
© 2010 ACM.
Reprinted with permission
Personalized User Interfaces for Product Configuration

Alexander Felfernig
Graz University of Technology
Applied Software Engineering
Inffeldgasse 16b, A-8010 Graz
alexander.felfernig@ist.tugraz.at

Monika Mandl
Graz University of Technology
Applied Software Engineering
Inffeldgasse 16b, A-8010 Graz
monika.mandl@ist.tugraz.at

Juha Tiihonen
Helsinki University of Technology
Computer Science and Engineering
02015 TKK, Finland
juha.tiihonen@tkk.fi

Monika Schubert
Graz University of Technology
Applied Software Engineering
Inffeldgasse 16b, A-8010 Graz
monika.schubert@ist.tugraz.at

Gerhard Leitner
University of Klagenfurt
Interactive Systems
Universitätsstrasse 65-67, A-9020 Klagenfurt
gerhard.leitner@uni-klu.ac.at

ABSTRACT
Configuration technologies are well established as a foundation of mass customization which is a production paradigm that supports the manufacturing of highly-variant products under pricing conditions similar to mass production. A side-effect of the high diversity of products offered by a configurator is that the complexity of the alternatives may outstrip a user’s capability to explore them and make a buying decision. In order to improve the quality of configuration processes, we combine knowledge-based configuration with collaborative and content-based recommendation algorithms. In this paper we present configuration techniques that recommend personalized default values to users. Results of an empirical study show improvements in terms of, for example, user satisfaction or the quality of the configuration process.

Author Keywords
Configuration systems, recommender systems, model-based diagnosis.

ACM Classification Keywords
I.2.5.Expert system tools and techniques.

General Terms
Human Factors, Design, Algorithms

INTRODUCTION
Configuration systems have a long tradition as a successful application area of Artificial Intelligence, see, for example, [1,9,15,18,22]. On an informal level, configuration can be interpreted as a special case of design activity where the artifact being configured is assembled from instances of a fixed set of well-defined component types which can be composed conforming to a set of constraints [18]. Constraints can represent technical restrictions, rules regarding production processes, or restrictions that are related to economic factors. Example domains where product configurators are applied are computers, cars, financial services, railway stations, and complex telecommunication switches.

Although configuration has many advantages such as a significantly lower amount of incorrect quotations and orders, shorter product delivery cycles, and higher productivity of sales representatives [1], customers (users) in many cases have the problem of not understanding the set of offered options in detail and are often overwhelmed by the complexity of those options. The other problem is that users typically do not know exactly which products or components they would like to have. This phenomenon is described by the theory of preference construction [2] which follows from the fact that users do not know their preferences beforehand but rather construct and adapt their preferences within the scope of (in our case) a configuration process. In such a situation it makes sense to support users with recommendations that are, for example, derived from preferences articulated by similar users [23].

In this paper we present functionalities that support personalized configuration of mobile phones and corresponding subscriptions. Our major contribution is the integration of recommendation technologies with knowledge-based configuration (a functionality that is not available in commercial systems). The remainder of the paper is organized as follows. In the next section we present the recommendation approaches useful for supporting personalized configuration. Thereafter we shortly discuss results of empirical evaluations. Finally, we discuss related work and conclude the paper.

CONFIGURING, RECOMMENDING, ORDERING
In this section we will provide technical details that help to understand how our prototype implementation determines
repair alternatives in situations where no solution can be found, how recommendations for features are determined, and how phones are ranked taking into account user preferences.

Supporting configuration tasks. The task of identifying a configuration for a given set of specified customer requirements can be defined as follows:

**Definition 1 (configuration task):** A configuration task can be defined as a constraint satisfaction problem \((V, D, C)\). \(V=\{x_0, x_1, \ldots, x_n\}\) represents a set of finite domain variables and \(D=\{\text{dom}_0, \text{dom}_1, \ldots, \text{dom}_n\}\) represents a set of domains \(\text{dom}_i\) where \(\text{dom}_i\) is assigned to the variable \(x_i\). Finally, \(C = C_{\text{KB}} \cup C_{\text{R}}\) where \(C_{\text{KB}} = \{c_0, c_1, \ldots, c_m\}\) represents a set of domain constraints (the configuration knowledge base) that restrict the possible combinations of values assigned to the variables in \(V\) and \(C_{\text{R}} = \{r_0, r_1, \ldots, r_k\}\) represents a set of customer requirements.

A simple example for a configuration task is \(V=\{\text{styleReq}, \text{webUse}, \text{GPSReq}, \text{pModel}, \text{pStyle}, \text{pHSDPA}, \text{pGPS}\}\) where \(\text{styleReq}\) expresses the user’s preferred phone style, \(\text{webUse}\) specifies how often the user intends to access internet with the phone, and \(\text{GPSReq}\) specifies whether the user wants to use GPS navigation functionality. Table 1 specifies the existing phone models \(\{\text{pModel}\}\), their styles \(\{\text{pStyle}\}\), whether the phone supports fast internet access \(\{\text{pHSDPA}\}\), and whether the phone supports GPS navigation \(\{\text{pGPS}\}\). The respective domains are \(D=\{\{\text{any}, \text{bar}, \text{clam}\}\}, \{\text{no, occasional, often}\}, \{\text{false, true}\}, \{\text{p1, p2, p3}\}, \{\text{bar, clam}\}, \{\text{true, false}\}, \{\text{true, false}\}\}.

<table>
<thead>
<tr>
<th>pModel</th>
<th>pStyle</th>
<th>pHSDPA</th>
<th>pGPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>bar</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>p2</td>
<td>clam</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>p3</td>
<td>clam</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

Table 1: Available phone models in working example.

Furthermore, we introduce a set of domain constraints \(C_{\text{KB}} = \{c_0, c_1, c_2, c_3\}\). Table 1 can be interpreted as a constraint in disjunctive normal form, which yields \(c_0\). The remaining constraints represent the following domain properties:

- \(c_1\): \((\text{webUse} = \text{often}) \rightarrow (\text{pHSDPA} = \text{true}) \) /* frequent web use requires a fast internet connection */
- \(c_2\): \((\text{styleReq} = \text{any})\) OR \((\text{styleReq} = \text{pStyle})\) /* the phone should support the user’s preferred phone style */
- \(c_3\): \((\text{GPSReq} = \text{true}) \rightarrow (\text{pGPS} = \text{true}) \) /* if GPS navigation is available, the phone must support it */

Finally, an example for customer requirements is \(C_{\text{R}} = \{r_0; \text{styleReq} = \text{clam}, r_1; \text{webUse} = \text{often}, r_2; \text{GPSReq} = \text{false}\}\).

On the basis of this definition of a configuration task we can now introduce the definition of a solution for a configuration task (also denoted as configuration).

**Definition 2 (configuration):** A solution (configuration) for a given configuration task \((V, D, C)\) is represented by an instantiation \(I = \{x_0 = v_0, x_1 = v_1, \ldots, x_n = v_n\}\), where \(v_i \in \text{dom}_i\). A configuration is consistent if the assignments in \(I\) are consistent with the constraints in \(C\). Furthermore, a configuration is complete if all the variables in \(V\) have a concrete value. Finally, a configuration is valid, if it is both consistent and complete.

An example for a valid configuration is the following: \(\{\text{styleReq} = \text{clam}, \text{webUse} = \text{often}, \text{GPSReq} = \text{false}, \text{pModel} = \text{p3}, \text{pStyle} = \text{clam}, \text{pHSDPA} = \text{true}, \text{pGPS} = \text{false}\}\).

Diagnosing inconsistent requirements. In situations where no configuration can be found for a given set of requirements, we have to activate a diagnosis functionality [6,7,8,17]. Let us assume the following set of customer requirements \(C_{\text{R}} = \{r_1; \text{styleReq} = \text{bar}, r_2; \text{webUse} = \text{often}, r_3; \text{GPSReq} = \text{true}\}\). The setting in \(C_{\text{R}}\) does not allow the calculation of a solution; consequently, we have to identify a minimal set of requirements that has to be changed in order to be able to restore consistency. We are interested in minimal changes since we want to keep the original set of requirements the same as much as possible. The calculation of a minimal set of requirements that has to be changed is based on the determination of conflict sets (see the following definition) [11,17].

**Definition 3 (conflict set):** A conflict set is a set \(C_S \subseteq C_{\text{R}}\) s.t. \(C_{\text{KB}} \cup C_S\) does not allow the calculation of a solution. Furthermore, \(C_S\) is said to be minimal if there does not exist a set of \(C_S' \subset C_{\text{R}}\) such that \(C_S' \cup C_{\text{KB}}\) does not allow the calculation of a solution. In our working example we can identify the two minimal conflict sets \(C_{S_1}=\{r_1, r_2\}\) and \(C_{S_2}=\{r_1, r_3\}\). Both are conflict sets since \(\{r_1, r_2\} \cup C_{\text{KB}}\) as well as \(\{r_1, r_3\} \cup C_{\text{KB}}\) is inconsistent. Furthermore, both conflict sets are minimal since for both there does not exist a proper subset with the conflict set property (see Definition 3). In order to restore consistency, we have to resolve each of the identified minimal conflict sets. A systematic way to this is to apply the concept of model-based diagnosis [17]. A customer requirements (CR) diagnosis problem and a corresponding CR diagnosis can be defined as follows.

**Definition 4 (CR diagnosis problem and CR diagnosis):** A CR diagnosis problem is defined as a tuple \((C_{\text{KB}}, C_{\text{R}})\) where \(C_{\text{R}}\) is a set of requirements and \(C_{\text{KB}}\) represents the constraints of the configuration knowledge base. A diagnosis for \((C_{\text{KB}}, C_{\text{R}})\) is a set \(d \subseteq C_{\text{R}}\), s.t. \(C_{\text{KB}} \cup (C_{\text{R}} - d)\) is consistent. A diagnosis is minimal if there does not exist a diagnosis \(d' \subset d\), s.t. \(C_{\text{KB}} \cup (C_{\text{R}} - d')\) is consistent.

For determining the complete set of minimal diagnoses we can apply the algorithm proposed by [17]. The core of the concept presented in [17] is the Hitting Set Directed Acyclic Graph (HSDAG) algorithm that is complete in the sense that all the existing diagnoses are found.

**Recommending feature values.** Besides the calculation of diagnoses in the case that no solution could be found, the recommendation of feature values and the calculation of user-individual rankings for phones are important functionalities. To calculate recommendations for feature values, valid configurations of previous sessions are stored.
in a database. On the basis of these configurations two basic
algorithms are supported in our prototype environment: nearest neighbors (see [23]) and Naïve Bayes voter [5,23].
The Naïve Bayes voter is discussed in detail in [5,23] and is taken into account in this paper. An empirical evaluation of the
performance of different feature recommendation algorithms is a goal of future work.

Nearest neighbor based feature value recommendation. The
idea of a nearest neighbor algorithm is to determine the
neighbor configuration conf*, which is closest to the active
user's already specified requirements, and to recommend
feature values from this nearest neighbor. The distance
between the already specified user requirements and a
neighbor configuration conf* is defined as the sum of
individual distances [20] between corresponding feature
values, weighted by feature importance weights.

To calculate distances between feature values,
Heterogeneous Value Difference Metric (HVDM) [25] can
be applied which help to cope with both symbolic and
numeric features. The individual similarity metric to be
used for calculating the similarity between two feature
values is chosen depending on the basic characteristic of the
feature (less is better, more is better, or nearer is better –
for details see, for example, [20]). The distance values are
normalized to usually be in range 0 to 1. The similarity of
symbolic values in a domain is learned automatically [25].
This is done by examining the probability that individual
feature values contribute to the classification of the samples
- in our case classification of configurations. The higher the
probability of a pair of feature values to be present in
identically classified configurations, the more similar these
feature values are considered [25].

Maintaining the consistency of recommendations. Note that
recommendations for feature values must be consistent with
the already specified set of customer requirements, i.e., if
the user accepts a recommended feature value, this
selection should not trigger an inconsistency and the
activation of the diagnosis & repair component. In cases
where none of the candidate nearest neighbors is able to
provide a value that can be recommended, feature
recommendation can be omitted.

Similarity-based ranking of phones. For the ranking of
phones to be presented to the user we follow a similarity-
approach. We determine the distance of each
previous configuration to the user's current configuration,
so that phones from nearest configurations are shown first.
Phones that are compatible with user requirements are
presented to the user.

FIRST EMPIRICAL RESULTS
We have evaluated our prototype configuration environ-
ment within the scope of an empirical study with n>500
participants.¹ This study showed significant improvements
in terms of qualitative measures such as trust in a configu-
ration or the willingness to buy [4] as well as in terms of
measures such as prediction quality of the used similarity
measures (precision). In all those dimensions personalized
configurator versions outperformed non-personalized ver-
sions as they are still in use in commercial environments.
One of the major results of this initial study is a clear obser-
vation of decision biases where selection probabilities sig-
nificantly changed depending on the configurator version.

RELATED AND FUTURE WORK
Main-stream recommender applications are based on
collaborative filtering [13] and content-based filtering [16]
approaches. These approaches are predominantly applied to
quality and taste products – a very well known example is
amazon.com [14]. The application of pure collaborative or
content-based recommendation is the exception of the rule
- in many cases only hybrid approaches can solve problems
such as the ramp-up problem (e.g., for a new user the
recommender system does not dispose of rating data which
makes the calculation of initial recommendations a challen-
ging task). A discussion of this and further issues regarding
the deployment of recommenders can be found in [3].

Configuration systems have a long and successful history in
the area of Artificial Intelligence [1,9,15,18,22]. Although
these systems support interactive decision processes with
the goal to determine configurations that are useful for the
customer, the integration of personalization technologies
has been ignored with only a few exceptions – see, for
example, [5,10]. The goal of the work presented here was to
implement and evaluate a system that integrates
recommendation technologies that actively support users in
a configuration process.

The integration of recommendation technologies with
knowledge-based configuration is still in a very early stage.
Most of the existing commercial configuration environ-
ments are lacking of recommendation functionalities – the
study presented in this paper points out potentials for
improvements. There exist some contributions that take into
account the application of personalization technologies in
the configuration context. The authors of [10] introduce an
approach to the integration of case-based reasoning
to adapt nearest neighbors identified for the current
problem. There exist a couple of approaches that are similar
to [10] – see, for example, [5]. All of those approaches do
not provide a clear concept for enabling minimal changes
and handling inconsistent feature value recommendations.

¹ A detailed discussion of these results has been omitted due
to space limitations and will be provided in an extended
version of this paper.
CONCLUSIONS
In this paper we provided an overview of basic recommendation techniques that can be used in the context of configuring complex products and services. These techniques show to be useful in terms of improving the user acceptance of the configurator interface.

ACKNOWLEDGEMENTS
The work presented in this paper has been conducted within the research projects WECARE (Austrian Research Promotion Agency) and COSMOS (TEKES Finland).

REFERENCES