Intelligent Debugging of Utility Constraints in Configuration Knowledge Bases

A. Felfernig, M. Mandl, and M. Schubert

Institute for Software Technology, Graz University of Technology
Inffeldgasse 16b, A-8010 Graz, Austria
{*alexander.felfernig*, *monika.mandl*, *monika.schubert@ist.tugraz.at*}

Abstract: Knowledge-based configurators support customers in preference construction processes related to complex products and services. In this context *utility constraints* (*scoring rules*) play an important role. They determine the order in which configurations are presented to customers. In many cases utility constraints are faulty, i.e., calculate configuration rankings which are not expected and accepted by marketing and sales experts. The repair of these constraints is extremely time-consuming and often an error-prone task. In this paper we present an approach which effectively supports *automated debugging and repair of faulty utility constraint sets*. This approach allows us to automatically take into account intended rankings of configurations specified by marketing and sales experts.

1 Introduction

Knowledge-based configurators [Fe07] exploit deep knowledge about the product or service domain in order to determine solutions fitting the wishes and needs of customers. Alternative configurations (complete or partial) must be ranked according to their utility for the customer. Due to serial position effects, which induce customers to preferably take a look at and select configurations at the beginning of a list, personalized rankings can significantly increase the trust in the presented configurations [La51]. We apply Multi-Attribute Utility Theory (MAUT) [WE86] for the determination of such rankings. When using MAUT, each configuration is evaluated according to a predefined set of interest dimensions. *Profit*, *availability*, and *risk* are examples of such interest dimensions in the domain of financial services. If a customer is more interested in *high return rates* than in the *availability* of a service, the dimension *profit* is very important. This way, customer requirements influence the importance of interest dimensions.

A successful application of configurators requires that utility constraints (scoring rules - see Figure 1) consistently reflect the marketing and sales strategies of a company. In the financial services domain an example for such a utility constraint is *if both, balanced funds and bonds are part of the configuration result, the utility of bonds should be higher than that of balanced funds. The manual repair of utility constraints to produce results consistent with the marketing and sales strategy is a time-consuming and error-prone task since those constraints are strongly interdependent.* Therefore our goal was to develop techniques that effectively support knowledge engineers in identifying faulty elements in

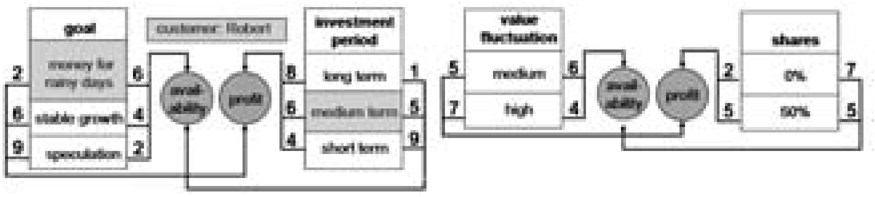

Figure 1: *Scoring rules for customer properties* {*goal*, *investment period*}, *customer requirements* {*customer*:*Robert*}, and *scoring rules for service properties* {*value fluctuation*, *shares*}.

utility constraint sets. In this paper we introduce debugging and repair techniques that automatically identify the sources of inconsistencies in utility constraint sets. By exploiting Model-Based Diagnosis (MBD) [Re87] we are able to automatically determine *minimal* sets of repair actions for faulty utility constraints.

The remainder of the paper is organized as follows. The next section introduces a set of utility constraints for the ranking of different financial service configurations. This constraint set is used as working example throughout this paper. Section 3 sketches our approach to transform a set of utility constraints into a corresponding Constraint Satisfaction Problem (CSP) which is used as a basis for applying MBD techniques. Section 4 sketches our approach to the automated debugging and repair of faulty utility constraints. Section 5 concludes the paper.

2 Working Example: Utility Constraints

We now present a simplified set of utility constraints used as a working example throughout the paper. The basic elements of Multi-Attribute Utility Theory (MAUT) [WE86] are *interest dimensions* that describe focuses of interest of customers. Our example contains two such interest dimensions: *profit* and *availability* (see Figure 1). *Profit* denotes the expected performance in terms of *return rate*. *Availability* is related to the *accessibility of the invested sum* within the targeted investment period. The degree to which a customer is interested in such dimensions can be directly derived from the requirements articulated within a configuration session. A customer interested in *short term investments* (*investment period* = *short term*) has a higher interest in *availability* than a customer who is interested in *long term investments* (*investment period* = *long term*). Similarly, customers interested in *stable-growth investments* (*goal* = *stable growth*) have a lower interest in very high profits than those interested in *speculations* (*goal* = *speculation*).

In our example we use the requirements shown in Figure 1: customer *Robert* is interested in *medium term* investments with the goal of *putting money by for a rainy day*. By interpreting the utility constraints of Figure 1 we can determine a distribution for the interest dimensions, i.e., to which extent *Robert* has a focus on the dimensions *profit* and *availability*. *Robert* requires a *medium term investment* solution which contributes an importance of 6 to the dimension *profit* and an importance of 5 to the interest dimension *availability* (see Figure 1). Furthermore *Robert* is interested in putting money by for a rainy day which contributes an importance of 2 to the dimension *profit* and an importance of 6 to *availability*. On the basis of customer preferences (expressed as the extent of customer interest on the interest dimensions – see Table 2) we evaluate the suitability of alternative configura-

customer	profit	availability
Robert	6+2=8	5+6=11

Table 1: Example customer interests.

configuration	shares	value fluctuation
balanced funds	50%	medium
bonds	0%	medium
bonds2	0%	high

Table 2: Example set of financial services (configurations).

tions. We introduce the following simplified assortment of financial service configurations $\{$*balanced funds*, *bonds*, *bonds2*$\}$ as a basis for our working example (see Table 2).

The next step is to define the utility of configurations w.r.t. interest dimensions. We define a dependence between configuration attribute values and the interest dimensions. For instance, financial services including shares support a higher *profit* (score 5 in Figure 1). Financial services without shares have a higher degree of *availability* (score 7) and financial services with a higher value fluctuation have a higher (expected) *profit* (score 7). By interpreting the utility constraints of Figure 1, we can determine the extent to which our financial services contribute to the interest dimensions *profit* and *availability* (see Table 3): *balanced funds* have a higher *profit* than bonds but by the majority a lower degree of *availability*. Vice-versa, bonds have a higher degree of *availability*.

On the basis of the identified configuration utilities, we can determine the customer-specific utility of each configuration (see Table 4). The utility of a configuration is determined using formula *utility*(x) = $\sum_{i=1}^{n} in_i con_i(x)$ where $utility(x)$ specifies the overall utility of a configuration x for a specific customer. The overall utility is defined as sum over the interest dimensions i of the customer's interest in dimension (in_i) (see Table 1) times the contribution of configuration x to dimension i (con_i). In our example, *balanced funds* have a higher utility for *Robert* than *bonds* and *bonds2*.

3 Utility Constraint Sets as CSP

In order to be able to apply Model-Based Diagnosis [Re87] to the identification and repair of faulty utility constraints, we transform a given set of utility constraints (representation of Figure 1) into a corresponding Constraint Satisfaction Problem (CSP) [Fe07, Ts93]. Model-based Diagnosis uses the description of a system as a starting point. In our case, this is the description of the intended behavior of a set of utility constraints which is spec-

configuration	profit	availability
balanced funds	5+5=10	5+6=11
bonds	2+5=7	7+6=13
bonds2	2+7=9	7+4=11

Table 3: Utilities of configurations regarding interest dimensions.

customer	configuration	profit	availability	utility
Robert	balanced funds	8*10	11*11	201
	bonds	8*7	11*13	199
	bonds2	8*9	11*11	193

Table 4: Utilities of configurations for example customer.

example	investment period	goal	ranking
e_1	medium term	for rainy days	utility(bonds) $>$ utility(balanced funds)
e_2	medium term	for rainy days	utility(bonds2) $>$ utility (balanced funds)
e_3	medium term	for rainy days	utility(bonds) $>$ utility(bonds2)

Table 5: Examples $e_i \in E$ for intended configuration orderings.

ified by a set of examples describing the intended ordering of configurations in a result set (our examples are introduced in Table 5). If the actual system behavior (rankings calculated by the utility constraints) conflicts with its intended behavior (rankings described by examples), the diagnosis task is to determine those components (utility constraints) which, when assumed to functioning abnormally, explain the discrepancy between actual and intended behavior.

Following the definitions of Figure 1, we introduce a set of utility constraints related to the required investment period and the personal goals of the customer. For instance, constraint c_1 denotes the fact that for customers requiring financial services with short term investment periods, the dimension *profit* is of medium importance, whereas *availability* aspects play a significantly more important role (c_2). $\{c_1, c_2\}$ are example utility definitions derived from Figure 1.

c_1: profit$_{(investmentperiod\_short)}$ = 4 c_2: availability$_{(investmentperiod\_short)}$ = 9

We denote each constraint defining utility values as utility constraint $c_i \in$ C. Since we are interested in a utility constraint set consistent with all the examples $e_i \in$ E, we have to check the consistency of the given set of utility constraints with $\bigcup e_i$. This consistency check requires a representation where each example is described by a separate set of finite domain variables. For instance, the contribution to *profit* provided by the customer attribute *investment period* in example e_1 is stored in *profit* $_{(investmentperiod\_e1)}$. The following representation can be directly applied by our diagnosis algorithm.

e_1: profit$_{(investmentperiod\_e1)}$= profit$_{(investmentperiod\_medium)}$ $\wedge$

availability$_{(investmentperiod\_e1)}$= availability$_{(investmentperiod\_medium)}$ $\wedge$

profit$_{(goal\_e1)}$=profit$_{(goal\_rainydays)}$ $\wedge$

availability$_{(goal\_e1)}$= availability$_{(goal\_rainydays)}$ $\wedge$ utility$_{(balancedfunds\_e1)}$ $<$ utility$_{(bonds\_e1)}$

The overall customer interest in the dimension *profit* is represented by profit(e_i). The values of these variables represent the sum over all defined contributions of customer requirements of example e_i to the dimension *profit*. This approach is analogously applied to the dimension *availability* (availability(e_i)). We denote constraints summing up customer utilities as $s_i \in$ S, e.g., s_1: profit(e_1)= profit$_{(investmentperiod\_e1)}$+profit$_{(goal\_e1)}$

For each configuration we specify its contribution to the interest dimensions. For instance, the shares percentage specified for *balancedfunds* defines an average contribution to the dimension *profit*. In our CSP, we define this fact as *profitshares*$_{(balancedfunds)}$ = 5.

Analogously, we define the relationship between the interest dimension *availability* and shares percentage as *availabilityshares*$_{(balancedfunds)}$ = 5.

We denote each constraint defining a utility value for a configuration (financial service) as utility constraint $p_j \in$ P. The following constraints exemplify the implementation of the definitions of Figure 1.

p_1: profitshares$_{(balancedfunds)}$=5 p_2: availabilityshares$_{(balancedfunds)}$=5

For each $c_i \in$ C (and each $p_i \in$ P) we add a corresponding *repair constraint* cr_i (pr_i) which specifies potential repairs for c_i (p_i). The idea behind repair constraints is that if c_i (p_i) is identified as a faulty element by the diagnosis component, we have to identify a consistent repair action for c_i (p_i). This repair should change the original c_i (p_i) as little as possible.[1] Therefore, we define an interval for the accepted changes for each $c_i \in$ C and each $p_i \in$ P. Each of the following example repair constraints allows changes of the given evaluations by at most one unit. We denote $\bigcup cr_i \cup \bigcup pr_i$ as the set of repair constraints R.

cr_1: profit$_{(investmentperiod\_short)}$ IN [3,4,5] pr_1: profitshares$_{(balancedfunds)}$ IN [4,5,6]

The *profit* of a financial service is defined by the sum of contributions of the values of shares and fluctuation. *Availability* of a service is as well defined by the sum of related contributions of those attribute values. We denote each rule summing up service utility values as $s_\alpha \in$ S. For example, the following constraint implements summing of profit utility values as defined in Table 3.

s_a: profit$_{(balancedfunds)}$=profitshares$_{(balancedfunds)}$ + profitfluctuation$_{(balancedfunds)}$

The following constraint exemplifies the calculation of service utilities, where utility(x_{ei}) specifies the utility of configuration x in the context of example e_i (see Table 4).

s_b: utility$_{(balancedfunds\_e1)}$= profit$_{(balancedfunds)}$*profit(e_1)+

availability$_{(balancedfunds)}$*availability(e_1)

The constraints introduced above are constituting elements of a Constraint Satisfaction Problem (CSP) [Fe07, Ts93] which is defined as tuple (V, D, Con), where *V* is a set of finite integer domain variables, *D* represents the domain of each variable, and *Con* is a set of constraints consisting of exactly those constraints defined in (C, P, R, S, E).

4 Debugging Utility Constraints

For utility constraint set repair we apply Model-Based Diagnosis (MBD) [Re87] techniques. In the sense of MBD, a faulty component (erroneous utility constraint) is expressed through an inconsistency between the utility constraint set and the specified examples. A MAUTKB diagnosis indicates faulty constraints whose repair will restore consistency. We exploit examples which are provided by domain experts or online customers. Those examples specify the expected ranking of configurations. Depending on a set of customer requirements, the configurator should rank configurations consistent with the rankings specified in the given set of examples (in our case $\{e_1, e_2, e_3\}$). With regard to our ex-

[1]Note that changes of at most one unit are only introduced for this example, the range of possible changes is more flexible. In the current implementation the range can be specified by knowledge engineers.

ample utility constraints, $\{e_1,e_2,e_3\}$ ∪ C ∪ P ∪ R ∪ S is inconsistent. Consistency can be restored by deleting $\{p_2\}$ from P. In this context, $\{p_2\}$ is a minimal diagnosis, i.e., a minimum set △ of constraints which have to be deleted from C ∪ P ∪ R (or adapted), s.t. $\{e_1,e_2,e_3\}$ ∪ ((C ∪ P ∪ R) - △) ∪ S becomes consistent. We now introduce the definition of a *MAUTKB Diagnosis Problem* and a corresponding *MAUTKB Diagnosis*.

Definition: MAUTKB Diagnosis Problem. A MAUTKB (MAUT Knowledge Base) Diagnosis Problem is defined as a quintuple (C, P, R, S, E), where C is a set of customer utility constraints, P is a set of utility rules related to offered configurations, R is set of repair constraints, and S is a set of summarization rules. Finally, E is a set of examples for the intended ordering of result sets.

Definition: MAUTKB Diagnosis. A MAUTKB Diagnosis for a MAUTKB Diagnosis Problem is a set △ ⊆ C ∪ P ∪ R, s.t. ((C ∪ P ∪ R) - △) ∪ S ∪ E is consistent.

The calculation of diagnoses is based on *minimal conflicts* induced by e_i ∈ E.

Definition: Conflict Set CS. A Conflict Set is defined as CS = $\{c_1, c_2, ..., c_k\}$ ⊆ C ∪ P ∪ R s.t. CS ∪ S ∪ E inconsistent. CS is minimal iff not(∃ CS': CS' ⊂ CS).

To calculate diagnoses we extended the algorithm proposed by [Re87]. For reasons of space limitations, we will not present the algorithm in detail here.

5 Conclusion

In this paper we have presented innovative knowledge engineering techniques that effectively support knowledge engineers and domain experts in the development and maintenance of utility constraint sets used for the calculation of configuration rankings. We proposed debugging and repair approaches for identifying faulty elements in utility constraint sets that are based on extending and applying Model-Based Diagnosis (MBD) techniques. These approaches can potentially save significant amounts of efforts in important but error-prone, difficult, and frustrating development tasks.

References

[La51] K.S. Lashley. The problem of serial order in behavior. In L. A. Jeffress (Ed.), Cerebral mechanisms in behaviour (pp. 112-136). New York: Wiley, 1951.

[PBJ93] J.W. Payne, J.R. Bettman, and E.J. Johnson. The Adaptive Decision Maker. Cambridge Univ. Press, 1993.

[Re87] R. Reiter. 1987. A theory of diagnosis from first principles. AI Journal, 23, 1, 5795, 1987.

[SW98] M. Stumptner, and F. Wotawa. A Survey of Intelligent Debugging. European Journal on Artificial Intelligence (AICOM), 11(1):35-51, 1998.

[WE86] D. Winterfeldt, W. Edwards. Decision Analysis and Behavioral Research, Cambridge University Press, Cambridge, England, 1986.

[Fe07] A. Felfernig, Standardized Configuration Knowledge Representations as Technological Foundation for Mass Customization, IEEE Trans. on Eng. Mgmt., 54(1), pp. 41-56, 2007.

[Ts93] A. Tsang, Foundations of Constraint Satisfaction, Academic Press, London, 1993.