This way different sets of values can be easily combined with different versions of rules and structure data.

How difficult/easy it is to apply a small change in an established structure with/without this separation will be defined by the viscosity of the system [15], which describes the flexibility of the system.

To supply such a flexible system it is important to make this separation. That also has to be taken care of in the user interface, so that the product manager knows, which parts (values vs. rules) of the product are stored where.

3.4 Separation of product design and UI design environments

We experienced, that product managers working with configurator management tools, which do not separate the user interface designer from the product designer, struggle with data duplication.

Some applications need to be displayed on different devices (desktop, tablets, smartphones), or on different channels (websites, apps, social media channels …), or some of them need to be refurbished to meet new requirements.

In order to create several different user interfaces for the same product, they have to also duplicate the product data. That leads to more data, which greatly increases the workload of the product manager, if this product data needs to be modified.

By separating the product design from the user interface design environment the product manager can address these requirements.

4 IMPLEMENTATION OF A PRODUCT CONFIGURATOR SYSTEM

To meet the needs described we developed a system for creating product configurators. In order to make the use of the system for the product managers as easy as possible, we have chosen to use these standards and technologies:

- HTML5: For maximum acceptance and further developments we have chosen to use HTML5 for presentation and communication. So the system is supported by every modern Browser without the need to install plugins or downloaded software the product manager would have to install first.
- Quick start: To allow new product managers an easy and quick start into the system, we decided to implement the solution as Software-as-a-Service (SaaS) and hosted it in the Cloud (in this case Microsoft Azure [3])
- Another aspect of SaaS is, that the product manager does not need to invest administration and maintenance work to run the system.

4.1 Graphical product representation

Visuals are crucial and the biggest part of the mission. Most users valuate product primarily on its visual appearance. This applies for consumers just as for product designers. We put a lot of effort into providing a flexible system for the visual representation of the product.

Changes in the visual will be applied in real-time. Instead of rasterized graphic formats we use scalable vector graphics (SVG) for a better image quality. The visual editor is interactive and you get what you see (WYSIWYG). After the configuration process is ended by a user each scalable vector graphic can be converted to a PNG or PDF file, making it easy for further processing or printing.

The usage of SVG also allows us the enable custom fonts, gradients, patterns, mask, filters and many other effects. For more information about SVG see [16].

4.2 Quick response times

Our system is built with low response times in mind, since researches have shown that higher latency times have a negative influence on the user acceptance, no matter if that is for consumers or product managers [17].

We experienced that latency times need to stay below 250ms otherwise most users interact with the same UI element again (e.g. clicking a button).

This speed must be achieved with any representation of the changes: simple values, results of complex chained calculations, visuals, and so on.

4.3 Calculation on server and client

To achieve quick response times, it is necessary to split the calculations on the server and client side. The product manager, however, must not be confused. It always must be clear on where the calculation will be done, because there are advantages and disadvantages with either method:

The advantages of server side computations [18]:

- Big data necessary for a calculation does not have to be transmitted to the client
- Product rules are knowhow of the company which needs to be protected. Some product managers don't want to transmit this knowledge to the browser of the client. Our system won't transmit the company's knowledge to the

client at any time. Just the results are transmitted and presented to the client.

- Virtual machines, which are hosted in a datacenter, offers far more computation power than a client device.
- Progress is always saved. If a user catches up later he may continue where he left the configuration (also on other devices).
- It would be possible that more users collaborate on the same product.

Disadvantage of server side computing:

- The system needs to be designed to scale up on demand. This can be done through distributed computing with automatically adding virtual servers if needed. This is especially difficult to implement if the machines work with stateful sessions.

Advantages of client side computing:

- Quick responses are possible
- No network communication needed

Disadvantage of client side computing:

- Only simple calculation should be done, because transmitting big raw data into the client browser can be ineffective

To take a good mix of both advantages it is possible let the system operates on server side calculation for the product design rules and on client side calculation for the UI design rules.

This way calculations of the product itself are done on the server and calculations concerning the user interface (i.e. jumping to a certain page based in input data) are done on the client.

The separation of product design and UI design environments (see 3.4) allows the system to intuitively distinguish between both methods.

4.4 One solution

Most systems on the market cover one part of the mission. The other parts are done by other software tools which needs to be connected via interfaces.

A common thing for instance is, to create a product configurator by using an ERP-software [19] and linking a CAD-software [20] with it to create the visuals.

That leads to this constellations: Both systems hold their own product data. Many parts of these 2 data sets need to be redundant on both systems. And there is a third set of data: the interface itself hold data, too.

Product managers struggle with the big amount of human resources needed to keep these data sets up to date.

It is less maintenance works if all the modules needed for product configuration are handled by one system [18] with one data set.

All the modules of this single system run on this one set of data, and any module of that system can directly and without conversion access the data needed to fulfil its function. Combeenation provides such a system.

5 CONCLUSION

We have highlighted the problems and drawbacks of current systems, which product managers use if they want to build a product configurator. Further on we emphasized key aspects which are needed for such a solution, including a responsive and easy to understand user interface and a strict separation of data and rules.

With these findings in mind we built the configurator management system Combeenation [9], which addresses all those problems and provide an all-in-one solution. This solution is trimmed to rapidity, easy to use, flexible and is highly scalable. First client projects are currently implemented with Combeenation and all usability and performance issues in these projects are monitored to provide further data for potential improvements.

REFERENCES

[1] T.A. Rogoll and F.T. Piller, *Konfigurationssysteme für Mass Customization und Variantenproduktion*, ThinkConsult. 2003.

[2] Blazek, P., Partl, M., Streichsbier, C. (2014): *Configurator Database Report 2014*, Vienna

[3] P. Ecker, *Sicherheitsaspekte bei der Entwicklung einer Software-as-a-Service-Lösung mit Windows Azure*, University of Applied Sciences Upper Austria, Hagenberg, 2012.

[4] D.n. Šormaz, *Distributed Agent-Based Integrative Model For Mass Customization Product Development*, Ohio University, Department of Industrial and Systems Engineering, 2010.

[5] C.M. Ricardo, *Databases Illuminated*, Jones & Bartlett Publ., 2011.

[6] *ISS+* from *MoveIT GmbH*, *Product Builder* in *Microsoft Dynamics NAV*

[7] Front Door Configurator from *TOPIC GmbH*, Austria

[8] Window Configurator from *IFN Internorm AG*, Austria

[9] www.combeenation.com

[10] T. Green and A. Blackwell, *Cognitive Dimensions of Information Artefacts: a tutorial*, 1998.

[11] G. Leitner, A. Felfernig, P. Blazek, F. Reinfrank, G. Ninaus, *User Interfaces for Configuration Environments*. In: A. Felfernig, L. Hotz, C. Bagley, J. Tiihonen (eds.), Knowledge-based Configuration: From research to business cases, 2014.

[12] J.R. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, S. Wiedenbeck, *The state of the art in end-user software engineering*, ACM Computing Surveys, 1-44, 2011.

[13] P. Blazek and K. Pilsl, *The Impact of the Arrangement of User Interface Elements on Customer Satisfaction in the Configuration Process*, In: T. D. Brunoe et al. (eds.), Proceedings of the 7th World Conference on Masss Customization, Personalization, and Co-Creation (MCPC 2014), Aalbor, Denmark, 2014.

[14] F.T. Piller, *Mass Customization - Ein wettbewerbsstrategisches Konzept im Informationszeitalter*, Deutscher Universitäts-Verlag, 2006.

[15] T. Green and M. Petre, *Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework* Journal of Visual Languages & Computing, 131-174, 1996.

[16] J. Ferraiolo, *Scalable Vector Graphics (SVG) 1.0 Specification*, http://www.w3.org/TR/2001/REC-SVG-20010904/REC-SVG-20010904.pdf, September 2001

[17] F.H. Nah, *A study on tolerable waiting time: how long are Web users willing to wait?* College of Business Administration, University of Nebraska-Lincoln, 2004.

[18] M. Enzelsberger, *Entwicklung von Hive, einer domänenspezifischen Sprache zur Spezifikation von Produktlogik durch Nicht-Programmierer*, University of Applied Sciences Upper Austria, Hagenberg, 2012.

[19] *Internet Pricing and Configurator* from SAP AG (*SAP IPC*)

[20] *Inventor* from Autodesk, *SolidWorks* from Dassault Systems, etc.

ReMax – A MaxSAT aided Product (Re-)Configurator

Rouven Walter and **Wolfgang Küchlin**[1]

Abstract. We introduce a product configurator with the ability of optimal re-configuration built on MaxSAT as the background engine. A product configurator supported by a SAT solver can provide an answer at any time about which components are selectable and which are not. But if a user wants to select a component which has already been disabled, a purely SAT based configurator does not support a guided re-configuration process. With MaxSAT we can compute the minimal number of changes of component selections to enable the desired component again. We implemented a product configurator — called ReMax — using state-of-the-art MaxSAT algorithms. Besides the demonstration of handmade examples, we also evaluate the performance of our configurator on problem instances based on real configuration data of the automotive industry.

1 Introduction

Using Propositional Logic encodings and SAT solving techniques to answer the question whether a formula is satisfiable or not has a wide range of applications [10]. The application of SAT solving for verification of automotive product documentation for inconsistencies, e.g. within the bill-of-materials, has been pioneered by Küchlin and Sinz [7].

In [16] we considered applications of MaxSAT in automotive configuration. We mentioned the possible usage of re-configuration with MaxSAT to make an invalid configuration valid again by keeping the maximal number of the customer selections. Re-configuration is of highly practical relevance [9]. For example, the after-sales business in the automotive industry wants to extend, replace or remove components with minimal effort while keeping the configuration valid.

In this paper, we extend this idea by considering product configuration in general. We focus on product configuration based on families of options, because this is the normal case when a user configures a product. Within a family of options, the user must select exactly one option out of a *regular family* or else may select at most one option out of an *optional family*. With the focus on families, we can distinguish two solving approaches:

1. **SAT Solving**: With a SAT aided product configurator, we can validate a configuration after each step of the configuration process.
2. **MaxSAT Solving**: With a MaxSAT aided product configurator, we can compute an optimal solution for an invalid configuration, such that a user has to make a minimal number of changes in the current configuration to regain validity.

We identify different use cases. We describe them in detail and make remarks about extensions or variants of them. We also show how a user process can look like using a MaxSAT aided product configurator.

This paper is organized as follows. Section 2 introduces all relevant mathematical definitions and notations needed for the later sections. In Section 3 we describe the basic concepts of SAT-based product configuration. Section 4 shows use cases of SAT aided product configuration. After that we describe use cases for MaxSAT aided product configuration in detail in Section 5 and illustrate a possible configuration process. Sections 6 and 7 describe the techniques we used for our implementation and experimental results with benchmarks based on industrial configuration instances. Section 8 describes related work and finally, Section 9 concludes this paper.

2 Preliminaries

We consider propositional formulas with the standard logical operators $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ over the set of Boolean variables X and with the constants $\bot$ and $\top$, representing false and true, respectively. Let $\mathrm{vars}(\varphi)$ be the set of variables of a formula φ. We call a formula φ *satisfiable*, if there exists an *assignment*, a mapping from the set of Boolean variables X to $\{0, 1\}$, under which the formula φ evaluates to 1. The evaluation procedure is assumed to be the standard evaluation for propositional formulas. The Boolean values 0 and 1 are also referred to as *false* and *true*. If no such assignment exists, we say the formula is *unsatisfiable*. The question whether a propositional formula is satisfiable or not is well-known as the *satisfiability (SAT) problem*, which is NP-complete.

In most cases a SAT solver accepts only formulas in *conjunctive normal form* (CNF). A formula in CNF is a conjunction of *clauses*, where a clause is a disjunction of *literals* (variables or negated variables). Let $\mathrm{var}(l)$ be the variable of a literal l.

If a formula $\varphi = \bigwedge_{i=1}^{k} \bigvee_{j=1}^{m_i} l_{i,j}$ in CNF is unsatisfiable, we can ask the question about the maximal number of clauses that can be satisfied at the same time. This optimization variant of the SAT problem is called *maximum satisfiability (MaxSAT) problem*. The corresponding question about the minimal number of unsatisfied clauses is analogously called *minimum unsatisfiabitlity (MinUNSAT) problem*. A solution to one of the two problems can be used to easily compute the solution of the other one, because the following relationship holds: $k = \mathrm{MaxSAT}(\varphi) + \mathrm{MinUNSAT}(\varphi)$. It is worth noting that a model of the optimum of the MaxSAT problem is also a model of the optimum of the MinUNSAT problem and vice versa. In general, there are several models for the optimum.

The MaxSAT problem can be extended in different ways: (i) we can assign a non-negative integer weight to each clause (denoted with (C, w) for a clause C and a weight w) and ask for the maximum sum of weights of satisfied clauses, which is known as the *Weighted MaxSAT problem*, (ii) we can split the clauses in *hard* and *soft* clauses and ask for the maximum number of satisfied soft clauses while sat-

[1] Symbolic Computation Group, WSI Informatics, Universität Tübingen, Germany, www-sr.informatik.uni-tuebingen.de, email: {walterr,kuechlin}@informatik.uni-tuebingen.de

isfying all hard clauses, which is known as the *Partial MaxSAT problem*, and finally (iii) we can combine both specifications, which is known as the *Weighted Partial MaxSAT problem*. The mentioned relationship above between the MaxSAT and MinUNSAT problem also holds for all MaxSAT variants.

Given a set of Boolean variables $\mathrm{F} = \{\mathrm{M}_1, ..., \mathrm{M}_\mathrm{n}\}$ and the restriction that *exactly one* variable has to be satisfied, $\sum_{i=1}^{n} M_i = 1$, we call the set F a *regular family* and the elements *members* of the family. For example, given a set of Boolean variables $\mathrm{E} = \{\mathrm{E}_1, \mathrm{E}_2, \mathrm{E}_3\}$ representing the selectable engines of a car. An engine is chosen if and only if the corresponding variable is set to true. A car has exactly one engine, which makes the set E a family.

Given a family $\mathrm{F} = \{\mathrm{M}_1, ..., \mathrm{M}_\mathrm{n}\}$ with the restriction that *at most one* variable has to be true, we call the set F an *optional family*. For example, given a set of Boolean variables $\mathrm{AC} = \{\mathrm{AC}_1, \mathrm{AC}_2, \mathrm{AC}_3, \mathrm{AC}_4\}$ representing the selectable air conditioners of a car. An air conditioner is an optional feature in a car, but there can be at most one air conditioner. This makes the set AC an optional family.

The restrictions of a regular family or an optional family are special cases of *cardinality constraints*, which restrict the number of satisified variables of a set of Boolean variables to be $\{\leq, <, =, >, \geq\}$ a non-negative integer k. The restriction for a regular family can be encoded in CNF with the following two formulas, while an optional family can be encoded by using only the second formula:

1. At least one satisfied variable: $\bigvee_{i=1}^{n} M_i$
2. At most one satisfied variable: $\bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n} (\neg M_i \vee \neg M_j)$

The given encodings for the two special cases $= 1$ and ≤ 1 are very simple and require only $\mathcal{O}\left(n^2\right)$ clauses without adding new auxiliary variables. There are also encodings using auxiliary variables in exchange for a fewer number of clauses [14].

Since we consider only regular and optional family types, more general cardinality constraints than the above-mentioned special cases are not necessary and thus not considered in this paper. In the context of automotive configuration, we usually deal with rules and families of certain model series. For example, the number of seats is fixed and therefore we do not need to handle a family of seats where we would need a cardinality constraint to restrict the selection of seats between two and four.

3 Product Configuration Concepts for SAT-Configuration

In this section, we describe the basic concept of SAT-based product configuration. We concentrate on rules in Propositional Logic, because in our main application context of automotive configuration we always deal with this type of rules. Along with the set of rules we consider families, which results in the following definition for product configuration:

Definition 1. *(Product Configuration Instance[2]) A* product configuration instance *is a triple* $(\mathcal{R}, \mathcal{F}, \mathcal{S})$*:*

- *Set* $\mathcal{R} = \{\varphi_1, \ldots, \varphi_k\}$*, where* φ_i *is a propositional formula.*
- *Set* $\mathcal{F} = \{\mathrm{F}_1, \ldots, \mathrm{F}_\mathrm{m}\}$*, where* $\mathrm{F_i}$ *is a family.*
- *Mapping* $\mathcal{S} : \bigcup_{i=1}^{m} \mathrm{vars}(F_i) \rightarrow \{\mathrm{no}, \mathrm{yes}\} \times (\mathbb{N}_{\geq 0} \cup \{\infty\})$.

[2] In the configuration literature a *product configuration instance* is a solution for a configuration problem, whereas we refer to the term as a description of a product configuration problem.

The following relation holds between rules and families: $\bigcup_{\mathrm{R} \in \mathcal{R}} \mathrm{vars(R)} \subseteq \bigcup_{\mathrm{F} \in \mathcal{F}} \mathrm{F}$.

The rules $\mathcal{R}$ describe the relationship among the family members of the different families. They determine the possible valid combinations. The set $\mathcal{F}$ contains all optional and regular families. The mapping $\mathcal{S}$ represents the selections and deselections of the family members in respect of a priority. For simplicity reasons we will only use the term selections to refer to both selections and deselections. There are three main cases for a member s:

1. $\mathcal{S}(s) = (c, 0)$ with $c \in \{\mathrm{no}, \mathrm{yes}\}$:
   The user made no decision about the member (priority 0).
2. $\mathcal{S}(s) = (c, p)$ with $c \in \{\mathrm{no}, \mathrm{yes}\}$ and $p \in \mathbb{N}_{\geq 1}$:
   The user made a selection (priority greater zero).
3. $\mathcal{S}(s) = (c, \infty)$ with $c \in \{\mathrm{no}, \mathrm{yes}\}$:
   The user made an indispensable (hard) selection (infinity priority).

We abbreviate the mapping $\mathcal{S}$ for a member s as follows: For a positive selection we write a positive literal s and for a negative selection we write the negative literal $\neg s$. We can then write a single tuple (s, p) with $p \in \mathbb{N}_{\geq 0} \cup \{\infty\}$ and describe the mapping $\mathcal{S}$ as a set of tuples. For simplicity reasons we leave each member s with priority 0 out of $\mathcal{S}$ in the given examples of this paper.

The set $\mathcal{S}$ of selections can be seen as a partial assignment given by the user of the product configurator and can be divided in two disjoint sets of positive and negative selections: $\mathrm{Pos}(\mathcal{S}) := \{(s, p) \mid (s, p) \in \mathcal{S} \text{ and } s \text{ is a positive literal}\}$ and $\mathrm{Neg}(\mathcal{S}) := \{(s, p) \mid (s, p) \in \mathcal{S} \text{ and } s \text{ is a negative literal}\}$.

The priority of a selected member is only relevant when it comes to the question of re-configuration. Then the priorities represent the users preferences.

Example 1. *We consider a product configuration instance* $(\mathcal{R}, \mathcal{F}, \mathcal{S})$*, where* $\mathcal{R}$ *and* $\mathcal{F}$ *describe components of a computer system and dependencies among them. Table 1 shows the families and Table 2 shows the rules. Let* $\mathcal{S} = \emptyset$*, which means a user has not made selections so far.*

Family	Type	Members
M (Mainboard)	regular	$\mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_3, \mathrm{M}_4$
V (Videocard)	regular	$\mathrm{V}_1, \mathrm{V}_2, \mathrm{V}_3, \mathrm{V}_4, \mathrm{V}_5$
C (CPU)	regular	$\mathrm{C}_1, \mathrm{C}_2, \mathrm{C}_3, \mathrm{C}_4$
P (Power Supply)	regular	$\mathrm{P}_1, \mathrm{P}_2$
CD (CD-Device)	optional	$\mathrm{CD}_1, \mathrm{CD}_2, \mathrm{CD}_3$
CR (Card-Reader)	optional	$\mathrm{CR}_1, \mathrm{CR}_2$

Table 1: Families $\mathcal{F}$ of the computer system

Rules		
M_1	$\rightarrow$	$((\mathrm{V}_1 \vee \mathrm{V}_2 \vee \mathrm{V}_4) \wedge (\mathrm{C}_1 \vee \mathrm{C}_3) \wedge \mathrm{P}_1 \wedge \neg \mathrm{CD}_1)$
M_2	$\rightarrow$	$((\mathrm{V}_2 \vee \mathrm{V}_5) \wedge (\mathrm{C}_2 \vee \mathrm{C}_3) \wedge (\mathrm{P}_1 \vee \mathrm{P}_2) \wedge \neg \mathrm{CD}_1)$
M_3	$\rightarrow$	$((\mathrm{V}_3 \vee \mathrm{V}_4) \wedge (\mathrm{C}_2 \vee \mathrm{C}_3 \vee \mathrm{C}_4) \wedge \mathrm{P}_1)$
M_4	$\rightarrow$	$((\mathrm{V}_1 \vee \mathrm{V}_2) \wedge (\mathrm{C}_1 \vee \mathrm{C}_4) \wedge \mathrm{P}_1 \wedge \neg \mathrm{CD}_2)$
C_1	$\rightarrow$	$((\mathrm{V}_2 \vee \mathrm{V}_3) \wedge \mathrm{P}_2)$
C_2	$\rightarrow$	$(\mathrm{V}_4 \vee \mathrm{V}_5)$
C_3	$\rightarrow$	$(\mathrm{V}_3 \vee \mathrm{V}_4)$

Table 2: Configuration rules $\mathcal{R}$ of the computer system

We will now define the criteria of a valid configuration:

Definition 2. *(Valid Configuration) A product configuration instance is called a* valid configuration *if the following formula is satisfiable:*

$$\bigwedge_{\mathrm{R}\in\mathcal{R}} R \wedge \bigwedge_{\mathrm{F}\in\mathcal{F}} \mathrm{CC(F)} \wedge \bigwedge_{(s,p)\in\mathcal{S},p\neq 0} s$$

Where $\mathrm{CC(F)}$ *are the appropriate cardinality constraints of a family (described in the preliminaries).*

If a configuration instance is valid, the corresponding (partial) variable assignment (also called model *or* configuration solution*) is of interest, because the variable assignment describes which members are chosen and which are not.*

A configuration solution is in general not complete, e.g. when the selections $\mathcal{S}$ made by a user contain selections with priority 0.

After defining the basic product configuration concepts, we will go into more detail in the next section by describing which use cases of a SAT aided product configurator exist and finally by showing an iterative process of SAT aided product configuration.

4 SAT aided Product Configuration

With SAT solving a product configurator can validate a user's selection and also compute the selectable members for the remaining families. The overall plan is quite simple: Each selection of an option results in a *true* valuation of that option. Regular families result in propagations of the value *false* to the remaining options, after one family member has been selected. Given a partial valuation, it is easy to compute by SAT solving which of the remaining options can still be selected, and which must be set to true or false, respectively, as a consequence of previous selections.

We describe these use cases in detail in the following subsections and afterwards consolidate them in an iterative user process.

4.1 Use Case: Validation & completion of a (partial) selection

Given a product configuration instance $(\mathcal{R}, \mathcal{F}, \mathcal{S})$, we can validate the selections with a SAT solver by checking the formula of Definition 2 for satisfiability. Algorithm 1 shows the procedure. Only selections with a priority $\neq 0$ are taken into account for the validation.

Algorithm 1: Validation & completion of a (partial) selection

Input: $(\mathcal{R}, \mathcal{F}, \mathcal{S})$
Output: $(\mathrm{result}, \mathrm{model})$, where result is `true` if the (partial) selection is valid, otherwise `false` and model is a complete variable assignment
return $\mathrm{SAT}\left(\bigwedge_{\mathrm{R}\in\mathcal{R}} \mathrm{CNF}(R) \wedge \bigwedge_{\mathrm{F}\in\mathcal{F}} \mathrm{CC(F)} \wedge \bigwedge_{(s,p)\in\mathcal{S},p\neq 0} s\right)$

Because most SAT solvers take CNF as input, we write $\mathrm{CNF}(R)$ to indicate the transformation of an arbitrary rule to its CNF representation. In practice we use a polynomial formula transformation [15, 12] to get an equisatisfiable formula to avoid the potentially exponential blow-up that occurs when using the distributive law.

If the configuration instance is valid, the algorithm also returns a complete variable assignment. This complete variable assignment gives an example which selections have to be made to complete the given configuration instance. In general, the given model is not uniqe and there exist several models.

Example 2. *We reconsider the computer system configuration Example 1. In the following two selection examples, we do not use priorities because we just want to check the validity of the selections.*

1. $\mathcal{S} = \{\mathrm{M}_1, \mathrm{V}_4\}$ *leads to a valid configuration, which can be completed to* $\{\mathrm{M}_1, \mathrm{V}_4, \mathrm{C}_3, \mathrm{P}_1, \mathrm{CD}_3, \mathrm{CR}_2\}$.
2. $\mathcal{S} = \{\mathrm{M}_1, \mathrm{C}_1\}$ *leads to an invalid configuration, because* M_1 *requires* P_1 *and* C_1 *requires* P_2*, but due to the family constraints, both cannot be selected at the same time.*

4.2 Use Case: Computation of selectable members

During the configuration process a user would like to know which of the remaining family members are still selectable, i.e. which selections lead to a valid configuration. We can compute the selectable members by validating the (partial) selections with a SAT solver. Algorithm 2 shows the procedure. We iteratively make one SAT call for each member and check if selecting this member is valid.

Algorithm 2: Computation of selectable members

Input: $(\mathcal{R}, \mathcal{F}, \mathcal{S})$
Output: Mapping V from $\left(\bigcup_{\mathrm{F}\in\mathcal{F}} F\right)$ to $\{\texttt{no}, \texttt{yes}\}$ indicating whether a member is selectable or not
$V \leftarrow$ Initialize mapping
foreach $m \in \bigcup_{\mathrm{F}\in\mathcal{F}} F$ **do**
  if $\mathrm{SAT}\left(\bigwedge_{\mathrm{R}\in\mathcal{R}} \mathrm{CNF}(R) \wedge \bigwedge_{\mathrm{F}\in\mathcal{F}} \mathrm{CC(F)} \wedge \bigwedge_{(s,p)\in\mathcal{S},p\neq 0} s \wedge m\right)$
  then
    $V \leftarrow (m, \texttt{yes})$
  else
    $V \leftarrow (m, \texttt{no})$
return V

After the computation of selectable members, the SAT aided product configurator can display the result to the user (i.e. by disabling all non-selectable members). Then the user knows about the selectable members.

Remarks:

1. In the special case $\mathcal{S} = \emptyset$, in which no selection has been made by the user so far, the computation of the selectable members implicitly brings up members which can never be part of a valid configuration (*redundant members*) and members which have to be part of each valid configuration (*forced members*).
2. The performance of Algorithm 2 can be improved. If a family already contains a positively selected member, then we know that all remaining members are not selectable anymore due to the family constraints. We just have to check families with no positively selected member.
   The performance can be improved further. We use an incremental and decremental SAT solver, which allows us to load all rules, family constraints and selections first and check each member m by adding and removing the unit clause m from the SAT solver. We do not have to load the invariant constraints repeatedly for each check.

Example 3. *We compute the selectable members for our computer system configuration (see Example 1):*

1. $\mathcal{S} = \emptyset$: *Table 3 shows the remaining selectable members.*

Family	Selectable Memb.	Non-Selectable Memb.
M (Mainboard)	M_1, M_2, M_3, M_4	
V (Videocard)	V_1, V_2, V_3, V_4, V_5	
C (CPU)	C_2, C_3, C_4	C_1
P (Power Supply)	P_1, P_2	
CD (CD-Device)	CD_1, CD_2, CD_3	
CR (Card-Reader)	CR_1, CR_2	

Table 3: Selectable members for an empty selection

2. $\mathcal{S} = \{M_1, V_4\}$: *Table 4 shows the remaining selectable members.*

Family	Selectable Memb.	Non-Selectable Memb.
C (CPU)	C_3	C_1, C_2, C_4
P (Power Supply)	P_1	P_2
CD (CD-Device)	CD_2, CD_3	CD_1
CR (Card-Reader)	CR_1, CR_2	

Table 4: Result of the selectable members computation

4.3 SAT aided configuration process

Figure 1 illustrates a possible SAT aided configuration process involving both Use Cases 4.1 and 4.2. After the user has made one or multiple selections, the SAT solver validates the current configuration. This results in two cases:

1. **Valid configuration:** In the case of a valid configuration, the user can continue selecting members. Additionally, to guide the user we can compute the selectable members for the current configuration. After new selections, the process iterates.
2. **Invalid configuration:** In the case of an invalid configuration, the user has to take back one or more of the previously made selections. The user can validate each backtracking step again until a valid configuration state is reached.

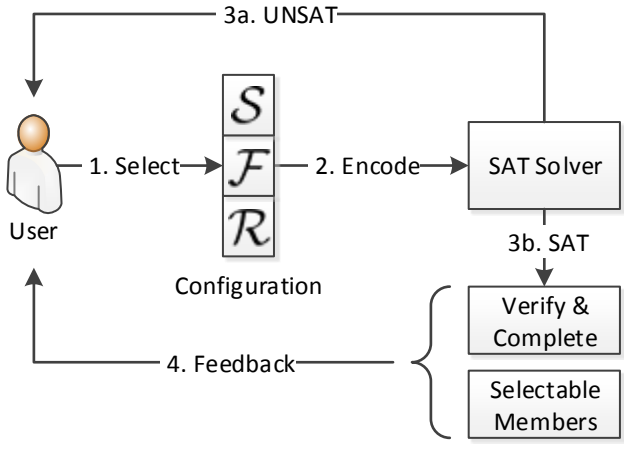


Figure 1. SAT aided configuration process

Remark: If a given complete example model $l_1 \wedge \ldots \wedge l_n$ in the SAT case does not satisfy the demands of the user, she can exclude this model by adding the hard clause $\neg l_1 \vee \ldots \vee \neg l_n$. Then another complete model will be produced if one exists, otherwise we encounter the UNSAT case.

In a SAT aided product configuration process described above, the user is left to herself when it comes to the question which selections should be undone to regain a valid configuration. Perhaps the user made a selection of a highly desired member, which she does not want to take back. Now the user has to try different configuration changes by herself and a guidance is missing which one to choose. This is the point where MaxSAT aided product configuration can help. We will describe re-configuration use cases in detail in the following section.

5 MaxSAT aided Product (Re-)Configuration

In this section we describe how re-configuration can be done with partial (weighted) MaxSAT as a background engine. We show two basic use cases, describe possible variations of them and finally integrate the re-configuration step into our iteractive user process.

5.1 Use Case: Re-configuration of the selections

During the configuration process we may reach a state where we have an invalid configuration. The cause of the conflict can be one or both of the following:

1. The selections $\mathcal{S}$ conflict with the rules $\mathcal{R}$.
2. The selections $\mathcal{S}$ conflict with the family constraints.

We have to re-configure either the rules or the selections to regain validity. For now we consider all rules as hard limitations that we can not soften, which is the common case. We will discuss re-configuration of rules later in Section 5.4.

Considering the rules as a hard restrictions, the question arises, how many of the selections can be kept maximally to reach a valid configuration. Remember, a user may have done multiple selections at once without validating the current configuration and without considering the selectable members. Therefore, removing only the last selection does not lead to a valid configuration again in general. Also the last selection could be of infinity priority, so it is no option for the user to remove the last selection.

To answer the question we set the selections as soft unit clauses and re-configure the selections with a partial MaxSAT solver. The following encoding represents our requirements:

$$\text{Hard} := \bigcup_{R \in \mathcal{R}} \text{CNF(R)} \cup \bigcup_{F \in \mathcal{F}} \text{CC(F)} \cup \bigcup_{(s,p) \in \mathcal{S}, p=\infty} \{s\}$$

$$\text{Soft} := \bigcup_{(s,p) \in \mathcal{S}, p \neq 0, p \neq \infty} \{s\}$$

Selections with priority ∞ are also considered as indispensable and will be encoded as hard unit clauses. Only dispensable selections will be re-configured. Algorithm 3 shows the re-configuration procedure.

With the resulting model, we can give the user an example of a complete selection which requires a minimal number of changes in order to regain a valid configuration compared to the original selections. Or, the other way round, the model gives an example about how to keep the maximal number of selections.

Algorithm 3: Re-Configuration of a (partial) selection

Input: $(\mathcal{R}, \mathcal{F}, \mathcal{S})$
Output: $(\text{optimum}, \text{model})$, where optimum is the minimal number of changes to regain a valid configuration and model is a model for the optimum

$\text{Hard} \leftarrow \emptyset$
$\text{Soft} \leftarrow \emptyset$
foreach $\text{R} \in \mathcal{R}$ **do**
  $\text{Hard} \leftarrow \text{Hard} \cup \text{CNF(R)}$
foreach $\text{F} \in \mathcal{F}$ **do**
  $\text{Hard} \leftarrow \text{Hard} \cup \text{CC(F)}$
foreach $(s, p) \in \mathcal{S} \wedge p \neq 0$ **do**
  if $p = \infty$ **then**
    $\text{Hard} \leftarrow \text{Hard} \cup \{s\}$
  else
    $\text{Soft} \leftarrow \text{Soft} \cup \{s\}$
$(\text{optimum}, \text{model}) \leftarrow \text{PartialMinUNSAT}(\text{Hard}, \text{Soft})$
return $(\text{optimum}, \text{model})$

Remark: As desribed before we use a transformation like Tseitin or Plaisted-Greenbaum instead of CNF(R) in practice. Even though the Tseitin and Plaisted-Greenbaum transformations are only equisatisfiable, this is not an issue for MaxSAT when converting formulas into hard clauses. Since the Tseitin and Plaisted-Greenbaum transformations share the same models on the original variables, one can easily verify that the search space between the converted and the original instance remains the same.

Extensions: The described use case can be extended as follows:

1. **User constraints**: A user can add additional constraints considered as hard clauses.
   If, e.g., mainboard M_1 is selected, the user definitely wants video card V_2 to be selected. But if mainboard M_2 is selected, the user definitely wants video card V_5 to be selected. Then we add the rules $(\text{M}_1 \rightarrow \text{V}_2) \wedge (\text{M}_2 \rightarrow \text{V}_5)$ as constraints to the rules $\mathcal{R}$.
2. **Focus on selection**: For each family an option "choose one of the selected" can be offered to add a constraint such that only positive selected members within a family will be considered during the re-configuration computation.
   E.g. if a user focuses on mainboards $\text{M}_1, \text{M}_3, \text{M}_4$, a hard clause $(\text{M}_1 \vee \text{M}_3 \vee \text{M}_4)$ will be added to the rules $\mathcal{R}$.

Example 4. *We continue our canonical Example 1: Table 5 shows multiple selections of members within families and a result model re-configuration. For all selections shown we choose priority* 1*, that means no selection in this example is an indispensable one.*

Family	Focus	Selections	Results
M	No	$(\text{M}_1, 1), (\text{M}_2, 1), (\neg\text{M}_3, 1)$	M_4
V	Yes	$(\text{V}_1, 1), (\text{V}_2, 1)$	V_1
C	No	$(\text{C}_2, 1), (\text{C}_3, 1)$	C_4
P	No		P_1
CD	No	$(\neg\text{CD}_1, \infty)$	CD_3
CR	No		CR_2

Table 5: Users selections and results

Result: *We have to make 5 changes minimally to regain a valid configuration. Without the focus set for the video cards family V, we would have to make 4 changes minimally, e.g. by choosing* M_2, V_5, C_2, P_1, CD_3, CR_2.

5.2 Use Case: Re-Configuration of the selections with priorities

In the previous use case we treated all soft clauses as equivalent. A user may prefer one member over the other, which results in priorization of the selected members. We can handle priorities with Partial Weighted MaxSAT solving. The encoding for this use case is basically the same as before, but now we bring priorities into play. Algorithm 4 shows the complete computation procedure.

Algorithm 4: Re-Configuration of a (partial) selection with priorities

Input: $(\mathcal{R}, \mathcal{F}, \mathcal{S})$
Output: $(\text{optimum}, \text{model})$, where optimum is the minimal number of priority points to change to regain a valid configuration and model is a model for the optimum

$\text{Hard} \leftarrow \emptyset$
$\text{Soft} \leftarrow \emptyset$
foreach $\text{R} \in \mathcal{R}$ **do**
  $\text{Hard} \leftarrow \text{Hard} \cup \text{CNF(R)}$
foreach $\text{F} \in \mathcal{F}$ **do**
  $\text{Hard} \leftarrow \text{Hard} \cup \text{CC(F)}$
foreach $(s, p) \in \mathcal{S} \wedge p \neq 0$ **do**
  if $p = \infty$ **then**
    $\text{Hard} \leftarrow \text{Hard} \cup \{s\}$
  else
    $\text{Soft} \leftarrow \text{Soft} \cup \{(s, p)\}$
$(\text{optimum}, \text{model}) \leftarrow$
$\text{PartialWeightedMinUNSAT}(\text{Hard}, \text{Soft})$
return $(\text{optimum}, \text{model})$

Extension: All extensions presented in Subsection 5 carry over to this use case.

Example 5. *We reconsider our re-configuration Example 4 and add a priority of 2 for member* V_2*. Table 6 shows our selections with the corresponding weights in parentheses and the results.*

Family	Focus	Selections	Results
M	No	$(\text{M}_1, 1), (\text{M}_2, 1), (\neg\text{M}_3, 1)$	M_4
V	Yes	$(\text{V}_1, 1), (\text{V}_2, 2)$	V_2
C	No	$(\text{C}_2, 1), (\text{C}_3, 1)$	C_4
P	No		P_1
CD	No	$(\neg\text{CD}_1, \infty)$	CD_3
CR	No		CR_2

Table 6: Users selections with priorities and results

Result: *We have to change 5 priority points minimally to regain a valid configuration. If we would still be choosing member* V_1 *instead of member* V_2 *we would have to change 6 priority points, because of the higher priority of* V_2*.*

5.3 A MaxSAT aided re-configuration process

We reconsider the process of Figure 1 in Step 3a. UNSAT where the user gets the feedback that her current selections lead to an invalid

configuration. With a SAT solver only, the user has to try by herself which selections have to be undone to regain a valid configuration. But now, we can help the user at this point by using re-configuration with MaxSAT. Figure 2 illustrates both Use Cases 5.1 and 5.2 embedded in a product configuration process using MaxSAT.

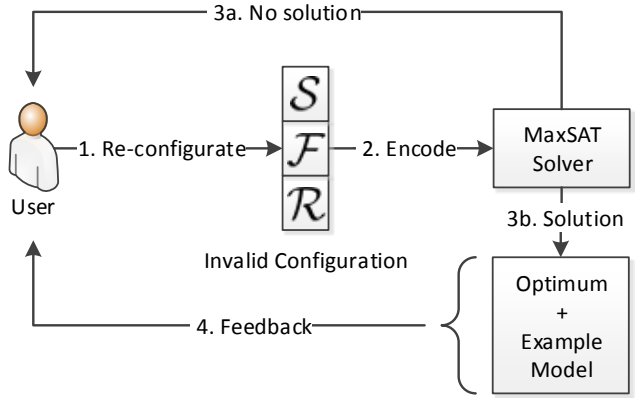


Figure 2. MaxSAT aided configuration process

After the user gets the feedback UNSAT, she can start a re-configuration of her current selections. This results in two cases:

1. **No solution:** If the indispensable selections (with priority ∞) collide with the rules or the family constraints, then there is no solution. In this case, the user has to weaken some of the indispensable selections in order to make a re-configuration possible. The user can use high priorities to weaken the desired members to ensure they will be preferred over other selections.
2. **Solution:** If the indispensable selections can be satisfied, then there exists a solution with an optimum for the prioritized selections. In this case, the user will be told about the optimum, i.e. about the number of minimal changes to regain a valid configuration. Also, an example model with the optimum will be given to the user.

Remark: Similiar to the SAT aided configuration process the following holds: If the given complete example model $l_1 \wedge \ldots \wedge l_n$ in the solution case does not satisfy the demands of the user, she can exclude this model by adding the hard clause $\neg l_1 \vee \ldots \vee \neg l_n$. Then another model with the same optimum will be produced, if one exists. If there is no other model with the same optimum, the next best optimum under the new conditions will be computed with an example model.

In case there is no solution and a user just do not want to weaken her selections with priority ∞, we can consider weakening the rules. In the next section, we will describe this possibility in detail.

5.4 Use Case: Re-configuration of rules

It is possible that the selections a user made have no solution when trying to re-configure them. Assuming the rules themselves are not contradictory, then the cause for no solution are too many selections with priority ∞. There are two cases which can occur or both at the same time:

1. **Violation of the family constraints:** If a user selects more than one member of a family with infinity priority, the family constraints are violated.
2. **Violation of rules:** If a user does not violate the family constraints, then the selected members with priority infinity are in collision with the rules.

The first case can be handled by a product configurator by simply not allowing to choose more than one member with priority infinity or giving the user a warning message when doing so.

In the second case, if the user is not willing to soften her selections, we can not re-configure the selections w.r.t. the rules. But when we have a closer look at the rules, there may be some rules, which we can soften, e.g. when a rule is not a physical or technical restriction, but only exists for marketing or similiar purposes. A company may be willing to violate or change some of these rules to build the product. Knowing the miminum number of rule changes in order to permit a desired vehicle configuration can help in managing the set of marketing rules.

For this use case, we extend Definition 1 by an additional mapping $\mathcal{S}_\mathcal{R} : \mathcal{R} \to (\mathbb{N}_{\geq 0} \cup \{\infty\})$, which represents the priorities of the rules a user made. After softening some of the rules this way we can re-configure the rules by maximizing the number of satisfied rules, respectively violating only a minimal number of rules. Algorithm 5 shows this procedure more formally.

Algorithm 5: Re-Configuration of rules

Input: $(\mathcal{R}, \mathcal{F}, \mathcal{S})$
Output: $(\text{optimum}, \text{model})$, where optimum is the minimal number of changes to regain a valid configuration and model is a model for the optimum

$\text{Hard} \leftarrow \mathcal{S}$
$\text{Soft} \leftarrow \mathcal{S}$
foreach $\text{F} \in \mathcal{F}$ **do**
  $\text{Hard} \leftarrow \text{Hard} \cup \text{CC(F)}$
foreach $\text{R} \in \mathcal{R} \wedge \mathcal{S}_\mathcal{R}(\text{R}) \neq 0$ **do**
  if $p = \infty$ **then**
    $\text{Hard} \leftarrow \text{Hard} \cup \text{CNF(R)}$
  else
    $\text{Hard} \leftarrow \text{Hard} \cup \text{CNF}(b_R \rightarrow \text{R})$
    $\text{Soft} \leftarrow \text{Soft} \cup \{b_R\}$
foreach $(s, p) \in \mathcal{S} \wedge p \neq 0$ **do**
  if $p = \infty$ **then**
    $\text{Hard} \leftarrow \text{Hard} \cup \{s\}$
  else
    $\text{Soft} \leftarrow \text{Soft} \cup \{s\}$
$(\text{optimum}, \text{model}) \leftarrow \text{PartialMinUNSAT}(\text{Hard}, \text{Soft})$
return $(\text{optimum}, \text{model})$

Since a rule R is an arbitrary formula, we can not just convert R to its CNF and add the resulting clauses as soft clauses. In general, some of these clauses will be satisfied and some not. Instead we want to maximize the number of rules. In other words, we are facing a *group MaxSAT problem* [2, 6], where each CNF(R) is a group of clauses. The goal of group MaxSAT is to satisfy the maximum number of groups. A group is satisfied if all clauses within the group are satisfied.

The group MaxSAT problem can be reduced to a partial MaxSAT problem as follows: For each non-indispensable rule R we introduce a *new* variable b_R and add the hard clauses $\text{CNF}(b_R \rightarrow \text{R})$. Addi-

tionally we add a unit soft clause $\{b_R\}$ for each new variable. Each satisfied variable b_R implies the whole group of clauses in $\mathrm{CNF}(\mathrm{R})$ to be satisfied. Therefore, satisfying a maximal number of the newly introduced variables satisfies a maximal number of the corresponding formulas. On the other hand, with the help of the newly introduced variables, we can identify, from the resulting model, which formulas are satisfied and show this result to the user. For a more detailed explanation, see [2, 6].

Extension: Of course, rules can also have different priorities and we can extend this use case by assigning priorities to rules and selections to compute the maximal sum of priority points. This extension can be realized analogously as described for Use Case 5.2, thus we will not describe it explicitly.

6 Implementation techniques

We implemented the above SAT-based and MaxSAT-based use cases in one product configurator — called *ReMax* — on top of our uniform logic framework, which we use for commercial applications within the context of automotive configuration. Our SAT solver provides an incremental and decremental interface. We maintain two versions (Java and .NET) and decided to implement ReMax using .NET 4.0 with C# along with the WPF Framework for the GUI. We implemented state-of-the-art partial (weighted) MaxSAT solvers Fu&Malik, PM2 and WPM1 on top of our SAT solver [5, 1].

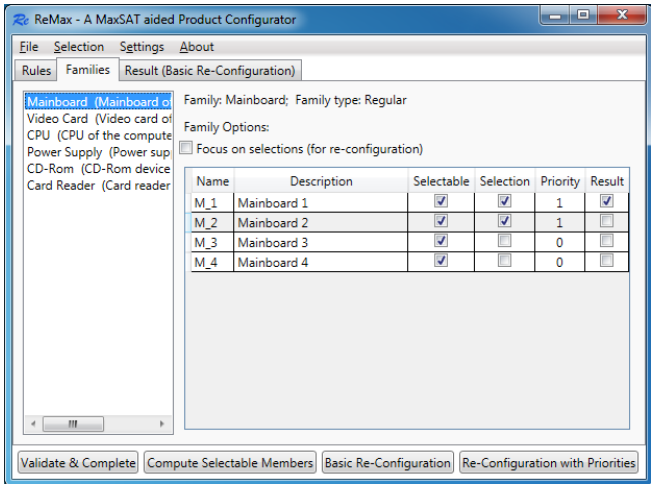


Figure 3. Screenshot of ReMax with open "Families" tab

Figure 3 shows an example screenshot from the ReMax GUI with the "Families" tab opened.

7 Experimental Results

Table 7 show statistics about the real configuration data from two different German car manufacturers, called `M01` and `M02`, which we used for our benchmarks. Car manufacturer `M01` uses arbitrary formulas as rules, whereas `M02` uses clauses as rules.

	Rules		Families	
Problem	Quantity	#Variables	Quantity	Avg. size
`M01_01`	2074	1772	34	34,294
`M01_02`	2430	2087	41	39,293
`M01_03`	1137	880	30	18,233
`M02_01`	11627	996	188	6,282
`M02_02`	4465	612	174	5,321

Table 7: Statistics about car manufacturer problems

For the following benchmarks we used two partial weighted MaxSAT solvers, which are based on the following principles:

1. **WPM1:** An unsat core-guided approach with iterative SAT calls. In each iteration a new blocking variable will be added to each soft clause within the unsat core [1].
2. **msu4:** An unsat core-guided approach with iterative SAT calls using a reduced number of blocking variables [11].

We implemented WPM1 on top of our own SAT solver while msu4 is an external solver[3].

Our environment for the benchmarks has the following hardware and software settings. Processor: Intel Core i7-3520M, 2,90 GHz; Main memory: 8 GB. WPM1, based on .NET 4.0, runs under Windows 7 while msu4 runs under Ubuntu 12.04.

For Use Case 5.2 we created three categories as follows: Out of 30%, 50% and 70% of the families one member is selected randomly with a random priority between 1 and 10. The rules have infinity priority. In general, this leads to an invalid configuration because the rules are violated. For each category we created 10 instances.

Table 8 shows the results for each category as average time in seconds. The abbreviation "exc." means that the time limit of 30 minutes was exceeded. As we can see, msu4 performs very well in all categories with reasonable times from less than one second up to about 25 seconds. Our solver WPM1 also has reasonable times from about 2 seconds up to about 28 seconds, but exeeds the time limit in two categories for the instance `M02_01`.

	30%		50%		70%	
Problem	WPM1	msu4	WPM1	msu4	WPM1	msu4
`M01_01`	7,34	0,66	12,70	1,08	15,59	1,84
`M01_02`	8,59	0,74	16,48	1,32	27,44	2,96
`M01_03`	2,10	0,33	4,10	0,45	5,80	0,85
`M02_01`	20,99	2,16	exc.	5,91	exc.	24,45
`M02_02`	3,90	0,48	9,60	1,56	13,01	4,77

Table 8: Results of Use Case 5.2 scenario

For Use Case 5.4 we created three categories as follows: Out of 30%, 50% and 70% of the families one member is selected randomly with infinity priority, which leads to an invalid configuration in general because the rules are violated. But this time, we assign all rules a priority of 1. For each category we created 10 instances.

Table 9 shows the results for each category as average time in seconds. As we can see, both solvers can handle all instances in each category in reasonable time. While WPM1 takes from about 3 seconds up to about 72 seconds, the external solver msu4 takes from less than one second up to about 9 seconds in the worst case.

[3] `http://logos.ucd.ie/web/doku.php?id=msuncore`

	30%		50%		70%	
Problem	WPM1	msu4	WPM1	msu4	WPM1	msu4
M01_01	9,35	2,39	16,19	3,93	20,63	4,35
M01_02	12,86	2,80	19,32	5,47	27,82	4,82
M01_03	2,54	0,78	5,71	1,45	6,76	1,74
M02_01	18,40	4,43	41,16	8,33	71,29	8,55
M02_02	5,13	0,49	9,88	1,04	16,32	1,48

Table 9: Results of Use Case 5.4 scenario

8 Related Work

Another approach for re-configuration uses answer set programming (ASP) on a decidable fragment of first-order logic [4]. Hence the used language is more expressive. With the growing performance of SAT solvers in the last decade, SAT solving in turn has been used to solve problem instances of ASP [8].

An algorithm for computing minimal *diagnoses* using a conflict detection algorithm is introduced in [13]. A minimal subset Δ of constraints is called a diagnosis if the original constraints without Δ are consistent. Although this approach is described for constraints of first-order sentences, the technqiues can be generalized to a wide range of other logics.

The indicated idea above is further improved in [3], where an algorithm — called FastDiag — is introduced which computes a preferred minimal diagnosis while improving performance.

We did not consider works dealing with explanations like MUS (Minimal Unsatisifable Subset) iteration. When using MUS iteration for re-configuration, a user not only has to manually solve each conflict, but also will not necessarily solve the conflicts in an optimal manner, i.e. only changing a minimal number of selections.

9 Conclusion

We described product configuration for propositional logic based rule sets which are widely used in the automotive industry. We showed applications of SAT solving by two use cases. Furthermore, we showed use cases of how MaxSAT can be used for product configuration when it comes to an invalid configuration. With MaxSAT we are able to re-configure an invalid configuration in an optimal way, i.e. we can compute the minimal number of necessary changes. We embedded both scenarios in configuration processes showing how a user can be guided during the configuration process.

We presented an implementation of a product configurator — ReMax — supporting all of the described use cases using state-of-the-art SAT and MaxSAT solving techniques. From real automotive configuration data from two different German premium car manufacturers we created synthetic product configuration benchmarks for the presented use cases. Besides our own MaxSAT solver we used the external solver msu4 to measure and compare the performance. As our experimental results show, we can re-configure those problem instances in reasonable time. Since some problem instances could be solved within a few seconds, our product configurator could be used as an interactive tool in these cases. Other problem instances took over a minute in the worst case, but is still more than adequate for a responsive batch service. While this may seem long, we were told that the manual configuration of an order without tool support by a trial and error process may well take on the order of half an hour.

We do not claim that our approach is currently fit for use as a consumer configurator. However, many business units of a car manufacturer, such as engineering or after sales are in need of a (re-)configurator that feeds directly off the engineering product documentation. E.g., many test prototypes must be built before start of production with a varying set of options.

Expert users sometimes need some complete car configurations which cover all valid combinations of a subset of options, e.g. for testing purposes. With a SAT based (re-)configurator, an expert user can start the configuration from the desired options instead of tediously following the given configuration process in a usual sales configurator. At any time, the user can ask the configurator for "any completion" or, using MaxSAT, for a "minimal completion" of the partial configuration to a complete configuration.

REFERENCES

[1] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy, 'Solving (weighted) partial MaxSAT through satisfiability testing', in *Theory and Applications of Satisfiability Testing - SAT 2009*, ed., Oliver Kullmann, volume 5584 of *Lecture Notes in Computer Science*, 427–440, Springer Berlin Heidelberg, (2009).

[2] Josep Argelich and Felip Many, 'Exact Max-SAT solvers for over-constrained problems.', *Journal of Heuristics*, **12**(4–5), 375–392, (September 2006).

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

[4] Gerhard Friedrich, Anna Ryabokon, Andreas A. Falkner, Alois Haselböck, Gottfried Schenner, and Herwig Schreiner, '(re)configuration using answer set programming', in *IJCAI-11 Configuration Workshop Proceedings*, eds., Kostyantyn Shchekotykhin, Dietmar Jannach, and Markus Zanker, pp. 17–24, Barcelona, Spain, (July 2011).

[5] Zhaohui Fu and Sharad Malik, 'On solving the partial MAX-SAT problem', in *Theory and Applications of Satisfiability Testing—SAT 2006*, eds., Armin Biere and Carla P. Gomes, volume 4121 of *Lecture Notes in Computer Science*, 252–265, Springer Berlin Heidelberg, (2006).

[6] Federico Heras, Antnio Morgado, and Joo Marques-Silva, 'An empirical study of encodings for group MaxSAT', in *Canadian Conference on AI*, eds., Leila Kosseim and Diana Inkpen, volume 7310 of *Lecture Notes in Computer Science*, pp. 85–96. Springer, (2012).

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

[8] Fangzhen Lin and Yuting Zhao, 'ASSAT: Computing answer sets of a logic program by SAT solvers.', *Artifical Intelligence*, **157**(1–2), 115–137, (August 2004).

[9] Peter Manhart, 'Reconfiguration – a problem in search of solutions', in *IJCAI-05 Configuration Workshop Proceedings*, eds., Dietmar Jannach and Alexander Felfernig, pp. 64–67, Edinburgh, Scotland, (July 2005).

[10] João Marques-Silva, 'Practical applications of boolean satisfiability', in *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*, 74–80, IEEE, (2008).

[11] João Marques-Silva and Jordi Planes, 'Algorithms for maximum satisfiability using unsatisfiable cores', in *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '08, pp. 408–413. IEEE, (2008).

[12] David A. Plaisted and Steven Greenbaum, 'A structure-preserving clause form translation', *Journal of Symbolic Computation*, **2**(3), 293–304, (September 1986).

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

[14] Carsten Sinz, 'Towards an optimal CNF encoding of boolean cardinality constraints', in *Principles and Practice of Constraint Programming—CP 2005*, ed., Peter van Beek, Lecture Notes in Computer Science, 827–831, Springer Berlin Heidelberg, (2005).

[15] G. S. Tseitin, 'On the complexity of derivations in the propositional calculus', *Studies in Constructive Mathematics and Mathematical Logic*, **Part II**, 115–125, (1968).

[16] Rouven Walter, Christoph Zengler, and Wolfgang Küchlin, 'Applications of MaxSAT in automotive configuration', in *Proceedings of the 15th International Configuration Workshop*, eds., Michel Aldanondo and Andreas Falkner, pp. 21–28, Vienna, Austria, (August 2013).

Sales Configurator Information Systems Design Theory

Juha Tiihonen[1] and **Tomi Männistö**[2] and **Alexander Felfernig**[3]

Abstract. We look for means to advance the field of configuration systems via research that is performed rigorously and methodologically with the aim of theory creation. Specifically, we explore the use of *Information Systems Design Theory* (*ISDT*) as a framework for defining a design science theory for sales configurator construction. ISDT is the primary output of Design Science research that "shows the principles inherent in the design of an IS artifact that accomplishes some end, based on knowledge of both IT and human behavior". The components of ISDT include purpose and scope, constructs, principles of form and function, artifact mutability, testable propositions, and justificatory knowledge. Generalizing from the novel principles of our earlier work applied in the construction of a sales configuration system called WeCoTin, we present the Sales Configurator Information Systems Design Theory SCISDT. SCISDT aims to support development of generic configurators (aka configuration toolkits) that enable the creation of configurator instantiations for individual companies or product lines to provide choice navigation capability.

1 Introduction

Underlying this paper is research that attempted to answer the research question "How to construct a practical and computationally well-founded sales configurator?" [1]. As a part of that research, a generic sales configurator was constructed and evaluated [2]. The configurator was named WeCoTin.

Numerous configurators have been developed both as research prototypes and as commercial software. The landmark R1/XCON was deployed at Digital Equipment Corporation in the early 1980s [3]. Major research efforts have been devoted to configurators applicable to solving general configuration tasks instead of a specific domain. These include COSSACK [4], PLAKON [5, 6] and its successor KONWERK [7, 8], and COCOS [9]. In addition, a large number of commercial general-purpose configurators exist. Trilogy SalesBUILDER [10] was among the first. ILOG offered a generic configuration engine to be used in other vendors' systems [11, 12]. Anderson [13] identified 30 vendors by their Web pages. In addition, prominent enterprise resource planning systems and CRM vendors have one or more configurators, e.g., SAP [14, 15] and Oracle [16-19].

There exists both numerous individual configurator instantiations and general-purpose configurators that enable the creation of such instantiations. However, developing such artifacts is not a scientific contribution as such and deeper principles are required. Many of the approaches to configurator construction could have been conducted within a Design Science framework but have not necessarily been presented as such. In addition, scientific knowledge on different approaches and means for building configurators has been published in different fields of research. For example, a procedure for implementing configurator instantiations based on generic configurators has been proposed [20] and sound principles and requirements on user interaction of configurators have been presented [21, 22]. However, any theories from the design perspective of generic configurator systems are still nonexistent. This view is supported by an identified need for formal configuration models and inference tools for providing systematic and comprehensive solutions to practitioners [22].

Thus, we see that it is possible to advance the field of configuration systems via research that is performed rigorously and methodologically with the aim of theory creation. Specifically, we explore the use of *Information Systems Design Theory* (*ISDT*) [23] as a framework for defining a design science theory for configurator construction. The underlying idea is that an ISDT can be applied as a prescription when constructing similar artefacts. However, an ISDT must be applied and interpreted in the context of application in an intelligent manner. For example, all aspects of the prescription may not apply in the context or other ISDTs may be applicable as sub-theories.

We use the construction of the WeCoTin sales configurator as a basis for the theory and as an example for illustrating the different parts of the theory.

In the following, we first briefly summarize the existing knowledge and principles behind the creation of configurator systems (Section 2). Thereafter in Section 3, we introduce the Design Science research approach. Section 0 outlines the WeCoTin sales configurator based on our earlier work [1, 2] and introduces ISDT and presents our proposal for the *Sales Configurator Information Systems Design Theory (SCISDT)*. Section 5 concludes.

2 Principles of configurators

Configuration has been a fruitful topic for artificial intelligence research, including problem-solving methods, their efficient implementation, and, to a lesser extent, conceptualizations and languages for representing configuration knowledge. System instantiations based on novel approaches have been described along with their business context.

2.1 Configuration knowledge modeling

Configuration knowledge modeling offers ways to represent configuration models, requirements, and configurations. Three primary types of configuration modeling conceptualizations can be

[1] Department of Computer Science and Engineering, Aalto University, Espoo, Finland, email: juha.tiihonen@aalto.fi
[2] Department of Computer Science, Helsinki University, Helsinki, Finland, email: tomi.mannisto@cs.helsinki.fi
[3] Institute for Software Technology, Graz University of Technology, Graz, Austria email: alexander.felfernig@ist.tugraz.at

identified. The first type is actually not a conceptualization. It is based on the idea that *configuration knowledge can be directly encoded in the presentation mechanisms of the problem-solving method*. At least rule-based approaches, constraint satisfaction and its dynamic extensions, several logic-based approaches, and different formalisms of propose-and-revise methods have been applied; for summaries, see Stumptner [24] Sabin and Weigel [25], and Hubaux et al. [26]. Of these methods, constraint satisfaction is the most widely applied. The second type is *configuration-domain-specific conceptualizations*, which are independent of problem-solving methods. These can be roughly classified as *connection-based* [27], *resource-based* [28], *structure-based* [5], or *function-based* [29] approaches. The conceptualizations have little in common, other than the central notion of a component.

The third and the most recent type of conceptualization includes *unified approaches* that combine the ideas of the individual approaches into a covering ontology or conceptualization. An example of such a conceptualization is [30]. Unified conceptualizations may include *component types* and their *compositional structure*, *attributes* and *topological concepts* such as *ports* for specifying connectivity. *Resources* model the production and use of some entity, such as power or expansion slots. The underlying idea is that some component individuals produce a *resource* and other component individuals use it. There must be enough production to cover use. *Functions* represent the functionality that a product individual provides to the customer, the product's user, or the environment. The idea of functions is to provide a non-technical view to the functionality and features of the product to be configured. These are then mapped to component individuals, attribute values, and connections that implement the desired functionality and features. Concepts discussed above are organized in a taxonomical structure with supertypes, subtypes, and support for inheritance. *Constraints* provide a general mechanism for specifying the interdependencies of entities. A constraint is a formal rule, logical or mathematical or a mixture of these, specifying a condition that must hold in a correct configuration. A similar synthesis as [30] is based on a representation that employs Unified Modeling Language (UML) [31] with specific stereotypes and Object Constraint Language (OCL) [32], was proposed for modeling configuration knowledge [33-37]. The stereotypes include the connection-oriented and resource-oriented concepts along with a taxonomical hierarchy of component types [33-35, 37].

2.2 Problem solving

Numerous problem-solving methods have been applied to configuration tasks; several overviews of the topic exist. A recent overview of problem solving in configurators is provided in [38]. In their taxonomy of types of problem-solving methods for design and configuration, Wielinga and Schreiber [39] consider *configuration problem-solving methods* a subtype of *design methods.* Configuration problem-solving methods can be further divided into *knowledge-intensive methods* and *uniform methods*. Uniform methods apply the same reasoning methods to all problems, whereas knowledge-intensive methods use (explicitly modeled) knowledge to constrain and direct problem solving. Knowledge-intensive methods (*propose, critique, and modify; case based,* and *hierarchical*) are not considered further in this work: the authors consider uniform methods to already be mature enough for supporting the configuration tasks in sales configuration of many products and services.

Uniform methods include *constraint solving* and *logic-based methods*. Constraint satisfaction (CSP) and its extensions have gained significant popularity [12, 40, 41]. Many authors, e.g., Desisto [42] and Haag, Junker & O'Sullivan [43][2], consider constraint-based methods ideal for solving configuration problems. Constraint-based methods can be extended with preference programming. Here, the idea is to express preferences and to provide inference that supports finding solutions that maximally satisfy preferences in such a way that more important preferences are satisfied before less important ones [45].

Several logic-based methods have been applied to solve configuration problems successfully. These include direct programming in Prolog or through a higher-level modeling layer [46]. Description logics [47] have been applied [48-50]. Constraint logic programming has also been applied [51]. Furthermore, a method has been proposed to translate configuration domain modeling concepts into weight constraint rules [52, 53]. Following this idea, an experimental system, OOASP, showed the feasibility of checking a configuration, completing a configuration, and performing reconfiguration [54].

Sometimes different problem-solving methods have been combined, such as description logic with constraint satisfaction [11].

2.3 Other aspects

Principles of configurators include numerous less technical aspects. An overview of configuration systems and current topics is given in [55]. Here, we do not attempt to provide a full treatment of these aspects, and we recognize that there is still significant room for future research. Examples of identified configuration related research challenges include personalized configuration, community-based configuration (by a group of users), standardized configuration knowledge representations, intelligent user interfaces for configuration knowledge acquisition, intelligent testing and debugging, and unobtrusive preference elicitation [56]. To our knowledge, it is not common for generic configuration systems to directly support providing the user support capabilities proposed to avoid the product variety paradox [21]: focused navigation, flexible navigation, easy comparison, benefit-cost communication, and user-friendly product-space description capabilities. Many sales configurators even struggle on aspects like consistency checking [22]. However, the application of configurators in business and corresponding effects (e.g., on organization, processes, business performance), and configurator user interaction aspects are relevant and gaining momentum [21, 22, 57-61]. A number of books guide companies on information management required by mass customization, configurator classifications, and selecting a configurator [20, 59, 62].

3 Design Science and theory

The Design Science approach creates and evaluates IT artifacts intended to solve identified organizational problems [63]. The approach is gaining popularity as a framework for research of constructive nature.

[2] An essay in [44] that is based on the Configuration Workshop of the 17th European Conference on Artificial Intelligence (ECAI 2006).

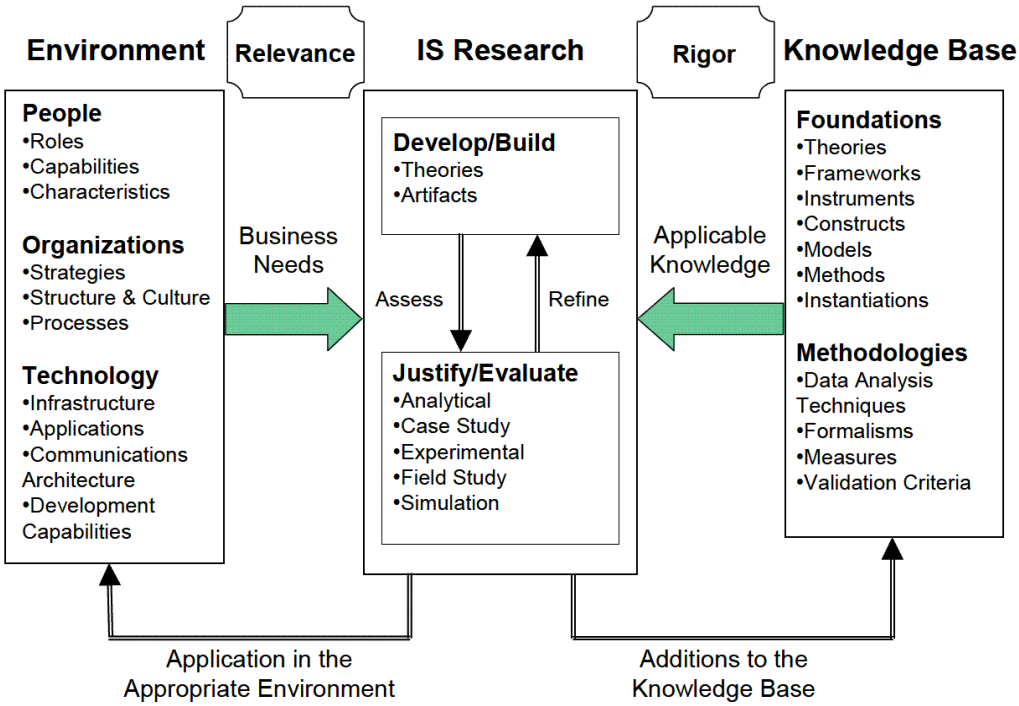


Figure 1. Information Systems Research Framework [63], redrawn

Hevner et al. [63] characterize the Design Science approach as follows (see Figure 1). The *environment* defines the problem space in which the phenomena of interest reside. In Information systems (IS) research, the environment consists of *people*, *organizations*, and *technology*. People in an organization perceive, assess, and evaluate *business needs* in the environmental context of their organization. The business needs perceived by the researcher stem from this context. Research *relevance* is assured by framing research to address business needs.

Design Science research is conducted through *building* and *evaluation* of *artifacts* designed to meet the identified business need, the ultimate goal being utility. The artifacts can be *constructs* (vocabulary and symbols), *models* (abstractions and representations), *methods* (algorithms and practices), or *instantiations* (implemented or prototype systems). Evaluation of an artifact often leads to refinements.

Research *rigor* stems from the appropriate use of the *knowledge base*. The knowledge base is formed by *foundations* used in the develop/build phase of research and *methodologies* used in the justify/evaluate phase. The knowledge base consists of previous contributions to IS research and related disciplines. *Contributions* in Design Science are assessed by their application to the identified business need in the appropriate environment.

Gregor [64] discussed the nature of theory in the discipline of Information Systems and presented five theory types (see Table 1). Of these, the most relevant to configuration research, and Design Science in more general, is theory type V: design and action, which "Says how to do something. The theory gives explicit prescriptions (e.g., methods, techniques, principles of form and function) for constructing an artifact." (p. 620). Continuing the idea, Gregor and Jones [23] posit that the primary output of Design Science is *Information Systems Design Theory* (ISDT). ISDT "shows the principles inherent in the design of an IS artifact that accomplishes some end, based on knowledge of both IT and human behavior. The ISDT allows the prescription of guidelines for further artifacts of the same type." Thus, contributions are not the artifacts themselves. Rather, contributions are more general prescriptions for artifacts of the same type. According to Gregor [64], a recipe-like *prescription* exists when theory enables an artifact to be constructed by describing a method or structure for its construction. Gregor and Jones [23] further refine the idea into *elements of information system theory*. They have identified 8 components; see **Table 2**.

Table 1. A Taxonomy of Theory Types in Information Systems Research [64](p. 620)

Theory Type	Distinguishing Attributes
I. Analysis	Says what is. The theory does not extend beyond analysis and description. No causal relationships among phenomena are specified and no predictions are made.
II. Explanation	Says what is, how, why, when, and where. The theory provides explanations but does not aim to predict with any precision. There are no testable propositions.
III Prediction	Says what is and what will be. The theory provides predictions and has testable propositions but does not have well-developed justificatory causal explanations.
IV. Explanation and prediction	Says what is, how, why, when, where, and what will be. Provides predictions and has both testable propositions and causal explanations.
V Design and action	Says how to do something. The theory gives explicit prescriptions (e.g., methods, techniques, principles of form and function) for constructing an artifact.

Table 2 Components of Information Systems Design Theory [23] and Sales Configurator Information Systems Design Theory (SCISDT).

Component	ISDT component Description [23]	SCISDT component description (as explicated by WeCoTin)
Core components		
1) Purpose and scope	"What the system is for," the set of meta-requirements or goals that specifies the type of artifact to which the theory applies and in conjunction also defines the scope, or boundaries, of the theory.	A web-based sales configurator that fulfills a set of major requirements
2) Constructs	Representations of the entities of interest in the theory.	Concepts of configuration knowledge [30], product configuration modeling language PCML, weight constraint rule language.
3) Principle of form and function	The abstract "blueprint" or architecture that describes an IS artifact, either product or method / intervention.	A high-level architecture and main functions of components was presented along with main working principles [2, 65, 66]
4) Artifact mutability	The changes in state of the artifact anticipated in the theory, that is, what degree of artifact change is encompassed by the theory.	WeCoTin has several internal interfaces that enable replacement of major components. It has also been designed to be flexible in numerous aspects, such as different ways to determine prices, and support for several languages.
5) Testable propositions	Truth statements about the design theory.	The main propositions were capability to model and configure real products. Another proposition is adequate performance. These aspects were tested with highly satisfactory results.
6)Justificatory knowledge	The underlying knowledge or theory from the natural or social or design sciences that gives a basis and explanation for the design (kernel theories).	The modeling constructs of PCML were given clear formal semantics by mapping them to the weight constraint rule language. This mapping also enables sound and complete inference by the Smodels system.
Additional components		
7) Principles of implementation	A description of processes for implementing the theory (either product or method) in specific contexts.	To be discussed in an extended version of this paper.
8) Expository instantiation	A physical implementation of the artifact that can assist in representing the theory both as an expository device and for purposes of testing.	WeCoTin. To be discussed in an extended version of this paper.

4 WeCotin and Sales Configurator Information Systems Design Theory

4.1 WeCoTin sales configurator

WeCoTin consists of two main components: a graphical modeling environment *Modeling Tool* and a web-based application *WeCoTin Configuration Tool* that supports the configuration task. WeCoTin Configuration Tool enables users to configure products over the web using a standard browser. The user interface for end users is dynamically generated.

WeCoTin Modeling Tool is used for creating and editing configuration models and additional information needed to generate a user interface for end users.

Configuration models are expressed in *Product Configuration Modeling Language* (PCML). PCML is object-oriented and declarative. PCML is conceptually based on a function-oriented subset of the configuration knowledge conceptualization of Soininen et al. [30].

WeCoTin is computationally well founded because it was constructed based on the idea of translation of configuration knowledge into weight constraint rules [52, 53]. In addition, WeCoTin incorporates tools that allow graphical configuration modeling, semi-automatic generation of user interfaces, and several other aspects that ease long-term management.

WeCoTin is implemented using the Java 2 Platform and Java programming language, except for the component Inference Engine, which consists of smodels and lparse programs of the Smodels system that are implemented in C++, and user interface components that employ some JavaScript to generate the HTML and CSS-based web-based UI. XML is applied for some user interface definitions, price lists, and calculation definitions.

4.2 Purpose and scope

Companies with a mass customization strategy need to provide choice navigation capability [67]. Configurators are the primary means to this end. In the scope of this work, generic configurators, aka *configuration toolkits,* enable the creation of configurator instantiations for individual companies or product lines. Configurators can provide numerous other benefits. On the other hand, taking a configurator into use, and operating and keeping it up to date, also incurs significant costs; the total cost of configurator ownership should be justifiable.

Although there are numerous individual configurator instantiations and generic-purpose configurators that enable such instantiations to be created, it was deemed that none met all the desirable properties that we considered important. The requirements are summarized in [2, 66] and they include: A (sales) configurator should enable

- easy set-up without programming (excluding integrations),
- fluent modeling of products by product experts based on a well-founded high-level modeling conceptualization,
- easy maintenance of configuration knowledge.

In addition, we wanted to experiment with applying answer set programming for problem solving combined with a higher-level configuration modeling and consistent and complete inference.

4.3 Constructs

ISDT *constructs* represent the entities that are of interest in the theory, and corresponding terms should be defined as clearly as possible [23].

In the context of this work, it is somewhat challenging to draw the line between the constructs and principles of form and function. Relevant constructs include at least the conceptualization of configuration knowledge, and object-oriented product configuration modeling language (PCML). A sales configurator (WeCoTin) as a whole and its major parts (Modeling Tool, Configuration Tool) also belong to the relevant constructs.

Underlying these as subsystems are the inference engine *Smodels* [68], its modeling language *weight constraint rule language (WCRL),* and the method of translating configuration knowledge to WCRL [53]. These underlying subsystems were developed outside the scope of the WeCoTin construction.

It is noteworthy that the conceptualization was constructed in such a way that that it retains the natural thinking patterns used in companies to describe the variation of product families. Compositional structure of products and configurable attributes are the main mechanisms for capturing variability. Taxonomy with inheritance generalizes the approach. The full conceptualization also supports connection-oriented constructs and resources that have proven to be useful in earlier work. All these can be given formal semantics by mapping them to a formal language.

4.4 Principle of form and function

Principles of form and function "define the structure, organization, and functioning of the design product or design method. The shape of a design product is seen in the properties, functions, features, or attributes that the product possesses when constructed" [23].

A configurator should have separate environments for the modelers and end users—the concerns are separate. Nevertheless, WeCoTin offers the modeler the capability to rapidly test the created or edited configuration model.

WeCoTin was built on a layered architecture. We propose this as a significant principle of configurator construction. This provided a clear separation of

- formal inference, which in this case is logic-based;
- high-level modeling constructs, which match how the product experts think of configuration and yet can be provided with formal semantics and automatically mapped to a form suitable for inference; and
- the end-user interface, creation of which does not require programming, but is, for example, generated utilizing the high-level modeling language.

The main functions of a configurator include checking for the consistency and completeness of a configuration, with the capability to prevent from ordering a product based on an incomplete or inconsistent configuration. Price is an integral element that must be managed within the scope of a configuration task.

A hierarchy of modeling languages needs to match the layered architecture. In the case of WeCoTin, the high-level configuration modeling language (PCML) is aimed to be adequate for modelers. This is compiled into a formal weight constraint rule language with variables. Finally, WCRL is compiled into a simple basic constraint rule language without variables. This principle provides theoretical grounding and allows for sound and complete inference.

We feel that future configurators should support recommendation functionality to support users with choice navigation. Case-based recommendation approaches seem to be potentially viable (e.g. [69]), but further research is required. Future sales configurator ISDTs should address user interaction more thoroughly, e.g. along the lines of [21, 22].

4.5 Artifact mutability

WeCoTin has several internal interfaces that enable replacement of major components. For example, Smodels could be relatively easily replaced with another inference engine based on answer set programming. There are interfaces for configuration model manipulation and manipulation of configurations. These make it easier to create different modeling environments and user interfaces for end users.

WeCoTin has also been designed to be flexible in numerous respects, such as different ways to determine prices, and built-in support for several end-user languages and tax models. Product changes do not require programming changes in the user interface for end users: a template gives the general visual appearance, and WeCoTin generates the product-specific part (the modeler can change the input control types and determine their sequencing).

However, architectural mutability and suitability for generic tasks including dimensioning and connections could potentially be higher. Generic dimensioning tasks would require integrating additional inference or calculation mechanisms; user-specified connections would require appropriate user interface support. In some configuration tasks, a dynamically determined flow of the configuration process based on previous answers would be necessary. There are no specific provisions for these needs.

4.6 Testable propositions

The main propositions were capability to model and configure real products and adequate performance in this context. These aspects were tested with highly satisfactory results.

Created 26 sales configuration models were characterized in terms of size and modeling constructs that were applied [70]. The sales configuration view of 14 real-world products was modeled in their entirety (some with extra demonstration features, one in 2 variants), and 8 partial products or concepts. These offerings came from 10 organizations representing machine industry, healthcare, telecommunications services, insurance services, maintenance services, software configuration, and construction. The created models were small, but representative of the Finnish industry. Among larger models was 'Broadband' that had 66 feature types, 453 effective attributes (the sum of inherited and locally defined attributes in concrete types) and 43 type level "generic" constraints. A semi-automatically generated Linux model had 626 feature types, 4369 effective attributes, and 2380 constraints.

WeCoTin had demonstrably adequate performance with the four models that were systematically tested [71]. We obtained additional performance evaluation by configuring all the characterized products using the WeCoTin user interface (Linux only partially) with a 2.4 GHz Intel Core 2 Duo laptop. All configuration models had a feeling of instant response, except the "Broadband" model's response time was slightly more than 3 seconds before an attribute with 436 possible values was specified, after which the response time decreased to less than a second. Linux was too slow to be

usable. Also, the compilation time from PCML to WCRL and then to BCRL was very satisfactory: a script that compiled all the characterized configuration models, except Linux, and a few additional test and sample models ran in 32 seconds. For the Linux model, achieving sufficient performance would require at least the capability to control when full inference (with finding a configuration) is performed, and possibly other optimizations.

Using WCRL and Smodels to provide inference seems to be a feasible proposition for building a sales configurator. The typical approach in previous work has been based on constraint satisfaction.

4.7 Justificatory knowledge

The configuration knowledge conceptualization is based on a synthesis of previous work and additional experiences from interviews in ten companies and two case studies [72-75].

PCML allows the variability of products to be expressed on a high level that product experts can understand. Furthermore, the modeling constructs of PCML were given clear formal semantics by being mapped to a weight constraint rule language. This mapping enables sound and complete inference by the Smodels system, giving a foundation to the claim that, if a sales configurator is built on such well-founded principles, a working sales configurator can be implemented.

New methods of characterizing configuration models and measuring configurator performance were developed [70, 71].

Numerous configuration models based on the variability of real offerings were developed [2, 70]. These show how WeCoTin could be applied in respective companies to provide choice navigation support.

5 Conclusions

In this paper, we presented, to our knowledge, the first attempt to construct an Information System Design in the context of configuration systems. An ISDT for sales configurators (SCISDT) fulfilling a set of major requirements was presented. SCISDT is based on the design of WeCoTin, a sales configurator that supports mass customization of complex products.

The main components of SCISDT are as follows. The purpose and scope are to construct a web-based sales configurator that fulfills a set of major requirements. The major constructs include a high-level object-oriented configuration modeling language that is based on a well-founded conceptualization that can be mapped to a language with an inference engine to support the configuration task. The principles of form and function include a high-level layered architecture with a matching hierarchy of modeling languages. Artifact mutability includes several internal interfaces and built-in flexibility with respect to numerous aspects that allow for application of the constructed sales configurator more widely than for one specific domain only. The main testable propositions are capability to model and configure real products and adequate performance. Justificatory knowledge includes providing the major modeling constructs clear formal semantics by mapping them to a language with appropriate formal semantics and support for the required inference capabilities.

Although we specifically addressed sales configurators, the Design Science approach can potentially be applied in other configuration related contexts. The authors view that applying the Design Science approach can help to ensure the rigor and relevance of configuration research. Contributions can be the additions to the knowledge base as suggested by Hevner et al. [63], or (ISDT) theories.

Acknowledgements

We thank DIGILE, TEKES and related companies for financial support; this work has been partially funded by DIGILE SHOK program Need 4 Speed (N4S). We also express our gratitude to companies that have offered us access in the context of earlier research that this work is based on.

References

[1] J. Tiihonen, *Support for Configuration of Physical Products and Services. Manuscript Submitted to pre-examination for the degree of Doctor of Science (Technology).* Helsinki, Finland: Unigrafia, 2014.

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

[3] J. McDermott, "R1: A Rule-based configurer of computer systems," *Artificial Intelligence,* vol. 19 (1), pp. 39-88, 1982.

[4] F. Frayman and S. Mittal, "COSSACK: A constraint-based expert system for configuration tasks," in *Knowledge-Based Expert Systems in Engineering: Planning and Design*, D. Sriram and R. A. Adey, Eds. Woburn, MA, USA: Computational Mechanics Publications, 1987, pp. 143-166.

[5] R. Cunis, A. Günter, I. Syska, H. Peters and H. Bode, "PLAKON - an approach to domain-independent construction," in *Proceedings of the Second International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE - 89),* Tullahoma, TN, USA, 1989, pp. 866-874.

[6] R. Cunis, A. Günter and H. Strecker, Eds., *The PLAKON-Book.* London, UK: Springer-Verlag, 1991.

[7] A. Günter and L. Hotz, "KONWERK - A domain independent configuration tool," in *Configuration Papers from the AAAI Workshop, 1999. AAAI Technical Report WS-99-05,* 1999, pp. 125-126.

[8] L. Hotz and A. Günter, "Konwerk," in *Knowledge-Based Configuration - from Research to Business Cases*, 1st ed., A. Felfernig, L. Hotz, C. Bagley and J. Tiihonen, Eds. Waltham, MA, USA: Morgan Kaufmann Publishers, 2014, pp. 281-295.

[9] M. Stumptner, A. Haselböck and G. E. Friedrich, "COCOS - a tool for constraint-based, dynamic configuration," in *10th IEEE Conference on Artificial Intelligence for Applications (CAIA-94),* San Antonio, TX, USA, 1994, pp. 373-380.

[10] H. L. Hales, "Automating and integrating the sales function: how to profit from complexity and customization," *Enterprise Integration Strategies,* vol. 9 (11), pp. 1-9, 1992.

[11] U. Junker and D. Mailharro, "The logic of ILOG (J)configurator: Combining constraint programming with a description logic," in *18th International Joint Conference on Artificial Intelligence (IJCAI-03), Configuration Workshop,* Acapulco, Mexico, 2003, pp. 13-20.

[12] D. Mailharro, "A classification and constraint-based framework for configuration," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM),* vol. 12 (4), pp. 383-397, 1998.

[13] A. Anderson, *Towards Tool-Supported Configuration of Services.* M. Sc. thesis, Espoo: Helsinki University of Technology, Department of Computer Science and Engineering, 2005.

[14] A. Haag, "Sales configuration in business processes," *IEEE Intelligent Systems,* vol. 13 (4), pp. 78-85, 1998.

[15] A. Haag, ""“Dealing” with configurable products in the SAP business suite," in *19th International Joint Conference on Artificial Intelligence (IJCAI-05), Configuration Workshop,* Edinburgh, Scotland, UK, 2005, pp. 68-71.

[16] S. R. Damiani, T. Brand, M. Sawtelle and H. Shanzer, "Oracle configurator developer user’s guide, release 11i," 2001.

[17] M. Sawtelle, "Oracle Configurator: Fusion Configurator Engine Guide, Release 12.1," 2010.

[18] Oracle, "Peoplesoft Enterprise Configurator - Oracle Data Sheet," vol. 2009, 2005.
[19] Oracle. Siebel product & catalog management - oracle data sheet. 2007. Available: http://www.oracle.com/us/products/applications/siebel/036241.pdf.
[20] L. Hvam, N. H. Mortensen and J. Riis, *Product Customization.* New York: Springer, 2008.
[21] A. Trentin, E. Perin and C. Forza, "Sales configurator capabilities to avoid the product variety paradox: Construct development and validation," *Comput. Ind.,* vol. 64 (4), pp. 436-447, 2013.
[22] E. K. Abbasi, A. Hubaux, M. Acher, Q. Boucher and P. Heymans, "The anatomy of a sales configurator: An empirical study of 111 cases," in *Advanced Information Systems Engineering - 25th International Conference, CAiSE 2013,* Valencia, Spain, 2013, pp. 162-177.
[23] S. Gregor and D. Jones, "The anatomy of a design theory," *Journal of the Association for Information Systems,* vol. 8 (5), pp. 312-335, 2007.
[24] M. Stumptner, "An overview of knowledge-based configuration," *AI Communications,* vol. 10 (2), pp. 111-125, 1997.
[25] D. Sabin and R. Weigel, "Product configuration frameworks — a survey," *IEEE Intelligent Systems,* vol. 13 (4), pp. 42-49, 1998.
[26] A. Hubaux, D. Jannach, C. Drescher, L. Murta, T. Männistö, K. Czarnecki, P. Heymans, T. Nguyen and M. Zanker, "Unifying software and product configuration: A research roadmap," in *Proceedings of the Workshop on Configuration at ECAI 2012 (ConfWS'12),* Montpellier, France, 2012, pp. 31-35.
[27] S. Mittal and F. Frayman, "Towards a generic model of configuration tasks," in *11th International Joint Conference on Artificial Intelligence (IJCAI-89),* Detroit, Michigan, USA, 1989, pp. 1395-1401.
[28] M. Heinrich and E. W. Jüngst, "A resource-based paradigm for the configuring of technical systems from modular components," in *Seventh IEEE Conference on Artificial Intelligence Applications (CAIA-91),* Miami Beach, FL, USA, 1991, pp. 257-264.
[29] O. Najmann and B. Stein, "A theoretical framework for configuration," in *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: 5th International Conference (IEA/AIE-92),* Paderborn, Germany, 1992, pp. 441-450.
[30] T. Soininen, J. Tiihonen, T. Männistö and R. Sulonen, "Towards a general ontology of configuration," *AI EDAM,* vol. 12 (4), pp. 357-372, 1998.
[31] J. Rumbaugh, I. Jacobson and G. Booch, *The Unified Modeling Language Reference Manual.* Reading, MA, USA: Addison-Wesley, 1999.
[32] J. Warmer and A. Kleppe, *The Object Constraint Language: Getting Your Models Ready for MDA.* Boston, MA, USA: Addison-Wesley, 2003.
[33] A. Felfernig, G. E. Friedrich and D. Jannach, "Generating product configuration knowledge bases from precise domain extended UML models," in *12 Th International Conference on Software Engineering and Knowledge Engineering (SEKE 2000),* Chicago, IL, USA, 2000, pp. 284-293.
[34] A. Felfernig, G. E. Friedrich and D. Jannach, "UML as domain specific language for the construction of knowledge-based configuration systems," *International Journal of Software Engineering and Knowledge Engineering,* vol. 10 (4), pp. 449-469, 2000.
[35] A. Felfernig, G. Friedrich, D. Jannach and M. Zanker, "Configuration knowledge representation using UML/OCL," in *UML 2002 — the Unified Modeling Language - Model Engineering, Concepts, and Tools, 5th International Conference, Proceedings (LNCS),* Dresden, Germany, 2002, pp. 49-62.
[36] A. Felfernig, G. Friedrich, D. Jannach, M. Stumptner and M. Zanker, "Configuration knowledge representations for Semantic Web applications," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM),* vol. 17 (1), pp. 31-50, 2003.
[37] A. Felfernig, "Standardized configuration knowledge representations as technological foundation for mass customization," *Engineering Management, IEEE Transactions On,* vol. 54 (1), pp. 41-56, 2007.
[38] L. Hotz, A. Felfernig, M. Stumptner, A. Ryabokon, C. Bagley and K. Wolter, "Configuration knowledge representation and reasoning," in *Knowledge-Based Configuration - from Research to Business Cases*, A. Felfernig, L. Hotz, C. Bagley and J. Tiihonen, Eds. Waltham, MA, USA: Morgan Kaufmann, 2014, pp. 41-72.
[39] B. Wielinga and G. Schreiber, "Configuration-design problem solving," *IEEE Expert,* vol. 12 (2), pp. 49-56, 1997.
[40] S. Mittal and B. Falkenhainer, "Dynamic constraint satisfaction problems," in *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90),* Boston, MA, USA, 1990, pp. 25-32.
[41] G. Fleischanderl, G. E. Friedrich, A. Haselböck, H. Schreiner and M. Stumptner, "Configuring large systems using generative constraint satisfaction," *IEEE Intelligent Systems,* vol. 13 (4), pp. 59-68, 1998.
[42] R. P. Desisto, "Constraints still key for product configurator deployments," Gartner, Inc., Stamford, CT, USA, Tech. Rep. T-22-9419, 1 June, 2004. 2004.
[43] A. Haag, U. Junker and B. O'Sullivan, "Explanation in product configuration," *IEEE Intelligent Systems,* vol. 22 (1), pp. 83-85, 2007.
[44] C. Sinz, A. Haag, N. Narodytska, T. Walsh, E. Gelle, M. Sabin, U. Junker, B. O'Sullivan, R. Rabiser, D. Dhungana, P. Grunbacher, K. Lehner, C. Federspiel and D. Naus, "Configuration," *IEEE Intelligent Systems,* vol. 22 (1), pp. 78-90, 2007.
[45] U. Junker and D. Mailharro, "Preference programming: Advanced problem solving for configuration," *AI EDAM,* vol. 17 (1), pp. 13-29, 2003.
[46] D. B. Searls and L. M. Norton, "Logic-based configuration with a semantic network," *The Journal of Logic Programming,* vol. 8 (1-2), pp. 53-73, 1990.
[47] F. Baader, "Description logics," in *Reasoning Web. Semantic Technologies for Information Systems*, S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M. Rousset and R. A. Schmidt, Eds. Berlin, Heidelberg: Springer, 2009, pp. 1-39.
[48] 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,* vol. 14 (3), pp. 69-80, 1993.
[49] J. R. Wright, D. L. McGuinness, C. H. Foster and G. T. Vesonder, "Conceptual modeling using knowledge representation: Configurator applications," in *14th International Joint Conference on Artificial Intelligence (IJCAI-95), Workshop on Artificial Intelligence in Distributed Information Networks,* Montreal, Quebec, Canada, 1995, .
[50] D. L. McGuinness and J. R. Wright, "An industrial-strength description logic-based configurator platform," *IEEE Intelligent Systems,* vol. 13 (4), pp. 69-77, 1998.
[51] N. Sharma and R. Colomb, "Mechanising shared configuration and diagnosis theories through constraint logic programming," *The Journal of Logic Programming,* vol. 37 (1-3), pp. 255-283, 1998.
[52] T. Soininen, *An Approach to Knowledge Representation and Reasoning for Product Configuration Tasks.* Ph.D. thesis, Espoo, Finland: Helsinki University of Technology, Department of Computer Science and Engineering, 2000.
[53] T. Soininen, I. Niemelä, J. Tiihonen and R. Sulonen, "Representing configuration knowledge with weight constraint rules," in *AAAI Spring Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge (AAAI Technical Report SS-01-01),* Stanford University, CA, USA, 2001, pp. 195-201.
[54] G. Schenner, A. Falkner, A. Ryabokon and G. E. Friedrich. Solving object-oriented configuration scenarios with ASP. Presented at 15th International Configuration Workshop. 2013, Available: http://ws-config-2013.mines-albi.fr/CWS-2013-Proceedings-Color.pdf.
[55] A. Felfernig, L. Hotz, C. Bagley and J. Tiihonen, *Knowledge-Based Configuration: From Research to Business Cases.* Waltham, MA, USA: Morgan Kaufmann, 2014.
[56] A. Felfernig, L. Hotz, C. Bagley and J. Tiihonen, "Chapter 15 - configuration-related research challenges," in *Knowledge-Based Configuration*, A. Felfernig, L. Hotz, C. Bagley and J. Tiihonen, Eds. Boston: Morgan Kaufmann, 2014, pp. 191-195.
[57] C. Forza and F. Salvador, "Managing for variety in the order acquisition and fulfilment process: The contribution of product configuration systems," *Int J Prod Econ,* vol. 76 (1), pp. 87-98, 2002.
[58] C. Forza and F. Salvador, "Product configuration and inter-firm coordination: an innovative solution from a small manufacturing enterprise," *Comput. Ind.,* vol. 49 (1), pp. 37-46, 2002.
[59] T. Blecker, G. Friedrich, B. Kaluza, N. Abdelkafi and G. Kreutler, *Information and Management Systems for Product Customization.* Boston: Springer, 2005.
[60] M. Heiskala, K. Paloheimo and J. Tiihonen, "Mass customization with configurable products and configurators: A review of benefits and challenges," in *Mass Customization Information Systems in Business*,

1st ed., T. Blecker and G. Friedrich, Eds. Hershey, PA, USA & London, UK: IGI Global, 2007, pp. 1-32.
[61] F. Salvador and C. Forza, "Principles for efficient and effective sales configuration design," *International Journal of Mass Customisation,* vol. 2 (1-2), pp. 114-127, 2007.
[62] C. Forza and F. Salvador, *Product Information Management for Mass Customization: Connecting Customer, Front-Office and Back-Office for Fast and Efficient Customization.* Hampshire, UK; New York, NY, USA: Palgrave Macmillan, 2006.
[63] A. R. Hevner, S. T. March, J. Park and S. Ram, "Design science in information systems research," *MIS Quarterly,* vol. 28 (1), pp. 75-105, 2004.
[64] S. Gregor, "The nature of theory in information systems," *MIS Quarterly,* vol. 30 (3), pp. 611-642, 2006.
[65] A. Anderson and M. Pasanen, "WeCoTin Requirements and architecture (unpublished)," 2003.
[66] J. Tiihonen, T. Soininen, I. Niemelä and R. Sulonen, "A practical tool for mass-customising configurable products," in *Proceedings of the 14th International Conference on Engineering Design,* Stockholm, Sweden, 2003, pp. CDROM, paper number 1290, 10 pp.
[67] F. Salvador, P. M. de Holan and F. T. Piller, "Cracking the code of mass customization," *MIT Sloan Management Review,* vol. 50 (3), pp. 71-78, 2009.
[68] P. Simons, I. Niemelä and T. Soininen, "Extending and implementing the stable model semantics," *Artif. Intell.,* vol. 138 (1-2), pp. 181-234, 2002.
[69] A. Felfernig, M. Mandl, J. Tiihonen and M. Schubert. Personalized product configuration. Presented at Multikonferenz Wirtschaftsinformatik 2010, 24. PuK-Workshop: Planung/Scheduling Und Konfigurieren/Entwerfen. 2010, Available: http://webdoc.sub.gwdg.de/univerlag/2010/mkwi/.
[70] J. Tiihonen, "Characterization of configuration knowledge bases," in *19th European Conference on Artificial Intelligence (ECAI-2010), Workshop on Intelligent Engineering Techniques for Knowledge Bases (IKBET),* Lissabon, Portugal, 2010, pp. 13-20.
[71] J. Tiihonen, T. Soininen, I. Niemelä and R. Sulonen, "Empirical testing of a weight constraint rule based configurator," in *15th European Conference on Artificial Intelligence (ECAI-2002), Configuration Workshop,* Lyon, France, 2002, pp. 17-22.
[72] J. Tiihonen, *Computer-Assisted Elevator Configuration.* M.Sc (Eng.) thesis, Espoo: Helsinki University of Technology, Department of Computer Science, 1994.
[73] T. Soininen and J. Tiihonen, "Sales configurator in Datex procuct data management process," 1995.
[74] J. Tiihonen and T. Soininen, "Product configurators - information system support for configurable products," Helsinki University of Technology, Espoo, Tech. Rep. TKO-B 137, 1997.
[75] J. Tiihonen, *National Product Configuration Survey — customer Specific Adaptation in the Finnish Industry.* Licentiate of technology (Eng.) thesis, Espoo: Helsinki University of Technology, Department of Computer Science, Laboratory of Information Processing Science, 1999.

Open Configuration: a New Approach to Product Customization

Linda L. Zhang[1] and **Xiaoyu Chen**[*1, 2] and **Andreas Falkner**[3] and **Chengbin Chu**[2]

Abstract. State-of-the-art product configuration enables companies to deliver customized products by selecting and assembling predefined configuration elements based on known relationships. This paper introduces an innovative concept, open configuration, in order to assist companies in configuring products that correspond exactly to what customers want. Superior to product configuration, open configuration involves both predefined configuration elements and new ones in configuring customized products. As a first step, this study explains the concept of open configuration and the basic principles. It also discusses in detail the challenges involved in open configuration, such as conceptual model development, open configuration optimization, and open configuration knowledge representation.

1 INTRODUCTION

With the advancement of design and manufacturing technologies, customers are no longer satisfied with standardized products. They increasingly demand products that could satisfy their individual needs. As a result, companies need to timely offer customized products at affordable costs to survive [1]. With traditional design approaches, companies cannot efficiently develop customized products [2, 3]. Product configuration has been proposed to enable companies to deliver customized products at low costs with short delivery times. Product configuration has been widely applied to a variety of industries, including computer, telecommunication systems, transportation, industrial products, medical systems and services [4]. It brings companies a number of advantages in delivering required products. These advantages include managing product variety [5], shortening delivery time [6], improving product quality [7], simplifying order acquisition and fulfilment activities [8], etc.

Product configuration has received much attention from industrial and academia alike. Researchers have approached product configuration from different perspectives and have developed diverse methods, methodologies, approaches, and algorithms to solve different configuration issues and problems. In spite of the diversities among these solution tools, they are developed based on a common assumption: the configuration elements, such as components, modules, attributes, functions, and their relationships are predefined. In relation to this assumption, the products that can be configured are known in principle even if not explicitly listable [2]. In this regard, product configuration cannot deal with such products that demand new functions and components in addition to the predefined ones. In another word, it cannot configure customized products in a true sense, i.e., to the full extent that it covers all reasonable and unforeseen customer requirements.

This study proposes an innovative concept 'open configuration' in order to help companies configure such products that can meet both predefined and unforeseen customer requirements, that is, to meet customer requirements as complete as possible without making too much compromise (see Section 2). In this regard, in configuring customized products, open configuration deals with not only the addition of new configuration elements, such as functions, components, but also the modification of existing configuration elements, more specifically components. Existing component modification is to accommodate the integration of new components with the predefined ones.

In the rest of this paper, Section 2 uses a fridge configuration example to illustrate the limitation of product configuration, i.e., the product configured lie in a known range in accordance with the predefined components. Section 3 introduces the concept of open configuration, its basic principles, and its process. Section 4 sheds lights on the challenges involved in open configuration. We end the paper in Section 5 by pointing out the ongoing research that we are working on.

2 PRODUCT CONFIGURATION

As a special design activity, product configuration capitalizes on design results, such as components, attributes and their relationships [9, 10]. It entails such a process that based on given customer requirements, suitable components are selected from the set of predefined component types; the selected components are evaluated and further arranged into products according to the configuration constraints and rules.

Take fridge configuration as an example. Assume in this example, there are 6 component types, including *Refrigerator (R)*, *Freezer (F)*, *Freezer drawer (Fd)*, *Variable compartment (V)*, *Base (B)*, *Outer casing (O)*. Each component type is defined by a set of attributes (number, size, price) and each attribute can assume a number of values. Table 1 summarizes these component types, the attributes, and attribute values.

For example, $N_R:(1,2)$ represents the number of *Refrigerators* in one fridge can be 1 or 2; $S_R:(\text{small, medium, large, extra-large})$ indicates the component *Refrigerator* has four different sizes: small, medium, large, extra-large. Price mentioned hereinafter states the price of the configured fridge.

[1] IESEG School of Management (LEM-CNRS), Lille-Paris, France
[2] Ecole Centrale Paris (Laboratoire Genie Industriel), Paris, France
[3] Siemens AG Österreich, Vienna, Austria
[*] Corresponding author: x.chen@ieseg.fr

Table 1. The attributes of the fridge components.

Component types	Number	Size	Price
Refrigerator	1-2	small, medium, large, extra-large	depending on size
Freezer	0-1	small, large, extra-large	depending on size
Freezer drawer	0-2	small	P(*Fd*) (i.e., a fixed price)
Variable compartment	0-1	small	P(*V*) (i.e., a fixed price)
Base	1	standard, wide	depending on size
Outer casing	1	standard, wide	depending on size

There are relationships among components, among attributes, and between components and attributes. For examples, $\{S_R = \text{large}, N_F = 1\} \rightarrow \{S_F = \text{small}\}$ means if one large sized *Refrigerator* and one *Freezer* are selected, the size of the *Freezer* is small; $N_{Fd} \neq 0 \rightarrow \{N_R = 2, S_R = \text{medium}\}$ states that if the component *Freezer drawer* is selected then two medium *Refrigerators* are required. The other relationships include: $\{S_R = \text{medium}, N_F = 0\} \rightarrow N_R = 2$; $\{S_F = \text{small}, S_R = \text{small}\} \rightarrow N_V = 1$; $\{S_R = \text{extra-large}, N_F = 1\} \rightarrow \{S_F = \text{extra-large}\}$; $\{S_F = \text{extra-large}\} \rightarrow \{S_B = \text{wide}, S_O = \text{wide}\}$; $\{N_V = 1, N_F = 0\} \rightarrow \{N_R = 1, S_R = \text{large}, S_V = \text{small}\}$; $\{S_R = \text{small}, N_F = 1\} \rightarrow \{S_F = \text{large}\}$.

There are four additional rules, including (1) $(N_R + N_V + N_F) \leq 3$, meaning the total number of *Refrigerator*, *Variable compartment*, and *Freezer* in one fridge should be no more than 3, (2) $N_R = 2 \rightarrow N_V + N_F = 0$, indicating if two *Refrigerators* are selected, the number of *Freezer* and *Variable compartment* is zero, (3) $N_{Fd} + N_F = 0$ representing that *Freezer* cannot be selected together with *Freezer drawer*, and (4) $N_{Fd} + N_V = 0$ indicating that *Freezer drawer* cannot be selected together with *Variable compartment*.

According to the above pre-defined components and their relationships, only 17 fridge configurations are available as possible solutions. While Fig. 1 shows 8 fridge configurations due to the space issue, different positions of components in Fig. 1.c, Fig. 1.d, Fig. 1.e, Fig. 1.f, and Fig. 1.g lead to the other 9 fridge configurations. All customized fridges to be configured based on customer requirements fall into this range of configuration solutions. (Note: Fridges from the left to the right are arranged based on the increase of price.) Take fridge f in Fig. 1 as an example to explain the components and their attributes in the configuration solution. This fridge configuration is represented as $FC_f = \{R:1, \text{small}; V:1, \text{small}; F:1, \text{small}; B:1, \text{standard}; O:1, \text{standard}\}$. It has one small *Refrigerator* on top, one small *Variable compartment* in the middle, one small *Freezer* at the bottom, one standard *Base*, and one standard *Outer casing*.

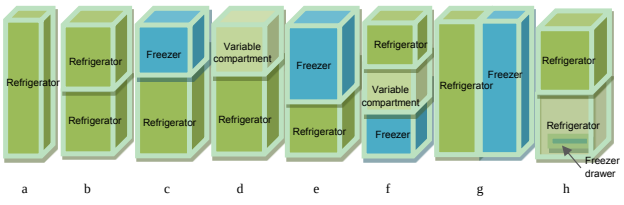


Figure 1. Fridge configuration solutions

Suppose the requirements from a customer include a cheaper fridge with a freezer and a large refrigerator. In accordance with these requirements, the constraints can be modeled as $\{R:1, \text{large}; N_F = 1; \min P\}$. The configured fridge must satisfy these constraints and additional rules mentioned earlier while fulfilling the customer requirements. In this regard, the constraints $\{R:1, \text{large}\}$ and $\{N_F = 1\}$ limit the possible choices to: $\{FC_c, FC_e\}$, i.e., the configuration solutions shown in Figs. 1.c and 1.e. The cost constraint $\{\min P\}$ indicating the minimal price results in the final solution to be $FC_c = \{R:1, \text{large}; F:1, \text{small}; B:1, \text{standard}; O:1, \text{standard}\}$.

As only predefined elements are involved, product configuration fails to provide customized products in a true sense or provides these products which can meet unforeseen customer requirements. Take the above fridge configuration as an example. Suppose that the requirements from another customer include any of the following:

- a fridge consisting of only one medium refrigerator,
- a fridge consisting of 2 freezers,
- an outer casing with a special color, and
- a cheaper fridge to be moved easily and with at least one freezer drawer.

In general, the first two requirements violate some predefined constraints (although the first one requires a new - lower - type of outer casing as a side-effect); the last two introduce new concepts. In more detail, the third requirement requires a new attribute value for the component outer casing. The last one is more complex. A part of it, i.e., being cheaper and with one freezer drawer, can be fulfilled by the predefined functions and components, while the rest cannot be fulfilled by the available functions, thus calling for a new function: 'to be movable'. This new function, in turn, needs new components, such as 'wheels', 'brakes', etc., which are necessary for delivering this function. Because of the lack of these components, product configuration can provide the customer with one of the fridges shown in Fig. 1 without satisfying all his requirements. The customer, thus, has to accept this fridge by making compromise (e.g., accept a cheapest fridge with a freezer drawer, which cannot be moved easily).

3 OPEN CONFIGURATION

In order to help companies configure customized products that correspond exactly to what a customer requires, this paper puts forward the concept of open configuration. The basic principle and general process of open configuration are introduced below.

3.1 Open configuration concept

Built on top of product configuration, open configuration is to configure customized products to meet customer requirements in a true sense. Similar as product configuration, it utilizes design results, selects components, and arranges the selected components according to constraints and rules. In extension to product configuration, it involves new component design, more specifically the specification of functions and the selection of the corresponding components. In addition, it deals with the modification of the predefined components, which allows the integration of new configuration elements.

3.2 Open configuration overview and process

Open configuration involves two types of knowledge: predefined knowledge and dynamic knowledge. Predefined knowledge relates

to predefined functions, components, and relationships; dynamic knowledge is associated with newly defined elements. In relation to these customer requirements, which can be fulfilled by the predefined functions (i.e., Type Ⅰ requirements in Fig. 2), the corresponding components are selected, while for these requirements, which cannot be fulfilled by the predefined functions (i.e., Type Ⅱ requirements in the figure), new functions and corresponding components are specified. The specification of these new configuration elements contributes to the extension of the dynamic knowledge. The relationships among the predefined elements and the newly defined elements are specified as well. This specification contributes to the interaction between the predefined knowledge and the dynamic knowledge. By respecting the constraints embedded in both the predefined and dynamic knowledge, all necessary components are selected, modified, and arranged into a customized product.

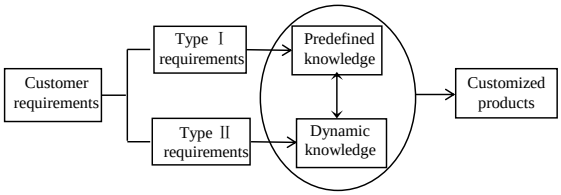


Figure 2. Open configuration overview

In more detail, suppose that given customer requirements are valid, complete and do not conflict with one another. These requirements are evaluated first to determine whether or not they can be fulfilled by the available configuration elements (i.e., functions and components). According to the evaluation results, these requirements are classified into Type Ⅰ and Type Ⅱ requirements. Fig. 3 summarizes this process.

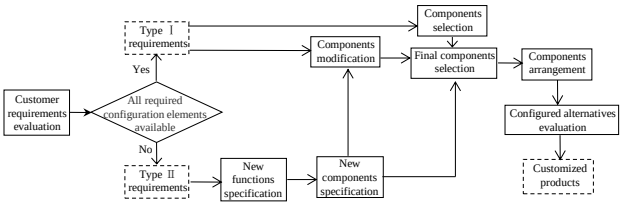


Figure 3. Open configuration process

For Type Ⅱ requirements, new functions are specified and all possible components which can realize these functions are subsequently determined. Also specified are the relationships among functions, among components, and between functions and components. This process contributes to the extension of the dynamic knowledge. For Type Ⅰ requirements, all possible components are selected from the predefined ones. In addition, to be compatible with the newly introduced components, some predefined components are modified by respecting constrains and rules embedded in the predefined and dynamic knowledge. This process reflects the interaction between the dynamic and predefined knowledge. From the modified components, newly introduced components, and selected predefined components, suitable components are further selected for forming configuration alternatives, which can meet customer requirements. In the selection, consistency and compatibility evaluations might be carried out. The selected components are arranged into product configuration alternatives by following the product structure described in the dynamic and predefined knowledge. These configuration alternatives are further evaluated under certain criteria. Based on the evaluation results, the optimal one or multiple are suggested to customers.

4 CHALLENGES INVOLVED IN OPEN CONFIGURATION

In accordance with the involvement of new configuration elements, open configuration changes the basic assumptions and reasoning processes of product configuration. In this regard, there are a number of potential challenges involved in open configuration. Due to the page limitation, this paper discusses five of these challenges, including open configuration modeling, system design and development, open configuration solving, open configuration optimization, and open configuration knowledge representation.

4.1 Open configuration modeling

Open configuration modeling addresses the modeling of open configuration knowledge and the reasoning mechanism for using the configuration knowledge. The modeling of open configuration knowledge is to model configuration elements, constraints, and rules. It involves two kinds of knowledge: predefined knowledge and dynamic knowledge. A product model and corresponding functional architectures should be developed for defining and further classifying the two different types of knowledge. The modeling of the reasoning mechanism is to shed light on (1) how new functions are specified, (2) how new components are determined, and (3) how components are selected and arranged into products.

In open configuration modeling, the components and functions are characterized by their attributes, while the inter-connections among the components are represented by connections and ports. The modeling of the dynamic knowledge needs to take into account the fact that new functions and components are added based on the unforeseen customer requirements. Thus, its modeling involves newly-added concepts, constraints, and rules. The modeling of the predefined knowledge needs to consider these predefined components, modified components, and their relationships. The interaction between predefined knowledge and dynamic knowledge needs to be modeled as well.

Open configuration modeling is more sophisticated than configuration modeling due to the involvement of the dynamic knowledge. In this regard, it is interesting to see whether or not these techniques which are suitable for modeling product configuration (e.g., Unified Modeling Language (UML), Alloy, and generative Constraint Satisfaction Problem (CSP) [11]) can be used to model open configuration. If these techniques are feasible, how can they be modified or adjusted to model open configuration. If these techniques are not feasible, new modeling formalisms and constructs are to be developed.

4.2 System design and development

System design and development for open configuration refers to the design and development of the computer information system to implement open configuration, i.e., open configurators. Open

configurators consist of a customer input module which deals with customer requirements evaluation, open configuration knowledge bases, reasoning and evaluation mechanisms, optimization and diagnosis mechanisms, and an output module which communicates the configuration results with users. Different from product configurators, open configurators involve two knowledge bases: a knowledge base for the predefined knowledge and the other for the dynamic knowledge. Joint reasoning mechanisms between the two knowledge bases are required, which mainly associate with interacting and integrating elements from the two knowledge bases. For the dynamic knowledge base, new elements design modules are needed to develop and maintain this knowledge base. The new elements design modules include the module for specifying new functions with respect to the requirements, the module for selecting new components to fulfill new functions and the module for interfacing with the predefined elements. For the predefined knowledge base, different from product configurators, there need to be a modification module for modify existing components to be compatible with the new ones.

In designing and developing open configurators, the techniques should have the ability to model dynamic knowledge and the interaction between dynamic knowledge and predefined knowledge. In this regard, the available system design techniques for product configuration may need to be modified in designing and developing open configurators.

4.3 Open configuration knowledge representation

Open configuration knowledge representation entails the effective organization of open configuration knowledge, including the predefined and dynamic knowledge. It logically uniforms the open configuration knowledge and enables the utilization of the knowledge in different configuration tasks.

The representation of open configuration knowledge includes the representation of predefined components, relationships, constraints and rules; the representation of newly-added components, relationships, constraints and rules; and the representation of the constraints and relationships between predefined knowledge and newly added knowledge. From the experience of the knowledge representation for product configuration, open configuration should be considered as both a classification problem (i.e., capturing the aspects of taxonomy and topology) and a constraint satisfaction problem (i.e., capturing the aspects of constraints and resource balancing). Considering the dynamic and indeterminate feature of open configuration, it might be potentially challenging to capture different aspects of open configuration knowledge (e.g., taxonomy, topology, constraints, and resource balancing) in one model. Further studies may try to design new models (or sub models to be embedded in the available tools) separately on each aspect and joint them together to represent the knowledge.

4.4 Open configuration solving

Open configuration solving relates to the development and application of algorithms or other tools to solve open configuration problems. In solving an open configuration problem, the problem needs to be modeled first with respect to customer requirements and configuration rules. To solve this model, algorithms need to be developed subsequently.

In the situation that customer requirements demand new functions, the dynamic knowledge will be specified. The modeling of open configuration problem will associate with the interaction between the customer requirements and two types of knowledge (predefined knowledge and dynamic knowledge). The main difficulties are (1) the modeling of new function specification, (2) the modeling of new components selection according to the customer requirements, (3) and the modeling of the interaction between new components and selected existing components. After modeling an open configuration problem, suitable algorithms need to be developed to solve the model. Because of the differences between product configuration and open configuration and the corresponding differences between a product configuration model and an open configuration model, these algorithms, which are suitable for product configuration solving, may not be applicable for open configuration solving. Thus, new algorithms are to be developed.

4.5 Open configuration optimization

During each step of open configuration, optimal functions, components and structures need to be specified from a number of alternatives. The dynamic feature of open configuration increases the degree of difficulty in optimizing the new functions, new components, and the interaction between new components and predefined ones. In this regard, an explicit optimization mechanism needs to be developed.

In accordance with the open configuration process discussed earlier, the optimization mechanism should evaluate the configuration elements at three levels. In the first level, the mechanism should evaluate all the possible function alternatives for fulfilling Type II requirements and decide on the optimal ones. This optimization might be based on, e.g., the performance and completeness of these function alternatives. In the second level, the mechanism should evaluate all the possible component alternatives for delivering the determined new functions and decide on the optimal ones. This optimization may take into account, e.g., the compatibility among the new components and the interaction with predefined components. In the third level, the mechanism should evaluate all the product configuration alternatives and decide on the optimal ones. This optimization may consider, e.g., product reliability.

5 CONCLUSION

In response to the limitation of product configuration, this paper proposed open configuration to help design customer-driven product in a true sense. It introduced the concept and process of open configuration. It also discussed several challenges involved in open configuration. Currently, we are working on the formulation of open configuration. In the formulation, new components, relationships among new components, and relationships between new components and existing components will be defined and modeled. This formulation is to rigorously define open configuration and shed light on the reasoning behind open configuration.

REFERENCES

[1] M. Heiskala, K.S. Paloheimo, and J. Tiihonen, *Mass customization of services: benefits and challenges of configurable services*, 206-221, Proceedings of Frontiers of e-Business Research, Tampere, Finland, 2005.

[2] D. Sabin and R. Weigel, 'Product configuration frameworks-a survey', *IEEE Intelligent Systems and Their Applications*, **13(4)**, 42-49, (1998).

[3] S. Schmitt and R. Bergmann, *Applying case-based reasoning technology for product selection and customization in electronic commerce environments*, 42-48, Proceedings of the 12th International Bled Electronic Commerce Conference, Bled, Slovenia, 1999.

[4] A. Trentin, E. Perin, and C. Forza, 'Product configurator impact on product quality', *International Journal of Production Economics*, **135(2)**, 850-859, (2012).

[5] 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(1)**, 87-98, (2002a).

[6] A. Haug, L. Hvam, and N.H. Mortensen, 'The impact of product configurators on lead times in engineering-oriented companies', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **25(2)**, 197-206, (2011).

[7] A. Trentin, E. Perin, and C. Forza, 'Overcoming the customization-responsiveness squeeze by using product configurators: Beyond anecdotal evidence', *Computers in Industry*, **62(3)**, 260-268, (2011).

[8] C. Forza and F. Salvador, 'Product configuration and inter-firm co-ordination: an innovative solution from a small manufacturing enterprise', *Computers in Industry*, **49(1)**, 37-46, (2002b).

[9] S. Mittal and F. Frayman, *Towards a generic model of configuration tasks*, 1395-1401, Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit, USA, 1989.

[10] D. Brown, 'Defining configuration', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **12(4)**, 301-305, (1998).

[11] A. Falkner, A. Haselböck, G. Schenner, and H. Schreiner, 'Modeling and solving technical product configuration problems', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **25(2)**, 115-129, (2011).

Towards an understanding of how the capabilities deployed by a Web-based sales configurator can increase the benefits of possessing a mass-customized product

Chiara Grosso[1] and **Alessio Trentin** [1] and **Cipriano Forza** [1]

Abstract. Manufacturers that adopt mass customization are paying a growing attention to understanding not only how product customization can be delivered efficiently, but also how this strategy can create value for their customers. As reported in literature, the customer-perceived value of a mass-customized product also depends on the uniqueness and self-expressiveness benefits that a customer may experience above and beyond the traditionally considered utility of possessing a product that fits with the customer's functional and aesthetical needs. Increasing customer-perceived value by delivering uniqueness and self-expressiveness benefits can therefore be one key in augmenting the customer's willingness to pay for a mass-customized product. This paper conceptually develops and empirically tests the hypotheses that five sales-configurator capabilities previously defined in literature increase uniqueness and self-expressiveness benefits of a mass-customized product, in addition to the traditionally considered utilitarian benefit. The hypothesized relationships have been tested by analyzing self-customization experiences made by engineering students using a set of real Web-based sales configurators of different consumer goods. The analysis results show that easy comparison, flexible navigation and focused navigation capabilities have a positive impact on each of the considered benefits, while user-friendly product space description and benefit-cost communication capabilities have a positive impact on utilitarian benefit only. The findings of this study complement previous research results on what characteristics sales configurators should have to increase consumer-perceived benefits of mass customization.

1 Introduction

According to Pine [42, p.48] mass customization is defined as ''developing, producing, marketing and delivering affordable goods and services with enough variety and customization that nearly everyone finds exactly what they want''. Nowadays, mass-customization strategies are more and more widespread and, therefore, mass customizers may need to identify unexploited sources of differentiation advantage [35].

[1] Università di Padova, Dipartimento di Tecnica e Gestione dei sistemi ind.li, Stradella S. Nicola 3, 36100 Vicenza, Italy. E-mail addresses: chiara.grosso@unipd.it (C.Grosso), alessio.trentin@unipd.it (A.Trentin), cipriano.forza@unipd.it (C.Forza).

In such a context, increasing the customer-perceived benefits of possessing a mass-customized product can be one key in delivering value that exceeds those of competing mass customizers' offerings. In particular, manufacturers that adopt mass customization need to take into account the various benefits that consumers can experience from mass-customization and the product value implication for customers [51]. While early literature emphasized the utilitarian benefit of possessing a product that better fit with one's idiosyncratic functional and aesthetical needs, the recent literature has developed more sophisticated knowledge of the value implications of mass customization to individual customers [20]. In particular, it has recently been acknowledged that providing other benefits in addition to the utilitarian one is crucial in augmenting customers' willingness to pay.

Since mass customizers are increasingly adopting Web-based sales configurators, it is important to understand what characteristics sales configurators should have to increase customer-perceived benefits of a mass-customized product. Previous research, however, has focused on how sales configurators should be designed to increase the traditionally considered utilitarian benefit of owning a self-customized product. The present paper offers additional insights into this issue by conceptually developing and empirically testing hypotheses on how capabilities deployed by a Web-based sales configurator can increase the benefits of possessing a mass-customized product.

2 Background

2.1 Consumer perceived benefits of a mass-customized product

According to Holbrook [33], every consumption experience involves an interaction between a subject and an object, where the subject of interest is a consumer or customer and the object of interest is some product or service. The value that the consumer gains from the consumption experience is created through that interaction [19]. Mass customization allows customers to ask for new personalized products at a level of individualized tailoring that was never possible before [1]. Addis and Holbrook [1] identified a trend that the same authors called 'an explosion of subjectivity' [1, p.2] to denote the emerging phenomenon of a more widespread role that individual subjectivity plays in consumption, where the term 'subjectivity' refers to a personal psychological state - that is, one's

own way of feeling, thinking, or perceiving. According to these authors, mass customization implicitly recognizes the growing importance of consumer subjectivity.

Previous mass-customization studies on mass-customized product value [26, 38, 26, 47] explain that, in addition to the well-researched utilitarian benefit, there are two benefits, namely uniqueness and self-expressiveness benefits, which a consumer could derive from the possession of a mass-customized product.

Utilitarian benefit, according to Merle et al. [38], is a benefit deriving from the closeness of fit between product objective characteristics (i.e. aesthetical and functional characteristics) and an individual's preferences. In other terms, utilitarian benefit derives from the fact that the self-customized product fulfills the individual's idiosyncratic functional and aesthetical needs [1].

The uniqueness benefit of possessing a mass-customized product is defined by Merle et al [38] as the benefit that a consumer derives from the opportunity to assert his/her personal uniqueness by using a customized product. Uniqueness benefit is related to the symbolic meanings a person attributes to the objects as a result of social construction [12, 52, 49, 53, 29, 39]. Brewer's [8] optimal distinctiveness theory posits that people have opposing motives to fit in and stand out from social groups. A series of studies by Brewer and colleagues e.g. [9] has shown that, whereas threats to one's inclusionary status produce increased attempts to fit in and conform, threats to one's individuality produce attempts to demonstrate how different one is from the rest of the group. Consequently, uniqueness benefit deriving from a mass-customized product will meet the individual need to assert his/her own personality by differentiating his/her self from others [21, 50].

Self-expressiveness benefit is defined by Merle et al. 38] as the benefit that originates from the opportunity to possess a product that is a reflection of the consumer's image. This is in accordance with the self-consistency motive underlying self-concept, where the term "self-consistency" denotes the tendency for an individual to behave consistently with his/her view of his/her self [48]. Like uniqueness, self-expressiveness benefit is related to the symbolic meanings a person attributes to the objects as a result of social construction [12, 52, 49, 53, 29, 39]. According to Belk [4], possessions are often extension of the self. As Belk states, "people seek, express, confirm, and ascertain a sense of being through what they have" [4, p.146]. The above statement implicitly relates identity with consumption. Consumers deliberately acquire things and engage in consumption practices to achieve a pre-conceived notion of their selves [46]. Thus, a mass-customized product will accomplish an individual's need for self-consistency through the possession of a product that is a reflection of his/her self.

2.2 Sales configurators

Consistent with previous research [23, 32, 30], we define sales configurators as knowledge-based software applications that support a potential customer, or a sales-person interacting with the customer, in completely and correctly specifying a product solution within a company's product offer.

The benefits and challenges of implementing and using a sales configurator have been the focus of several researches e.g., [54, 23, 34, 57, 58, 30-31]. Relatively less studies, however, have addressed the question of what characteristics a sales configurator should have to increase such benefits and alleviate such challenges. For example, Randall et al. [43] suggest that, depending on a customer's expertise with a product, a sales configurator should present either product functions and product performance characteristics or design parameters to the potential customer. Another example is Chang et al.'s [13] recommendation that a sales configurator provides potential customers with examples of configured products, in order to offer them guidance about what to do. More recently, Trentin et al. [56] have conceptualized five sales-configurator capabilities based on previous research recommendations. The definitions of such capabilities are reported in Table 1.

Table 1. Sales-configurator capabilities [55]

Capability	Definition
Benefit-cost communication	The ability to effectively communicate the consequences of the configuration choices made by a potential customer both in terms of what he/she would get and in terms of what he/she would give
User-friendly product-space description	The ability to adapt the description of a company's product space to the individual characteristics of a potential customer as well as to the situational characteristics of his/her using of a sales configurator
Easy comparison	The ability to support sales-configurator users in comparing product configurations they have previously created
Flexible navigation	The ability to let sales-configurator users easily and quickly modify a product configuration they have previously created or are currently creating
Focused navigation	The ability to quickly focus a potential customer's search on those solutions of a company's product space that are most relevant to the customer himself/herself

Previous studies on sales configurators, however, have typically regarded the mass-customized product only as a source of utilitarian benefits related to the fulfillment of customers' functional and aesthetical needs. As discussed in the previous section, however, a mass-customized product can also be a source of benefits resulting from uniqueness and self-expressiveness. What characteristics a sales configurator should have to increase uniqueness and self-expressiveness benefits is therefore a question that deserves additional research, as previously pointed out by Schreier [47] or Franke and Schreier [28].

3 Research hypotheses

In addressing the question raised at the end of the previous section, we draw upon the five sales-configurator capabilities conceptualized by Trentin et al.[55, 56] based on prior research on sales configurators. For each of these capabilities, we develop hypotheses about its effects on both uniqueness benefit and self-expressiveness benefit, as well as on the traditionally considered utilitarian benefit of possessing a mass-customized product.

In the existing literature, a number of studies make the point that, to increase the utilitarian benefit of possessing a mass-customized product, a sales configurator should support a company's potential customer in learning about the options available within the company's solution space, in learning about how these options are

useful in fulfilling his/her preferences and in learning about his/her preferences themselves e.g., [62, 43, 44] The more a sales configurator supports such a learning process about one or more of these aspects during the configuration task, the more a potential customer is enabled to create, within a company's product space, the configuration that best fits with his/her objective needs [59, 25]. Prior research has focused on product fit with an individual's functional and aesthetical needs, which leads to the traditionally considered utilitarian benefit. However, this also applies to product fit with an individual's need for asserting his/her own personality by differentiating his/her self from others. Consequently, such a learning process also augments the uniqueness benefit that a customer will enjoy from the possession of the configured product. Finally, this also applies to product fit with an individual's need for behaving consistently with his/her view of his/her self by possessing a product that reflects his/her self concept. Accordingly, such a learning process also increases the self-expressiveness benefit that a customer will derive from the product configuration eventually purchased.

Clearly, the more effective the learning process enabled by a sales configurator, the greater the utilitarian benefit, the uniqueness benefit and the self-expressiveness benefit of possessing the configured product. While Franke and Hader [25, p.16] find that the learning effects of single self-customization experiences lasting only a few minutes with sales configurators "that were not even specifically designed for learning purposes are remarkable", we argue that such learning effects are greater if a sales configurator deploys a higher level of each of the capabilities conceptualized by Trentin et al. [55, 56] based on prior research on sales configurators.

A sales configurator with a higher level of flexible navigation capability allows a potential customer to go through a greater number of complete trial-and-error cycles to evaluate the effects of his/her prior choices and to improve upon them. This is because this kind of sales configurator allows its users to change, at any step of the configuration process, the choice they made at any previous stage without having to begin the process all over again and allows them to immediately recover a previous configuration in case they decide to reject the newly-created one [56]. By conducting more trial-and-error tests, the potential customer learns more about the available choice options and the value he/she would derive from them [59, 60].

A sales configurator with a higher level of user-friendly product space description capability promotes a potential customer's learning process by increasing the congruence between the challenges of the configuration task and the abilities of the configurator user. This is because a sales configurator with this capability presents product space information to potential customers using the most suitable format (e.g., text, image, animation,…) depending on their skill levels and cognitive styles and offers different types of choices (e.g., among product functions and performance levels rather than among product components, or vice versa) according to the users' prior knowledge about the product [56]. In addition, such a sales configurator allows its users to decide for themselves how many feedback details they want to tackle, without forcing them to process information content they do not value [56]. By tailoring the sales configuration experience to each individual user's characteristics on both the content and presentation levels [36], a sales configurator with higher user-friendly product space description reduces the risk that the configuration task is too difficult and, therefore, the user reacts with frustration. At the same time, such a sales configurator alleviates the risk that the configuration task is too easy and, thus, the individual gets bored. In both cases, the effectiveness of the learning process would be undermined [3, 63, 41].

A sales configurator with a higher level of focused navigation capability increases learning effects by tailoring the sales configuration experience to each individual user's characteristics on the interaction level [36]. A sales configurator with this capability enables its users to freely prioritize their choices regarding the various attributes of a product and, therefore, allows them to quickly eliminate options they regard as certainly inappropriate from further consideration [56]. In addition, such a sales configurator enables its users to decide for themselves how many configuration options they want to tackle, as not all potential customers are necessarily interested in, and/or able to fully exploit the potential of customization offered by a company [43]. In this manner, this kind of sales configurator reduces the risk that the configuration task is frustrating as well as the risk that it is boring, and both of these situations would undermine the effectiveness of the learning process [3, 63, 41].

A sales configurator with a higher level of benefit-cost communication capability promotes a potential customer's learning process by providing him/her with better pre-purchase feedback on the effects of his/her configuration choices. Such a sales configurator is more effective in explaining the benefits the customer would derive from consumption of the configured product, as well as the monetary and nonmonetary sacrifices that the customer would bear for obtaining that product [56]. For example, a sales configurator with a higher level of benefit-cost communication capability takes advantage of three-dimensional Web and virtual try-on technologies to more closely simulate customers' real-world interactions with their configured products [18, 14]. As the feedback provided by the sales configurator improves, so does the effectiveness of the potential customer's learning process [10].

Finally, a sales configurator with a higher level of easy comparison capability increases learning effects by providing better pre-purchase feedback on the effects of the configuration choices made by a potential customer. This is because such a sales configurator allows its users to compare previously-saved configurations on the same screen and to rank-order them based on some criterion that is meaningful to the users [56]. Again, the better the feedback provided, the more effective the customer's learning process [10].

As each of the sales configurator capabilities mentioned above make the learning process more effective and the effectiveness of such a learning process increases the utilitarian benefit, the uniqueness benefit and the self-expressiveness benefit of the configured product eventually purchased, we posit the following hypotheses, which are graphically summarized in Figure 1.

HXa. *The higher the level of flexible navigation capability (H1a), focus navigation capability (H2a), benefit-cost communication capability (H3a), user-friendly product space description (H4a), easy comparison capability (H5a) deployed by a sales configurator, the greater the utilitarian benefit that a consumer derives from a product self-customized using that configurator.*

HXb. *The higher the level of flexible navigation capability*

(H1b), focus navigation capability (H2b), benefit-cost communication capability (H3b), user-friendly product space description (H4b), easy comparison capability (H5b) deployed by a sales configurator, the greater the uniqueness benefit that a consumer derives from a product self-customized using that configurator.

***HXc**. The higher the level of flexible navigation capability (H1c), focus navigation capability (H2c), benefit-cost communication capability (H3c), user-friendly product space description (H4c), easy comparison capability (H5c) deployed by a sales configurator, the greater the self-expressiveness benefit that a consumer derives from a product self-customized using that configurator.*

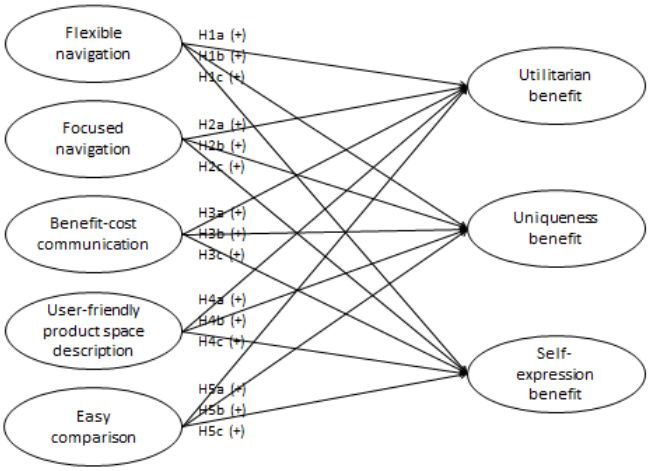


Figure 1. Research hypotheses overview

4 Method

To test our hypotheses we conducted an empirical analysis using data collected from a sample of 675 sales-configuration experiences made by 75 students at the authors' university (age range: 24-27; 30% females). Each participant was asked to make one mass-customization experience on each of nine pre-assigned Web-based sales configurators and, for each experience, to fill out a questionnaire covering the constructs of interest (see Appendix A), for a total of 675 mass-customization experiences. Each experience involved browsing the sales-configuration website and configuring one product from start to finish, on that website, according to one's own preferences. The nine sales configurators assigned to each participant were chosen from a set of 30 real Web- based configurators of consumer goods. The set included ten configurators of notebooks/laptops (e.g., www.dell.com), nine configurators of sports shoes/sneakers (e.g., www.converse.com) and eleven configurators of economy cars (e.g., www.volkswagen.com). The inclusion of multiple product categories, ranging from relatively simple products with relatively few configuration steps to more complex products with more configuration steps, was motivated by the aim of increasing the variation ranges of the independent variables within our sample. To further increase the differences among the mass-customization experiences comprising our sample, we assigned sales configurators to participants according to the following rules: (i) no pairs of participants were assigned the same combination of configurators, (ii) each participant was assigned three configurators for each product category, and (iii) each of the triples assigned to each participant included at least one product configurator with a high mean score of the five capabilities within the corresponding product category and at least one configurator with a low mean score of the five capabilities within the same product category.

The data were analyzed through structural equation modeling, using LISREL 8.80. Following Anderson and Gerbing [2], we decided to adopt a two-step approach, assessing construct validity before the simultaneous estimation of the measurement and structural models. Moreover, since our variables did not meet the assumption of multivariate normal distribution (Mardia's test significant at $p<0.001$), we applied the Satorra-Bentler correction to produce robust maximum likelihood estimates of standard errors and Chi-square. Prior to conducting the analysis, Prior to conducting the analysis, we decided to control for possible effects of participants' characteristics. Consequently, and consistent with prior studies (e.g., [37, 56]), we regressed our observed indicators on 75 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 subsequently employed to assess unidimensionality, convergent validity, discriminant validity, and reliability of our measurement scales. We tested a CFA model specifying the posited relations of the observed variables to the underlying latent constructs, with these constructs allowed to correlate freely [2]. Our CFA model showed good fit indices (RMSEA (90% CI)= 0.0489 (0.0445; 0.0533), GFI=0.927, NFI=0.987), meaning that the hypothesized factor structure reproduced the sample data well. The standardized factor loadings were all in the anticipated direction, greater than 0.50 and statistically significant at $p<0.001$. Altogether, these results suggested unidimensionality (i.e., a set of empirical indicators reflect one, and only one, underlying latent factor) and good convergent validity (i.e., the multiple items used as indicators of a construct significantly converge) of our measurement scales [11, 2]. 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 [22] procedure. For each latent construct, the square root of the average variance extracted (AVE) exceeded the correlation with all the other latent variables, thus suggesting that our measurement scales represent distinct latent variables [22]. Reliability of the measurement scales was assessed using both AVE and the Werts, Linn and Joreskog (WLJ) composite reliability (C.R.) method [61]. All the WLJ composite reliability values were greater than 0.70 and all the AVE scores largely exceeded 0.50. This indicates that a large amount of the variance is captured by each latent construct rather than being due to measurement error [22, 40].

5 Results

After establishing measurement scale reliability and validity for the focal constructs, we estimated the full model including the hypothesized relationships among the same constructs. Our hypotheses were that all five sales-configurator capabilities increase consumer-perceived utilitarian benefit, uniqueness benefit and self-expressiveness benefit of a mass-customized product. Accordingly, all five capabilities were modeled as impacting both utilitarian benefit and uniqueness benefit and self-expressiveness benefit. Table 2 reports the LISREL estimates of the path coefficients and the corresponding *t* values. In assessing whether a hypothesis is

supported or not, we adopted a p value of 5% as a threshold. This is a conservative choice, as a cut-off value of 10% is often used in literature.

Table 2. Path coefficients of the estimated model

		BCC	EC	FlexN	FocN	UFD
UT	Coeff.[§]	0,283***	0,102***	*0,132***	*0,379****	*0,146**
	t value[†]	*3,654*	*3,669*	*2,735*	*5,237*	*2,451*
UN	Coeff.[§]	0,004	0,299***	0,304***	0,253*	0,034
	t value[†]	*0,036*	*6,773*	*4,106*	*2,537*	*0,42*
SE	Coeff.[§]	0,148	0,19***	0,151**	0,337***	0,06
	t value[†]	*1,82*	*5,346*	*2,612*	*4,137*	*0,95*

UT = *utilitarian benefit*; *UN* = *uniqueness benefit*; *SE* = *self-expressiveness benefit*

BCC = *benefit-cost communication*; *EC* = *easy comparison*; *UFD* = *user-friendly product-space description*; *FlexN* = *flexible navigation*; *FocN* = *focused navigation*

Significant at: *** $p < 0.001$; ** $p < 0.01$; * $p < 0.05$.

[†]Cut-off t value: 10%: 1.645; 5%: 1.960; 1%: 2.576; 0.1%: 3.29.

As regards utilitarian benefit, all the estimated path coefficients were positive, as hypothesized, and statistically significant at $p< 0.05$, indicating that all our hypotheses regarding the utilitarian benefit are supported. As regards uniqueness benefit, the estimated path coefficients were positive, as hypothesized, and statistically significant at $p< 0.05$ for easy comparison, flexible navigation and focused navigation capabilities, but not for benefit-cost communication and user-friendly product space description capabilities. Therefore, only three of our five hypotheses are supported. The same pattern of results was found with regard to self-expressiveness benefit. It is worthwhile noting, however, that the estimated path coefficient between benefit-cost communication capability and self-expressiveness benefit is statistically significant at $p< 0.10$, though not at $p< 0.05$.

6 Conclusions

6.1 Discussion of results and related work

The analysis results support the hypotheses that easy comparison, flexible navigation and focused navigation capabilities raise not only the utilitarian benefit of possessing a mass-customized product, but also its uniqueness and self-expressiveness benefits. These findings improve our understanding of how product configurators should be designed to increase customers' willingness to pay for a mass-customized product by triggering uniqueness and self-expressiveness benefits, in addition to utilitarian benefit.

As regards user-friendly product space description and benefit-cost communication capabilities, however, only the hypotheses that they increase utilitarian benefit are supported, while the others are not. Two possible explanations can be provided for these unexpected findings. One explanation revolves around the notion of functional fixedness. Functional fixedness is the phenomenon in which an individual finds difficulties in attributing and recognizing different types of relationships between objects presented to him/her during decision-making processes or problem-solving situations [15]. Another possible explanation is that the existing sales-configurators, even when they deploy higher levels of benefit-cost communication and user-friendly product space description capabilities, provide feedback information with content and format that are appropriate for promoting potential customers' learning about the possibility to fulfill customers' functional and aesthetical needs through the consumption of a configured product, but are not appropriate for supporting the same learning process as far as satisfaction of uniqueness and self-consistency needs are concerned. However, these are conjectures; further research is needed on this issue.

The present paper contributes to the debate as to what characteristics sales configurators should have to increase consumers' willingness to buy as well as consumers' willingness to pay for a mass-customized product. This debate has typically focused on a twofold objective: (i) alleviating the difficulty that a consumer experiences in self-customizing a product with a sales configurator and in making a purchase decision and (ii) increasing the utilitarian benefit deriving from the closeness of fit between the objective characteristics of the configured product and the consumer's functional and aesthetical needs. Several recommendations have been made by prior, both conceptual and empirical studies joining this debate, and many of these recommendations are subsumed by the five sales-configurator capabilities considered in this study [56]. Higher levels of these capabilities have been found as predicting both higher levels of satisfaction with the configured product and higher levels of purchase intention [56]. More recently, the debate has been enriched by the consideration of the benefits that a consumer can gain from the experience of self-customizing a product using a sales configurator above and beyond those deriving from the possession of the configured product. In particular, Trentin et al. [55] find that the same five sales-configurator capabilities considered in the present study increase hedonic benefit, which stems from the capacity of the experience to be gratifying per se, regardless of the completion of the configuration task, and creative-achievement benefit, which derives from the capacity of the experience to arouse, in combination with the configured product, the positive emotion of pride of authorship. The present study makes an additional contribution to this debate by examining the impacts of the same five sales-configurator capabilities on another two benefits that a consumer can enjoy by purchasing a mass-customized product, in addition to the traditionally considered utilitarian benefit: namely, the benefits of uniqueness and self-expressiveness.

Related work has been conducted in the domain of recommender technologies. Like Web-based sales configurators, recommender applications are intended to support online customers in making purchase decisions [45]. With a focus on knowledge-based recommender applications, Felfernig et al. [16] empirically examine the effects of a number of possible features of such applications on a variety of outcome variables, including a consumer's willingness to buy and his/her trust in that the application recommended the optimal solution. The examined features include the provision of a justification for why a product fits to a certain customer, the possibility of making product comparisons, and the fitting of the interactive user-recommender dialog to the user's product domain knowledge. These features are captured by the capabilities of benefit-cost communication, easy comparison and user-friendly product space description which are considered in the present study. Interestingly, Felfernig et al. [16] find that the recommender versions exhibiting such features are associated with higher ratings of users' trust in the recommended products, which in turn is

positively associated with users' willingness to buy the products. This result is echoed by our findings that benefit-cost communication, easy comparison and user-friendly product space capabilities predict the utilitarian benefit deriving from the possession of a mass-customized product.

6.2 Limitations and further research

The present research is not without limitations, which might be addressed in future research. A primary limitation lies in the fact the empirical study was conducted with engineering students and using only three categories of consumer goods. While engineering students are undeniably potential buyers of the considered products, they constitute a biased sample of the potential customers of such goods. In addition, these products represent only a small subset of consumer goods. A wider set of products would strengthen the generalizability of the results. Consequently, future research should seek to replicate our findings in truly representative samples of potential customers and should use a wider set of consumer goods.

Another limitation of the present study is its focus on the main effects [17] of the five considered sales-configurator capabilities on the three consumer-perceived benefits of interest. In line with this focus, we neglect possible interaction effects between the five capabilities as well as possible contingency effects. Future studies should be designed to overcome this limitation.

6.3 Managerial implications

While having its limitations, our study not only reinforces the importance of the research on the role of sales configurators in mass-customization strategies, but also provides useful managerial implications. By considering additional benefits, besides the utilitarian one, our study increases practitioners' awareness that sales/product configurators can be an effective tool to augment the consumer-perceived benefits of possessing a mass-customized product. Exploiting such sources of differentiation advantages as the fulfillment of consumers' needs for uniqueness and self-expressiveness can be one key for a company to augment the value of its mass-customization strategy. For those firms that are interested in fulfilling consumers' needs for uniqueness and self-expressiveness, our theoretical explanations and our empirical results highlight the importance of adopting sales configurators with higher levels of easy comparison, flexible navigation and focused navigation capabilities. This is another step in the direction of providing practitioners with prescriptive indications on how sales configurators should be designed to increase the benefits of possessing mass-customized products.

ACKNOWLEDGEMENTS

We acknowledge the financial support of the University of Padova, Project ID CPDA129273.

APPENDIX A. Measurement instrument

Benefit-cost communication capability[a]

BCC1	Thanks to this system, I understood how the various choice options influence the value that this product has for me.
BCC2	Thanks to this system, I realized the advantages and drawbacks of each of the options I had to choose from.
BCC3	This system made me exactly understand what value the product I was configuring had for me.

Easy comparison capability[a]

EC1	The system enables easy comparison of product configurations previously created by the user.
EC2	The system lets you easily understand what previously created configurations have in common.
EC3	The system enables side-by-side comparison of the details of previously saved configurations.
EC4	The systems lets you easily understand the differences between previously created configurations.

User-friendly product-space description capability([a]

UFD1	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.
UFD2	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.
UFD3	The choice options are adequately presented for both the expert and inexpert user of the product.

Flexible navigation capability[a]

FlexN1	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.
FlexN2	With this system, it takes very little effort to modify the choices you have previously made during the configuration process.
FlexN3	Once you have completed the configuration process, this system enables you to quickly change any choice made during that process.

Focused navigation capability[a]

FocN1	The system made me immediately understand which way to go to find what I needed.
FocN2	The system enabled me to quickly eliminate from further consideration everything that was not interesting to me at all.
FocN3	The system immediately led me to what was more interesting to me.
FocN4	This system quickly leads the user to those solutions that best meet his/her requirements.

Utilitarian benefit([b]

UT1	This product is exactly what I had hoped for.
UT2	I could create the product that was the most adapted to what I was looking for.
UT3	I could create the product I really wanted to have.

Uniqueness benefit(b)

UN1	With this product, I will not look like everybody else.
UN2	With this program, I could design a product that others will not have.
UN3	With this product, I have my small element of differentiation compared to others.

Self-expressiveness benefit(b)

SE1	I could create a product that is just like me.
SE2	This product reflects exactly who I am.
SE3	This product is in my own image.

(a) Trentin et al.[56]
(b) Merle et al. [38]

REFERENCES

[1] M. Addis and M.B. Holbrook, 'On the conceptual link between mass customisation and experiential consumption: an explosion of subjectivity'. *Journal of Consumer Behaviour*, **1**(1), 50-66, (2001).

[2] J.C. Anderson, D.W. Gerbing, 'Structural equation modeling in practice: a review and recommended two-step approach', *Psychological Bulletin*, **103**(3), 411–423, (1988).

[3] A. Bandura, 'Perceived self-efficacy in cognitive development and functioning', *Educational Psychologist*, **28**(2), 117–48, (1993).

[4] R.W. Belk, 'Possessions and the extended self', *Journal of Consumer Research*, **15**, 139-168, (1989).

[5] R.W. Belk, J.F. Sherry and M. Wallendorf, 'A naturalistic inquiry into buyer and seller behavior at a swap meet', *Journal of Consumer Research*, **14**(4), 449-470, (1988).

[6] T. Blecker and G. Friedrich, *Mass Customization Information Systems in Business*, IGI Global, London, UK, 2007.

[7] T. Blecker, N. Abdelkafi, B Kaluza and G. Friedrich, 'Key metrics system for variety steering in mass customization', in: Piller, F. T./Reichwald, R./Tseng, M. (Eds.): Competitive Advantage Through Customer Interaction: Leading Mass Customization and Personalization from the Emerging State to a Mainstream Business Model. *Proceedings of the 2nd Interdisciplinary World Congress on Mass Customization and Personalization- MCPC'03*, Munich, October 6-8, (2003).

[8] M.B. Brewer, 'The social self: On being the same and different at the same time', *Personality and Social Psychology Bulletin,* **17**, 475-482, (1991).

[9] M.B. Brewer, J.M Manzi and J.S. Shaw, 'In-group identification as a function of depersonalization, distinctiveness, and status', *Psychological Science*, **4**(2), 88-92, (1993).

[10] D.L. Butler and P.H. Winne, 'Feedback and self-regulated learning:A theoretical synthesis', *Review of Educational Research*, **65**(3), 245–81, (1995).

[11] D.T. Campbell and D.W. Fiske, 'Convergent and discriminant validation by the multitrait-multimethod matrix', *Psychological bulletin*, **56**(2), 81, (1959).

[12] R.L. Celsi, R.L. Rose and T.W. Leigh, 'An Exploration of High-Risk Leisure Consumption through Skydiving', *Journal of Consumer Research*, **20**, June, 1-23, (1993).

[13] C.C. Chang, H.Y. Chen and I.C. Huang, 'The interplay between customer participation and difficulty of design examples in the online designing process and its effects on customer satisfaction: mediational analyses', *CyberPsychology & Behavior*, **12**(2),147–154, (2009).

[14] K. Dai, Y. Li, J. Han, X. Lu, and S. Zhang, 'An interactive web system for integrated three-dimensional customization', *Computers in Industry*, **57**(8-9), 827-37, (2006).

[15] K. Duncker, 'The Structure and Dynamics of Problem-Solving Processes', *Psychological monographs*, **58**(5), 1-112, (1945).

[16] A. Felfernig, B. Gula, E. Teppan, 'User Acceptance of Knowledge-based Recommenders, Machine Perception and Artificial Intelligence', *World Scientific Publishers*, **70**, 249-276, 2007.

[17] J.W. Finney, R.E. Mitchell, R.C. Cronkite and R.H. Moos, 'Methodological issues in estimating main and interactive effects: examples from coping/social support and stress field', *Journal of Health & Social Behavior*, **25**(1) 85–98, (1984).

[18] A.M Fiore, S.E. Lee, and G. Kunz. 'Individual differences, motivations, and willingness to use a mass customization option for fashion products', *European Journal of Marketing,* **38**(7), 835-49, (2004).

[19] A.F. Firat, and A Venkatesh, 'Liberatory Postmodernism and the Reenchantment of Consumption', *Journal of Consumer Research*, **22**, December, 239-67, (1995).

[20] F.S. Fogliatto, G.J.C. da Silveira and D. Borenstein, 'The mass customization decade: an updated review of the literature', *International Journal of Production Economics,* **138**(1),14–25, (2012).

[21] H.L. Fromkin, 'A social psychological analysis of the adoption and diffusion of new products and practices from a uniqueness motivation perspective'. In D.M. Gardner (Ed.), *Proceedings of the 2nd annual Conference of the Association far Consumer Research*, College Park, MD: Association for Consumer Research, 464-469, (1971).

[22] C. Fornell and D.F. Larcker, 'Evaluating structural equation models with unobservable variables and measurement error', *Journal of Marketing Research*, **18**(1), 39–50, (1981).

[23] C. Forza and F. Salvador, 'Application support to product variety management', *International Journal of Production Research*, **46**(3), 817–836, (2008).

[24] C. Forza and F. Salvador, 'Product configuration and inter-firm co-ordination: an innovative solution from a small manufacturing enterprise', *Computers in Industry*, **49**(1), 37–46, (2002).

[25] N. Franke and C. Hader, 'Mass or Only "Niche Customization"? Why We Should Interpret Configuration Toolkits as Learning Instruments', *Journal of Product Innovation Management,* **31**(5), in press (2013).

[26] N. Franke, M. Schreier and U. Kaiser, 'The "I designed it myself" effect in mass customization', *Management Science*, **56**(1), 125–140, (2010).

[27] N. Franke and M. Schreier, 'Why customers value self-designed products: the importance of process effort and enjoyment', *Journal of Product Innovation Management*, **27**(7), 1020–1031, (2010).

[28] N. Franke and M. Schreier, 'Product uniqueness as a driver of customer utility in mass customization', *Marketing Letters* **19**(2), 93–107, (2008).

[29] G. Ger, S. Askegaard and A. Christensen, 'Experiential Natural of Product- Place Images: Image as a Narrative' in Arnould, E. J. and Scott, L. M. (Eds), *Advances in Consumer Research*, 26, Association for Consumer Research, Provo, UT, 165-9, (1999).

[30] L. Haug, L. Hvam and H.N. Mortensen, 'The impact of product configurators on lead- times in engineering-oriented companies', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **25**(2), 197–206, (2011).

[31] A. Haug, L. Hvam and N.H. Mortensen, 'Definition and evaluation of product configurator development strategies', *Computers in Industry*, **63**(5), 471–481, (2012).

[32] M. Heiskala, J. Tiihonen, K.S. Paloheimo and T. Soininen, 'Mass customization with configurable products and configurators: a review of benefits and challenges, in: T. Blecker, G. Friedrich (Eds.), *Mass Customization Information Systems in Business*, IGI Global, London, UK, 1–32, 2007.

[33] M.B Holbrook, *Introduction to Consumer Value* in Holbrook, M. B. (Ed), Consumer Value: A Framework For Analysis and Research, Routledge, London, 1-28, 1999.

[34] L. Hvam, S. Pape and M.K. Nielsen, 'Improving the quotation process with product configuration', *Computers in Industry*, **57**(7), 607–621, (2006).

[35] P. Jiang, 'Exploring consumers' willingness to pay for online customisation and its marketing outcomes', *Journal of Targeting Measurement & Analysis for Marketing*, **11**(2), 168–183, (2002).

[36] G. Kreutler and D. Jannach, 'Personalized needs acquisition in Web-based configuration systems', in: T. Blecker, G. Friedrich (Eds.), Mass Customization, Concepts-Tools-Realization, *Proceedings of the International Mass Customization Meeting 2005* (IMCM'05), GITO-Verlag, Berlin, Germany, 293-302, (2005).

[37] G. Liu, R. Shah and R.G. Schroeder, 'Linking work design to mass customization: a sociotechnical systems perspective', *Decision Sciences*, **37**(4), 519–545, (2006).

[38] 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 & Operations Management,* **19**(5), 503–514, (2010).
[39] K. O'Donnell, 'Good Girls Gone Bad, The Consumption of Fetish Fashion and the Sexual Empowerment of Women' in E. J Arnould and L.M. Scott, (Eds), *Advances in Consumer Research*, 26, Association for Consumer Research, Provo, UT, 184-89, (1999).
[40] S.W. O'Leary-Kelly and R.J. Vokurka, 'The empirical assessment of construct validity', *Journal of Operations Management*, **16**(4), 387–405, (1998).
[41] R. Pekrun, T. Goetz, W. Titz, and R. P. Perry, 'Academic emotions in students' self-regulated learning and achievement: A program of qualitative and quantitative research', *Educational Psychologist*, **37**(2), 91–106, (2002).
[42] B.J. Pine II, *Mass Customization – The New Frontier in Business Competition*, Harvard Business School Press, Cambridge, MA, 1993.
[43] T. Randall, C. Terwiesch and K.T. Ulrich, 'Principles for user design of customized products', *California Management Review,* **47** (4), 68–85, (2005).
[44] F. Salvador and C. Forza, 'Principles for efficient and effective sales configuration design', *International Journal of Mass Customisation*, **2**(1–2), 114–127, (2007).
[45] J.B. Schafer, J.A. Konstan, and J. Riedl, 'E-commerce recommendation applications', *Data Mining and Knowledge Discovery*, **5**(1-2), 115–153, (2001).
[46] H.J. Schau,'Consumer Imagination, Identity and Self-Expression', in NA - Advances in *Consumer Research Volume* **27**, (Eds.) J. Stephen Hoch and R.J. Meyer, Provo, UT: Association for Consumer Research, 50-56, (2000).
[47] M. Schreier, 'The value increment of mass-customized products: an empirical assessment', *Journal of Consumer Behaviour*, **5**(4), 317–327, (2006).
[48] M. J Sirgy, 'Self-concept in consumer behavior: a critical review', *Journal of consumer research*, **9**(3), 287-300, (1982).
[49] D. Slater, *Consumer Culture and Modernity*. Polity Press, Cambridge, UK,1997.
[50] C.R. Snyder, 'Product scarcity by need for uniqueness interaction: A consumer catch-22 carousel?', *Basic Appl. Soc. Psychol,* **13**(1), 9–24, (1992).
[51] B. Squire, S. Brown, J. Readman and J. Bessant, 'The impact of mass customisation on manufacturing trade-offs, *Production & Operations Management*, **15**(1),10–21, (2006).
[52] C.J., Thompson, 'Caring Consumers, Gendered Consumption Meanings and the Juggling Lifestyle', *Journal of Consumer Research*, **22**(4),388-407, (1996).
[53] C.J. Thompson, and D. L. Haytko, 'Speaking of Fashion, Consumers' Uses of Fashion Discourses and the Appropriation of Countervailing Cultural Meanings', *Journal of Consumer Research*, **24**(1),15-42, (1997).
[54] J. Tiihonen, T. Soininen, T. Männistö and R. Sulonen, *State-of-the-practice in product configuration – a survey of 10 cases in the Finnish industry,* in: T. Tomiyama, M. Mäntylä, & S. Finger (Eds.) ; Knowledge intensive CAD, Chapman & Hall, London, UK, 95–114, 1996.
[55] A.Trentin, E. Perin and C. Forza, 'Increasing the consumer-perceived benefits of a mass-customization experience through sales-configurator capabilities' *Computers in Industry*, **65**(4), 693-705, (2014).
[56] A. Trentin, E. Perin and C. Forza, 'Sales configurator capabilities to avoid the product variety paradox: construct development and validation', *Computers in Industry,* **64**(4), 436–447, (2013).
[57] A.Trentin, E. Perin and C. Forza, 'Product configurator impact on product quality', *International Journal of Production Economics*, **135** (2), 850–859, (2012).
[58] A.Trentin, E. Perin and C. Forza, 'Overcoming the customization-responsiveness squeeze by using product configurators: Beyond anecdotal evidence', *Computers in Industry*, **62**(3), 260–268, (2011).
[59] E. von Hippel, 'Perspective: user toolkits for innovation', *Journal of Product Innovation Management,* **18**(4) 247–257, (2001).
[60] E. von Hippel and R. Katz, 'Shifting Innovation to Users via Toolkits', *Management Science,* **48**(7), 821–833, (2002).
[61] C.E. Werts, R.L. Linn and K.G. Jo, 'Reskog, Intraclass reliability estimates: testing structural assumptions', *Educational & Psychological Measurement*, **34**(1), 25–33, (1974).
[62] J. Wind and A. Rangaswamy, 'Customerization: The next revolution in mass customization', *Journal of Interactive Marketing,* **15**(1), 13–32, (2001).
[63] P.H. Winne. 'Experimenting to bootstrap self-regulated learning', *Journal of Educational Psychology,* **89**(3), 397–410, (1997).
[64] J.L. Zaichkowsky, 'Conceptualizing involvement', *Journal of advertising*, **15**(2), 4-34, (1986).

Towards Open Configuration

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

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

1 Introduction

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

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

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

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

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

2 Community-based Knowledge Engineering

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

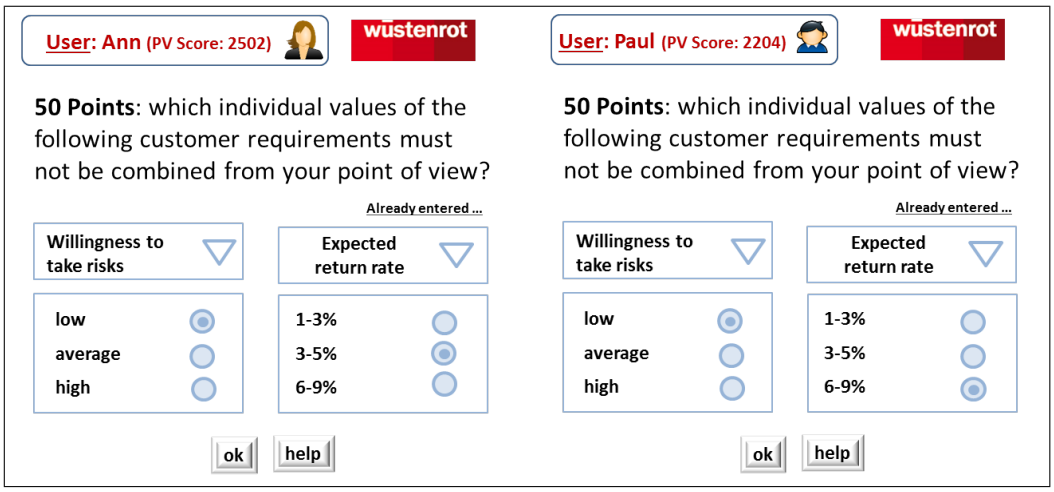


Figure 1. Sketch of a user interface for game-based knowledge acquisition. The overall goal of the game is that both players agree on the set of incompatible value combinations of a given set of variables. This user interface can be regarded as a micro task template for the acquisition of incompatibility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

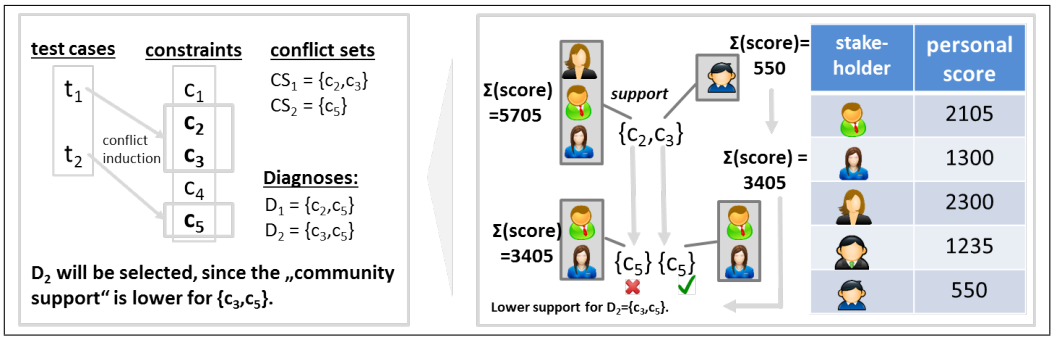


Figure 2. Group-based diagnosis of a faulty configuration knowledge base. Diagnoses are selected by taking into account the expertise of users/knowledge engineers: the higher the *personal score* (value derived from his/her personal contributions), the higher the weight given to his/her opinion.

in the configuration knowledge base introduced in Section 2.

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

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

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

3 Group-based Configuration

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4 Flexible Product Enhancement

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

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

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

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

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

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

5 Related and Future Work

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

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

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

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

6 Conclusions

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

charge of evaluating alternatives [13].

ACKNOWLEDGEMENTS

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

REFERENCES

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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