Automated Debugging of Recommender User Interface Descriptions

Alexander Felfernig[1,2], Erich Teppan[1], Kostyantyn Shchekotykhin[1]

[1]University Klagenfurt, Computer Science and Manufacturing

Universitätsstrasse 65-67, A-9020 Klagenfurt, Austria

[2]ConfigWorks, Lakeside, B01, A-9020 Klagenfurt, Austria

contact: alexander.felfernig@uni-klu.ac.at

Abstract

Complex assortments of products and services offered by online selling platforms require the provision of sales support systems assisting customers in the product selection process. Knowledge-based recommenders are intelligent sales assistance systems which guide online customers through personalized sales dialogs and automatically determine products which fit their needs and wishes. Such systems have been successfully applied in application domains such as financial services or digital cameras. In this context, the construction of recommender user interfaces is still a challenging task. In many cases faulty models of recommender user interfaces are defined by knowledge engineers and no automated support for debugging such models is available. In this paper we present a formal model for defining the intended behaviour of recommender user interfaces and show the application of model-based diagnosis concepts which allow the automated debugging of those definitions. An empirical evaluation shows significant time savings in recommender user interface development and maintenance processes.

1 Introduction

The selection of products from a complex assortment is still a challenging task since many online selling environments offer simple query interfaces based on the assumption that users know technical product details. In this context, recommendation technologies [1, 2, 3, 9, 13, 19, 20, 30, 26] are of great importance for making product assortments more accessible. Basically, there are three technological approaches to the implementation of a recommender application:

- First, *content-based filtering* [20] derives recommendations by exploiting similarities between the preferences of the current customer and existing product descriptions. In this case, the recommender proposes products which are similar to those the customer has liked in the past. If a customer has bought books related to the *SAP system*, similar books will be recommended in future advisory sessions, i.e., no serendipity effects can be exploited in this context.

- Second, *collaborative filtering* [13, 22, 26] is based on preferences of a large set of customers. Recommendations are derived by taking into account preferences of customers with similar purchasing patterns, for instance, movies not yet bought by the current customer but positively rated by customers with similar purchasing behavior will be recommended to the current customer.

- Third, *knowledge-based recommendation* [1, 3, 15, 23, 27, 30] exploits deep knowledge about the product domain. Compared to domains such as books or movies, customers purchasing complex products such as computers or financial services are much more in the need of intelligent interaction mechanisms supporting the calculation of appropriate solutions. Therefore, we need an explicit representation of the existing product, marketing, and sales knowledge [9] which makes it possible to (a) derive recommendations which comply with existing marketing and sales strategies and suit the wishes of a customer, (b) to explain those recommendations (why has a specific product been recommended to a customer?), and (c) support customers in situations where the recommender is unable to find a solution for the given requirements (repair of requirements).

When developing knowledge-based recommenders, two basic aspects have to be taken into account. First, a *recommender knowledge base* [3, 6] has to be defined which consists of a structural description of the provided products, a description of the possible customer requirements and constraints restricting the allowed combinations of customer requirements and related product recommendations. Second, a *process model* has to be defined which describes the intended behavior of the recommender user interface [3, 10], i.e., which questions have to be posed to a specific customer in which contexts? Thereafter, both, knowledge bases and process definitions can be automatically translated into a recommender application [3, 9, 5].

In the remainder of this paper we focus on a situation where knowledge engineers develop a model of the intended behavior of a recommender user interface (process model development). Knowledge acquisition as a collaborative process conducted by technical and domain experts is still a very time-consuming task. In this context, automated debugging support is an important contribution to the effective deployment of recommender applications [4]. 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 in user interface descriptions play an important role in recommender development processes.

The intended behavior of a recommender user interface can be described by a finite state model [10, 14, 32]. Each state of such a model represents an input unit of the application where a user can articulate his/her preferences by answering questions posed by the recommender. Figure 1 depicts a simple example for the model of the intended behavior of a financial services recommender application [9]. Figure 2 depicts the corresponding user interface of our process modeling environment (Process Designer). This environment is part of a commercially available recommender development environment [3, 8, 9]. Basically, this interface allows the specification of a finite state model where the states represent input units allowing customers to articulate their requirements, e.g., in state q_2 the customer is asked about the preferred duration of investment.[1] Having defined such a model, our development environment

[1]Note that *duration_of _investment* (*id*) is the identifier for the corresponding question posed by the recommender application (*what is your required duration of investment*?).

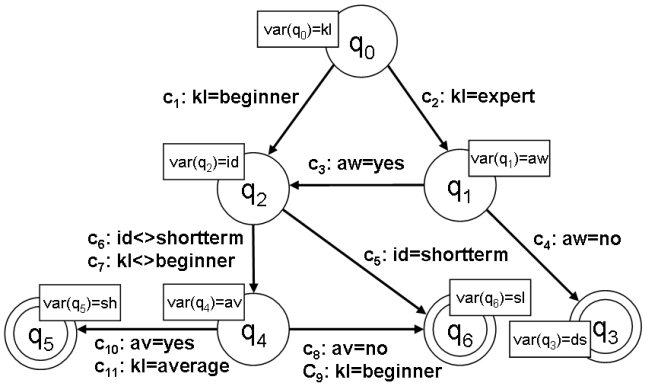

Figure 1: Example recommender user interface description.

automatically generates a corresponding application [3, 9].

Depending on the preferences articulated by the customer, the automaton of Figure 1 changes its state, e.g., an expert (kl = expert) who is not interested in financial services advisory (aw = no), is forwarded to the state q_3, where a direct product search can be performed. Consequently, different navigation paths determine different subsets of input variables relevant for the preference elicitation process. After the completion of the preference elicitation process (a final state of the process definition has been reached), the recommender application can calculate and present a corresponding set of solutions [3, 9].

Note that the specification of our financial services recommender interface in Figure 1 is faulty. A financial services expert (kl = expert) who wants to be advised by a financial services recommender (aw = yes) and is interested in long-term investments (id $\neq$ shortterm) and doesn't have any available funds (av = no) comes to a standstill at the input of availability (the transition condition sets $\{c_2, c_9\}$ and $\{c_2, c_{11}\}$ are contradictory). In such situations, the developer of a knowledge base needs additional debugging support in order to effectively identify the sources of the inconsistency.

In this paper we demonstrate the application of Model-Based Diagnosis (MBD) [21] with the goal to be able to automatically identify minimal sets of faulty transition conditions in recommender user interface descriptions. For this purpose we derive a

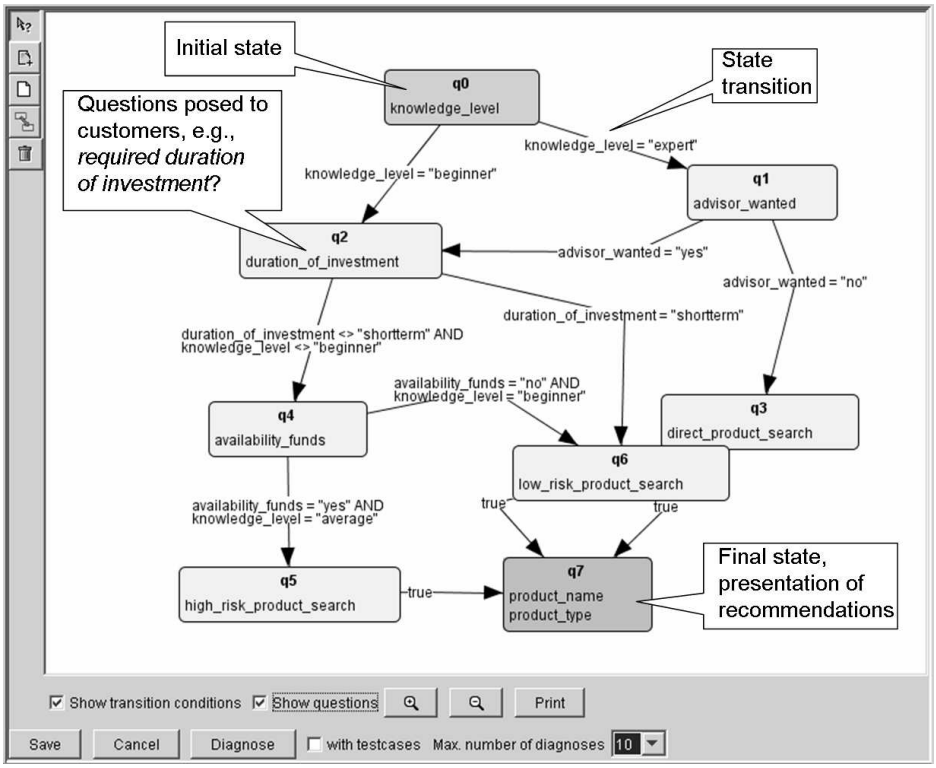

Figure 2: Modeling environment for user interface descriptions.

logical representation (system description) of a given finite state representation of a recommender user interface (see Figure 1) which serves as input for the calculation of hitting sets (diagnoses) introduced in [21].

The remainder of this paper is organized as follows. In Section 2 we introduce a finite state representation formalism for modeling the navigational behavior of recommender user interfaces. Using the concepts of Model-Based Diagnosis (MBD) [21], we present an approach to the automated identification of minimal sets of faulty transition conditions in recommender user interfaces (see Section 3). In Section 4 we evaluate the performance of the presented debugging algorithm and present results of an empirical study. Finally, Section 5 contains a discussion on related work.

2 Finite State Models of Recommender User Interfaces

For the definition of the intended behavior of a recommender user interface, we introduce the concept of Predicate-based Finite State Automata (PFSA) [10, 32] (see Figure 1) which are a specific variant of finite state automata [14]. This type of automaton is more compact in the way state transitions can be defined (domain restrictions of finite domain variables), which makes it an excellent formalism for the graphical design and maintenance of recommender user interfaces. This representation of recommender user interfaces is integrated into the recommender development environment presented in [3, 9]. Note that in the context of building knowledge-based recommender applications, we are primarily interested in *acyclic* automata.

Definition 1 (PFSA): a Predicate-based Finite State Automaton (recognizer) (PFSA) is defined as a 6-tuple (Q, Σ, Π, 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 (ϕ_α = $\{c_r, c_s, ..., c_t\} \subseteq \Pi$), postc($q_i$) = $\{\psi_1, \psi_2, ..., \psi_n\}$ is the set of postconditions of q_i (ψ_β= $\{c_u, c_v, ..., c_w\} \subseteq \Pi$), 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.
- Σ = $\{x_i = d_{ij} \mid x_i \in$ var(q_i), $x_i = d_{ij} \in$ dom(x_i)$\}$ is a finite set of variable assignments, the input alphabet.
- Π = $\{c_1, c_2, ..., c_q\}$ 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 $\times$ Π $\times$ Q.
- S $\subseteq$ Q is a finite set of start states.
- F $\subseteq$ Q is a finite set of final states. $\square$

Preconditions of a state q_i (prec(q_i) = $\{\phi_1, \phi_2, ..., \phi_m\}$) can be automatically derived from the reachability tree of a PFSA. Figure 3 depicts the reachability tree for the PFSA of Figure 1. The state q_2 is accessed twice in the reachability tree, consequently, we

can derive two preconditions for the state q_2 which directly correspond to the transition conditions of paths in the reachability tree leading to q_2, i.e., prec(q_2) = {{c_1}, {c_2, c_3}} where the different subsets are interpreted as being part of a disjunction (not every precondition has to be fulfilled). 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}}}. Figure 4 depicts the textual representation of the PFSA of Figure 1.

The set of input sequences leading to a final state is also denoted as the language accepted by the PFSA. 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 (see [10]).

When developing user interfaces, mechanisms have to be provided which support the effective identification of violations of *well-formedness properties*, e.g., if a path in the process definition reaches a state q_i, there must be at least one extension of this path to a final state. Regarding our example of Figure 1, there exist accepted input sequences visiting the states [q_0, q_1, q_2, q_4], but none of those sequences can be propagated to any of the following states {q_5, q_6}. Path expressions form the basis for expressing well-formedness properties on a PFSA (see Definition 2a,b).

Definition 2a (path): we define a sequence (of transitions) p = [(q_1,C_1,q_2), (q_2,C_2,q_3), ..., (q_{i-1},C_{i-1},q_i)] ((q_α,C_α,q_β) $\in$ E) as *path* of a given PFSA. □

Definition 2b (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_α,C_α,q_β) $\in$ E) be a path from a state $q_1 \in$ S to a state $q_i \in$ Q. p is *consistent* (consistent(p)) iff $\bigcup C_\alpha$ is consistent. □

Following this definition of a consistent path we introduce a set of well-formedness rules which specify important 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. Note that if additional well-formedness rules are needed, our framework allows the introduction of further domain-specific properties.

Extensibility. 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_{i+1}) propagating the path (i.e.,

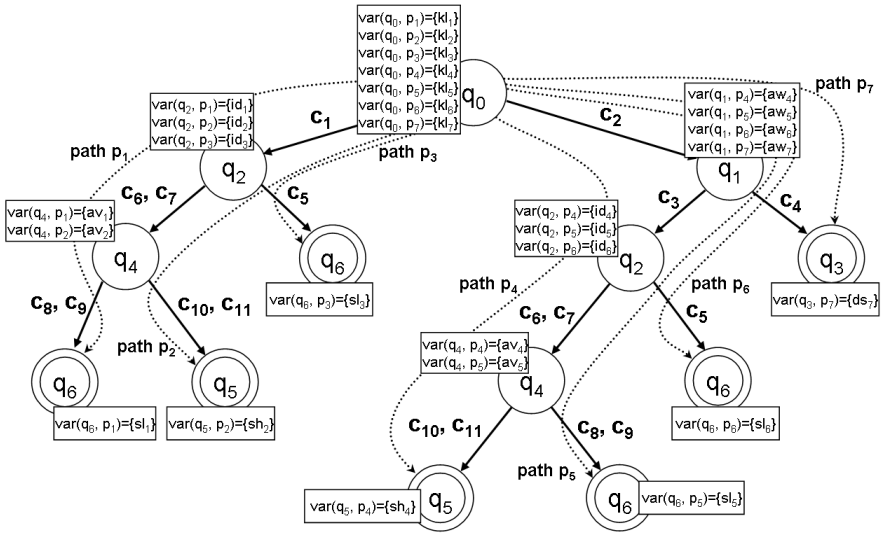

Figure 3: Reachability tree of a PFSA.

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 consistent 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 consistent. □

Figure 5 (a) depicts a *non-extensible path*, since the conditions {c_0, c_1} of p=[(q_0,{c_0:x_0>3},q_1), (q_1,{c_1:x_1>=3},q_2)] are inconsistent with both conditions of postc(q_2)={{c_2},{c_3}}. Similarly, Figure 1 includes a non-extensible path: [(q_0,{c_2:kl=expert},q_1), (q_1,{c_3:aw=yes},q_2), (q_2,{c_6:id $\neq$ shortterm, c_7:kl $\neq$ beginner},q_4)] is inconsistent with the conditions of postc(q_4)={{c_{10}, c_{11}},{c_8, c_9}}.

Determinism. 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

```
Q = {q0, q1, q2, q3, q4, q5, q6}.

/* knowledge level */
var(q0) = {kl}.
/* advisory wanted */
var(q1) = {aw}.
/* duration of investment */
var(q2) = {id}.
/* direct product search */
var(q3) = {ds}.
/* availability of
 financial resources*/
var(q4) = {av}.
/* high risk products */
var(q5) = {sh}.
/* low risk products */
var(q6) = {sl}.

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(q0) = {{true}}.
prec(q1) = {{c2}}.
prec(q2) = {{c1}, {c2, c3}}.
prec(q3) = {{c2, c4}}.
```

```
postc(q0) = {{c2, c4}, {c2, c3, c5},
          {c2, c3, c6, c7, c8, c9},
          {c2, c3, c6, c7, c10, c11},
          {c1, c5},
          {c1, c6, c7, c8, c9},
          {c1, c6, c7, c10, c11}}.
postc(q1) = {{c4}, {c3, c5},
          {c3, c6, c7, c8, c9},
          {c3, c6, c7, c10, c11}}.

/* ...  */

postc(q4) = {{c8, c9}, {c10, c11}.

postc(q3) = {{true}}.
postc(q5) = {{true}}.
postc(q6) = {{true}}.

Σ = {kl=beginner, kl=average,
    kl=expert,
    aw=yes, aw=no, ...,
    sl=savings, sl=bonds}.

Π = {c1, c2, ..., c11}.
E= {(q0, {c2}, q1), (q0, {c1}, q2),
   (q1, {c4}, q3),
   (q1, {c3}, q2), (q2, {c6, c7}, q4),
   (q2, {c5}, q6),
   (q4,{c8, c9}, q6),
   (q4, {c10, c11}, q5)}.
S= {q0}.
F= {q3, q5, q6}.
```

Figure 4: Textual version of PFSA in Figure 1.

deterministic iff $\forall(q_i, C_{i1}, q_j), (q_i, C_{i2}, q_k) \in \mathrm{E} : C_1 \cup C_2 \cup ... \cup C_{i-1} \cup C_{i1} \cup C_{i2}$ is contradictory ($C_{i1} \neq C_{i2}$). □

Well-formedness rules related to deterministic states require that each input sequence consistent with the transition conditions $\{C_1, C_2, ..., C_{i-1}\}$ is consistent with at most one of the following conditions $\{C_{i1}, C_{i2}\}$. Figure 5 (b) depicts an *indeterministic state*, since the conditions $\{c_0, c_1\}$ are consistent with $\{c_2, c_3\}$. By introducing negated neighbour conditions (C_{i1} is the neighbour of C_{i2}), we can achieve the inconsistency required by Definition 4. Exchanging neighbour conditions is basically realized by a pairwise exchange of the negations of existing transition conditions.[2] This approach is exemplified in Figure 6 where in case (a) the original semantics is preserved and in case (b) an inconsistency ($c_2 : x_0 > 4 \wedge c_1 : \neg x_0 > 3$) is triggered which has to be resolved by the diagnosis process (see Section 3).

[2]Note that this approach to a pairwise exchange of negated conditions is as well applicable in situations with more that two post-conditions of a given state.

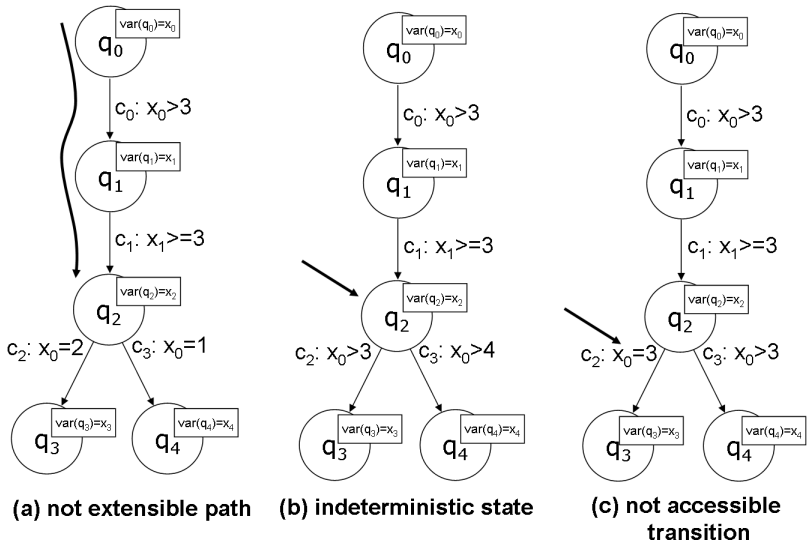

Figure 5: Counter examples for well-formedness rules.

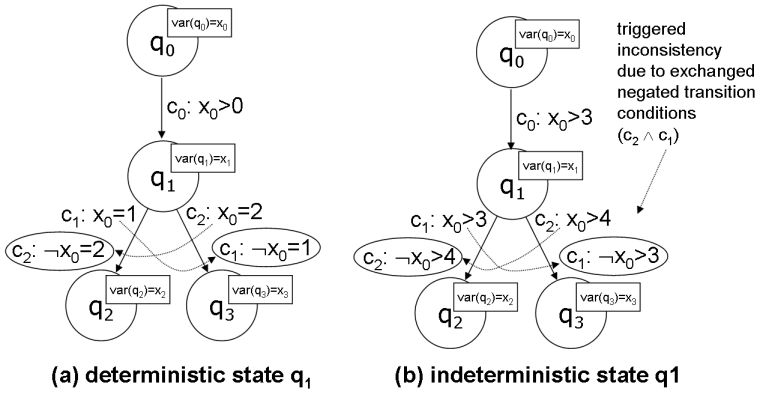

Figure 6: Handling of indeterministic transition conditions: state q_1 is indeterministic (b); on the basis of a pairwise exchange of negated neighbour conditions, we can trigger an inconsistency which has to be handled by the diagnosis process.

Accessibility. 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 consistent. □

Figure 5 (c) depicts a *not accessible transition*, since c_2 (transition (q_2,{c_2},q_3)) is inconsistent with {c_0, c_1}, the conditions of the only path leading to q_2. Similarly, Figure 1 contains a non-accessible transition: none of the possible paths are consistent with the transition conditions of (q_4,{c_{10}:av=yes, c_{11}:kl=average},q_5).

Well-formed PFSA. A PFSA is well-formed, if the defined 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). □

Having specified necessary properties of a PFSA, we now present our approach to the calculation of minimal sets of faulty transition conditions in a PFSA.

3 Debugging Finite State Models

Given a faulty (not well-formed) PFSA, we want to automatically identify a minimal set of faulty transition conditions. In order to solve this task, we define a PFSA diagnosis problem which is solved using the concepts provided by model-based diagnosis (MBD) [21]. Model-based diagnosis starts with the description of a system (SD) which is in our case the structural description of the intended behavior of a PFSA (Definitions 3-5). If the actual behavior of the system conflicts with its intended behavior, 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 behavior.

In order to apply MBD concepts, we transform a PFSA into a corresponding constraint-based representation which is composed of the following sets:

1. STAT: finite domain variables representing possible user inputs.
2. WF: constraints representing well-formedness rules.
3. TRANS: constraints representing the transition conditions of the PFSA.

The goal of a diagnosis task is to identify a minimal set of transition conditions ($\subseteq$TRANS) which are responsible for the faulty behavior of the PFSA, i.e., are inconsistent with the given set of well-formedness rules. Note that diagnoses do not need to be unique, i.e., there can be different explanations for faulty transition conditions in the PFSA. We define a PFSA Diagnosis Problem as follows.

Definition 7 (PFSA Diagnosis Problem): A PFSA Diagnosis Problem is represented by a tuple (SD, TRANS), where SD = STAT $\cup$ WF. STAT is the structural description of a PFSA represented by a set of finite domain variables. WF is the intended behavior of a PFSA (set of well-formedness rules) which is represented by a set of constraints on STAT. Finally, TRANS represents a set of transition conditions (as well represented by constraints on STAT). $\square$

Note that (STAT, WF, TRANS) defines a Constraint Satisfaction Problem (CSP) [31]. STAT is a set of finite domain variables related to paths of the reachability tree, e.g., {kl_3, id_3, sl_3} are variables related to the path p_3 (Figure 3 depicts the relationship between paths and variables $\in$ STAT, e.g., var(q_0, p_3) = $\{kl_3\}$). The complete set of variables related to paths of the reachability tree is included in Example 1. Note that each of the variables $\in$ STAT is either active ($\mathcal{ACT}$) or inactive ($\mathcal{IACT}$). This notion of variable activity has been introduced by [25]. Although we apply a simplified version of this approach (every variable is either active or inactive without any additional activation constraints), this representation perfectly supports our goal of testing well-formedness properties of process definitions. The set of solutions to the CSP defined by (STAT, WF, TRANS) represents all possible interaction sequences

(accepted runs). The projection of those solutions to, e.g., the variables {kl_3, id_3, sl_3} represents those input sequences accepted by path p_3, i.e., [kl=beginner, id=shortterm, sl=savings], [kl=beginner, id=shortterm, sl=bonds]. For our example PFSA, STAT is defined as follows.[3]

Example 1 (STAT): STAT = {

kl_1, id_1, av_1, sl_1, /* path p_1 */

kl_2, id_2, av_2, sh_2, /* path p_2 */

kl_3, id_3, sl_3, /* path p_3 */

kl_4, aw_4, id_4, av_4, sh_4, /* path p_4 */

kl_5, aw_5, id_5, av_5, sl_5, /* path p_5 */

kl_6, aw_6, id_6, sl_6, /* path p_6 */

kl_7, aw_7, ds_7 /* path p_7 */}. □

Since in our case a reachability tree (see, e.g., Figure 3) 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 to be inactive. In order to assure that all variables of a path are either active or inactive, we introduce meta-constraints defined for each path in the reachability tree, e.g., for path p_3: $\mathcal{ACT}(kl_3) \wedge \mathcal{ACT}(id_3) \wedge \mathcal{ACT}(sl_3) \vee \mathcal{IACT}(kl_3) \wedge \mathcal{IACT}(id_3) \wedge \mathcal{IACT}(sl_3)$. This constraint denotes the fact that either all variables of path p_3 must be active or all variables are inactive. In order to simplify the construction of well-formedness rules, we introduce additional meta-constraints which define the *activity state* of a path variable, e.g., for path p_3: $\mathcal{ACT}(kl_3) \wedge \mathcal{ACT}(id_3) \wedge \mathcal{ACT}(sl_3) \leftrightarrow \mathcal{ACT}(p_3)$, and $\mathcal{IACT}(kl_3) \wedge \mathcal{IACT}(id_3) \wedge \mathcal{IACT}(sl_3) \leftrightarrow \mathcal{IACT}(p_3)$. In order to introduce such meta-constraints, we have to introduce $\{p_1, p_2, ..., p_k\} \subset$ TRANS.

In the following we give examples for the construction of well-formedness rules (WF) needed for the identification of minimal sets of faulty transition conditions in the given example PFSA.

First, we give an example for the construction of an *accessibility* rule related to the

[3]The corresponding variable domains are depicted in Figure 4.

transition (q_2, $\{c_6, c_7\}$, q_4) of our example PFSA (see Example 2).

Example 2 (well-formedness rules for accessibility): WFaccessibility ((q_2, $\{c_6$, $c_7\}$, q_4)) = $\{\mathcal{ACT}(p_1) \vee \mathcal{ACT}(p_2) \vee \mathcal{ACT}(p_4) \vee \mathcal{ACT}(p_5)\}$. □

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 (see Figure 3), i.e., at least one of the variable sets $\{kl_1, id_1, av_1, sl_1\}$, $\{kl_2, id_2, av_2, sh_2\}$, $\{kl_4, id_4, av_4, sh_4\}$, $\{kl_5, id_5, av_5, sl_5\}$ must be active in a solution for the CSP defined by (STAT, WF, TRANS).

Second, we give an example for the construction of an *extensibility* rule related to the consistent path p = [(q_0,$\{c_2\}$,q_1)].

Example 3 (well-formedness rules for extensibility): WFextensibility ([(q_0, $\{c_2\}$, q_1)]) = $\{\mathcal{ACT}(p_4) \vee \mathcal{ACT}(p_5) \vee \mathcal{ACT}(p_6) \vee \mathcal{ACT}(p_7)\}$. □

This rule denotes that fact that at least one of the paths p_4, p_5, p_6, p_7 must be extensible in the state q_1, i.e., the corresponding variables must be active.

Third, we give an example for the construction of a well-formedness rule related to the *determinism* of the state q_1.

Example 4 (well-formedness rules for determinism): WFdeterminism (q_1) = $\{(kl_7{\neq}kl_4 \vee aw_7{\neq}aw_4 \vee \mathcal{IACT}(p_4) \vee \mathcal{IACT}(p_7)) \wedge (kl_7{\neq}kl_5 \vee aw_7{\neq}aw_5 \vee \mathcal{IACT}(p_5) \vee \mathcal{IACT}(p_7)) \wedge (kl_7{\neq}kl_6 \vee aw_7{\neq}aw_6 \vee \mathcal{IACT}(p_6) \vee \mathcal{IACT}(p_7))\}$. □

This rule denotes that fact that each instantiation of $\{kl_7, aw_7\}$ must be different from the possible instantiations of variables of the paths p_4, p_5 and p_6. Figure 7 depicts a simple example which demonstrates the application of this type of well-formedness rule. This rule allows to determine whether there exists an extension of a given PFSA which is well-formed regarding the determinism property. This is useful in situations where the diagnosis process assumes that a certain transition is faulty (in Figure 7: $\{c_1, c_2\}$ are assumed to be faulty). If this is the case, we have to identify a substitution of the faulty transition conditions which fulfills the given set of well-formedness rules. For our simple example of Figure 7 we have to define the following determinism well-formedness rule: $(x_1 \neq x_2 \vee \mathcal{IACT}(p_1) \vee \mathcal{IACT}(p_2)) \wedge (y_1 \neq y_2 \vee \mathcal{IACT}(p_1) \vee \mathcal{IACT}(p_2))$. In the case of Figure 7.a such an extension exists for the transition conditions $\{c_1, c_2\}$ (e.g., $\{c_1 : x = 1 \wedge y = 1, c_2 : x = 2 \wedge y = 2\}$), the

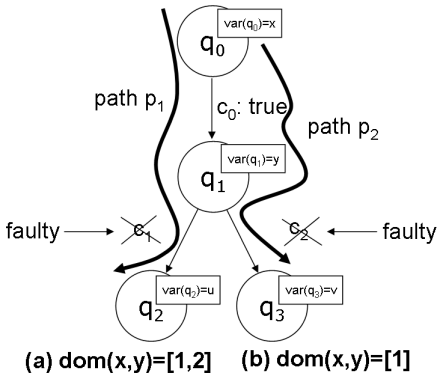

Figure 7: Calculation of extensions for a PFSA (the transition conditions c_1, c_2 are assumed to be faulty): in case (a) an extension is possible (e.g., $\{c_1 : x = 1 \wedge y = 1$, $c_2 : x = 2 \wedge y = 2\}$), case (b) does not allow the calculation of an extension.

case of Figure 7.b does not allow such an extension which means that no diagnosis can be found in this case.

An example for the definition of a transition condition (TRANS) is the following. We represent the transition condition c_1 of our example PFSA.

Example 5 (TRANS for PFSA): $\{c_1$: (kl$_1$ = beginner $\vee$ $\mathcal{IACT}(kl_1)) \wedge$ (kl_2 = beginner $\vee$ $\mathcal{IACT}(kl_2$)) $\wedge$ (kl_3 = beginner $\vee$ $\mathcal{IACT}(kl_3))\} \subseteq$ TRANS. $\square$

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 $\{kl_1, kl_2, kl_3\}$ must be *beginner* if the corresponding variable is active.

Given a specification of (SD, TRANS), a PFSA Diagnosis is defined as follows.

Definition 8 (PFSA Diagnosis): A PFSA Diagnosis for a PFSA Diagnosis Problem (SD, TRANS) is a set S $\subseteq$ TRANS s.t. SD $\cup$ TRANS - S consistent. $\square$

Given a PFSA Diagnosis Problem (SD, TRANS), a Diagnosis S for (SD = STAT $\cup$ WF, TRANS) exists under the reasonable assumption that STAT $\cup$ WF is consistent. Assuming that SD = STAT $\cup$ WF is inconsistent, it follows from the definition of a diagnosis S that SD $\cup$ TRANS-S is inconsistent $\forall$S $\subseteq$ TRANS. Assuming that SD $\cup$

TRANS-S is consistent, it follows that STAT ∪ WF is consistent.

The calculation of diagnoses is based on the concept of minimal conflict sets [16].

Definition 9 (Conflict Set): a Conflict Set (CS) for (SD, TRANS) is a set {c_1, c_2, ..., c_n} ⊆ TRANS, s.t. {c_1, c_2, ..., c_n} ∪ SD is inconsistent. CS is minimal iff ¬∃ CS' ⊂ CS : conflict set (CS'). □

The algorithm for calculating a set of minimal diagnoses for a process definition is the following (Algorithm 1).

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}.

The labelling of the search tree (Hitting Set Directed Acyclic Graph - HSDAG) is based on the labelling of the original HSDAG (details on the calculation of hitting sets can be found in [21]). 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). 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}, i.e., {c_2, c_9} have to be changed in order to make to PFSA consistent w.r.t. to the given set of well-formedness rules.

Applicability for Interactive Settings. The process flow diagnosis concepts presented in this paper have been implemented as a component of the knowledge-based recommender development environment presented in [3, 9]. Our approach complements the existing knowledge acquisition interface with a set of intelligent mechanisms allowing the automated identification of faulty transition conditions in process definitions, i.e., does not fundamentally change the structure of the knowledge acquisition interface. Typically, process definitions include between 15 and 40 transition conditions (see Table 1). Faulty PFSA definitions typically exhibit 2-5 conflicts which makes our approach applicable to settings where a knowledge engineer is interactively

transitions	2 conflicts	3 conflicts	4 conflicts	5 conflicts	10 conflicts
5	0.05	0.071	0.12	0.16	-
10	0.141	0.171	0.251	0.321	0.36
20	0.401	0.621	1.152	1.343	2.268
30	0.891	1.092	1.421	1.692	2.826
40	1.352	1.791	2.096	2.813	3.912

Table 1: Performance of PFSA Diagnosis (in secs).

developing a recommender application (see Table 1). Conflicts of this magnitude occur in settings where knowledge engineers develop recommender process definitions who have a basic understanding of the well-formedness properties discussed in this paper.

The presented debugging concepts are a first approach to make development and maintenance of recommender process definitions more effective. 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 (specific type of accessibility well-formedness rule). 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 first.

4 Empirical Analysis

Analysis Approach. We have conducted an experiment with the goal to highlight and quantify potential reductions of development and maintenance efforts facilitated by our process flow debugging environment. For the experiment we have defined three example (faulty) process definitions (pd_1, pd_2, pd_3) with an increasing complexity regarding the number of states and transition conditions. The types of erroneous transition conditions are similar to those occurring in industrial recommender development projects [9]. On the basis of these process definitions, participants of the experiment had to identify solutions for the following types of tasks:

1. Diagnosis task (d) - identify a minimum cardinality set of faulty transition conditions (without an automated debugging support): participants had to provide an answer to the question which minimum cardinality set S of faulty transition conditions has to be removed from the set of given transition conditions in the PFSA, such that the transition conditions in the resulting PFSA are consistent with the well-formedness rules (SD $\cup$ TRANS - S has to be consistent).

2. Repair task (r) - select consistent repair actions: the participants had to select a maximum set of consistent repair actions T out of a proposed set of (partially faulty) repair actions, s.t. SD $\cup$ TRANS - S $\cup$ T consistent.

The participants of the experiment were interacting with an online questionnaire where each participant had to solve the given tasks autonomously. The time efforts needed to complete a given task were stored in an underlying knowledge base. Participants were randomly assigned to one of the two testgroups shown in Table 2. For each process definition, the members of one testgroup had to solve a diagnosis and repair task whereas the members of the other testgroup had only to solve a corresponding repair task. Following this approach, we were able to compare process definition maintenance efforts with and without a corresponding debugging support.

The participants (Computer Science students at the Klagenfurt University) of the study (n=40) had knowledge engineering experiences in the development of recommender applications. The types of error identification tasks which had to be solved

	pd_1	pd_2	pd_3
$testgroup_1(n = 20)$	pd_{1dr}	pd_{2r}	pd_{3dr}
$testgroup_2(n = 20)$	pd_{1r}	pd_{2dr}	pd_{3r}

Table 2: Assignment of error identification and repair tasks to test groups (*dr* = manual diagnosis and repair, *r* = automated diagnosis and manual repair), e.g., $testgroup_1$ had a diagnosis and repair task for process definition pd_1.

within the scope of the experiment were similar to those tasks knowledge engineers have to solve within the scope of commercial projects. The similarity of the participants' education level as well as the similarity of the posed error identification tasks to real-world settings clearly show the external validity (similarity-based) of the results of our experiment. The participants of the study had to solve the given error identification and repair tasks autonomously. For the comparison of time efforts related to diagnosis and repair tasks, we applied an independent (two-sample) t-test (parametric statistical test), which is applicable since the error identification and repair times are normally distributed and the effort data sets for the two testgroups are independent.

Result. Our experiment clearly shows the applicability of our debugging approach in terms of time savings related to development and maintenance processes. The goal of our analysis was to investigate differences in time efforts related to the identification and repair of faulty process definitions depending on whether a corresponding automated debugging support was available or not.

Hypothesis: *Automated debugging support for process definitions leads to significant time savings in the detection and repair of faulty transition conditions.*

The average *repair effort* for process definition pd_1 was 71.458 seconds (std. dev. 28.915 seconds), the corresponding average *diagnosis and repair* effort was 110.750 seconds (std. dev. 63.704 seconds). Similar results have been obtained by examining the remaining example (faulty) process definitions which allows us to accept the defined hypothesis.

Process definition	$mean_{dr}$(sec.)	t-score	p	$mean_r$ (sec.)
pd_1	110.750	2.263	0.034	71.458
pd_2	155.417	2.277	0.033	78.417
pd_3	246.167	3.308	0.003	72.458

Table 3: Reduced error detection/repair times with debugging support ($mean_r$) compared to error detection/repair times without debugging support ($mean_{dr}$).

5 Related Work

Colloborative filtering [13], content-based filtering [20] and knowledge-based recommendation [1, 3, 30] are the three basic approaches to the implementation of a recommender application. 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). Using knowledge-based approaches, the relationship between customer requirements and offered products is explicitly modeled [7] - compared to collaborative and content-based filtering, knowledge-based approaches do exploit deep knowledge about the application domain.

Such knowledge representations are the major precondition for the application of model-based diagnosis techniques [21, 12]. An overview of the application of model-based diagnosis techniques in software debugging can be found in [28]. The complexity of configuration knowledge bases motivated the application of model-based diagnosis (MBD) [21] in knowledge-based systems development [6]. Similar motivations led to the application of model-based diagnosis in technical domains such as the development of hardware designs [11], onboard diagnosis for automotive systems [24] and in software development [18]. The work presented in this paper has a special relationship to the work presented in [6]. [6] focus on the identification of faults in configuration knowledge bases, where a set of test cases is used to induce conflicts with a configuration knowledge base. In contrast to this work, we provide an abstract representation of finite state models which is checked against a set of well-formedness

rules, i.e., well-formedness rules correspond to test cases presented in [6]. Test cases used in [6] are, e.g., configurations calculated by previous versions of configuration knowledge bases. In many cases, such test cases have to be defined by domain experts which makes testing and debugging a time-consuming task. Although our debugging approach for recommender process definitions allows the specification of test cases as well (input sequences which have to be accepted by the process definition), one of the major strengths of the approach is that well-formedness rules (generic test cases) can be automatically derived from given process definitions. In contrast to the evaluation presented in this paper, [4] contains an analysis of the effectiveness of recommender knowledge base diagnosis concepts.

The representation of recommender processes in the form of finite state representations is discussed in [3, 10]. 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. The automated debugging of such process definitions on the basis of MBD [21] has so far not been discussed in the literature. Related work can be found, e.g., in [17], where an algorithm for checking the consistency of workflow definitions is presented. Compared to our work, [17] focus on assuring workflow properties such as *each component of the workflow has at least one output parameter* or *all components of the workflow are executable*. Compared to these basic types of consistency checks, our work provides intelligent mechanisms which effectively support the automated indication of potential sources of inconsistencies.

A number of studies have been conducted related to the evaluation of Knowledge Acquisition (KA) tools. A corresponding overview on those approaches can be found in [29]. All those studies concentrate on different aspects of user behaviour when interacting with a certain knowledge acquisition environment. Examples for hypotheses tested in these experiments are: *all users would employ the same set of commands even if told nothing in advance about the modeling environment*; *users will make less mistakes during KA tasks using KA tools*; or *users will be able to complete a KA task in less time using KA tools*. Compared to these evaluations, our experiment focuses on

the specific aspect of debugging support in the context of recommender user interface development which to our knowledge has not been conducted so far.

6 Conclusions and Future Work

Automated debugging support for the design of recommender user interfaces can significantly reduce related development and maintenance efforts. In this paper we have presented concepts supporting the identification of minimal sets of faulty transition conditions in finite state models of recommender user interfaces. Although this paper focused on the development of recommender user interfaces, the presented approach is not restricted to this domain but is generally applicable to settings where a finite state model of a user interface is given. The proposed approach has been implemented as part of a commercially available recommender development environment.

References

[1] R. Burke. Knowledge-based Recommender Systems. *Encyclopedia of Library and Information Systems*, 69(32), 2000.

[2] R. Burke. Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

[3] A. Felfernig. Koba4MS: Selling Complex Products and Services Using Knowledge-Based Recommender Technologies. In G. Müller and K. Lin, editors, *7^{th} IEEE International Conference on E-Commerce Technology (CEC'05)*, pages 92–100, Munich, Germany, 2005.

[4] A. Felfernig. Reducing Development and Maintenance Efforts for Web-based Recommender Applications. *International Journal of Web Engineering and Technology*, page to appear, 2006.

[5] A. Felfernig, G. Friedrich, D. Jannach, and M.Zanker. An Integrated Environment for the Development of Knowledge-Based Recommender Applications. *International Journal of Electronic Commerce*, 11(2):11–34, 2006.

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

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

[8] A. Felfernig, K. Isak, and C. Russ. Knowledge-based Recommendation: Technologies and Experiences from Projects. In G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, editors, *17^{th} European Conference on Artificial Intelligence (ECAI06)*, pages 632–636, Riva del Garda, Italy, 2006.

[9] A. Felfernig and A. Kiener. Knowledge-based Interactive Selling of Financial Services using FSAdvisor. In *17^{th} Innovative Applications of Artificial Intelligence Conference (IAAI'05)*, pages 1475–1482, Pittsburgh, Pennsylvania, 2005.

[10] A. Felfernig and K. Shchekotykhin. Debugging User Interface Descriptions of Knowledge-based Recommender Applications. In *Workshop Notes of the IJCAI'05 Workshop on Configuration*, pages 13–18, Edinburgh, Scottland, 2005.

[11] G. Friedrich, M. Stumptner, and F. Wotawa. Model-based diagnosis of hardware designs. *AI Journal*, 111(2):3–39, 1999.

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

[13] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. Riedl. Evaluating Collaborative Filtering Recommender Systems. *ACM Trans. on Information Systems*, 22(1):5–53, 2004.

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

[15] B. Jiang, W. Wang, and I. Benbasat. Multimedia-Based Interactive Advising Technology for Online Consumer Decision Support. *Communications of the ACM*, 48(9):93–98, 2005.

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

[17] J. Kim, M. Spraragen, and Y. Gil. An Intelligent Assistant for Interactive Workflow Composition. In *International Conference on Intelligent User Interfaces (IUI-2004)*, pages 125–131, Madeira, Portugal, 2004.

[18] Mateis, M. Stumptner, and F. Wotawa. Modeling Java programs for diagnosis. In *14th European Conference on Artificial Intelligence*, pages 171–175, Berlin, Germany, 2000.

[19] M. Montaner, B. Lopez, and J. De la Rose. A Taxonomy of Recommender Agents on the Internet. *Artificial Intelligence Review*, 19:285–330, 2003.

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

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

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

[23] F. Ricci, A. Venturini, D. Cavada, N. Mirzadeh, D. Blaas, and M. Nones. Product Recommendation with Interactive Query Management and Twofold Similarity. In 5^{th} *International Conference on Case-Based Reasoning (ICCBR 2003)*, pages 479–493, Trondheim, Norway, 2003.

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

[25] S.Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In 8^{th} *National Conference on Artificial Intelligence*, pages 25–32, Detroit, MI, 1990. MIT Press.

[26] B. Smyth, E. Balfe, O. Boydell, K. Bradley, P. Briggs, M. Coyle, and J. Freyne. A Live User Evaluation of Collaborative Web Search. In 19^{th} *International Joint Conference on Artificial Intelligence*, pages 1419–1424, Edinburgh, Scotland, 2005.

[27] M. Stolze, S. Field, and P. Kleijer. Combining Configuration and Evaluation Mechanisms to Support to Selection of Modular Insurance Products. In 8^{th} *European Conference on Information Systems*, pages 858–865, 2000.

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

[29] M. Tallis and Y.G. Kim. User studies of knowledge acquisition tools: methodology and lessons learned. In *KAW-99*, 1999.

[30] C. Thompson, M. Göker, and P. Langley. A Personalized System for Conversational Recommendations. *Journal of Artificial Intelligence Research*, 21:393–428, 2004.

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

[32] G. VanNoord and D. Gerdemann. Finite State Transducers with Predicates and Identities. *Grammars*, 4(3):263–286, 2004.