

# Automatic Software Customization: A Methodology for Learning Individual Preferences

Bowen Hui  
Dept. of Computer Science  
University of Toronto  
bowen@cs.utoronto.ca

## 1 Problem and Motivation

In order to accommodate a wide variety of user needs, skills, and preferences, today’s software is typically packed with functionality aimed at suiting all types of users. As a result, software interfaces are complicated, functionalities are unexplored (and hence unused), and users are dissatisfied. Hence, providing software that is *personalized* to individuals has the potential to increase work productivity, ease interaction with software, minimize software development cost, and increase the user’s sense of ownership of a product. To achieve this goal, researchers have proposed *pre-configuration* methods where developers customize products for specific clients [7], *adaptable* methods that let users tailor the software themselves [5], or *adaptive* methods that automatically learn user-specific information and then customize the software accordingly [6]. The third approach, when successful, completely avoids manual tweaking of products and the requirement that users know their own preferences and invest time and effort tailoring the software. This work proposes a design methodology for supporting the adaptive approach, where a novel set of user features are assessed within a new framework that models costs and tradeoffs of system actions.

In this work, we consider customization in a broad sense, to include infrequent or one-time changes (such as turning off certain functionality, or choosing a preferred font size) as well as changes that may be continually updated as the user works (such as changing the frequency and amount of automatic help that is offered to the user, depending on the user’s current *state*). We also distinguish between three dimensions along which customization is performed: (i) presentation of information, (ii) function availability, and (iii) navigation structure. Consider as an example a text editor that adaptively predicts what the user is typing and offers help in the form of word completion. Rather than offering word completion for every character typed, the system adapts the availability of help (an example of (ii)), only offering completion when the benefits of help are estimated to outweigh the costs (due to interruption, distraction, occlusion caused by popping up a suggested completion, etc.) An example of (i) in this editor would be changing where a suggested completion is displayed (e.g. at the cursor, in the margin, etc.), and an example of (iii) would be changing the organization of menus in the editor. In general, any of these customizations carries a potential benefit and cost, the balance of which should be estimated to determine if and when to perform the adaptation.

Automatically customizing software on the user’s behalf is complicated by the uncertainty inherent in the domain. Since it is not possible to know the user’s preferences for certain, the system need to probabilistically assess them. This is especially challenging since user preferences depend not only on the objective task at hand, but also on the user’s internal state. For example, a user who is *frustrated* may not accept word completion help even when it is accurate, while a user who is *needy* may even accept partially correct help. How might we estimate a user’s current level of frustration or neediness? What other kinds of user features influence whether the user accepts, or even considers, automated help? If the user is easily distracted, popping up a suggestion risks deterring the user from their goal. The value of automatic help may also depend on the user’s tendency to prefer working independently versus accepting help from others. On the other hand, if the user has trouble writing and needs help, the value of help increases. We conjecture that user features related to these factors are necessary in developing a *user model* that allows the system to probabilistically assess the current user state — the user’s current attitudes and receptiveness toward the system. Doing so enables the system to quantify the uncertainty in its beliefs about the user.

Furthermore, since it is not always possible to completely satisfy all the user’s preferences, the system must be able to model tradeoffs in the face of uncertainty. For example, if the system had to decide between making a highly

beneficial suggestion now when the user happened to be frustrated, versus making a less beneficial suggestion but offering it when the user is less frustrated, how should the system decide what to do? Asking the user what she wants is one possibility, but there are many variables and decisions involved and the system should not resort to explicit questions each and every time. Even if the system could ask unlimited questions to resolve the uncertainty, the user may not even know her own preferences, because the preferences themselves may change over time. Moreover, the system needs to model existing interaction phenomena, such as the cost of *interruption* and the value of *perceived savings*, so that user preferences are expressed in terms of the current interaction settings, subject to the current user state. In order to make the right decision, the system must model the costs and benefits of each action, with respect to interaction factors and the probabilistic assessment of user features.

To fully handle complex decision making problems encumbered by a high degree of uncertainty, we turn to decision theory coupled with probabilistic user modeling techniques. We advocate an explicit model of user features that influence the user’s interaction style with the system, such as the user’s levels of frustration, neediness, distractibility, and independence. Consider these examples: someone who is easily distracted may find automated assistance costly because it prevents the user from completing the task; someone who currently needs help with a difficult task may benefit greatly from partial suggestions that helps the user identify the next steps; someone who is generally dependent may not mind receiving imperfect suggestions as much as someone who is highly independent; someone who is frustrated with the system now is likely to become more frustrated with further interruptions and suggestions. The relevant user features illustrated by these examples are distractibility, neediness, independence, and frustration. Knowing the user state allows the system to make more informed decisions about how best to help her. Although a few researchers suggest modeling these kinds of user features, they do not provide a mechanism to learn them [2,8,4]. One possible explanation is that inferring the user state as an online task is computationally expensive in the large models which are often needed to provide a more accurate assessment. As our first contribution, we argue that modeling user features is necessary and we provide an abstraction of events that enables fast computation using a standard probabilistic inference procedure called *belief state monitoring*.

In a number of settings, decision-theoretic models have been adopted to allow a system to make the right decisions based on principled tradeoffs [6,8,4,1]. We adopt this general perspective, but tailor our approach so that decisions are influenced by the system’s beliefs about the generally evolving user state. As our second contribution, we present a general decision-theoretic system architecture for automatic software customization. This framework ascribes the economic notion of *expected utility* to the system’s actions with respect to the probabilistic user state, so that the system always chooses actions with the maximum expected utility (MEU).

In the following, we present the background on the techniques used in our work in Section 2 and present our methodology and system implementation in Section 3. In Section 4, we describe our results using the text editor example and summarize our findings.

## 2 Background and Related Work

Decision theory is the combination of probability theory and utility theory for making decisions under uncertainty. In the context of software customization, we apply probability theory to infer the user state and apply utility theory to quantify the usefulness of system actions. Together, the system takes the most useful action for the particular user at hand.

To probabilistically assess the user state, we use a *Dynamic Bayesian Network* (DBN) [3]. A DBN is a graphical representation of random variables represented by nodes and causal dependencies represented by directed edges. Each node has an associated *conditional probability distribution* (CPD), defining the stochastic dependencies of the node’s values based on its parents’ values. DBNs have a temporal dimension so that dependencies over time can be represented as part of the model’s dynamics.

Figure 1 illustrates an example of a DBN with two discrete time slices, where the dashed line indicates the separation in time. This figure shows that someone who *needs help* ( $N$ ), causes actions such as erasing text, browsing, and pausing, while an *easily distracted* ( $D$ ) person causes browsing and pausing actions. A table is used to define the prior distribution  $Pr(D)$ , indicating that people are typically not easily distracted (with probability  $Pr(D = no) = 0.55$ ), with lower chances of some people being somewhat distracted ( $Pr(D = somewhat) = 0.35$ ) and very easily distracted ( $Pr(D = yes) = 0.1$ ). A CPD is defined for *Erase*, as a function of its parent node. Since its parent has 3 possible values, each row in this table corresponds to a distribution summing up to 1. Note that both  $N$  and  $D$  have temporal dependencies. Roughly speaking, a person’s previous neediness level influences the current neediness level, with some probability of it escalating or decreasing (e.g., according to how difficult

the task becomes). However, a person’s distractibility tends to lessen over time, so that the person slowly becomes “undistracted”. In the same spirit, our system’s user model follows this DBN example, but extended to model the user’s frustration and independence.

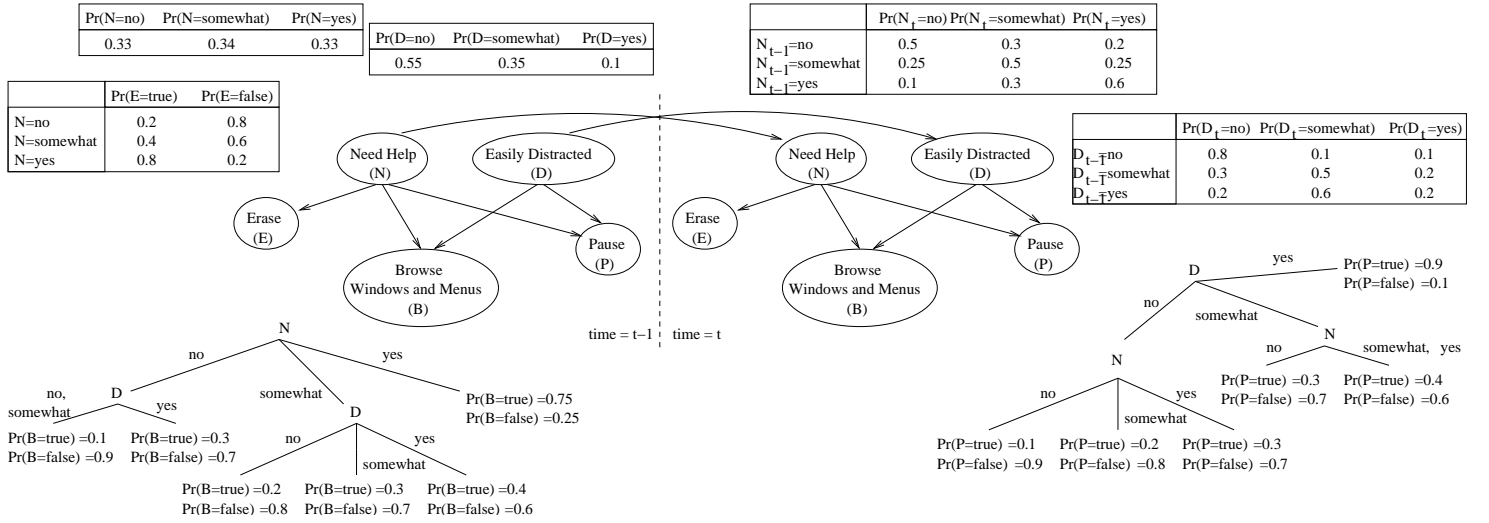


Figure 1: An example of a two-slice discrete Dynamic Bayesian Network (DBN).

It is easy to see that when a node depends on many parents, a table representation becomes exponentially large. One solution is to use a decision tree representation, such as the one shown for *Browse* ( $B$ ). In this tree, 3 values of  $N$  are first considered. When  $N = no$ , we consider the values of the second parent,  $D$ . On the other hand, when  $N = yes$ , we aggregate all the values of  $D$  into one scenario so that  $Pr(B = true) = 0.75$ . The remaining tree is interpreted in this fashion. The advantage of using decision trees is that shared structure does not need to be replicated, because they can branch to the same distribution.

Moreover, DBNs embody the assumption that each node is independent of its non-descendants given its parents. Taking all the local conditional distributions in the network, a DBN specifies the joint probability of all the random variables in the network as the product of the local conditional distributions. This result yields a compact representation and efficient computation for inference procedures.

Through a series of interactions, the system observes user behaviour (such as erasing text, browsing, pausing) and infers the user’s current neediness and distractibility levels. Formally, the system computes  $Pr(N_t, D_t | Ev_{1:t})$ , which is the joint probability of  $N$  and  $D$  at the current time  $t$ , given the entire history of observed evidence from time 1 to  $t$ . This inference step is called *filtering* and various algorithms are available to compute it in time linear to the size of the model. Note that  $Pr(N_t, D_t | Ev_{1:t}) \propto Pr(Ev_t | N_t, D_t) * Pr(N_t, D_t | Ev_{1:t-1})$ , where the first term corresponds to a single inference step in the current time slice only, and the second term corresponds to maintaining a prior distribution from the previous time slice. This two-part decomposition makes direct use of the DBN structure. In this way, the system no longer has to maintain the full history. Modeling the joint state of  $N$  and  $D$  as the system’s belief and inferring its distribution online is referred to as *belief state monitoring*. With large, complicated models, belief state monitoring is intractable. However, we show that our model affords good online performance with exact inference.

The second aspect to decision making under uncertainty is the use of utility theory. In economics, *utility* is a term expressing one’s “happiness” via a *utility function* ( $U$ ), which maps outcomes,  $O$ , to reals. These outcomes describe situations that a person has preferences over, such as the outcome of having a bundle of goods or being in an emotional state. Therefore, an outcome with a higher utility is preferred (because it yields greater happiness). In the real world, we cannot predict perfectly whether an outcome takes place. Thus, the notion of *expected utility* defines the utility of an outcome in expectation of its occurrence, i.e.,  $EU(O) = Pr(O)U(O)$ . Imagine an intelligent agent with the above user model and a set of actions  $A$ . We define the action’s utility with respect to the agent’s environment state jointly as  $U(S, A)$ . If the state is partially observable, i.e., the agent cannot tell with absolute certainty the exact state it is in, then the agent’s action utility is taken in expectation of the distribution of the state,  $EU(A) = \sum_S Pr(S)U(S, A)$ . In the user model above, this computation becomes  $EU(A) = \sum_{N,D} Pr(N, D)U(N, D, A)$ . This formulation gives us a utility function with user states as param-

ters. Expressing action utility in this manner enables us to model user preferences as a function of the user’s internal state and keep up with evolving preferences by updating the system’s belief over user states.

### 3 Approach and Uniqueness

We take a decision-theoretic approach to design a general methodology for automatic software customization. In bottom-up fashion, we derive a set of factors that influence the user’s acceptance of automated help. We formalize the results of the derivation as a DBN user model with a set of user variables, behavioral observations, and interface states. Examples of behavioral observations include jamming into the keyboard (showing frustration), pausing (showing neediness and distraction), and rejecting highly useful help (showing independence).

More formally, we define the *user state* as the joint outcome of the user’s frustration level toward the system ( $F$ ), the user’s neediness level toward the difficulty of the task ( $N$ ), the user’s tendency to be easily distracted ( $TD$ ) and the user’s tendency to work independently ( $TI$ ) on a computer. Thus, the system’s belief of the user state is  $Pr(F, N, TD, TI)$ . Variables  $TD$  and  $TI$  are *static*, reflecting specific user traits that do not change over the course of an interaction session. In contrast,  $F$  and  $N$  are *transient*, reflecting user attitudes that change frequently during a session. How these transient variables evolve can also be modeled by assuming additional static user traits. For this purpose, we propose latent variables  $TF$  and  $TN$ , representing the user’s tendencies to be frustrated and needy. These influence the stochastic evolution of  $F$  and  $N$ . We define a *user type* to be the state of all static user traits, represented as  $\{TF, TN, TD, TI\}$ . Learning the user state enables the system to adapt its behavior to best suit the current user state. Learning the user type enables the system to create a long term profile that can be transferred to other applications.

To quantify the usefulness of actions, we define a utility function to represent the value of automated help. To define an individualistic utility function, we parameterize the utility function with user variables. For example, auto-completing the current word may save the user 10 keystrokes. However, to a needy user, this objective savings has higher value, while to an independent user, it has lower value. With an objective savings,  $Qual$ , the perceived savings is defined relative to the user features as  $PS(F, N, TD, TI, Qual)$ . Also, every time the system makes a suggestion, there is an associated cost of interruption that varies depending on the user. For example, a highly frustrated user would assign a higher interruption cost to a pop-up than a needy user would. Thus, we define an interruption cost function as  $CI(F, N, TD, TI)$ . Together, we define the overall utility function as  $U(F, N, TD, TI, Qual) = PS(F, N, TD, TI, Qual) + CI(F, N, TD, TI)$ . In this way, the system can take the expected utility of each action by computing  $EU(Qual) = \sum_{F, N, TD, TI} Pr(F, N, TD, TI)U(F, N, TD, TI, Qual)$  and choose the action with the highest expected utility.

Rather than developing specialized techniques, our objective is to define a generic design methodology that supports the development of software that adapts to the user, as the user’s needs, skills, and preferences are revealed through the course of the interaction. Our approach is unique in that the system’s reasoning process is influenced by its belief of the user state, which we model as a composite of personality and affect variables. To successfully infer the user state online, we abstract low-level events into behaviors that are meaningful for designers and manageable for computation. Unlike other user modeling work, we apply belief state monitoring and show that inference is tractable in our prototype. Typically, probabilistic systems choose actions based on the user’s *most likely* goal. In contrast, our system chooses actions based on what is *most useful* for the user, in expectation of the most likely goal. Specifically, the system has a domain model that estimates the most likely goal. In a typing task, the system’s domain model is a language model that returns a distribution of words, representing each word’s probability of being the target word based on what has been typed. Usefulness is quantified as the amount of savings a completion offers the user (e.g., number of keystrokes). This objective quantity is then modified by user features, thus, providing a formal definition of subjective, *perceived savings*. This quantity taken in expectation of the most likely goal, yields an expected perceived savings for each system action. Thus, the system’s action selection process is driven by both how useful an action is and how likely it will help the user achieve her goal.

### 4 Results and Contributions

To demonstrate our methodology, we implemented our system in a text editor that offers word predictions via a pop-up box with  $J$  words, where  $J = 3$  in the experiments. The language model here uses bigrams trained on 40% of the 100 million word British National Corpus. Unlike other word prediction software, our system does not offer completions whenever a letter is typed. Rather, it learns the user’s traits and needs and make suggestions only when

it believes that the user can benefit from them. This methodology is generalizable to more complex software and tasks.

In our prototype, the user variables are discrete, with variables  $F$ ,  $N$ ,  $TD$ , and  $TI$  being tertiary and  $TN$  and  $TD$  being binary. For example,  $F = 1$  denotes that the user is not frustrated,  $F = 2$  the user is somewhat frustrated, and  $F = 3$  the user is very frustrated. Other variables are defined similarly. In total, there are 81 user states and 36 user types.

To assess our user model, we ran simulations by sampling our user model as the simulated user for all 36 types averaged over 100 trials. The test text consisted of unseen sentences (approximately 200 words long) drawn randomly from 10% of the British National Corpus. Belief state monitoring is currently implemented in Matlab 6.5. On average, this computation takes approximately 0.57 second on a Pentium M, 1.2G Hz CPU, 386 MB RAM processor. This performance is reasonable and can be accelerated considerably via simple code enhancements, algorithm approximations, or model abstraction techniques. More importantly, the results show that the system’s beliefs converged to the true type in all 36 cases. The time it took the system to reach convergence varied from about 20 to 150 words. Examples of convergence curves for three user types (as a function of the number of observations, such as characters typed, pause events, browse events, etc.) are shown in Figure 2.

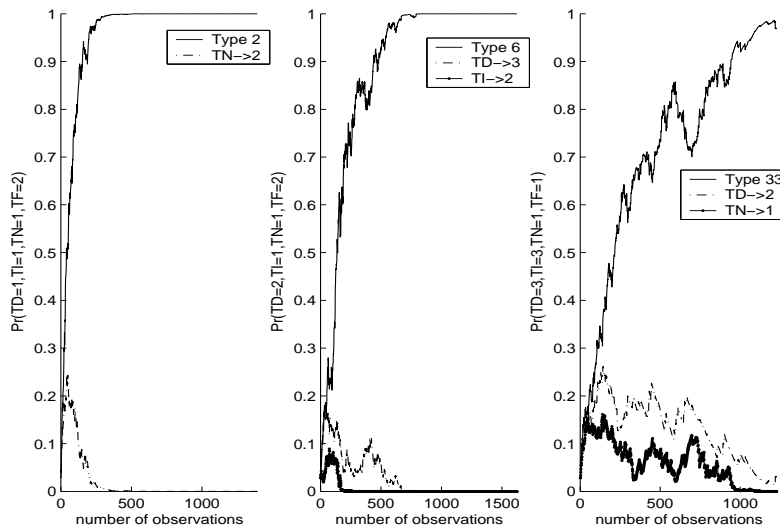


Figure 2: Examples of belief monitoring showing convergence of the true user type with respect to alternative user types, ranging from early convergence (left) to late convergence (right).

The system’s overall utility is defined as the accumulated rewards and costs received throughout the interaction with the user, using the function defined above,  $U(F, N, TD, TI, Qual)$ . Figure 3 shows examples of the patterns of system behavior for different user types. In general, the system’s behavior adapts to user’s responses — more acceptances encourage more suggestions, and vice versa. Across user types, the patterns also show that more needy and dependent types receive higher overall utility, while more frustrated, distractible, and independent types receive lower utility.

For comparison purposes, we conducted experiments with other system policies. We chose two static policies, always pop up suggestion (ALWAYS) and never pop up suggestions (NEVER), and one adaptive policy, pop up suggestion only if  $Qual$  is over 80% (THRESHOLD). We refer to our system policy as MEU.

Table 1 compares average reward per time step for these policies and several representative user types. Generally, ALWAYS outperforms the other policies with dependent users who tend to need help, as shown in the first row of the table. However, it does poorly (often extremely) in all other cases. The second row shows that even with dependent users who are easily distracted or frustrated, the users may benefit more from adaptive policies. In the remaining cases, an independent user, either easily frustrated or easily distracted or neither, benefits most from a system that learns to back off when help is undesired. These cases illustrate that a static policy, such as ALWAYS, or a policy that disregards the user type, such as THRESHOLD, suffers most. Overall, MEU dominates THRESHOLD for 17 of the 36 user types (sometimes quite significantly), while the difference in the other cases are insignificant. Although NEVER receives zero rewards in all cases, it is unable to detect when the user in fact needs help.

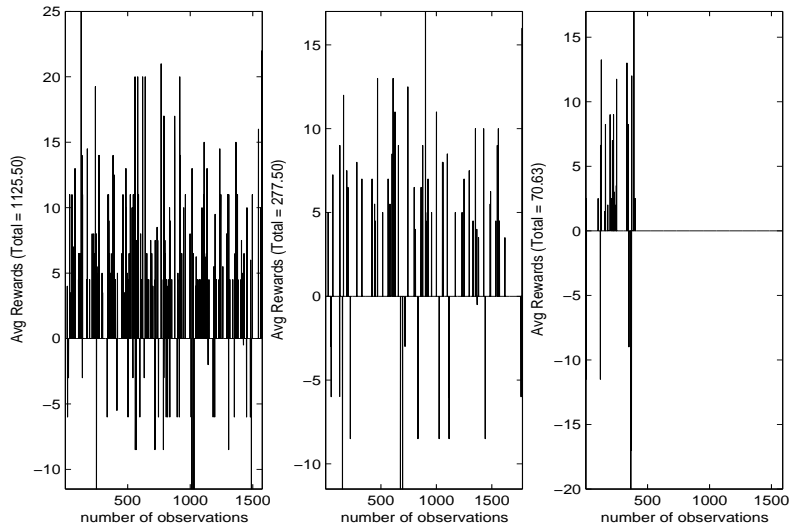


Figure 3: Examples of system behaviors. Left: a user who welcomes help so the system offers them regularly. Middle: a sporadic user with sparse help. Right: a user who rejects help so the system learns to back off.

User Type	ALWAYS	MEU	THRESHOLD	NEVER
{1,2,1,1}	1.64	0.93	0.91	0
{2,1,2,1}	0.46	0.62	0.65	0
{1,1,3,2}	-0.64	0.39	0.31	0
{1,1,1,3}	-10.29	-2.15	-2.29	0
{2,1,1,3}	-10.89	-1.89	-2.93	0
{2,1,3,3}	-6.04	-0.07	-1.43	0

Table 1: Comparison of policies using average rewards by user type {TFTN,TD,TI}.

A small pilot usability experiment was conducted with 4 users comparing the above 4 policies via a Java text editor. Post-questionnaires were used to elicit the true user type. Both the questionnaire and the belief state monitoring identified all 4 users as generally dependent and needy. According to our simulations, these users would prefer ALWAYS over other policies, and indeed, this is what we found. Between the two adaptive policies, all 4 users reported that the quality of suggestions made by the MEU policy were noticeably higher than those made by the THRESHOLD policy. However, in three of the four log files, the percentage of correct suggestions were higher in the THRESHOLD policy. Also, about 20% of accepted suggestions were partially correct, leading to users erasing incorrect endings. This suggests that the users perceive a subjective aspect to the suggestions, which we believe to be the utility of character savings prescribed in our system.

We have outlined a general methodology for incorporating user models in automated assistance that encompasses a wide range of user types. Specifically, we modeled user features explicitly so that they can be inferred and learned over the course of interaction. We demonstrated our approach in the word prediction domain using simulations and usability experiments. Our results show that the model is able to adapt to different user types and to evolving user states during the course of the interaction. The adaptive nature of our system’s policy allows greater reward to be obtained over a wider range of user types than other fixed policies. In the future, we will extend the methodology to encompass a larger range of interaction phenomena, including visual occlusion and disruption, and apply it to a more complicated test-bed, such as PowerPoint.

## References

- [1] J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and A. Mihailidis. A decision-theoretic approach to task assistance for persons with dementia. Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI), 1293-1299, Edinburgh, Scotland, 2005.
- [2] C. Conati, A. Gertner, and K. VanLehn. Using Bayesian networks to manage uncertainty in student modeling. User Modeling and User-Adaptive Interaction, 12(4):371-417, 2002.
- [3] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. Computational Intelligence, 5(3):142-150, 1989.
- [4] K. Gajos and D.S. Weld. SUPPLE: Automatically generating user interfaces. In Proceedings of the International Conference on Intelligent User Interfaces (IUI), 93-100, Madeira, Portugal, 2004.
- [5] J. McGrenere, R.M. Baecker, and K.S. Booth. An evaluation of a multiple interface design solution for bloated software. In Proceedings of ACM CHI Letters 4(1), 163-170, 2002.
- [6] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The Lumière Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users. In Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI), 256-265, Madison, WI, 1998.
- [7] B. Hui, S. Liaskos, and J. Mylopoulos. Requirements Analysis for Customizable Software: A Goals-Skills-Preferences Framework. In Proceedings of the 11th IEEE International Conference on Requirements Engineering (RE), 117-126, Monterey Bay, CA, 2003.
- [8] A. Jameson, B. Großmann-Hutter, L. March, R. Rummer, T. Bohnenberger, and F. Wittig. When actions have consequences: Empirically based decision making for intelligent user interfaces. Knowledge Based Systems, 14(1-2):75-92, 2001.