
Constraint-based personalized configuring of product and service bundles.

Markus Zanker*, Markus Aschinger and Markus Jessenitschnig

Institute of Applied Informatics ,
University Klagenfurt, Austria
E-mail: {firstname.lastname}@uni-klu.ac.at
*Corresponding author

Abstract: The configuring of product bundles such as tourism packages, financial services or compatible skin care products is a synthesis task that requires the support of a knowledgeable information system. We propose a personalized constraint-based configuration approach that is capable of solving such tasks in e-commerce environments. Here, a knowledge-based configurator is hybridized with collaborative methods from the domain of recommender systems in order to exploit user preferences to guide the problem solving process through large product spaces. The system implementation is based on a service-oriented architecture and the *Choco* open source constraint solver. It supports a model-driven approach for knowledge base design, acquisition and maintenance. An evaluation of the system suggests that it can solve realistic problem instances within an acceptable period of time¹.

Keywords: Personalized configuration; Recommender Systems; E-Tourism applications

Reference to this paper should be made as follows: Markus Zanker, Markus Aschinger and Markus Jessenitschnig (xxxx) 'Constraint-based personalized configuring of product and service bundles', *Int. J. of Mass Customization*, Vol. x, No. x, pp.xxx-xxx.

Biographical notes: Markus Zanker is an assistant professor in the Department for Applied Informatics at the University of Klagenfurt. He is also a co-founder and director of ConfigWorks GmbH, a provider of interactive selling solutions. He received his MS and Ph.D. degrees in Computer Science and his MBA in business administration from the University of Klagenfurt. His research interests focus on knowledge-based systems, particularly in the fields of interactive sales applications such as product configuration and recommendation. He also works on knowledge acquisition and user modeling for personalization.

Markus Aschinger has been a master student at the University of Klagenfurt. After receiving his MS degree in Computer Science in 2008 he is now a Ph.D. candidate in Computer Science at the University of Klagenfurt. His research interests include constraint satisfaction techniques, product configuration and planning.

Markus Jessenitschnig has been a research assistant and lecturer at the University of Klagenfurt since 2008. From 2006 to 2008 he was a member of the eTourism Competence Center Austria. After receiving his MS degree in Computer Science

¹This article improves and extends the work presented in Zanker et al. (2008) and Zanker et al. (2007)



he is now a Ph.D. candidate in Computer Science at the University of Klagenfurt. His research interests include user modeling, recommender systems and system architectures.

1 Introduction

In many e-commerce situations consumers are not looking for a single product item but rather require a set of several different products. For instance on e-tourism platforms online users may either selectively combine different items, such as accommodation, travel services, events or sights, or they might choose from an array of pre-configured packages. While bundling different items on their own users are performing a synthesis task comparable to configuration.

Mittal and Frayman (1989) defined configuration as a special type of design activity, with the key feature that the artifact being designed is assembled from a set of pre-defined components. When bundling different products together, product categories represent these pre-defined components. Additional knowledge is required that states which combinations of components are allowed and which restrictions need to be observed. For instance, proposed leisure activities should be acceptably close to the guest's accommodation or recommended sights need to be appropriate for children if the user represents a family. Nevertheless, the problem of computing product bundles that are compatible with a set of domain constraints differs from traditional configuration domains in the sense that fewer restrictions apply. For instance a car configurator must compute a valid vehicle variant satisfying the user's requirements and all applicable commercial and technical restrictions derived from the manufacturer's marketing and engineering policies. In contrast, few strict limitations apply to a product bundle, because it represents an intangible composition of products rather than a new artifact. As a consequence, an additional order of magnitude of component combinations are possible and the question of finding an optimal configuration becomes crucial.

Optimality can be either interpreted from the provider perspective, e.g. the configuration solution with the highest profit margin, or from the customer perspective. In the latter case, the system should propose the product bundle that best fits the customer's requirements and preferences. Typically, recommender systems are employed to derive a ranked list of product instances for an abstract goal such as maximizing user's utility or online conversion rates (Adomavicius and Tuzhilin, 1978). The most commonly used recommendation technique is collaborative filtering, which exploits clusters of past users with similar interests (peer users) to propose products that were well liked by these peers (Resnick et al., 1994). We exploit this type of recommender system to order personalized preferences for each type of product in our configuration problem. Our contribution thus lies in integrating soft preference information obtained from recommender systems (based on for instance the collaborative filtering paradigm) into a constraint-based configuration approach allowing user preferences to guide the system towards finding optimal product bundles.

In contrast to the work of Ardissono et al. (2003) and of Pu and Faltings (2004), we do not require explicit preference elicitation via questioning or example-critiquing, but depend on the underlying recommendation paradigm of community knowledge and past transaction data.

The paper is structured as follows: in Section 2 we present an extensive survey of



related work before introducing a motivating example in Section 3. Furthermore, we elaborate on the system's implementation and usage in Section 4 and conclude by summarizing practical experiences and results.

2 Related Work

Configuration systems are one of the most successful applications of AI-techniques. In industrial environments, they support the configuration of complex products and services and, compared to manual processes, help to reduce error rates and increase throughput. Depending on the underlying knowledge-representation mechanism, a rule-based, model-based or case-based framework may be employed for product configuration (Sabin and Weigel, 1998). Configurators that utilize the constraint satisfaction problem (CSP) paradigm are within the family of model-based approaches (Fleischanderl et al., 1998; Mailharro, 1998) and include an explicit knowledge base that is distinct from the system's problem solving strategy. In technical domains such as telephone switching systems, large problem instances with tens of thousands of components exist. Efficient strategies for solving problems exploit the functional decomposition of the product structure to determine valid interconnections of the different components (Fleischanderl et al., 1998). Pure sales configuration systems, such as online car or pc configuration systems^a, are much simpler from a computational point of view. They allow their users to explore the variant space of different options and add-ons and ensure that users place orders that are technically feasible and correctly priced. However, these systems are typically not personalized, i.e. they do not adapt their behavior according to their current user.

The CAWICOMS project was among the first to address the issue of personalization in configuration systems (Ardissono et al., 2003), developing a framework for personalized, distributed Web-based configurators. The system's dynamic user interfaces adapt their interaction style according to abstract user properties such as experience level or needs. The system decides on the questioning style (e.g. asking for abstract product properties or detailed technical parameters) and computes personalized default values if the user's assumed expertise is insufficient. Pu and Faltings (2004) present a decision framework based on constraint programming and demonstrate the suitability of soft constraints for supporting preference models. Their work concentrates on explicitly stated user preferences and presents an example critiquing interaction model to elicit tradeoff decisions from users. Given a specific product instance users may provide critique on one product property and specify which other properties they would be willing to compromise on. Soft constraints with priority values are revised in such an interaction scenario and guide the solution search.

In contrast to the work in Ardissono et al. (2003) and Pu and Faltings (2004), we do not solely rely on explicitly stated user feedback, but integrate a recommender system into the configuration process to include assumed user preferences.

Recommender systems constitute a base technology for personalized interaction and individualized product propositions in electronic commerce (Adomavicius and Tuzhilin, 1978). However, they do not support synthesis tasks like configuration. Given sets of items and users, recommender systems compute for each single user an individualized list of ranked items according to an abstract goal such as buyer interest or the likelihood of a sale (Adomavicius and Tuzhilin, 1978). Burke (2002) differentiates between five differ-

^aFor instance, see <http://www.bmw.com> or <http://store.apple.com>

ent recommendation paradigms: *collaborative*, *demographic* and *content-based* filtering as well as *knowledge* and *utility-based* recommendation. Collaborative filtering is the most well known technique that utilizes clusters of users that showed similar preferences in the past to provide recommendations to users in the same neighborhood (Resnick et al., 1994; Pazzani, 1999). Content-based filtering records the items that were liked by the user in the past and proposes similar ones. Successful application domains are, for instance, the suggestion of news or Web documents in general, where the system learns user preferences in the form of vectors of term categories (Balabanovic and Shoham, 1997). Demographic filtering builds upon the assumption that users with similar social, religious or cultural backgrounds share similar views and tastes. Knowledge and utility-based methods rely on a domain model of assumed user preferences that is developed by a human expert. Felfernig et al. (2006) developed a sales advisory system that maps explicitly stated abstract user requirements onto product characteristics and computes the set of best matching items. Case-based recommender systems exploit former successful user interactions denominated as cases. When interacting with a new user, the system retrieves and revises stored cases in order to make a proposition. Subsequently, a human expert is required to define efficient similarity measures for case retrieval (O'Sullivan et al., 2005). Burke (2000) and Ricci and Werthner (2002) have carried out extensive research within this field and have developed several successful recommendation systems. Hybrid recommendation strategies have been proposed to overcome common shortcomings such as cold start problems or considerable ramp-up effort by combining two or more recommendation paradigms (Burke, 2002). A *mixed* hybrid is a hybridization design where the recommendations of different recommender systems are presented together at the user interface level. In Cotter and Smyth (2000) a personalized TV guide is presented that composes a viewing schedule from the result lists of different recommender systems. In this example, predefined precedence rules are used to resolve conflicts. Analogously, the application described in this article can be seen as a specific instance of a mixed hybrid recommender that composes bundles of tourism packages, employing a CSP solver to resolve conflicts and to ensure that only consistent sets of items are bundled together.

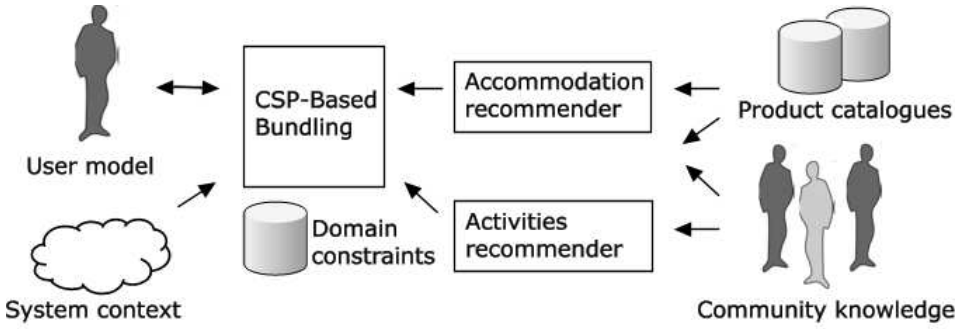
We chose to integrate a polymorphic recommendation service into our configurator that can be instantiated with an arbitrary recommendation paradigm. Technical descriptions on the system framework and the recommendation services can be found in (Zanker and Jessenitschnig, 2007; Jessenitschnig and Zanker, 2009b,a), while details on integrating the configuring of bundles with item recommendations are provided in Section 4.

Preference-based searches require a more interactive and dynamic approach for the personalized retrieval of items. Rather than relying on a static preference model, they build and revise the preference model of the specific user during interaction. One of the first applications of interactive assessment of user preferences was the Automated Travel Assistant (Linden et al., 1997). Further work on interactive preference elicitation has been conducted recently by Shimazu (2001); Smyth et al. (2004); Viappiani et al. (2006a) and Viappiani et al. (2006b) includes an extensive overview.

The work on preference-based search is orthogonal to our contribution as we primarily focus on computing bundles of recommendations. Our implementation supports interactivity between the system and the user during exploration of the search space. Therefore, additional preference constraints can be added and revised during each round of interaction (see Subsection 4.6).



Figure 1 Example scenario



3 Motivating example

To describe our approach, we start by giving a motivating example. Figure 1 depicts a service configuration scenario from the e-tourism domain. Based on the user model describing the needs of the current user and the system context, a CSP-based bundling approach computes personalized product portfolios by exploiting the domain constraints and recommendation lists of the collaborative recommender system. For reasons of simplicity we restrict the motivating example to two product categories: accommodation and activities. Table 1 lists the catalogue data for these two product categories. The *User model* contains a set of specific requirements for the current user such as that she is interested in a travel package consisting of a place to stay and activities that are appropriate for a family with children. In addition, contextual parameters such as the weather forecast or the current season, i.e. elements of the *System context*, should also be considered. In this example, we assume that the weather outlook is good. The CSP-based bundling service invokes several recommender systems to propose a ranked list of items for each product category based on the user model and available community knowledge. For instance a collaborative recommender instance derives recommendations by computing and analyzing similar peers of the current user based on item ratings or online behavior. In a second step domain knowledge in the form of constraints is exploited to find matching service bundles as outlined in Table 2. For instance, only activities that are appropriate for children should be proposed to families with kids and activities and accommodation should be close to one another. In addition, each constraint is associated with a weight that indicates its relative importance, e.g. the similarity of locations and the appropriateness for children is more important than the constraint to propose only indoor activities if the weather is bad. These constraints are typically defined by domain experts who formulate conventional knowledge and marketing restrictions applicable to the domain with the help of a knowledge acquisition environment (see Section 4.3).

Additional preference information is included when configuring the bundles because higher ranked recommendation items are more likely to be included in the configuration solution. Thus, if we assume that product items are ranked according to their ids by the recommendation services for accommodations and activities, then the system would propose the 2-tuples $\{(Hotel\ Post, Christmas\ market), (Hotel\ Goldener\ Hirsch, Hockey\ game)\}$ to the current user. Both highest ranked items from both categories are consistent with the domain constraints and are therefore bundled together. The second ranked hotel does not fit

Table 1 Product catalogue data

Accommodations		
Id	Property	Value
1	Name	<i>Hotel Post</i>
	Category	****
	City	<i>Innsbruck</i>
2	Name	<i>Hotel Adler</i>
	Category	****
	City	<i>Landeck</i>
3	Name	<i>Hotel Goldener Hirsch</i>
	Category	****
	City	<i>Innsbruck</i>
4	Name	<i>Hotel Kranebitter Hof</i>
	Category	***
	City	<i>Innsbruck</i>
Activities		
Id	Property	Value
1	Name	<i>Christmas market</i>
	Indoor	<i>no</i>
	Appr. children	<i>yes</i>
	City	<i>Innsbruck</i>
2	Name	<i>Hockey game</i>
	Indoor	<i>yes</i>
	Appr. children	<i>yes</i>
	City	<i>Innsbruck</i>
3	Name	<i>Rock concert</i>
	Indoor	<i>yes</i>
	Appr. children	<i>no</i>
	City	<i>Landeck</i>

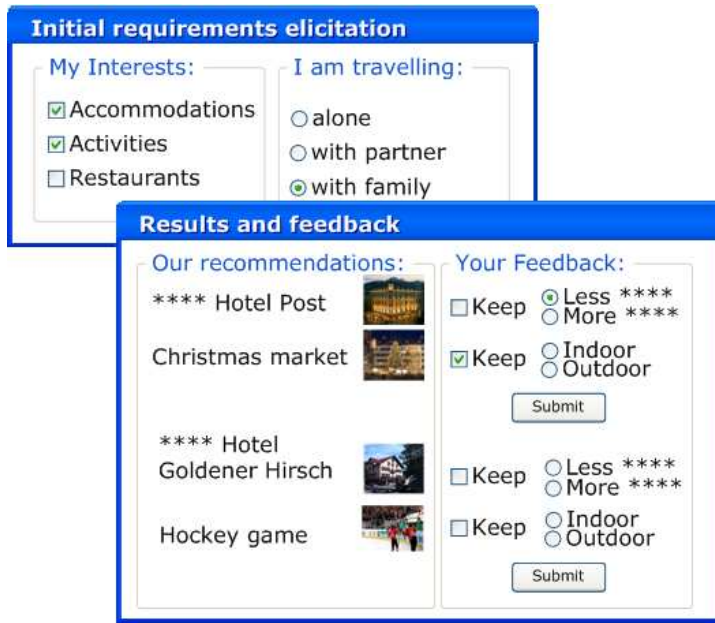
Table 2 Domain constraints

Id	Constraint	Weight
1	If user model.guest type* = ' <i>family with children</i> ' then activities.appropriate for children = ' <i>yes</i> '.	0.9
2	If system context.weather = ' <i>bad</i> ' then activities.indoor = ' <i>yes</i> '.	0.5
3	activities.city = accommodations.city	1

*Note, that we are using a object oriented notation where properties are qualified by the product catalogue, the user model or the system context.

together with any valid activity, as the only activity that can be done in the city of *Landeck* according to the third constraint violates the first constraint (cmp. Tables 1 & 2). Note, that in this example the two recommended bundles do not overlap. In Section 4.5 different solving strategies with respect to the overlap of items in recommended bundles are discussed. The user may now either accept one of the proposed bundles or give some additional feedback in the form of additional constraints. For instance, as sketched in Figure 2 the user might like the *Christmas market* included in the top-ranked bundle but would prefer a less

Figure 2 Sketch of user interaction



expensive hotel, i.e. a lower category. Therefore, two additional constraints are added for the subsequent solving step, namely $activities.id = 1$ and $accommodations.category < ****$. Consequently, only a single solution bundle remains to be recommended: $\{(Hotel\ Kranebitter\ Hof, Christmas\ market)\}$.

To summarize: given a user model, a system context and a set of domain constraints, the task of the system is to find a set of products from different categories that is optimal with respect to preference information derived from external recommender systems. The following section presents the implementation of the system.

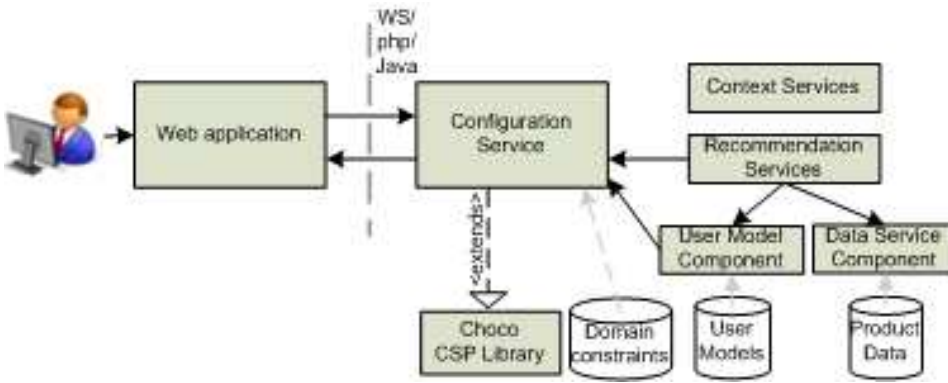
4 Implementation

While designing the system we decided to use the constraint programming paradigm for knowledge representation and problem solving - comparable to most of the configuration systems referenced in Section 2. The Java *Choco* open source constraint library (Jussien et al., 2008) forms the basis of our implementation. The following subsections describe the representation of the domain model through constraints, various aspects of knowledge acquisition as well as our system’s architecture.

4.1 Architecture

Figure 3 illustrates the system’s architecture. It consists of a configuration service set on top of a CSP library and several recommender service instances for delivering personalized instance rankings from a given class of products. The implementation utilizes a service-oriented architecture that supports communication via Web services (WS), a php-API and a Java-API, enabling flexible integration with a wide range of Web appli-

Figure 3 System architecture



cations and distributed deployment scenarios. Furthermore, it ensures that the system can be extended to include additional recommendation services. The latter requires sharing the identities of users and product instances as well as the semantics of user and product characteristics among all components. This is realized by central user and product model components that also offer service APIs.

The user interacts with a Web application that itself requests personalized product bundles from the configurator via the service-API. We have implemented variants of collaborative and content-based filtering recommenders as well as utility and knowledge-based ones as sketched in Zanker and Jessenitschnig (2007). The evaluation of contextual parameters is requested from a further service component. A more detailed description of the different recommendation strategies available can be found in Zanker et al. (2007); Zanker (2008); Zanker and Jessenitschnig (2007). Next, we examine the domain model for the configuration service itself.

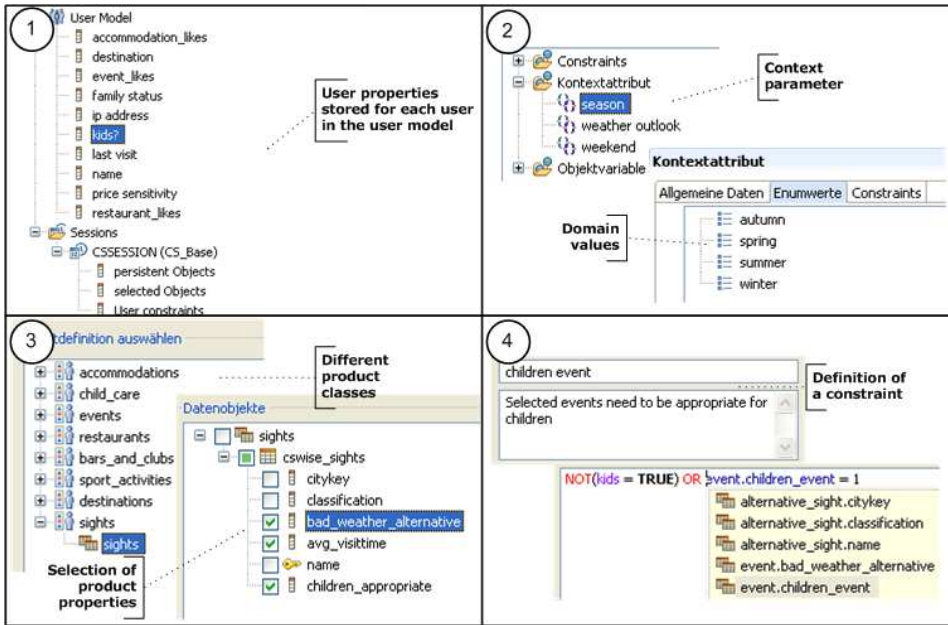
4.2 Model representation

As stated above, the constraint satisfaction paradigm is employed for knowledge representation. A Constraint Satisfaction Problem (CSP) is defined as follows Tsang (1993):

A CSP is defined by a tuple $\langle X, D, C \rangle$, where $X = \{x_1, \dots, x_n\}$ is a set of variables, $D = \{d_1, \dots, d_n\}$ a set of corresponding variable domains and $C = \{c_1, \dots, c_m\}$ a set of constraints.

Each variable x_i may only be assigned a value $v \in d_i$ from its domain. A constraint c_j further restricts the valid assignments within a set of variables. For each partial value assignment to variables it is possible to determine if a constraint has been violated or not. In addition, all constraints $c_j \in C$ are defined to be either *hard* ($c_j \in C_{hard}$) or *soft* ($c_j \in C_{soft}$), where $C = C_{hard} \cup C_{soft}$ and $C_{hard} \cap C_{soft} = \emptyset$. Soft constraints may be violated by variable assignments if such violations are required to obtain a solution. Each violation is typically associated with a penalty value, the sum of which is minimized for an optimal solution. For further details of CSPs we refer the reader to Tsang (1993).

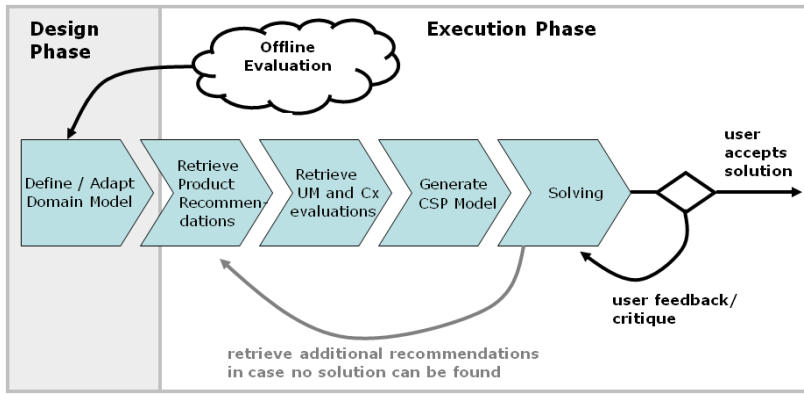
Figure 4 Knowledge acquisition workbench



Next, based on this formalization we present our domain model. It consists of a tuple $\langle X_{\{UM, Cx, P, PM, OPT\}}, D_{\{P, PM\}}, C_{\{hard, soft\}} \rangle$, where:

- $X = X_{UM} \cup X_{Cx} \cup X_P \cup X_{PM} \cup X_{OPT}$ the set of variables subdivided into several disjoint subsets as explained below,
- $D = D_P \cup D_{PM}$ the sets of the corresponding domains for the product classes and their properties,
- $C = C_{hard} \cup C_{soft}$ the set of constraints subdivided into hard and soft constraints,
- $X_{UM} = \{u_1, \dots, u_j\}$ a set of variables from the user model,
- $X_{Cx} = \{cx_1, \dots, cx_k\}$ a set of variables modeling the system context,
- $X_P = \{p_1, \dots, p_i\}$ a set of index variables representing the product classes, each of which is associated with a recommendation service that delivers a personalized item ranking upon request,
- $weight(p_i)$ the relative weight of product class p_i used in the overall optimization function,
- $X_{PM} = \{p_1.x_1, \dots, p_1.x_m, \dots, p_i.x_1, \dots, p_i.x_n\}$ a set of variables modeling product properties, where $p.x$ denotes the product property x of product class p and $p[j].x$ the concrete evaluation of x for product instance j ,
- $X_{OPT} = \{rv_{prod}, rv_{soft}, rv_{opt}, sc_1, \dots, sc_q\}$ a set of resource variables for the optimization and a set of penalty variables for representing the soft constraints,
- $D_P = \{d_1, \dots, d_i\}$ a set of corresponding domains for the product classes,

Figure 5 Design and execution phase



- $D_{PM} = \{p_1.d_1, \dots, p_1.d_m, \dots, p_i.d_1, \dots, p_i.d_n\}$ a set of corresponding domains for product properties,
- $C_{hard} = \{c_1, \dots, c_p\}$ a set of hard constraints on variables in X ,
- $C_{soft} = \{c_1, \dots, c_q\}$ a set of soft constraints on variables in X and finally
- $pen(c_j)$ the penalty value for relaxing soft constraint c_j .

This domain model is defined and maintained solely by domain experts in order to reduce the traditional knowledge acquisition bottleneck as outlined in the next subsection.

4.3 Model definition and CSP generation

Model definition and maintenance is supported by a modular editor environment based on the Eclipse RCP (Rich-Client Platform) technology^b. Figure 4 gives an overview of the interaction with the knowledge acquisition workbench. First, the relevant characteristics for configuring a bundle are retrieved from the user model repository. Second, additional external services providing contextual data such as the current season of the year or weather information are selected. In a third step, the set of product categories or classes P and their associated recommender services are integrated. For each class of products relevant properties are selected from the underlying repository. Finally, hard and soft constraints are defined using a context-sensitive editor.

Figure 5 depicts the complete process. This can be separated into a design phase, where the model is defined and maintained, and an execution phase. During the latter, the configurator is invoked for a specific user u . In the following subsections each of the phases is described in greater detail. First, product rankings are retrieved from recommendation services and then corresponding product characteristics are requested from the product model repository. In the next step, all variables in $X_{UM} \cup X_{Cx}$ are assigned values by the particular user model repository and the context services.

Then a CSP model is generated on the fly and transformed using the following procedure:

^bSee <http://www.eclipse.org/rcp> for reference.

- Create all variables in $X_{UM} \cup X_{Cx}$ in the CSP model and assign them their respective domains.
- For each index variable $p_i \in P$, create a related domain $d_i = \{1, \dots, n\}$, where n is the number of recommendations received for product class i . Product instance $p_i[1]$ denotes the highest ranked recommendation and $p_i[n]$ the lowest ranked respectively. The index represents the preference information about the instances of a product class. If two product instances fulfill all constraints then the one with the lower index value should be included as part of the recommended bundle.
- Create all variables in X_{PM} and assign them domains as follows: $\forall p_i \in X_P, \forall p_i.x_j \in X_{PM}, \forall idx \in d_i \quad p_i[idx].x_j \in p_i.d_j$, i.e. for a given product property x_j from product class p_i , all characteristics of recommended instances need to be in the domain $p_i.d_j$.
- Furthermore, integrity constraints are required to ensure that the value of index variable p_i of product class i is consistent with the values assigned to its product properties $p_i.x$: $\forall p_i \in X_P, \forall p_i.x_j \in X_{PM}, \forall idx \in d_i \quad p_i = idx \rightarrow p_i.x_j = p_i[idx].x_j$, i.e. when choosing the idx -th instance of product class i ($p_i = idx$), the related product property $p_i.x_j$ is assigned the value of the idx -th instance. This methodology enables us to represent complex product records types, although the *Choco* CSP formalism does not support structured variable representations by itself.
- Insert all domain constraints from $C_{hard} \cup C_{soft}$.
- For each soft constraint $c_i \in C_{soft}$, create a variable sc_i that holds the penalty value $pen(c_i)$ if c_i is violated or 0 otherwise:

$$\text{Penalty-Variable } sc_i = \begin{cases} pen(c_i) & \text{if } c_i \text{ is violated} \\ 0 & \text{else} \end{cases}$$

- Create the resource variables rv_{prod} , rv_{soft} and rv_{opt} . rv_{prod} represents the adjusted sum of all index variables and rv_{soft} represents the adjusted sum of all variables holding penalty values for soft constraints. rv_{opt} is defined as the weighted sum of rv_{prod} and rv_{soft} . The optimization model will be discussed in more detail in the next subsection.

4.4 Optimization

The optimization process manages the trade-off between satisfying soft constraints and including the highest ranked items from recommender results, balancing soft constraints violations against the inclusion of lower indexed product items in bundles. This is achieved using two resource variables rv_{prod} and rv_{soft} that are normalized on an interval $[0 \dots 1]$, where 0 means only top ranked items are included or all soft constraints satisfied, and 1 indicates that the lowest ranked product instances are chosen or all soft constraints are violated. The computation of both resource variables is explained in the following.

Recommendation results are represented by index variables p_i for every product class. For each product class p_i , a parameter $weight(p_i)$ indicates the relative weight of the product class in the overall optimization function. The resource variable rv_{prod} is calculated as follows:

$$rv_{prod} = \sum_{i=1}^{|X_P|} \frac{p_i}{|d_i|} * \frac{weight(p_i)}{Sum_{prod}}$$

$|X_P|$ indicates the total number of product classes in the model and $|d_i|$ represents the number of instances for product class p_i . Sum_{prod} is the sum of weights of all product classes:

$$Sum_{prod} = \sum_{i=1}^{|X_P|} weight(p_i)$$

The optimization of the soft constraints is similar to that of the index variables defined above.

$$rv_{soft} = \sum_{i=1}^{|C_{soft}|} \frac{sc_i}{Sum_{soft}}$$

$|C_{soft}|$ represents the total number of soft constraints and sc_i is a variable employed to hold an optional penalty value for use in the case that soft constraint c_i is violated and 0 otherwise. Sum_{soft} is the overall sum of all penalty values of soft constraints, i.e.:

$$Sum_{soft} = \sum_{i=1}^{|C_{soft}|} pen(c_i)$$

Scaling both parts of the optimization model is necessary so that a tradeoff factor Δ indicating the relative importance of rv_{prod} versus rv_{soft} can be set to balance relaxing soft constraints against adding items with higher product indexes (i.e. less recommended) to the computed bundles. The tradeoff factor Δ can be set to a value in the interval $[0 \dots 10]$ in the knowledge acquisition workbench. The resource variable rv_{opt} has to be minimized in order to find optimal bundles, i.e.:

$min \ rv_{opt}$, where

$$rv_{opt} = \Delta * rv_{prod} + (10 - \Delta) * rv_{soft}$$

4.5 CSP solving

Once the CSP model has been generated, the *Choco* solver is invoked. Its optimization algorithm aims to find suitable values for all variables in the CSP model such that all hard constraints are satisfied while minimizing the resource variable rv_{opt} . We extended the branch and bound optimization algorithm in Jussien et al. (2008) to compute the top n solution tuples instead of solely a single product bundle (see Algorithm 1). A trivial approach would be to compute the best bundle in one run, add constraints to exclude this solution, run the search again to get the second best solution and so on. Our approach computes the desired number of solutions in a single run which requires on average just a few milliseconds longer than one run of the standard algorithm. This leads to a significant speed-up in the overall search process. The pseudo-code of the algorithm is listed in Algorithm 1 below. Initially the first n solutions are stored in a sorted list without setting bounds. In subsequent iterations the upper bound is set to the value of the current n -th best solution. Aside from this, the algorithm mainly proceeds in a similar fashion to the

standard algorithm. Whenever a new solution is found, it is inserted at the right position in the sorted list of solutions and the upper bound is updated with the new n -th best solution. A similar approach for modifying branch and bound can be found in Torrens (2002).

```

Input:  $n$  . . . desired number of product bundles
Output: solutions . . . sorted list of solutions in descending order
 $upperBound \leftarrow +\infty$ 
solutions  $\leftarrow$  array  $[1, \dots, n]$  of integer
while ( $\#solutions < n$ ) and (new solution found) do
  | solution  $\leftarrow$  getNextSolution()
  | insert solution in solutions in descending order
if  $\#solutions < n$  then
  | return solutions
 $upperBound \leftarrow$  value of the current  $n$ -th best solution  $- 1$ 
while new solution found do
  | solution  $\leftarrow$  getNextSolution()
  | insert solution in solutions in descending order
  |  $upperBound \leftarrow$  value of the current  $n$ -th best solution  $- 1$ 
return solutions

```

Algorithm 1: modified branch & bound algorithm

The solver is capable of following different strategies for computing n solution instances depending on the required level of diversity between them. The *1-different* strategy represents an extreme case, ensuring that each tuple of product instances in the set of n solutions contains at least one product instance that is different to every other solution bundle. This strategy thus results in solution sets including product bundles with minimal diversity. For instance, let us assume a set of fictitious product bundles denoted in Table 3 that are enumerated with decreasing utility, i.e. Bundle 1 is the highest ranked solution. When applying a *1-different* solving strategy, the set of solutions would encompass all bundles in Table 3, because all possible pairs of bundles differ by at least one product instance. The *all-different* variant, on the other hand, guarantees that the intersection between two product bundles from the set of solutions is empty, i.e. an instance of one product class such as accommodations may only occur in a single solution tuple. In contrast to the *1-different* strategy, *all-different* results in maximum diversity between the calculated product bundles. In the scenario denoted in Table 3, only the set of bundles $\{1, 4\}$ would fulfill the requirement of pairwise disjoint item sets according to the *all-different* strategy. Note, that this strategy has different semantics than the well-known *all-different* constraint from the constraint programming community (van Hoesve, 2001). While the *all-different* constraint would restrict variable assignments in a single solution bundle (i.e. in each row of Table 3), the *all-different* solving strategy checks that the assignment of product instances to product classes is disjoint for each column. Another solving strategy called *π -different* permits a projection π of product categories to be defined for which the intersection of product items between any two computed bundles has to be empty. For instance, if the *π -different* strategy were applied with $\pi = \{Accommodation\}$, the potential combination of consistently configured bundles would consist of number 1 and 3. In the case that $\pi = \{Restaurant, Activity\}$, recommended solutions contain bundles number 1, 2 and 4.

Bundle ID	Accommodation	Restaurant	Sightseeing
1	A1	R1	S1
2	A1	R2	S3
3	A2	R1	S1
4	A2	R3	S2

Table 3 Examples for different product bundles

If a single solution bundle is unable to satisfy all domain constraints, the variable domains could be extended by retrieving additional lower ranked recommendations and restarting the model generation and solving steps (compare to the shaded arrow in Figure 5). However, this option is not implemented because it is more efficient to retrieve larger recommendation sets initially instead of invoking the recommendation services several times. In Section 5 we further explore the performance of CSP generation and solving steps.

4.6 Interactivity

The system also supports conversational user interaction throughout the configuration process. As described in Figure 5, the user either accepts a proposed solution or requests a modified configuration.

In the latter case he or she may provide some feedback by explicitly accepting or rejecting some components of the proposed bundle which results in the addition of explicit equality or inequality constraints or some form of critique that refines the user's preference model, e.g. *category < ***** in the introductory example. The interface of the configuration solver thus accepts all forms of constraint-based user feedback and supports therefore an incremental preference elicitation strategy comparable to Viappiani et al. (2006b). Thus in each round of interaction additional constraints are added to the CSP model.

Furthermore, interaction sessions can be restored and resumed at a later point in time. Then, stored solutions can be reused, where a subset of product categories of a specific product bundle is fixed and new instances for the remaining product classes are recalculated. This is in particular relevant when long-lasting user sessions need to be supported, where users continuously receive bundled recommendations that need to be consistent with each other over time. The *etPlanner* mobile recommendation and guidance framework supporting travelers during several phases of their trip gives an example for this (Hoepken et al., 2006).

5 Evaluation

Based on our e-tourism application domain we developed an example scenario consisting of 5 product classes with a total of 30 different product properties. Their domains are strings, bounded integers or boolean values. We defined a total of 23 representative domain constraints (13 hard and 10 soft constraints) as configuration knowledge. Constraints have been defined either on a single or on two variables, thus they are either unary or binary. The evaluation focuses solely on computation time and doesn't consider the overall quality of the final recommendations, as this relies mainly on the quality of the knowledge base and the employed recommenders which are treated by the system as black boxes.

Model	Number of Recs.	Number of Vars	Average Domain Size	Number of Constraints	Generation time in ms
M1	5	58	7,45	206	10
M2	10	58	8,73	374	20
M3	30	58	13,55	1010	60
M4	50	58	16,5	1355	95
M5	100	58	23,23	2093	135

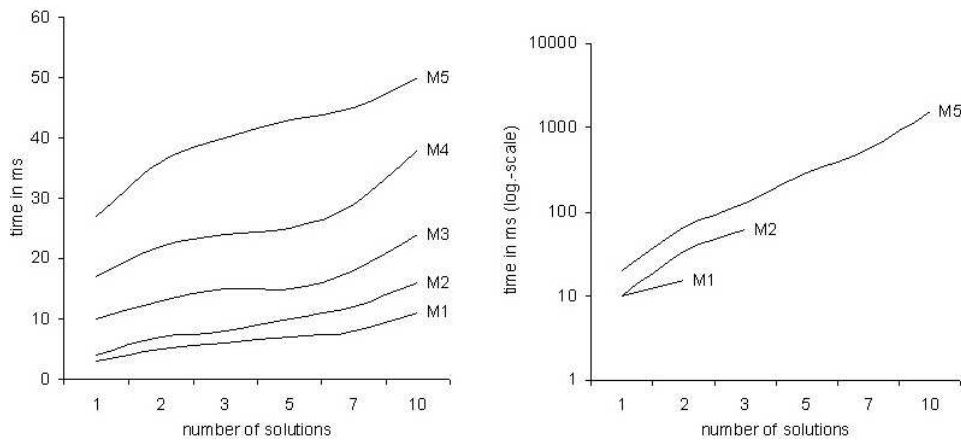
Table 4 Model sizes used in evaluation

We decided to adopt an experimental research design and manipulated the problem size whilst measuring time and space complexity. Space complexity is given as the number of generated variables and constraints while time complexity is measured in terms of model generation time and the time required by the constraint solving step. We created 5 different CSP models (denoted M1 to M5) requiring between 5 and 100 product instances. Details of the problem sizes and computation times for 'on-the-fly' generation of CSP models are given in Table 1. Clearly, the number of variables does not depend on the number of recommendations and is constant for all models. However, the average size of variable domains increases from M1 to M5 due to the higher number of product instances per product class. The number of constraints depends mainly on the aforementioned integrity constraints that model arrays of product records. M5, for example, contains ten times the number of constraints found in M1. As can be seen in Table 1, the times for generating even the large M5 model are acceptable for interactive applications.

In order to evaluate the performance of the system experiments were conducted using a computer containing a standard Pentium 4 3GHz processor. The time measurements for the solving steps are depicted in Figure 6. A series of 100 test runs (representing 100 different users) were executed for each model to obtain an average computation time. We evaluated the two extreme cases of possible solving strategies, i.e. the *1-different* and the *all-different* variant. In each run we computed a set consisting of the n top-ranked solution tuples, where n was varied between 1 and 10. The solve times (y-axis) for the *all-different* strategy are presented using a logarithmic scale. Moreover, graphs for M3 and M4 are omitted for readability. The *1-different* strategy is significantly less complex and therefore solving times did not exceed 50 ms. In contrast, *all-different* requires significantly longer.

Nevertheless, performance is still satisfactory. The highest value for model M5 (for 10 solutions) was computed in about 1.5 seconds. Although typical e-commerce situations require at most 10 different product bundle suggestions, requests for as many as 100 solution tuples can still be computed within an acceptable time period. For instance, when using the *1-different* solution strategy the solver required around 220 ms to calculate the top 100 solutions for model M5. In the case of the *all-different* strategy, a maximum of 15 solutions could be found in this problem instance. The associated calculations required 6 seconds which indicates that this would be the likely bottleneck of an interactive system.

Nevertheless, our results indicate that the integration of different recommender systems into a configurator is efficient enough to solve standard online product bundling tasks. Further experiments in different example domains will be conducted as future work.

Figure 6 Results for solution strategies *1-different* (left) and *all-different* (right)

6 Conclusion

We presented the development of a generic Web configurator that combines recommendation functionality with a constraint solver to compute product bundles. Our contribution lies in the system's novel ability to personalize product bundles as well as in presenting a specific type of mixed hybrid recommendation strategy. The system observes both domain restrictions represented by a constraint base and user preferences derived from recommender systems. It is therefore novel in its approach to integrating preference information derived from recommendation techniques such as collaborative filtering with constraint-based inferencing. A pilot application was developed within the scope of an industrial research project in the e-tourism domain which subsequently showed that the system performs acceptably for realistic problem sizes. Future work will deal with the relaxation of constraints and diagnosing possible conflicts between constraints as well as experimenting with different recommendation strategies.

References

- Adomavicius, G. and Tuzhilin, A. (2005) 'Towards the next generation of recommender systems: A survey of the state-of-the-art and possible extensions', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 2, pp. 734–749.
- Ardissono, L., Felfernig, A., Friedrich, G., Goy, A., Jannach, D., Petrone, G., Schäfer, R. and Zanker, M. (2003) 'A Framework for the Development of personalized, distributed web-based Configuration Systems', *AI Magazine*, Vol. 24, No. 3, pp. 93–108.
- Balabanovic, M. and Shoham, Y. (1997) 'Fab: Content-based, collaborative recommendation', *Communications of the ACM*, Vol. 40, No. 3, pp. 66–72.
- Burke, R. (2000) 'The Wasabi Personal Shopper: A Case-Based Recommender System', in 11th *Conference on Innovative Applications of Artificial Intelligence (IAAI)*, Trento, IT, AAAI, pp. 844–849.

- Burke, R. (2002) 'Hybrid recommender systems: Survey and experiments', *User Modeling and User-Adapted Interaction*, Vol. 12, No. 4, pp. 331–370.
- Cotter, P. and Smyth, B. (2000) 'PTV: Intelligent Personalized TV Guides', in 11th *Innovative Applications of Artificial Intelligence (IAAI)*, Trento, IT, AAAI, pp. 957-964.
- Felfernig, A., Friedrich, G., Jannach, D. and Zanker, M. (2006) 'An integrated Environment for the Development of knowledge-based Recommender Applications', *International Journal of Electronic Commerce*, Vol. 11, No. 2, pp. 11–34.
- Fleischanderl, G., Friedrich, G., Haselböck, A., Schreiner, H. and Stumptner, M. (1998) 'Configuring large systems using generative constraint satisfaction', *IEEE Intelligent Systems*, Vol. 17 (July-August), pp. 59–68.
- Hoepken, W., Fuchs, M., Zanker, M., Beer, T., Eybl, A., Flores, S., Gordea, S., Jessenitschnig, M., Kerner, T., Linke, D., Rasinger, J. and Schnabl, M. (2006) 'etPlanner: An IT Framework for Comprehensive and Integrative Travel Guidance', in 13th *International Conference on Information Technology and Travel & Tourism (ENTER)*, Lausanne, Switzerland, pp. 125–134.
- Jessenitschnig, M. and Zanker, M. (2009a) 'A generic user modeling component for hybrid recommendation strategies', in 11th *IEEE Conference on Commerce and Enterprise Computing (CEC)*, Vienna, Austria.
- Jessenitschnig, M. and Zanker, M. (2009b) 'ISeller: A Flexible Personalization Infrastructure for e-Commerce Applications', in 10th *International Conference on Electronic Commerce and Web Technologies (EC-Web)*, Linz, Austria.
- Jussien, N., Rochart, G. and Lorca, X. (2008) 'The CHOCO constraint programming solver', in *CPAIOR'08 Workshop on Open-Source Software for Integer and Constraint Programming (OSSICP)*, Paris, France.
- Linden, G., Hanks, S. and Lesh, N. (1997) 'Interactive assessment of user preference models: The automated travel assistant', in 5th *International Conference on User Modeling (UM)*, Lyon, France, pp. 67–78.
- Mailharro, D. (1998) 'A classification and constraint-based framework for configuration', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Vol. 12, 383–397.
- Mittal, S. and Frayman, F. (1989) 'Toward a generic model of configuration tasks', in 11th *International Joint Conferences on Artificial Intelligence*, Menlo Park, CA, USA, pp. 1395–1401.
- O'Sullivan, D., Smyth, B. and Wilson, D. (2005) 'Understanding case-based recommendation: A similarity knowledge perspective', *International Journal of Artificial Intelligence Tools*, Vol. 14, No. 1-2.
- Pazzani, M. (1999) 'A framework for collaborative, content-based and demographic filtering', *Artificial Intelligence Review*, Vol. 13, No. 5-6, pp. 393–408.
- Pu, P. and Faltings, B. (2004) 'Decision tradeoff using example-critiquing and constraint programming', *Constraints*, Vol. 9, pp. 289–310.

- Resnick, P., Iacovou, N., Suchak, N., Bergstrom, P. and Riedl, J. (1994) 'GroupLens: An open architecture for collaborative filtering of netnews', in *Computer Supported Collaborative Work (CSCW)*, Chapel Hill, NC, USA, pp. 175-186.
- Ricci, F. and Werthner, H. (2002) 'Case base querying for travel planning recommendation', *Information Technology and Tourism*, Vol. 3, pp. 215-266.
- Sabin, D. and Weigel, R. (1998) 'Product configuration frameworks - a survey', *IEEE Intelligent Systems*, Vol. 17 (July/August), pp. 42-49.
- Shimazu, H. (2001) 'Expert clerk: Navigating shoppers' buying process with the combination of asking and proposing', in *17th International Joint Conference on Artificial Intelligence (IJCAI)*, Seattle, WA, USA, pp. 1443-1448.
- Smyth, B., McGinty, L., Reilly, J. and McCarthy, K. (2004) 'Compound critiques for conversational recommender systems', in *IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, Washington, DC, USA, IEEE Computer Society, pp. 145-151.
- Torrens, M. (2002) *Scalable intelligent electronic catalogs*, Thesis No. 2690, Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland.
- Tsang, E. (1993) *Foundations of Constraint Satisfaction*, Academic Press, London, UK.
- van Hoes, W. J. (2001) 'The Alldifferent Constraint: A Survey', in *6th Annual Workshop of the ERCIM Working Group on Constraints*, Prague, Czech Republic.
- Viappiani, P., Faltings, B. and Pu, P. (2006a) 'Evaluating preference-based search tools: a tale of two approaches', in *22th National Conference on Artificial Intelligence (AAAI)*, Boston, USA, pp. 205-211.
- Viappiani, P., Faltings, B. and Pu, P. (2006b) 'Preference-based search using example-critiquing with suggestions', *Artificial Intelligence Research*, Vol. 27, pp. 465-503.
- Zanker, M. (2008) 'A collaborative constraint-based meta-level recommender', in *Second ACM International Conference on Recommender Systems (RecSys)*, Lausanne, Switzerland, ACM Press, pp. 139-146.
- Zanker, M., Aschinger, M. and Jessenitschnig, M. (2007) 'Development of a collaborative and constraint-based web configuration system for personalized bundling of products and services', in *8th International Conference on Web Information Systems Engineering (WISE)*, Nancy, France, Springer, pp. 273-284.
- Zanker, M., Aschinger, M. and Jessenitschnig, M. (2008) 'Constraint-based personalized bundling of products and services', in *18th European Conference on Artificial Intelligence (ECAI) - Workshop on Configuration Systems*, Patras, Greece, pp. 23-28.
- Zanker, M. and Jessenitschnig, M. (2007) 'ISeller - a generic and hybrid recommendation system for interactive selling scenarios', in *15th European Conference on Information Systems (ECIS)*, St. Gallen, Switzerland, pp. 70-80.
- Zanker, M. and Jessenitschnig, M. (2009) 'Case-studies on exploiting explicit customer requirements in recommender systems', *User Modeling and User-Adapted Interaction, Special Issue on Data Mining for Personalization*, Vol. 19, No. 1-2, pp. 133-166.

Zanker, M., Jessenitschnig, M., Jannach, D. and Gordea, S. (2007) 'Comparing recommendation strategies in a commercial context', *IEEE Intelligent Systems*, Vol. 22(May-June), pp. 69–73.

