Debugging User Interface Descriptions of Knowledge-based Recommender Applications

Alexander Felfernig
University Klagenfurt
Universitätsstrasse 65-67
A-9020 Klagenfurt, Austria
++43/463/2700/3754
alexander.felfernig@uni-klu.ac.at

Kostyantyn Shchekotykhin
University Klagenfurt
Universitätsstrasse 65-67
A-9020 Klagenfurt, Austria
++43/463/2700/3768
kostya@ifit.uni-klu.ac.at

ABSTRACT

The complexity of product assortments offered by e-Commerce platforms requires intelligent sales assistance systems alleviating the retrieval of solutions fitting to the wishes and needs of a customer. Knowledge-based recommender applications meet these requirements by allowing the calculation of personalized solutions based on an explicit representation of product, marketing and sales knowledge stored in an underlying recommender knowledge base. Unfortunately, in many cases faulty models of recommender user interfaces are defined by knowledge engineers and no automated support for debugging such process designs is available. This paper presents an approach to automated debugging of faulty process designs of knowledge-based recommenders which increases the productivity of user interface development and maintenance. The approach has been implemented for a knowledge-based recommender environment within the scope of the Koba4MS project.

Categories and Subject Descriptors

I.2.5 [Programming Languages and Software].: *Expert system tools and techniques.*

General Terms

Algorithms, Design, Human Factors.

Keywords

Knowledge-based Recommenders, Knowledge Acquisition, Personalization, Diagnosis.

1. INTRODUCTION

Buying products and services (e.g., digital cameras, computers, financial services, or books) in online selling environments is still a challenging task for customers since many companies offer simple query interfaces under the assumption that customers are experts in the product domain [1,27]. Customers have to know how to specify requirements on a technical level in order to find solutions fitting to their wishes and needs. *Recommender technologies* [1,3,4,8,12,17,19,21,27] improve this situation by providing solution alternatives for the customer which are automatically derived from a set of customer requirements. These technologies are of great importance for making product assortments accessible to users without technical domain knowledge. There are three basic approaches to the implementation of recommender applications. *Collaborative Filtering* [12,19,21] is based on the concept of storing preferences of a large set of customers. Based on the assumption that human preferences are correlated, recommendations given to a customer are derived from preferences of a group of customers with similar interests, i.e., no deep knowledge about product properties is needed. Similarly, using *Content-based Filtering* [4,17], products are described by a set of keywords (categories) which are stored in a customer profile in the case that a customer buys a certain product. The next time, the customer enters the recommender application, the stored preferences are used for identifying additional products which are assigned to similar categories. Finally, *Knowledge-based Recommender* applications (advisors) [1,3,8] exploit deep knowledge about the product domain in order to determine solutions which exactly fit to the wishes and needs of the customer. When selling complex products such as financial services, a customer's taste is not of primary concern - primarily solutions and explanations must be correct in every case. This requirement can only be met by explicitly representing product, marketing, and sales knowledge [6,16], i.e., *Knowledge-based Recommender* applications are the natural choice in this context.

Two basic aspects have to be considered when implementing a knowledge-based recommender application. Firstly, the relevant set of products (or services) has to be identified and transformed into a corresponding formal representation, i.e., a *recommender knowledge base* [7,8,16] has to be defined. Such a knowledge base consists of a structural description of the provided set of products (or services), a description of the possible set of customer requirements and a set of constraints restricting the possible combinations of customer requirements and product properties (see Figure 1 - customer properties and constraints in textual form). Secondly, a model of a *recommender process* [8] has to be defined which serves as the basis for the execution of customer-specific dialogs based on personalized navigation

IUI'06, January 29–February 1, 2006, Sydney, Australia.

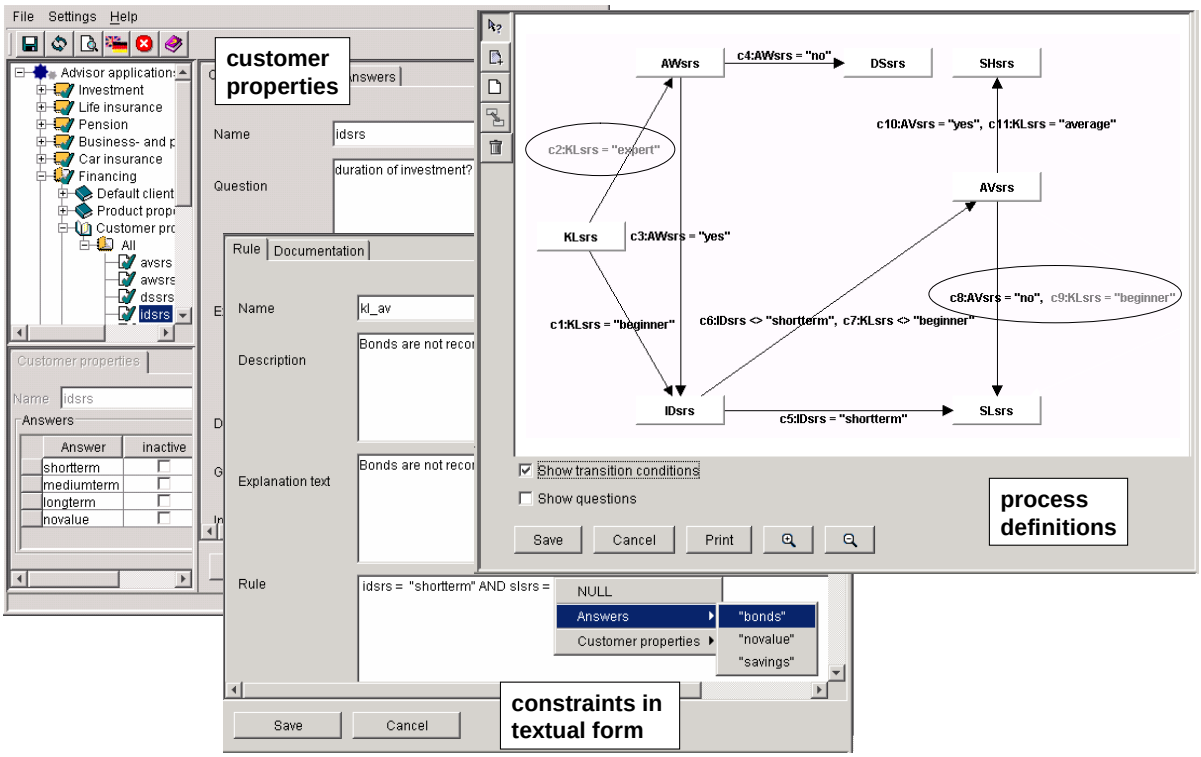

Figure 1: Koba4MS Development Environment.

paths through a recommender application (see Figure 1 - process definitions). Knowledge bases and process definitions can be automatically (no programming is needed) translated into an executable advisor, where each state of the process definition corresponds to a Web-page in the generated application. Figure 3 depicts the interface of a financial services recommender application. After a user has specified his requirements, the *Next* button triggers the determination of the following state. In each state of a recommender process a user can articulate his requirements and depending on the given answers the process control determines the following state. Having reached a final state, a corresponding set of product recommendations is displayed to the customer. Examples for such results are concrete types of digital cameras or financial services such as different types of savings or bonds. In the case that no solution can be found for the given set of customer requirements, constraint violation handling and repair of customer requirements are triggered.

In this paper we focus on a situation where knowledge engineers or domain experts develop a model of the user interface of a knowledge-based recommender application (e.g., a financial service recommender application or one for digital cameras). The interfaces of such interactive applications are described by a finite number of states. State transitions are triggered by user requirements represented by a set of variable settings which serve as input for the constraint solver of the recommender environment. Finite state automata [13] can be used for expressing the expected behaviour of a recommender user interface [2,26]. Figure 1 (process definitions) depicts a simple example for the model of the intended behaviour of a financial services recommender application (advisor) as it is represented in our Koba4MS[1] environment. Figure 2 shows the corresponding finite state automaton (PFSA - Predicate-based Finite State Automaton), Example 1 provides details related to the process definition. Typically, customers make decisions by specifying values (requirements) for a subset of a given set of variables. Depending on the answers the automaton changes its state, e.g., an expert (kl = expert) who isn't interested in financial services advisory (aw = no), is forwarded to the state q_3, where a direct product search (search based on product parameters) can be performed, i.e., different subsets of variables are defined by different paths in the automaton.

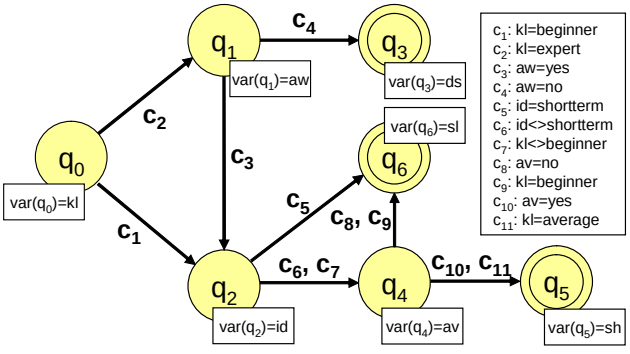

Figure 2: Example (faulty) PFSA. $\{c_1, c_2, ..., c_{11}\}$ are transition conditions between states q_i, $var(q_i)$ represents the variable which is instantiated by the user in q_i. We use a short-cut notation (e.g., *klsrs* in Figure 1 is denoted as *kl*).

[1]Koba4MS is an acronym for Knowledge-based Advisors for Marketing&Sales (Austrian Research Fund Project 808479).

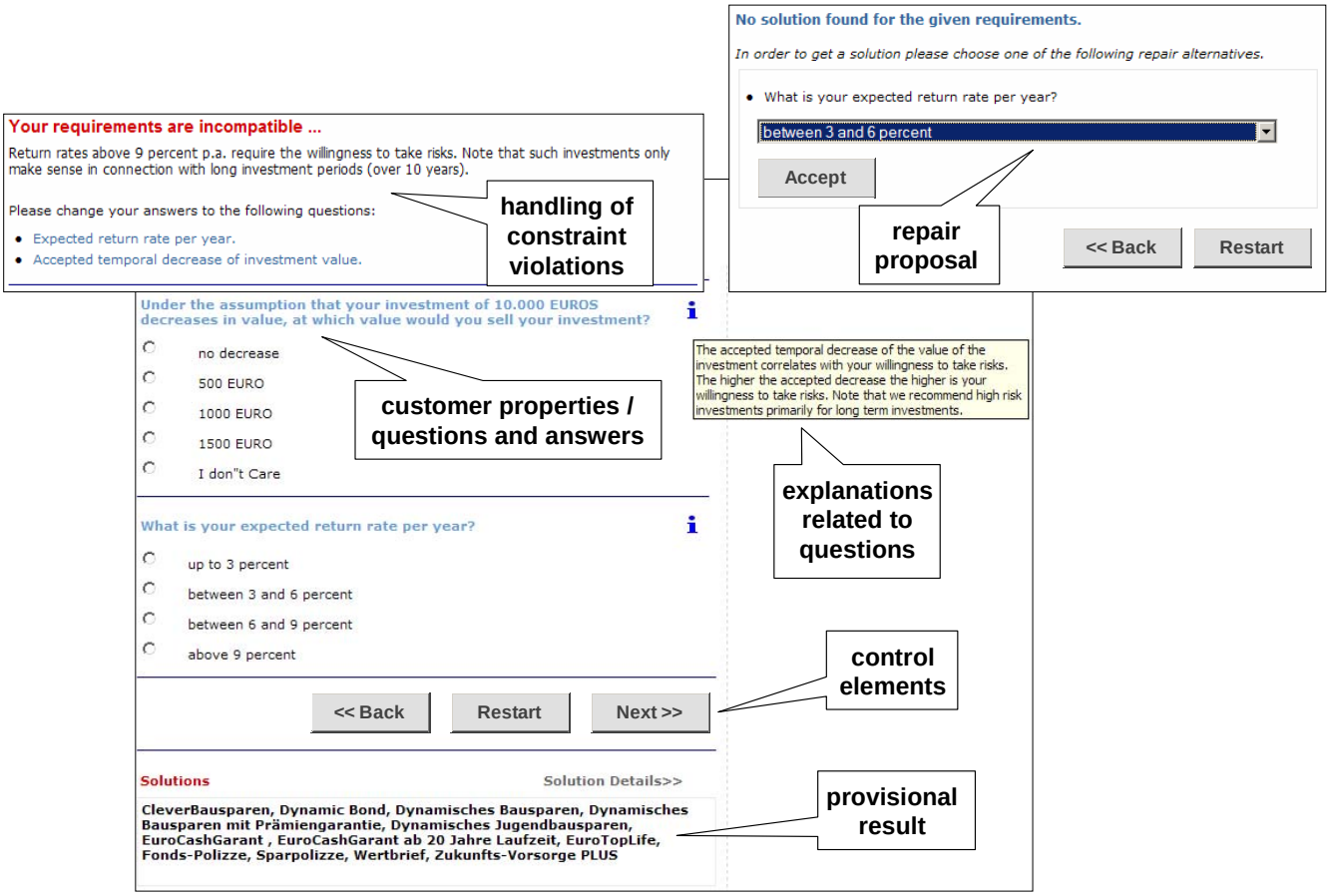

Figure 3: User Interface of Financial Services Recommender.

Note that the design in Figure 2 is faulty, since an expert (kl = expert) who wants to be guided by a financial advisory process (aw = yes) and is interested in long-term investments (id = longterm) and doesn't have any available funds (av = no) comes to a standstill at the input of availability (the conditions $\{c_2, c_9\}$ and $\{c_2, c_{11}\}$ are contradictory). In such a situation, additional design support is needed which provides an effective way for identifying the sources of inconsistencies, i.e., the potential sources of inconsistencies should clearly be indicated to the developer of the recommender process definition. In the following we show how concepts from model-based diagnosis (MBD) [18] are applied for identifying a minimal set of changes allowing consistency recovery in finite state representations of recommender user interfaces. The presented concepts have been implemented within the scope of the Koba4MS project, where finite state models are used to specify the intended behaviour of a user interface (see Figure 2).

The remainder of this paper is organized as follows. In Section 2 we introduce Predicate-based Finite State Automata (PFSA), as formalism for modelling navigational behaviour of knowledge-based recommender applications. In Section 3 we discuss our approach to the automated debugging of PFSA-based representations of recommender processes. In Section 4 we present results from evaluating the presented concepts within the scope of a study related to the investigation of effort reductions in development and maintenance caused by the application of model-based diagnosis concepts (Reiter, 1987).

2. FINITE STATE REPRESENTATIONS OF RECOMMENDER USER INTERFACES

In contrast to conventional finite automata (see, e.g., [13]) we introduce a variant of Predicate-based Finite State Automata (PFSA) [25] (see, e.g., Figure 2) which is a more natural and compact approach to define state transitions (transitions are defined in terms of domain restrictions of finite domain variables). For the following discussions we introduce the notion of a Predicate-based Finite State Automaton (PFSA), where we restrict our discussions to the case of *acyclic* automata.

Definition 1 (PFSA**):** we define a Predicate-based Finite State Automaton (recognizer) (PFSA) to be a 6-tuple (Q, Σ, ∏, E, S, F), where

- $Q = \{q_1, q_2, ..., q_j\}$ is a finite set of states, where $var(q_i) = x_i$ is a finite domain variable assigned to q_i, $prec(q_i) = \{\phi_1, \phi_2,..., \phi_m\}$ is the set of preconditions of q_i ($\phi_k = \{c_r, c_s, ..., c_t\} \subseteq \prod$), $postc(q_i) = \{\psi_1, \psi_2, ..., \psi_n\}$ is the set of postconditions of q_i ($\psi_l = \{c_u, c_v, ..., c_w\} \subseteq \prod$), and $dom(x_i) = \{x_i=d_{i1}, x_i=d_{i2}, ..., x_i=d_{ip}\}$ denotes the set of possible assignments of x_i, i.e., the domain of x_i.

- $\Sigma = \{x_i = d_{ij} \mid x_i = var(q_i), (x_i = d_{ij}) \in dom(x_i)\}$ is a finite set of variable assignments (input symbols), the input alphabet.
- $\prod = \{c_1, c_2, ..., c_q\}$ is a set of constraints (transition conditions) restricting the set of words accepted by the PFSA.
- E is a finite set of transitions $\subseteq Q \times \prod \times Q$.
- $S \subseteq Q$ is a set of start states.
- $F \subseteq Q$ is a set of final states. □

The set $prec(q_i) = \{\phi_1, \phi_2, ..., \phi_m\}$ represents a disjunction of invariants in the state q_i which can be automatically derived from the reachability tree of a PFSA (a common approach to study properties of finite state models). Figure 4 represents the reachability tree for the PFSA depicted in Figure 2. The state q_2 is accessed twice in the reachability tree, the preconditions of q_2 (i.e., $prec(q_2)$) can be directly derived from the transition conditions of the paths leading to q_2, i.e., $\{\{c_1\}, \{c_2, c_3\}\}$. Similarly, $postc(q_i)$ represents the set of possible postconditions of the state q_i which are as well derived from the reachability tree, e.g., the state q_4 has two postconditions, namely $\{\{c_8, c_9\}, \{c_{10}, c_{11}\}\}$. The following is an example for a PFSA describing the behaviour of our financial services advisory dialog (see Figure 2). States represent questions posed to customers. Depending on the answers given by a customer, the subsequent state is determined. An advisory process is completed when a final state is reached (the result page of a recommender application). The set of input sequences leading to a final state is also denoted as the language accepted by the PFSA.

Example 1 (PFSA**):**

$\{Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$.
$var(q_0) = kl.$ /* knowledge level of the customer */
$var(q_1) = aw.$ /* advisory wanted */
$var(q_2) = id.$ /* duration of investment */
$var(q_3) = ds.$ /* direct specification of services */
$var(q_4) = av.$ /* availability of financial resources*/
$var(q_5) = sh.$ /* high risk products */
$var(q_6) = sl.$ /* low risk products */
dom(kl)={kl=beginner,kl=average,kl=expert}.
dom(aw)={aw=yes,aw=no}.
dom(id)={id=shortterm,id=mediumterm,id=longterm}.
dom(ds)={ds=savings,ds=bonds,ds=stockfunds,ds=singleshares}
dom(av)={av=yes,av=no}.
dom(sh)={sh=stockfunds,sh=singleshares}.
dom(sl)={sl=savings,sl=bonds}.
$prec(q_0) = \{\{true\}\}$. $prec(q_1) = \{\{c_2\}\}$.
$prec(q_2) = \{\{c_1\}, \{c_2, c_3\}\}$. $prec(q_3) = \{\{c_2, c_4\}\}$.
...
$postc(q_0) = \{\{c_2, c_4\}, \{c_2, c_3, c_5\},$
$\{c_2, c_3, c_6, c_7, c_8, c_9\}, \{c_2, c_3, c_6, c_7, c_{10}, c_{11}\},$
$\{c_1, c_5\}, \{c_1, c_6, c_7, c_8, c_9\},$
$\{c_1, c_6, c_7, c_{10}, c_{11}\}\}$.
$postc(q_1) = \{\{c_4\}, \{c_3, c_5\},$
$\{c_3, c_6, c_7, c_8, c_9\}, \{c_3, c_6, c_7, c_{10}, c_{11}\}\}$.
...
$postc(q_4) = \{\{c_8, c_9\}, \{c_{10}, c_{11}\}\}$.
...
Σ = {kl=beginner, kl=average, kl=expert, aw=yes,
aw=no, ..., sl=savings, sl=bonds}.
$\prod = \{c_1, c_2, ..., c_{11}\}$.
E= $\{(q_0, \{c_2\}, q_1), (q_0, \{c_1\}, q_2), (q_1, \{c_4\}, q_3),$
$(q_1, \{c_3\}, q_2), (q_2, \{c_6, c_7\}, q_4), (q_2, \{c_5\}, q_6),$
$(q_4, \{c_8, c_9\}, q_6), (q_4, \{c_{10}, c_{11}\}, q_5)\}$.
S= $\{q_0\}$. F= $\{q_3, q_5, q_6\}$.} □

A word $w \in \Sigma^*$ (i.e., a sequence of user inputs) is accepted by a PFSA if there is an accepting run of w in the PFSA (for details see [5]).

When developing user interfaces, mechanisms have to be provided which support the effective identification of violations of well-formedness properties, e.g., if an input sequence reaches state q_i, there must be at least one extension of this input sequence to a final state. Path expressions form the basis for expressing such well-formedness properties on a PFSA (see Definition 2).

Definition 2 (consistent path): Let $p = [(q_1,C_1,q_2), (q_2,C_2,q_3), ..., (q_{i-1},C_{i-1},q_i)]$ $((q_\alpha,C_\alpha,q_\beta) \in E)$ be a path from a state $q_1 \in S$ to a state $q_i \in Q$. p is consistent (consistent(p)) *iff* $\cup C_j$ is satisfiable. □

Based on the definition of a consistent path we introduce the following set of *well-formedness rules* which specify structural properties of a PFSA (counter examples for these properties are depicted in Figure 5). These rules have shown to be relevant for the implementation of knowledge-based recommender applications. However, due to the generality of the presented diagnosis approach (see Section 3), further rule types can be introduced for domain-specific purposes.

For each consistent path in a PFSA leading to a state q_i there must exist a corresponding *direct* postcondition, i.e., (q_i,C_i,q_j) propagating the path (i.e., each consistent path must be *extensible*) (see Definition 3).

Definition 3 (extensible path): Let $p = [(q_1,C_1,q_2), (q_2,C_2,q_3), ..., (q_{i-1},C_{i-1},q_i)]$ be a path from a state $q_1 \in S$ to a state $q_i \in Q - F$. p is extensible (extensible(p)) *iff* $\exists (q_i,C_i,q_{i+1})$: $C_1 \cup C_2 \cup ... \cup C_{i-1} \cup C_i$ is satisfiable. □

Each state q_i is a decision point for the determination of the next state. This selection strictly depends on the definition of the *direct* postconditions for q_i, where each postcondition has to be unique for determining the subsequent state. A state q_i is *deterministic* if each of its postconditions is unique for determining subsequent states (see Definition 4).

Definition 4 (deterministic state): Let $p = [(q_1,C_1,q_2), (q_2,C_2,q_3), ..., (q_{i-1},C_{i-1},q_i)]$ be a path from a state $q_1 \in S$ to a state $q_i \in Q - F$. A state (q_i) is deterministic(q_i) *iff* $\forall (q_i, C_{i1}, q_j), (q_i, C_{i2}, q_k) \in E : C_1 \cup C_2 \cup ... \cup C_{i-1} \cup C_{i1} \cup C_{i2}$ is contradictory $(C_{i1} \neq C_{i2})$. □

Each transition should be *accessible*, i.e., for each transition there exists at least one corresponding path (see Definition 5).

Definition 5 (accessible transition): A transition $t = (q_i,C_i,q_{i+1})$ (postcondition of state q_i) is accessible (accessible(t)) *iff* there exists a path $p = [(q_1,C_1,q_2), (q_2,C_2,q_3), ..., (q_{i-1},C_{i-1},q_i)]$ $(q_1 \in S)$: $C_1 \cup C_2 \cup ... \cup C_{i-1} \cup C_i$ is satisfiable. □

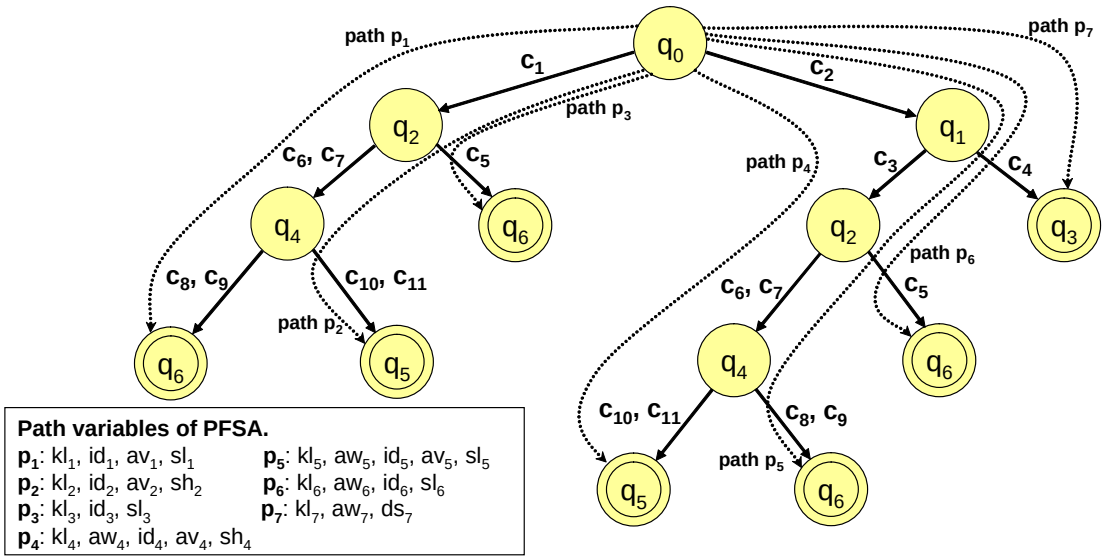

Figure 4: Reachability tree of a PFSA. The tree represents the expansion of the PFSA shown in Figure 2. {p_1, p_2, ..., p_7} are the possible paths of the PFSA. For each path p_i the corresponding path variables are shown.

A PFSA is well-formed, if the given set of well-formedness rules is fulfilled (see Definition 6).

Definition 6 (well-formed PFSA**):** A PFSA is well-formed *iff*

- each consistent path $p = [(q_1,C_1,q_2), (q_2,C_2,q_3), ..., (q_{i-1},C_{i-1},q_i)]$ ($q_1 \in S$, $q_i \in Q - F$) is extensible to a consistent path $p' = [(q_1,C_1,q_2), (q_2,C_2,q_3), ..., (q_{i-1},C_{i-1},q_i), (q_i,C_i,q_j)]$.
- $\forall q_k \in Q$: deterministic(q_k).
- $\forall t = (q_k,C_k,q_l) \in E$: accessible(t). □

3. DEBUGGING RECOMMENDER USER INTERFACES

In a situation where a recommender process is interactively tested by a knowledge engineer or a domain expert, faulty definitions should be identified as soon as possible. Consequently, not well-formed PFSAs require an automated identification of a minimal set of transitions which are responsible for the wrong behaviour. We apply model-based diagnosis (MBD) [18] by introducing the notion of a PFSA *Diagnosis Problem* and a PFSA *Diagnosis*. The MBD approach starts with the description of a system (SD) which is in our case the structural description and the intended behaviour of a PFSA (Definitions 3-5). If the actual behaviour of the system conflicts with its intended behaviour, the diagnosis task is to determine those components (transitions) which, when assumed to functioning abnormally, explain the discrepancy between the actual and the intended system behaviour. In order to apply MBD concepts, we transform a given PFSA definition into a corresponding representation of a constraint satisfaction problem (CSP) [24] consisting of constraints representing well-formedness rules and constraints representing transition conditions of the PFSA. The goal of a diagnosis task is to identify a *minimal* set of transition conditions which are responsible for the faulty behaviour of the PFSA, i.e., are inconsistent with the given set of well-formedness rules. Note that diagnoses need not to be unique, i.e., there can be different explanations for a faulty behaviour. Based on the description of a PFSA=(Q, Σ, ∏, E, S, F), a PFSA Diagnosis Problem can be defined as follows (see Definition 7).

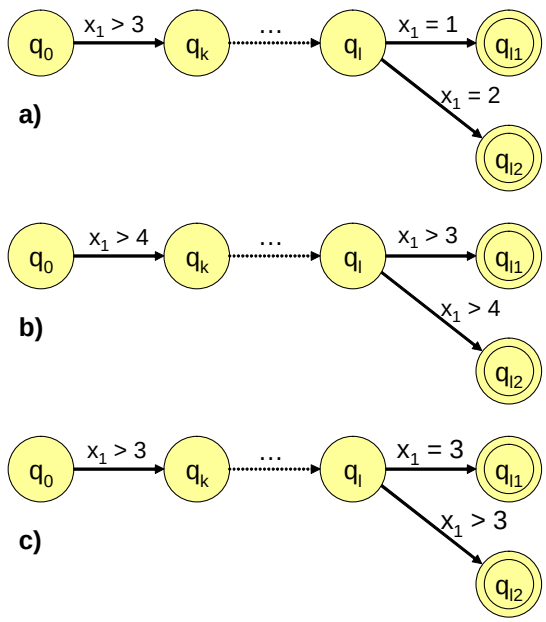

Figure 5: Counter examples for well-formedness rules. a) the path $p = x_1 > 3...$ is *not extensible* in state q_l since the direct postconditions $x_1 = 1$ and $x_1 = 2$ are both inconsistent with $x_1 > 3$. b) the state q_l is *indeterministic* since the direct postconditions $x_1 > 3$ and $x_1 > 4$ are consistent with $x_1 > 4$, i.e., $x_1 > 3 \wedge x_1 > 4 \wedge x_1 > 4$ is consistent. c) the transition $x_1 = 3$ is not accessible for path p, if p is the only path leading to q_l, the transition behind $x_1 = 3$ is *not accessible* (in general).

Definition 7 (PFSA Diagnosis Problem): A PFSA Diagnosis Problem is represented by a tuple (SD, TRANS), where SD = STAT ∪ WF. STAT is the structural description of a PFSA represented by a set of finite domain variables. WF is the intended behaviour of a PFSA which is represented by a set of constraints on STAT. Finally, TRANS represents a set of transition conditions. □

Note that (STAT, WF, TRANS) defines a Constraint Satisfaction Problem (CSP) [24]. The set of solutions to such a CSP represents all possible interaction paths (runs accepted by the PFSA). STAT is a set of finite domain variables related to paths of the reachability tree (see Figure 4), e.g., $\{kl_1, id_1, av_1, sl_1\}$ are variables related to the path p_1. The complete set of variables related to paths of the reachability tree is shown in Figure 4. Note that the projection of all solutions for the CSP defined by (STAT, WF, TRANS) to, e.g., the variables $\{kl_1, id_1, av_1, sl_1\}$ represents those input sequences accepted by path p_1. For our example PFSA, STAT is defined as follows.

Example 2 (STAT**):**
STAT = $\{kl_1,id_1,av_1,sl_1,kl_2,id_2,av_2,sh_2,kl_3,id_3,sl_3,kl_4,aw_4,id_4,av_4, sh_4,kl_5,aw_5,id_5,av_5,sl_5,kl_6,aw_6,id_6,sl_6,kl_7,aw_7,ds_7\}$ □

We exemplify the construction of well-formedness rules (Definitions 3-5) WF = $WF_{extensibility} \cup WF_{accessibility} \cup WF_{determinism}$ by showing the construction of an accessibility rule related to the transition (q_2, $\{c_6, c_7\}$, q_4) of our example PFSA (see Figure 2).

Example 3 (well-formedness rules for accessibility):
$WF_{accessibility}$ (q_2, $\{c_6, c_7\}$, q_4) = $\{id_1 \neq noval \vee id_2 \neq noval \vee id_4 \neq noval \vee id_5 \neq noval.\}$. □

This rule denotes the fact that the transition (q_2, $\{c_6, c_7\}$, q_4) must be accessible for at least one of the paths p_1, p_2, p_4, p_5, i.e., at least one of the variables id_1, id_2, id_4, id_5 must have a value ≠ noval in a solution for the CSP defined by (STAT, WF, TRANS). Since in our case a reachability tree (see, e.g., Figure 4) represents the complete expansion of a corresponding PFSA, not all the paths are necessarily consistent. If a path of the reachability tree represents such an illegal trajectory, i.e., no consistent value assignment exists for the corresponding variables, all variables of this path have the assignment noval. This is assured by meta-constraints defined for each path in the reachability tree, e.g., for path p_1: $kl_1 \neq noval \wedge id_1 \neq noval \wedge av_1 \neq noval \wedge sl_1 \neq noval \vee kl_1 = noval \wedge id_1 = noval \wedge av_1 = noval \wedge sl_1 = noval$. An example for the definition of a transition condition (TRANS) is the following (see Example 4). We represent the transition condition c_1 of our example PFSA.

Example 4 (TRANS **for** PFSA**):** TRANS={c_1: ($kl_1 = beginner \vee kl_1 = noval$) ∧ ($kl_2 = beginner \vee kl_2 = noval$) ∧ ($kl_3 = beginner \vee kl_3 = noval$). ...}. □

This generated condition for c_1 contains those variables which belong to paths including the transition (q_0, $\{c_1\}$, q_2), i.e., the variables $\{kl_1, kl_2, kl_3\}$ which belong to the paths $\{p_1, p_2, p_3\}$. For the CSP defined by (STAT, WF, TRANS) the possible values of the variables $\{kl_1, kl_2, kl_3\}$ are defined by condition c_1, i.e., the value of $\{kl_1, kl_2, kl_3\}$ can be either beginner or noval (in the case that the corresponding path is an illegal trajectory). Given a specification of (SD, TRANS), a PFSA Diagnosis can be defined as follows.

Definition 8 (PFSA **Diagnosis):** A PFSA Diagnosis for a PFSA Diagnosis Problem (SD, TRANS) is a set S ⊆ TRANS s.t. SD ∪ TRANS − S consistent. □

A diagnosis exists under the reasonable assumption that STATS ∪ WF is consistent [5]. The calculation of diagnoses for a given PFSA definition is based on the concept of minimal conflict sets.

Definition 9 (Conflict Set): a Conflict Set (CS) for (SD, TRANS) is a set $\{c_1, c_2, ..., c_n\} \subseteq$ TRANS, s.t. $\{c_1, c_2, ..., c_n\} \cup$ SD is inconsistent. CS is minimal *iff* $\neg\exists$ CS‘ ⊂ CS : conflict set (CS’). □

The algorithm for calculating a set of minimal diagnoses for a process definition is the following (Algorithm 1). The labelling of the search tree (Hitting Set Directed Acyclic Graph - HSDAG) is based on the labelling of the original HSDAG [18]. A node n is labelled by a corresponding conflict set CS(n). The set of edge labels from the root to node n is referred to as H(n).

Algorithm 1: PFSA-Diagnosis (SD, TRANS):
(a) Generate a pruned HSDAG T for the collection of conflict sets induced by transitions of TRANS in breadth-first manner (we generate diagnoses in order of their cardinality). With every theorem prover (TP) call at a node n of T the consistency of (TRANS - H(n) ∪ SD) is checked. If there exists an inconsistency, a conflict set CS is returned, otherwise ok is returned. If (TRANS - H(n) ∪ SD) is consistent, a corresponding diagnosis H(n) is found.

(b) Return {H(n) | n is a node of T labeled with ok}. □

Conflict sets determined by TP calls are $\{c_2, c_{11}\}$, $\{c_7, c_9\}$, $\{c_2, c_9\}$, and $\{c_1, c_9\}$. One minimal diagnosis S for our example PFSA is $\{c_2, c_9\}$. Figure 1 depicts a screenshot of the Koba4MS environment (transitions c_2 and c_9 are indicated as faulty, i.e., are part of a diagnosis). Currently, the debugging process is triggered each time, the process design is stored by the user. In the case of faulty transition conditions, the debugging component highlights the corresponding conditions in the graphical representation of the recommender process.

4. EVALUATION

Our approach to introduce Model-Based Diagnosis (MBD) [18] into the development of knowledge-based recommender applications doesn’t fundamentally change the structure of the interface for designing recommender processes, but complements the existing interface with a set of intelligent mechanisms allowing the automated identification of faulty transition conditions in process definitions. The presented approach is not restricted to the application in knowledge-based recommendation, but is generally applicable in situations, where a user interface is explicitly defined in terms of a finite state model. The presented debugging support for process definitions of recommender applications has been implemented within the scope of the Koba4MS project. The diagnosis component is a Java-based implementation of the diagnosis algorithm presented in [11,18]. The calculation of minimal conflict sets is based on the algorithm presented in [14]. In general the diagnosis

component is applicable to interactive settings, where a knowledge engineer or a domain expert is developing and maintaining a recommender application. A performance evaluation [5] shows the applicability of the presented algorithm in real-world settings where a knowledge engineer is interactively developing a recommender application. Typically, recommender process definitions consist of about 15-40 states – a size which makes the application of the presented debugging concepts meaningful. Experiences from applying the presented process debugging concepts show that the approach is applicable and supports knowledge engineers and domain experts when designing recommender process definitions. Knowledge acquisition and maintenance as a collaborative process conducted by technical and domain experts is still a very time-consuming task. In this context, automated support of debugging is an important precondition for the effective deployment of recommender systems in industrial environments. The time of a domain expert which can be dedicated to the development of a recommender application is strictly limited, i.e., time savings related to the identification of faults play an important role in the recommender development process. The Koba4MS debugging functionality is not applied to simple process definitions with, e.g., the complexity of our working example. However, in situations where the number of states is about 15 or more, additional debugging mechanisms are useful. The presented debugging concepts are a first approach to make development and maintenance of recommender process definitions more effective. However, due to the feedback from knowledge engineers and domain experts, further concepts will be integrated into future versions of our software. Domain experts (as well as knowledge engineers) often tend to think in terms of examples. We intend to integrate example-driven testing mechanisms where the developer himself provides an example set of paths which should be accessible. On the technical level such examples represent additional well-formedness rules for a given process definition. In many cases there exist a number of alternative diagnoses explaining the sources of inconsistencies in a given process definition. In this context we will include additional ranking mechanisms for diagnoses which take into account the probability of a transition condition to be faulty (e.g., the probability of a transition condition to be faulty is higher if there exists a large number of incoming paths to the related state, similarly, the probability is higher, if the complexity of the transition condition in terms of number of referenced variables etc. is high). Currently, the selection of a diagnosis strictly depends on its cardinality, i.e., diagnoses with the lowest number of transition conditions are presented. We have compared development efforts related to projects without debugging support for process definitions and with a corresponding debugging support. Our development environment has a logging functionality which stores each activity in the recommender design environment (e.g., change/edit a transition condition, view a product/customer property, change/edit of product data, or add a new transition condition). Six projects were investigated which have a similar structure w.r.t. number of product properties, customer properties, constraints, number of states in the PFSA etc. The observation of time efforts shows that the usage of the debugging concepts can significantly reduce efforts related to the development and maintenance of recommender user interfaces (reduced efforts in percent of the overall development efforts). In order to empirically show the direct positive influence of debugging techniques on a broader basis, we have conducted an experiment with ninety-seven participating students all having experiences in developing knowledge-based recommender applications. In this experiment participants were interacting with a web-based application providing the basic functionality of our test environment. The participants had to solve the task of identifying a *minimal* set of constraints in given recommender process definitions which should be changed in order to make the process definition consistent with the well-formedness properties presented in this paper. The participants had to propose a set of repair actions which make the process definition consistent with the well-formedness rules. The hypothesis of the experiment was that automated debugging support leads to effort reductions in the detection and repair of faulty transition conditions. The major result of the experiment was a significant decrease of time efforts due to the application of the presented debugging concepts (on an average of 36%). A more detailed analysis of the consequences of applying automated debugging concepts in recommender application development is currently ongoing (e.g. we investigate dependencies between the complexity of transition conditions and the corresponding effort reductions caused by the application of debugging techniques). This work is within the line of our research focus to improve the effectiveness of recommender development and maintenance processes.

5. RELATED WORK

Basically there are three approaches to the development of a recommender application. *Collaborative Filtering* [12,21] and *Content-based Filtering* [4,17] do not exploit deep knowledge about the product domain. Collaborative Filtering is based on the assumption that customer preferences are correlated, i.e., similar products are recommended to customers with similar interest profiles. Content-based filtering focuses on the analysis of a given set of products already ordered by a customer. Based on this information, products are recommended which resemble products already ordered (products related to similar categories). Additionally, there exist a number of approaches combining these basic approaches in order to gain an improved quality of the resulting solutions (see, e.g., [22]). Using *knowledge-based* approaches, the relationship between customer requirements and offered products is explicitly modelled [6]. Such model-based knowledge representations are the major precondition for the application of model-based diagnosis and testing techniques. The complexity of configuration knowledge bases motivated the application of model-based diagnosis (MBD) [18] in knowledge-based systems development [7]. Similar motivations led to the application of model-based diagnosis in technical domains such as the development of hardware designs [10], onboard diagnosis for automotive systems [20] and in software development [15]. Experiences related to the implementation of the concepts presented in [7] are discussed in [9]. A value-based model of program execution is introduced in a.o. [15], which is an extension of the approach of debugging imperative languages [10]. Using MBD techniques, the location of errors is based on the analysis of a given source code. Additionally, a set of test cases specifying the expected behaviour of the program is required - this set encompasses concrete values for variables, assertions, reachability constraints for paths, valid call sequences etc. An overview on the application of model-based diagnosis techniques in software debugging can be found in [23]. The representation of recommender processes in the form of finite state representations is discussed in [5,8]. This

approach is novel in the context of developing knowledge-based recommender applications and due to its formal basis it allows a direct and automated translation of the graphical model into a corresponding recommender application.

6. CONCLUSIONS

We have presented a set of concepts supporting the automated debugging of process definitions representing possible interaction sequences of knowledge-based recommender applications. Predicate-based Finite State Automata are applied to represent models of possible interaction paths. For debugging purposes the automaton is translated into a representation of a constraint satisfaction problem. The proposed approach has been evaluated using typical process definitions existing in commercial recommender projects.

7. ACKNOWLEDGEMENTS

The work presented in this paper has been done in the context of the projects Koba4MS (FFG-808479) and WSDIAMOND (IST-2005-516933).

8. REFERENCES

[1] Ardissono, L., Felfernig, A., Friedrich, G., Jannach, D., Petrone, G., Schaefer, R., and Zanker, M. A Framework for the development of personalized, distributed web-based configuration systems. *AI Magazine* 24, 3, 93–108, 2003.

[2] Bass, L., and Coutaz, J. Developing Software for the User Interface. *The SEI Series in Software Engineering*, Addison Wesley, Massachusetts, USA, 1991.

[3] Burke, R. Knowledge-based Recommender Systems. *Encyclopedia of Library & Information Systems*, 69, 32, 2000.

[4] Burke, R. Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, 331–370, 2002.

[5] Felfernig, A., and Shchekotykhin, K. Debugging User Interface Descriptions of Knowledge-based Recommenders. *Workshop Notes of the IJCAI'05 Workshop on Configuration*, 13–18, 2005.

[6] Felfernig, A., Friedrich, G., Jannach, D., Stumptner, M., and Zanker, M. Configuration knowledge representations for Semantic Web applications. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 17, 31–50, 2003.

[7] Felfernig, A., Friedrich, G., Jannach, D., and Stumptner, M. Consistency-based Diagnosis of Configuration Knowledge Bases. *Artificial Intelligence* 2, 152, 213–234, 2004.

[8] Felfernig, A. Koba4MS: Selling Complex Products and Services Using Knowledge-Based Recommender Technologies. *7th IEEE International Conference on E-Commerce Technology*, 92–100, 2005.

[9] Fleischanderl, G. Suggestions from the software engineering practice for applying consistency-based diagnosis to configuration knowledge bases. *13th International Workshop on Principles of Diagnosis* (DX-02), 2002.

[10] Friedrich, G., Stumptner, M., and Wotawa, F. Model-based diagnosis of hardware designs. *Artificial Intelligence* 111, 2, 3–39, 1999.

[11] Greiner, R., Smith, B., and Wilkerson, R. A correction to the algorithm in Reiter's theory of diagnosis. *Artificial Intelligence* 41, 1, 79–88, 1989.

[12] Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems* 22, 1, 5–53, 2004.

[13] Hopcroft, J., and Ullman, J. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley Publishing Company, Massachusetts, USA, 1979.

[14] Junker, U. QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. *19th National Conference on AI* (AAAI04), 167–172, 2004.

[15] Mateis C., Stumptner, M., and Wotawa, F. Modeling Java programs for diagnosis. *14th European Conference on Artificial Intelligence*, 171–175, 2000.

[16] Mittal, S., and Frayman, F. Towards a Generic Model of Configuration Tasks. *11th International Joint Conf. on Artificial Intelligence*, 1395–1401, 1989.

[17] Pazzani, M. A Framework for Collaborative, Content- Based and Demographic Filtering. *Artificial Intelligence Review* 13, 5-6, 393–408, 1999.

[18] Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 23, 1, 57–95, 1987.

[19] Resnik, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. *ACM 1994 Conference on Computer Supported Cooperative Work*, 175–186, 1994.

[20] Sachenbacher, M., Struss, P., and Carlen, C. A Prototype for Model-Based On-Board Diagnosis of Automotive Systems. *AI Communications* 13, 2, 83–97, 2000.

[21] Sarwar, B., Karypis, G., Konstan, J. A., and Riedl, J. T. Item-based collaborative filtering recommendation algorithms. *10th Int. World Wide Web Conf.*, 285–295, 2001.

[22] Shih, Y.-Y., and Liu, D.-R. Hybrid recommendation approaches: collaborative filtering via valuable content information. *38th Hawaii International Conference on System Sciences* (HICSS'05), 217b, 2005.

[23] Stumptner, M., and Wotawa, F. A Survey of Intelligent Debugging. *AI Communications* 11, 1, 35–51, 1998.

[24] Tsang, E. Foundations of Constraint Satisfaction. Academic Press, London, 1993.

[25] V.Noord, G., and Gerdemann, D. Finite State Transducers with Predicates and Identities. *Grammars* 4, 3, 263–286, 2001.

[26] Wasserman, A. Extending state transition diagrams for the specification of human-computer interaction. *IEEE Transactions on Software Engineering* 11, 699–713, 1985.

[27] Xiao, B., Aimeur, E., and Fernandez, J. PCFinder: An Intelligent Product Recommendation Agent for ECommerce. *IEEE International Conference on E-Commerce* (CEC'03), 181–188, 2003.